INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DISEÑO Y VALIDACIÓN DE UN PROTOCOLO DE AUTENTICACIÓN EN UN SISTEMA DE MANUFACTURA.

TESIS QUE PARA OPTAR EL GRADO DE MAESTRO EN CIENCIAS COMPUTACIONALES PRESENTA

EDUARDO MEZA CARMONA

Asesor: Dr. Oscar Hernández Pérez

Comité de tesis: Dr. Luis Ángel Trejo Rodríguez
Dr. Raúl Monroy Borja

Atizapán de Zaragoza, Estado de México, octubre del 2004.

DEDICATORIA.

A mis padres.
Por toda la paciencia que me han tenido
y el apoyo que me han dado.

CONTENIDO

INDICE DE FIGURAS

INDICE DE TABLAS

1 INTRODUCCIÓN

Desde la antigüedad, las personas han desarrollado diversas técnicas para comunicarse a grandes distancias. Las primeras civilizaciones se establecían en lugares cercanos, por ejemplo varias tribus en el mismo valle, donde las comunicaciones escritas y verbales eran transmitidas rápidamente. Cuando las tribus empezaron a establecerse en zonas más amplias fue necesario crear otros métodos para establecer la comunicación con tribus en lugares más distantes. Como ejemplos de estos métodos se pueden citar las señales de humo y palomas mensajeras [29].

Cuando Samuel Morse inventó el telégrafo en 1838 [30], se inició una nueva era en la comunicaciones, se inició la era de las comunicaciones eléctricas. En los años siguientes, el desarrollo de los sistemas de comunicación que empleaban señales eléctricas se hicieron más sofisticados lo que originó la invención de otros sistemas de comunicación como el teléfono, la radio, la televisión, el radar y los enlaces de microondas.

Casi un siglo después de la invención del telégrafo se empiezan a utilizar las primeras computadoras, en la década de los 50's. Éstos eran sistemas aislados que ocupaban cuartos enteros en donde solo personal muy especializado las podía utilizar. No existía una comunicación interactiva entre los usuarios y las computadoras, todo el trabajo tenía que ser capturado en tarjetas perforadas y de esta manera se introducían a los sistemas de cómputo para realizar el trabajo por lotes, en donde el resultado era obtenido a través de una impresora [30].

A pesar de las dificultades que tenía el manejo de las primeras computadoras éstas eran muy útiles, ya que permitían realizar operaciones complejas (matemáticas en la mayoría de los casos) y/o operaciones repetitivas de manera más rápida y libre de errores.

En la década de los años 60's, con los avances en los sistemas de transmisión de datos así como en las máquinas para escribir información, se desarrollaron terminales tontas[1]. Estas terminales eran utilizadas para comunicarse con el sistema de procesamiento central, en donde los usuarios escribían los comandos, éstos eran enviados al sistema de procesamiento y se mostraban los resultados en la pantalla de la terminal. Por este motivo surge la necesidad de crear las primeras

[1] Computadora que no tenía procesador ni memoria, sino solamente un monitor y teclado así como un dispositivo de comunicación que servía como enlace con el sistema de procesamiento y recursos.

redes para conectar varias terminales tontas a un sistema de cómputo que se encargaba de procesar las instrucciones [30].

En los años 80's, con la aparición de los circuitos integrados, la tecnología mejoró los circuitos que se encargaban del procesamiento de la información y de las instrucciones lo que permitió que las computadoras empezaran a reducir su tamaño y se pudieran desarrollar las primeras computadoras personales. Los primeras organizaciones en tener una computadora personal fueron las escuelas y, de manera especial, los investigadores que trabajaban en las escuelas. Debido a la necesidad que tenían los investigadores de intercambiar información y resultados que se obtenían de los proyectos en los que trabajaban para que otros investigadores analizaran estos resultados, surge la necesidad de realizar una interconexión de los sistemas de cómputo entre organizaciones.

Las instituciones comerciales también empezaron a hacer uso de las computadoras a finales de los años 70's ya que les permitió realizar operaciones repetitivas de manera más rápida. Como ejemplo se cita el área de contabilidad, donde se puede sistematizar los cálculos de una nómina de manera más rápida y eficiente sin los errores que ocurrían cuando no se utilizaba la computadora. Otra ejemplo de los sistemas de cómputo dentro de las empresas es en el manejo de información de inventarios, donde con la ayuda de las computadoras se pueden generar los reportes de existencias sin necesidad de contar cada uno de los artículos existentes en el almacén.

Uno de los aportes principales al desarrollo de las redes fue realizada por la agencia gubernamental de investigación de Estados Unidos ARPA (*Advanced Research Project Agency*) a principios de la década de los 60's. El objetivo principal de esta agencia era desarrollar un sistema militar de comunicaciones en red que permitiera conectar varias computadoras de manera descentralizada de forma que siguiera funcionando aún en el caso de que una o varias de las computadoras fueran destruidas durante un ataque enemigo. Uno de los responsables del proyecto, el Dr. J. C. R. Licklider [28], fue también uno de los principales promotores para lograr que la tecnología de comunicaciones fuera utilizada para conectar a las universidades dentro de los EUA, por lo que a principios de los años 70's ya se habían interconectado varias universidades.

Con la difusión y utilización de las redes de computadoras dentro del entorno de una institución o entre instituciones, se hizo necesario proteger los recursos que se almacenaban en los sistemas computacionales de la institución. Para este fin, se desarrollaron procedimientos de autenticación de usuarios para asegurar que sólo los usuarios válidos utilizarán los recursos informáticos dentro de la institución.

Como un ejemplo donde se ve claramente lo que es la autenticación de un usuario para proteger la confidencial de los recursos es en el correo electrónico (donde sus recursos son los mensajes que están almacenados en el servidor de correo electrónico). En este ejemplo el usuario que tiene una conexión, mediante una red institucional o Internet, con el servidor de correo electrónico que le pide un nombre de usuario y una palabra clave necesarios para comprobar la autenticidad del usuario permitiéndole o denegándole el acceso a los mensajes almacenados.

En el ejemplo del correo electrónico se puede apreciar la forma de realizar intercambio de información para validar que el usuario es el correcto. Los elementos de seguridad utilizados son débiles debido a que en el correo electrónico no se maneja el cifrado de la cuenta ni de la palabra clave que se transmiten entre el servidor del correo y el usuario. Adicionalmente, las contraseñas son escogidas por los mismos usuarios y éstos tienden a utilizar palabras sencillas de recordar (como puede ser el nombre de una persona, un cumpleaños o información sencilla de recordar), lo que las hace fácil de adivinar. Lo ideal sería escoger una contraseña con un mínimo de 8 caracteres de longitud, que mezcle las letras mayúsculas, minúsculas y números lo que las hace prácticamente imposibles de adivinar.

El principal entorno en donde es utilizada la autenticación de usuarios es en las organizaciones ya que son las más interesadas en que la información que almacenan en sus servidores se mantenga confidencial a las personas que no pertenecen a la organización. Incluso dentro de la misma organización pueden existir niveles jerárquicos para establecer qué nivel tiene derecho a qué información, según las políticas establecidas dentro de la empresa. Es en este entorno interno de la organización donde es necesario establecer protocolos de seguridad robustos que incluyan elementos seguros para realizar el intercambio de la información de autenticación y la validación del usuario, tanto para identificar a las personas que trabajan dentro de ello como, en especial, para las personas que por motivos de trabajo necesitan hacer uso de la información desde un sitio remoto que se encuentre fuera de la seguridad física de la empresa.

Para realizar la autenticación de los usuarios existen varios métodos que van desde la tupla <*cuenta, contraseña*> hasta procedimientos más sofisticados de seguridad que incluyen criptografía o pruebas biométricas.

1.1 AUTENTICACIÓN

Se entiende por *autenticación* a la validación de la supuesta identidad de un usuario. La autenticación tiene como resultado la autenticidad, lo que quiere decir que el sistema que verifica (*verificador*) las credenciales de un usuario puede estar seguro de que el usuario verificado (*solicitante*) es quien dice ser.

Es práctica común dividir las técnicas utilizadas para la autenticación en tres categorías, dependiendo de la información en que se basan:

- Algo que el solicitante sabe (prueba por conocimiento)
- Algo que el solicitante posee (prueba por posesión)
- Algunas características biométricas del solicitante (prueba por propiedad)

Como ejemplos de la primera categoría (prueba por conocimiento) se pueden citar los *números de identificación personales*; como ejemplos de la segunda categoría (prueba por posesión) se pueden citar las claves, las tarjetas de identificación y otros dispositivos físicos o elementos personales. Como ejemplo de la tercera categoría (prueba por propiedad) las características

biométricas mas utilizadas para la autenticación son las huellas dactilares, aunque también se agrupan en esta categoría las imágenes de retina y los patrones de voz.

En las redes de computadoras actuales la mayor parte de los mecanismos de autenticación que se utilizan se basan en la prueba por conocimiento.

1.1.1 AUTENTICACIÓN BASADA EN CONTRASEÑA

Este tipo de autenticación es un ejemplo de la prueba por conocimiento. En la mayoría de las redes de computadoras, la protección de los recursos se consigue protegiendo con contraseñas el acceso a los sistemas. Los usuarios seleccionan sus contraseñas y las transmiten cuando desean establecer una comunicación con el sistema junto con la cuenta de forma clara y no protegida.

En este sistema se presentan varios problemas de los que se mencionarán a continuación solo algunos:

- Los usuarios tienden a seleccionar contraseñas que no se distribuyen aleatoriamente como son las palabras fáciles de adivinar. Se trata de un problema bien conocido y no necesariamente relacionado a los sistemas de cómputo, sino a la forma estructurada de la mente humana que la hace relacionar los nombres o palabras a los objetos para que sean más fáciles de recordar.
- No es conveniente que un usuario que posea varias cuentas en sistemas diferentes tenga que recordar una contraseña diferente para cada una de ellas e introducirla cada vez que cambia el sistema. Es mejor que el usuario sea reconocido como uno solo por toda la red de computadoras dentro de una organización.
- La transmisión de la propia contraseña está expuesta a escuchas pasivas y a posteriores ataques de réplica.

Debido principalmente al último problema, la autenticación basada en contraseña no es adecuada para ser utilizada en un entorno de red de cobertura amplia. Esto se debe a que las contraseñas que se envían por las redes son demasiado fáciles de interceptar mediante las herramientas adecuadas como serían los *sniffers*[2], por lo tanto se puede suplantar a sus propietarios una vez que se obtuvo la contraseña validándose como si fuera el propietario.

1.1.2 AUTENTICACIÓN CRIPTOGRÁFICA

Este es un ejemplo de autenticación basado en la prueba por posesión. La idea básica de la autenticación criptográfica es que el solicitante A pruebe su identidad a un verificador B realizando una operación criptográfica sobre un valor que ambos conocen o que B suministra. La operación criptográfica realizada por A está basada en una clave criptográfica, que puede ser una

[2] Es un programa de computación que se encarga de capturar todos los paquetes transmitidos dentro de una red.

clave secreta en la criptografía[3] simétrica o una clave privada en la criptografía asimétrica o criptografía de llave pública [27].

La autenticación criptográfica es más segura que la autenticación basada en contraseñas ya que la información se intercambia de manera más confiable, porque aunque el paquete de información que contiene la contraseña sea capturado, éste contendrá información ilegible para alguien que no posea las llaves con que se realizó el cifrado.

A pesar de la aparente simplicidad de los principios básicos de diseño de los protocolos de autenticación, el diseño de protocolos realistas en notoriamente difícil, y varios de los protocolos diseñados han mostrado problemas de seguridad sustanciales o sutiles. Durante la última década, los esfuerzos de investigación se han dirigido hacia la provisión de las herramientas necesarias para el desarrollo de protocolos de autenticación y de distribución de claves con garantía formal de seguridad. Se han desarrollado herramientas formales para la prueba de protocolos criptográficos, como por ejemplo la lógica BAN [4]. En lugar de desarrollar protocolos específicos, estas metodologías proporcionan los medios para comprobar que la ejecución de un protocolo en donde se utilice criptografía cumpla con la seguridad requerida.

1.2 TÉCNICAS CRIPTOGRÁFICAS.

El término *criptología* se refiere a la ciencia de las comunicaciones seguras. La criptología comprende tanto la criptografía como el criptoanálisis. La criptología se divide en tres partes: funciones *hash* unidireccionales, criptografía con clave secreta y criptografía con clave pública.

1.2.1 FUNCIONES DE *HASH* UNIDIRECCIONALES

Una función *hash* unidireccional es sencilla de calcular pero difícil de invertir. Esto es, dado un conjunto de bits como entrada se puede calcular un resultado mediante una función matemática, pero que a partir del resultado no haya manera de calcular cuáles eran el conjunto de bits de entrada. Entre algunos de los ejemplos más comunes de funciones *hash* unidireccionales utilizados en la actualidad se puede citar MD5[4][18] y SHA 1[5][17].

La desventaja de las funciones *hash* unidireccionales es que, debido a que no se requiere que sea invertible, varias entradas distintas pueden tener el mismo valor de salida por lo que una función *hash* debe de tener un dominio lo suficientemente grande para evitar la duplicación de valores iguales para entradas distintas en donde se utilicen varios bits de información. Por ejemplo MD5 tiene un dominio de 32 bits (4, 294, 967,296 cantidades distintas) para los valores devueltos y lo expone menos a la duplicación de valores. Al tener un dominio de valores devueltos tan amplios

[3] Es un conjunto de técnicas que alteran los símbolos de la información sin alterar el contenido, de esta manera los símbolos originales pueden ser recuperados posteriormente.

[4] *Message Digest*, el algoritmo es la versión 5.

[5] *Secure Hash Algorithm*.

se puede utilizar una función *hash* para autenticar a los usuarios, como se menciona en el artículo *One-Way Functions for Authentication* [14].

1.2.2 CRIPTOGRAFÍA CON CLAVE SECRETA O SIMÉTRICA

En la criptografía con clave secreta, las entidades dentro de la comunicación establecen y comparten una clave secreta, ya sea por un canal seguro o entregándola personalmente, que se utiliza posteriormente para cifrar y descifrar los mensajes. Por esta razón, la criptografía con clave secreta es conocida también como criptografía simétrica.

La criptografía con clave secreta se ha utilizado desde hace miles de años en sus diversas formas. El ejemplo más conocido de esta forma de criptografía es el cifrado de César. En esta forma de cifrado una letra es sustituida por otra, por ejemplo, la letra A es sustituida por la letra C, la letra B es sustituida por la letra D y así de manera consecutiva. Esta forma de cifrado fue utilizada por el emperador romano Julio César para comunicarse con su ejército en época de guerras [28].

Un ejemplo más reciente de la utilización de criptografía simétrica fue realizado con la máquina Enigma durante la II Guerra Mundial. La máquina Enigma fue construida por Arthur Scherbius y Richard Ritter [28]. Esta máquina constaba de tres elementos: un teclado para introducir texto sin cifrar, una unidad que se encargaba de "revolver" la letra introducida para obtener una letra cifrada y un tablero que indicaba cuál era la letra cifrada.

Las versiones modernas de la criptografía con clave secreta están basadas en operaciones aritméticas que se pueden realizar de forma muy eficiente en las computadoras. Estas operaciones utilizan una combinación de transposiciones y permutaciones para, de esta manera, crear un sistema extremadamente difícil de analizar ya que un símbolo en la información original puede tener distintos símbolos en la información cifrada que dependen de las operaciones que hicieron la transformación de los símbolos.

La historia demuestra que la criptografía de clave secreta es fácil de romper, como se demostró cuando la máquina Enigma fue descifrada por las fuerzas aliadas de la II Guerra Mundial. Esto se debió a que las transformaciones que se podían efectuar para cifrar la información se encontraba limitada antes de la llegada de las computadoras.

1.2.3 CRIPTOGRAFÍA CON CLAVE PÚBLICA O ASIMÉTRICA

En la criptografía de clave pública cada usuario posee una pareja de claves relacionadas matemáticamente. Una de ellas es una clave pública, que puede ser publicada sin dañar la seguridad del sistema, y la otra es una clave privada, que nunca deja de estar en poder de su dueño. Tanto la clave privada como la pública son imposibles de obtener conociendo la otra. En la actualidad el criptosistema con clave pública más utilizada es RSA [26].

La criptografía con clave pública puede parecer más conveniente que la criptografía con clave secreta, ya que evita que las dos entidades que desean autenticarse tengan que compartir la misma clave secreta. No se requiere un sistema de distribución de claves que sea seguro como en el caso de la criptografía simétrica, ya que solo es necesario intercambiar las claves públicas para realizar el cifrado siendo utilizada la clave privada para realizar el descifrado. Además, la criptografía con clave pública permite que el sistema de autenticación esté bajo el control directo del usuario del sistema.

Sin embargo, los sistemas con clave pública requieren generalmente operaciones aritméticas que son difíciles de realizar en los microprocesadores antiguos con capacidad de proceso limitada. Esto puede causar problemas en el diseño de los sistemas de autenticación, ya que a menudo es necesario que los algoritmos criptográficos se ejecuten en sistemas pequeños que tienen capacidad de proceso limitada.

1.3 TRABAJO DE TESIS

La problemática actual para realizar la autenticación se basa en las pruebas más utilizadas, la prueba por conocimiento permite que el cliente pueda conectarse desde cualquier máquina pero no proporciona la seguridad dentro de la transmisión de información para evitar que un usuario distinto al cliente pueda conocer el conocimiento transmitido, la prueba por posesión utiliza dispositivos que el usuario cliente debe de tener para poder autenticarse pero no hay manera de saber si el dispositivo sigue en manos de su legítimo dueño cuando es utilizado o si ha sido robado por alguien ajeno.

El objetivo de esta tesis es desarrollar un protocolo de autenticación que combine la prueba por conocimiento y la prueba por posesión para minimizar las desventajas y maximizar las ventajas de estos métodos de autenticación cuando se usan en sistemas de comunicación que requieren de mecanismos de seguridad robustos y difíciles de penetrar.

2 ANTECEDENTES Y MARCO TEÓRICO

Un protocolo de autenticación de usuarios tiene como única función validar que el usuario pertenezca al sistema. Aunque puede parecer una tarea muy simple, su función es muy importante debido a la confidencialidad y seguridad de los datos que están almacenados en los sistemas computacionales que pertenecen a la institución. Un protocolo de autenticación es el proceso más importante dentro de la comunicación de redes seguras para evitar que usuarios no autorizados puedan tener acceso a los recursos y datos del sistema.

La mayoría de los protocolos de autenticación están constituidos por tres actores para realizar dicha autenticación. Dos de los actores son: el cliente que desea acceder a la información y el sistema computacional que guarda la información. La tercera entidad es un servidor de autenticación que se encarga de verificar que el cliente esté autorizado para acceder a la información contenida en el servidor.

2.1 JUSTIFICACIÓN

El protocolo de autenticación desarrollado en esta tesis forma parte de un proyecto de telemonitoreo, telecontrol y mantenimiento para un sistema de manufactura. El proyecto realizará telemonitoreo, telecontrol y telemantenimiento para procesos de manufactura en donde se manejará equipo delicado que resultaría costoso tanto en tiempo como económicamente en caso de que se dañara. Por este motivo se necesita un protocolo de autenticación para evitar que un usuario no autorizado interactué con el sistema y aumente el riesgo de daño físico del sistema.

Los protocolos de autenticación utilizan comúnmente un servidor cuya única función es verificar que el cliente que se autentica es una entidad autorizada dentro del sistema. Pero utilizar un servidor dedicado únicamente para autenticar a los usuarios puede dar como resultado un desperdicio del procesador si el número de usuarios que se autenticarán es poco, por ejemplo 20 usuarios, ya que el servidor podría estar inactivo durante mucho tiempo. También puede ser necesario adquirir licencias para utilizar el servidor de autenticación por lo que habría que agregar el costo de las licencias al costo del equipo, lo que resultaría en un costo elevado para una institución de pocos usuarios.

También existen protocolos de autenticación que no requieren de un servidor para verificar a los usuarios pero estos protocolos solo están basados en la autenticación por conocimiento sin una protección adicional como podría ser un algoritmo criptográfico.

Por eso es necesario desarrollar un protocolo de autenticación que incluya algoritmos criptográficos que no requieran de un servidor dedicado cuando los usuarios a administrar son pocos pero que ofrezcan la misma confianza en la seguridad que los protocolos de autenticación comerciales.

El protocolo de autenticación a desarrollar debe ser sencillo de implementar, también sería conveniente que utilizara algoritmos criptográficos de llave secreta ya que este tipo de criptografía utiliza operaciones aritméticas más simples (como sumas, restas o transposiciones) y ocupan menos ciclos de procesador de lo que ocuparían las operaciones en la criptografía de llave pública lo que permitiría que se implantara en cualquier sistema o en dispositivos portátiles (como los dispositivos de mano) que tienen procesadores con poca capacidad de proceso y memoria reducida.

2.2 PLANTEAMIENTO DE OBJETIVOS

2.2.1 OBJETIVOS GENERALES

Diseñar, validar e implementar un protocolo de autenticación de usuarios para un sistema de telecontrol, telemantenimiento y telemonitoreo; el desarrollo del protocolo de autenticación combinará la autenticación mediante conocimiento en el que habrá preguntas que solo podrá responder el usuario al que se le envían las podrá contestar junto con una función criptográfica para cifrar algunos de los mensajes que se transmitan, el protocolo debe ser robusto y funcionar sin importar los medios de transmisión de información que se utilicen.

El propósito del protocolo de autenticación es que el servidor confíe que un usuario válido es el que está solicitando recursos para utilizarlos. El diseño del protocolo se debe realizar con independencia de los medios de comunicación que se empleen para realizar la comunicación, considerando la portabilidad a cualquier sistema operativo.

2.2.2 OBJETIVOS ESPECÍFICOS

Una vez que hemos definido el objetivo general que se desea alcanzar se propondrán los objetivos concretos que serán de ayuda para alcanzar el objetivo general. Estos son:

2.2.2.1 Revisar las técnicas para el desarrollo de un protocolo.

Revisar los requisitos que se deben de considerar para llevar a cabo un buen desarrollo de un protocolo de comunicaciones libre de errores.

2.2.2.2 Revisar y seleccionar las herramientas existentes para la validación del protocolo.

Revisar qué herramientas son utilizadas para el análisis y la validación de un protocolo de comunicaciones correcto libre de errores.

2.2.2.3 Seleccionar el algoritmo criptográfico a utilizar.

Analizar los algoritmos criptográficos recomendados en la actualidad por los estándares o que hayan sido propuestos para algún estándar, de esta manera no será necesario efectuar los análisis de fuerza del algoritmo debido a que se realiza por las instituciones que proponen los estándares. El algoritmo seleccionado se utilizará para cifrar la información que será transmitida.

2.2.2.4 Seleccionar las técnicas para la validación de un algoritmo de autenticación.

Así como existen herramientas y técnicas para la validación de un protocolo de comunicaciones que no contiene elementos criptográficos, la validación de un protocolo de seguridad tiene sus propias herramientas y técnicas de validación. Estas técnicas están enfocadas a la seguridad que el protocolo está proporcionando y que no haya fallos en la transmisión de la información que puedan hacer que la información en el sistema este comprometida.

2.2.2.5 Implementar el algoritmo de autenticación.

Implementar una aplicación para demostrar que el protocolo de autenticación propuesto es funcional y, por lo tanto, puede ser llevado a un entorno operativo. Al finalizar los objetivos específicos anteriores se implementará la aplicación que se utilizará tanto en el sistema servidor que realizará la validación del usuario como la aplicación que se utilizará en el sistema cliente.

2.3 MARCO TEÓRICO

En esta tesis se desarrollará un protocolo de autenticación de usuarios que protegerá la información de autenticación, utilizando algoritmos criptográficos para cifrar esta información que se transmite cuando un usuario se conecta a un sistema de cómputo así como validar que el usuario sea el correcto. La información estará conformada por la identificación del usuario y una serie de valores que serán únicos para un usuario. Esto quiere decir que dos usuarios válidos no necesariamente podrán utilizar los mismos valores para ser considerados correctos, por esto un usuario A no podrá utilizar los valores de un usuario B aunque los dos sean usuarios válidos.

Para empezar con el desarrollo del proyecto de tesis se deben de conocer algunos conceptos que serán útiles durante el desarrollo del trabajo, esto permitirá que las ideas sean homogéneas y se tenga una base común para la lectura de la tesis.

2.3.1 CONCEPTOS BÁSICOS

De acuerdo con los entornos de seguridad que se están desarrollando por parte del Comité Técnico Conjunto 1 (JTC1 – *Joint Technical Committee 1*) de la Organización Internacional para la Estandarización (ISO – *International Organization for Standarization*), y del Comité Electrotécnico Internacional (IEC – *International Electrotechnical Committee*) se utilizará el término *principal* para referirnos a una entidad (que puede ser humano o sistema) que se ha registrado y es autenticable para una red de computadores. Los usuarios, los sistemas y los procesos se consideran comúnmente principales: [22]

- Un *usuario* es una entidad que se convierte en responsable, en último término, de sus actividades dentro de una red de computadores o sistema distribuido. Los usuarios son las personas que interactúan con las computadoras.
-
- Un *sistema* es una entidad dentro de una red de computadores. La entidad es seleccionada generalmente utilizando su nombre o su dirección de capa de red. Comúnmente son computadoras.
-
- Un *proceso* es una realización de un programa que se ejecuta en un sistema concreto. Es común utilizar el *modelo cliente / servidor* para distinguir entre los procesos clientes y servidores:
-
- Un *proceso cliente* (cliente) es un proceso que solicita y eventualmente consigue un servicio de red
- Un *proceso servidor* (servidor) proporciona el servicio. En esta terminología, un servicio es un conjunto coherente de funcionalidades abstractas, y un servidor es generalmente un *daemon* (demonio en entorno Linux) de ejecución continua que se especializa en proporcionar esa funcionalidad.

Pero hay servicios proporcionados que deben ser permitidos solo a determinados usuarios. Esta distinción de los usuarios la debe de hacer el administrador de acuerdo a ciertas políticas, por ejemplo con base en el puesto que ocupa, en este caso hay que utilizar seguridad computacional. Se definirá el objetivo de la seguridad computacional, que se divide en las comunicaciones que se llevan a cabo y los sistemas de cómputo de donde se origina esta información.

- El objetivo de la *seguridad en el sistema de cómputo* es preservar los recursos de cómputo contra usos y abusos no autorizados, así como proteger contra daños los datos que representan y codifican la información de revelaciones y modificaciones accidentales o deliberadas.
- El objetivo de la *seguridad en las comunicaciones* es proteger los datos que representan y codifican la información durante su transmisión en las redes de computadoras.

Las definiciones anteriores son tomadas del libro *Sistemas de Autenticación para Seguridad en Redes* [22] y serán suficientes para establecer lo que será la seguridad que se desea alcanzar para el proyecto de la tesis.

2.3.2 PROBLEMAS DE SEGURIDAD CAUSADOS POR LA ESTANDARIZACIÓN.

El uso de estándares permite que la información sobre el diseño de una red esté disponible para un gran número de personas y pueda ser aplicado para intercambiar información entre los usuarios, pero también permite que un usuario mal intencionado pueda encontrar vulnerabilidades en el estándar por lo que puede ser una amenaza para las redes implementadas que utilicen el estándar.

Una vulnerabilidad se refiere a una debilidad que se puede explotar para violar un sistema o acceder a la información que contiene. El término *amenaza* indica una circunstancia, condición o evento que tiene el potencial de causar una violación de la seguridad o de causar daño a los recursos del sistema. Las redes de computadoras son susceptibles de sufrir una gran variedad de amenazas, provenientes bien de intrusos, o de usuarios legítimos. De hecho, los usuarios legítimos son los que mayor daño pueden causar ya que poseen información sobre el estado del sistema como los identificadores de usuario y contraseñas, información que generalmente no está disponible para los intrusos.

Las amenazas en las redes de computadoras, tomados de las definiciones de Oppliger [22] se diferencian entre compromisos de comunicaciones y de sistemas:

- Un *compromiso de sistema* es el resultado de una subversión de un sistema aislado dentro de una red de computadores. Son posibles varios grados de subversión, desde el caso relativamente benigno de corrupción de la información de estado de los procesos hasta el caso de toma absoluta de control del sistema.

- Un *compromiso de comunicaciones* es el resultado de una subversión de una línea de comunicaciones dentro de una red de computadoras:
  - Un *ataque pasivo* amenaza la confidencialidad de los datos que se transmiten. Esta situación se describe de la siguiente manera: los datos que envía el remitente al receptor son observados por algún intruso, las posibilidades del intruso para interpretar la información que los datos transmitidos pueden representar o codificar, se debe diferenciar entre ataques de intervención pasiva y ataques de análisis de tráfico.
    - En un *ataque de intervención pasiva*, el intruso puede analizar los datos y extraer la información que contiene.
    - En un *ataque de análisis de tráfico* el intruso no puede hacerlo.

  - Un *ataque activo* amenaza la integridad y/o la disponibilidad de los datos que se están transmitiendo. En este ataque el intruso puede modificar,

ampliar, borrar y replicar unidades de datos. Además, puede saturar al receptor y causar una denegación de servicio.

La posibilidad de que se realice un ataque pasivo depende del medio físico de transmisión que se utilice. Las líneas de comunicaciones móviles son fáciles de intervenir ya que la información se propaga en el aire lo que permite que cualquier persona con el equipo adecuado pueda captar las señales, mientras que los medios de transmisión utilizando cables requieren algún tipo de acceso físico. Las fibras ópticas pueden ser también intervenidas, pero es más costosa la intervención que en una línea telefónica. También el uso de técnicas de concentración y de multiplexación, dificultan los ataques pasivos a una línea de comunicaciones. Existen varios paquetes de software que permiten monitorizar el tráfico por la red, principalmente para las tareas de administración de la misma. Sin embargo, estos mismos paquetes son también útiles para escuchar y eventualmente capturar contraseñas cuando se transmiten por las líneas de comunicaciones, un ejemplo de este tipo de software es el programa *Ethereal*.

Los ataques activos y pasivos se utilizan conjuntamente para causar de forma efectiva un compromiso de comunicaciones. Por ejemplo, se puede lanzar una escucha pasiva para obtener información de autenticación transmitida directamente, y esa información puede utilizarse posteriormente para un ataque tipo réplica. Con esta posibilidad en mente, es obvio que la autenticación basada en contraseñas no es lo suficientemente segura en entornos de red.

2.3.3 ESPECIFICACIÓN Y VERIFICACIÓN DE PROTOCOLOS

Los protocolos reales y los programas que se desarrollan que implementan estos protocolos son, por lo general, bastante complicados. Primero se debe de especificar la estructura del protocolo, y cumplir ciertas reglas básicas esenciales de diseño de un protocolo.

2.3.3.1 Estructura de un protocolo

Un protocolo contiene reglas, formatos y procedimientos necesarios para poner de acuerdo a dos entidades. Los protocolos contienen acuerdos que son utilizados para:

- Iniciación y terminación de intercambio de datos.
- Sincronización del emisor y del receptor.
- Detección y corrección de los errores de transmisión.
- Codificación y formato de los datos.

Cada uno de estos elementos tiene su propia capa dentro del modelo OSI de acuerdo a la función que cumple. A su vez estos protocolos pueden estar agrupados para formar los estándares.

2.3.3.2 Cinco elementos de un protocolo.

Para considerar de manera completa el desarrollo de un protocolo se deben de considerar cinco elementos esenciales [21] [29], los cuáles son:

2. El *servicio* que será provisto por el protocolo
3. Las *suposiciones* sobre el entorno en donde el protocolo será ejecutado
4. El *vocabulario* de los mensajes utilizados para implementar el protocolo
5. La *codificación (formato)* de cada uno de los mensajes del vocabulario
6. Las *reglas de intercambio* que se mantiene en la consistencia de los mensajes intercambiados

El quinto elemento de la especificación del protocolo es el más difícil de diseñar y también el más difícil de verificar, cada parte de la verificación del protocolo puede definir una jerarquía de los elementos, en donde elementos en una capa superior son construidos a partir de elementos que se encuentren en las capas inferiores. Para facilitar la tarea de diseñar un protocolo se deben de seguir ciertas reglas que simplificarán este proceso.

2.3.3.3 Diez reglas del diseño de protocolos

Una metodología útil cuando se desea diseñar un protocolo consiste en seguir las siguientes reglas, que fueron tomadas del libro de Holzmann [29] y que aquí se transcriben textualmente:

1. Asegurarse que el problema ha sido definido de manera correcta. Todos los criterios del diseño, requerimientos deben ser enumerados antes del comienzo del diseño.
2. Definir el servicio que será provisto en cada nivel de abstracción antes de decidir qué estructura debe ser utilizada para realizar estos servicios (esto es, lo que *hay que hacer* antes de cómo *hacerlo*).
3. Diseñar la funcionalidad externa antes de diseñar la funcionalidad interna. Se debe de considerar la solución primero como una caja negra definiendo como interactuará con su entorno, una vez realizado se definen los procedimientos internos de esta caja negra.
4. Mantenerlo simple. Muchas veces el diseñador no puede separar los problemas que serán cubiertos en problemas menores que sean más simples, dando como resultado que se creen protocolos que son muy grandes, difíciles de implementar, de verificar y la mayoría de las veces menos eficientes.
5. No conectar lo que es independiente.
6. Se debe de crear un diseño genérico, esto es de fácil implementación y que se pueda extender para realizar otras funciones en un futuro.
7. Antes de implementar el diseño, construir un prototipo que se pueda verificar.
8. Implementar el diseño, medir su desempeño, y si es necesario optimizarlo.
9. Comprobar que la implementación final es equivalente al prototipo que fue verificado en el punto 7.
10. No saltarse de la regla 1 a la 7.

Las diez reglas para el diseño de protocolos son necesarias para lograr un buen desarrollo del protocolo en donde se logren alcanzar los cinco elementos que lo conforman para lograr el primer objetivo de la tesis: Revisar las técnicas para el desarrollo de un protocolo.

La definición de un protocolo es comparable a la definición de un idioma, ya que tienen puntos comunes como el vocabulario o las reglas de intercambio en los mensajes. Pero existen diferencias con respecto a la definición del idioma ya que, al ser el protocolo parte de un lenguaje de computadora, éste no debe ser ambiguo al realizar el intercambio de mensajes, esto es, cada mensaje debe de tener un propósito específico. Un protocolo debe de especificarse para la ejecución concurrente de los procesos. Esta concurrencia crea una nueva clase de problemas, ya que se tiene que tratar con fallos que pueden llegar a ocurrir durante la ejecución del protocolo como *interbloqueos*, *sincronización* y *control de flujo*. Como la secuencia posible de los eventos no puede ser anticipada, el número de combinaciones posibles de cómo puede ocurrir el intercambio de mensajes puede ser tan extenso que es casi imposible el análisis manual de los casos que pueden presentarse durante la ejecución de un protocolo por lo que es necesario la utilización de técnicas automatizadas para la validación de protocolos.

También existen reglas para el diseño de protocolos de comunicación, también existen las reglas que se deben de aplicar al diseño de protocolos seguros, estas reglas o principios son definidos por *Abadi* [39] y se mencionan a continuación.

2.3.3.4 Principios para el diseño de protocolos de seguridad.

Principio 1
Cada mensaje debe indicar lo que significa, la interpretación del mensaje debe depender solo de su contenido. Por lo que debe ser posible escribir una sentencia que describa el contenido del mensaje en base a lo que se transmite.

Principio 2
Las condiciones sobre las que debe de actuar un mensaje deben ser definidas claramente. De esta manera alguien que revise un diseño pueda ver si las condiciones son aceptables o no.

Principio 3
Si la identidad de un principal es esencial para el significado del mensaje, es prudente mencionar el nombre del principal explícitamente.

Principio 4
Dejar en claro porque es hecho el cifrado. El cifrado no es un proceso sencillo de realizar y no saber porque es realizado puede llevarnos a la redundancia. El cifrado no es sinónimo de seguridad y el uso inapropiado de esté puede llevarnos a errores.

Principio 5
Cuando un principal firma un mensaje que ha sido cifrado no se debe de inferir que el principal conoce el contenido del mensaje. Por otra parte, es correcto inferir que el principal que firma un mensaje y después lo cifra para asegurar la privacidad conoce el contenido.

Principio 6
Dejar en claro que propiedades se asumen sobre los *nounces.* Lo que se puede usar para asegurar sucesión temporal podría no utilizarse para asegurar asociación que relaciona a un principal con su mensaje y la asociación puede ser establecida mejor por otros medios.

Principio 7
El uso de cantidades predecibles (como el valor de un contador) puede servir para garantizar que es reciente en un intercambio de un protocolo de reto-respuesta. Pero si una cantidad predecible será efectiva esta debe ser protegida de manera que un intruso no pueda simular un reto y posteriormente replicar una respuesta.

Principio 8
Si el tiempo es usado para garantizar frescura mediante la referencia a un tiempo absoluto, la diferencia de los relojes en las máquinas que participan en la autenticación debe ser mucho menor que la edad permitida para que un mensaje se considere válido. Por lo que el mecanismo de mantenimiento del tiempo debe ser parte de una base computacional confiable.

Principio 9
Una llave puede haber sido usada recientemente, por ejemplo para cifrar un *nounce* pero esto no indicaría que la llave no haya sido utilizada antes y puede estar comprometida. El uso reciente no hace que la llave se considere nueva.

Principio 10
Si una codificación es usada para presentar el significado de un mensaje, entonces debe ser posible decir que codificación se esta usando. En el caso común donde la codificación es dependiente del protocolo, debe ser posible deducir que el mensaje pertenece a ese protocolo y de hecho a una ejecución particular del protocolo y saber su posición dentro del protocolo.

Principio 11
El diseñador del protocolo debe saber de que relaciones de confianza depende su protocolo y por que la dependencia es necesaria. Las razones para que las relaciones de confianza particulares sean aceptables deben ser explicitas y estarán fundamentadas en juicios y políticas más que en la lógica.

Estos principios buscan cubrir los requerimientos de diseño de los protocolos de autenticación existentes. Pero debido a que los protocolos de autenticación pueden buscar resultados distintos como puede ser la autenticación de usuarios o el intercambio de llaves para cifrado de información utilizando métodos distintos para lograr su objetivo, no es necesario que todos los principios sean aplicados durante el desarrollo de un protocolo y eso no indicaría un problema de seguridad en el protocolo de autenticación diseñado.

2.3.4 MODELOS DE VALIDACIÓN DE PROTOCOLOS DE COMUNICACIONES.

A continuación detallaremos las maneras de representar un protocolo de comunicaciones para su validación posterior mediante una herramienta automatizada. Se puede realizar la validación manualmente pero aún en protocolos pequeños son tantos los caminos posibles que se consiguen en una ejecución que hace la validación manual muy difícil.

2.3.4.1 Modelos de máquina de estado finito

En varios modelos de protocolos se utiliza la *máquina de estado finito* para modelar protocolos. Con esta técnica, cada *máquina de protocolo*, ya sea el emisor o el receptor, siempre se encuentra en un estado específico en cualquier instante del tiempo. Su estado está constituido por todos los valores de sus variables [23].

Desde el punto de vista formal, el modelo de máquina de estado finito de un protocolo puede interpretarse como un cuádruplo: <S, M, I, T>, en donde:

S Es el conjunto de estados en los que los procesos y el canal puede encontrarse
M Es el conjunto de tramas que pueden intercambiarse en el canal
I Es el conjunto de estados iniciales de los procesos
T Es el conjunto de transiciones que se llevan a cabo entre estados

El estado del sistema completo es la combinación de todos los estados de las dos máquinas de los protocolos y del canal. El estado del canal está determinado por su contenido, que puede variar dependiendo de las tramas de los mensajes que se intercambien entre las máquinas del protocolo.

A partir de cada estado, se puede tener cero o más *transiciones* posibles a otros estados, las cuales ocurren siempre que se lleve a cabo algún evento. Para una máquina de protocolo, la transición podría ocurrir en el momento en que se transmite una trama, cuando llega una trama, cuando el contador de tiempo se desactiva, cuando sucede una interrupción, etc. Para un canal, los eventos que se presentan son, típicamente, la inserción de una trama nueva en el canal mediante la máquina de protocolo, la entrega de una trama a una máquina de protocolo, o bien, la pérdida de una trama debido a una ráfaga de ruido. Dada una descripción completa de la máquina de protocolo y de las características del canal, es posible diseñar un grafo dirigido que muestre todos los estados como nodos y todas las transiciones como arcos dirigidos [23].

En el diseño se considera como *estado inicial* a un estado particular, este estado corresponde a la descripción del sistema cuando comienza a funcionar, o bien, cuando se define algún inicio conveniente localizado poco después. A partir de este estado inicial, alguno, o todos, los estados restantes se pueden alcanzar mediante una secuencia de transiciones. Con el empleo de técnicas basadas en la teoría de grafos es posible determinar los estados que se pueden alcanzar, y los que no. Esta técnica se denomina *análisis de alcanzabilidad*, este análisis llega a ser muy útil para determinar si un protocolo es, o no es, correcto.

Al comienzo, todos los procesos se encuentran en sus estados iniciales. Después, comienzan a presentarse los eventos, como las tramas que llegan a estar disponibles para su transmisión, o bien, los vencimientos de los contadores de tiempo. Cada evento hará que uno de los procesos o algún canal lleve a cabo una acción y conmute a un estado nuevo. Con la enumeración cuidadosa de cada uno de los posibles sucesores para cada estado, se puede construir un grafo de alcanzabilidad y hacer un análisis del protocolo.

El análisis de alcanzabilidad se puede utilizar para detectar una variedad de errores en la especificación del protocolo. Por ejemplo, si es posible que ocurra cierta trama en un cierto estado, y la máquina de estado finito no dice qué acción debería tomarse, la especificación será errónea (es incompleta). Si existiera un conjunto de estados del cual no exista salida, y del cual no puede salirse (recibiendo tramas correctas), se tiene otro tipo de error (bloqueo). Un error menos serio en la especificación de un protocolo, es indicar cómo tratar un evento en un estado en el cual el evento no puede ocurrir, por ejemplo, el caso de una transición extraña.

Una propiedad importante de los protocolos es la ausencia de bloqueos mutuos. Un *bloqueo mutuo* es una situación en la cual el protocolo no puede seguir progresando (esto es, entregar paquete a la capa de red), sin importar qué secuencia de eventos suceda. En términos del modelo de grafo, un bloqueo mutuo se caracteriza por la existencia de un subconjunto de estados, al cual se puede llegar a partir del estado inicial y que tiene dos propiedades:

1. No hay transición alguna que salga del subconjunto
2. No hay transiciones en el subconjunto que hagan progresar

Una vez que se encuentra en la situación del bloqueo mutuo, el protocolo permanecerá ahí para siempre. Por lo que debería de existir un mecanismo para evitarlo, como el uso de contadores de tiempo.

Se mostrará un ejemplo de un protocolo modelado mediante una máquina de estados finitos. El protocolo elegido es el protocolo *stop-and-wait*; este protocolo tiene dos procesos: un emisor y un receptor. El emisor envía un número de tramas de datos al receptor que éste debe de responder como recibidas para que el emisor pueda enviar las demás tramas, de ahí el nombre *stop-and-wait*[6] [23], cuyo diagrama de estados se muestra en la Figura 2-1 [23] y el comportamiento de las transiciones del diagrama esta en la Tabla 2-1.

En el ejemplo de la Figura 2-1 los estados tienen una etiqueta de tres caracteres, XYZ. El carácter X representa la trama que el emisor trata de enviar que puede ser 0 ó 1, el carácter Y representa la trama que el receptor espera que también es 0 ó 1 y el carácter Z representa el estado del canal que puede ser la trama 0, 1, A (*Ack*, reconocimiento) o – (canal vacío).

[6] Para más información, referirse al libro de Redes de Ordenadores de Tannenbaum [23]

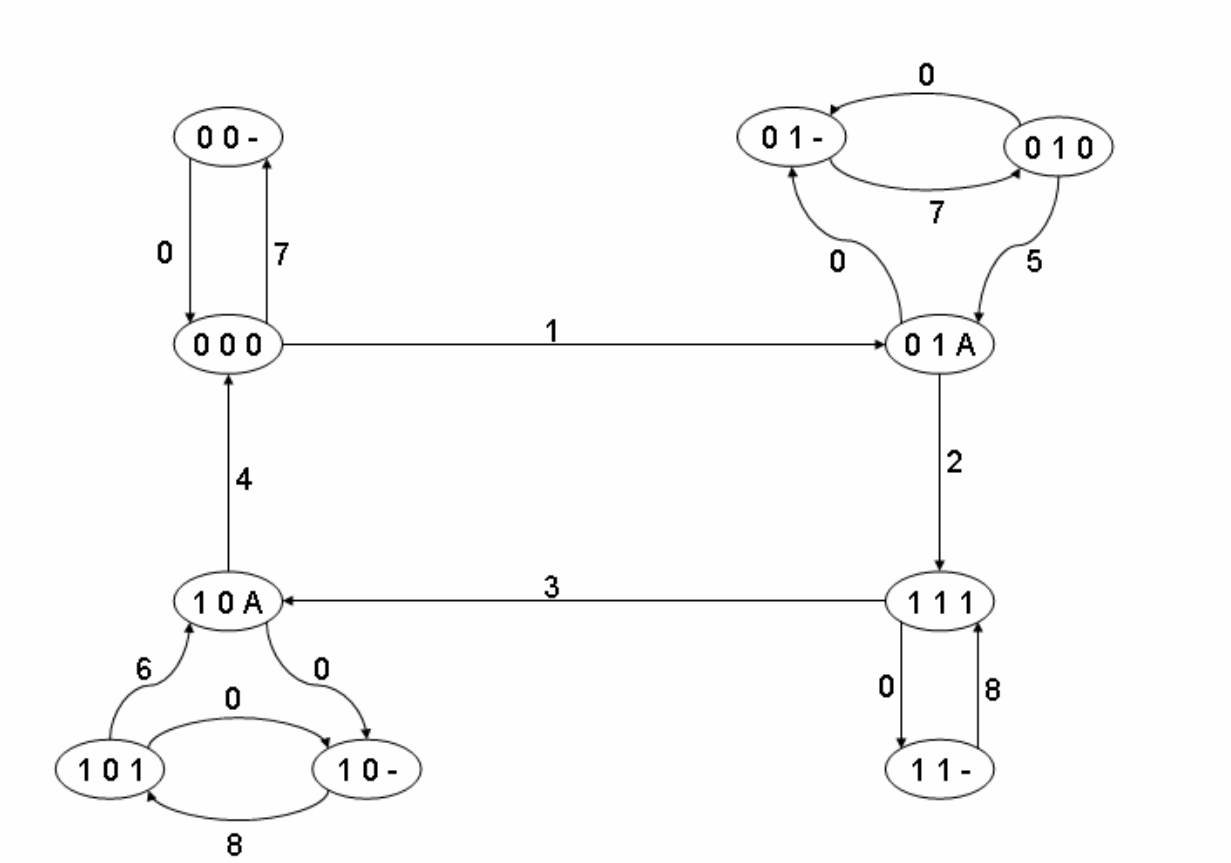

Figura 2-1 Diagrama de estados para el protocolo *stop-and-wait.*

Transición	¿Quién funciona?	Trama Aceptada	Trama Emitida	A la capa de red
0	—	(trama perdida)	(trama perdida)	—
1	R	0	A	Si
2	E	A	1	—
3	R	1	A	Si
4	E	A	0	—
5	R	0	A	No
6	R	1	A	No
7	E	(Temporización)	0	—
8	E	(Temporización)	1	—

Tabla 2-1 Transiciones para el protocolo *stop-and-wait*

2.3.4.2 Modelos de redes de Petri

La máquina de estado finito no es la única técnica utilizada para especificar formalmente protocolos, existe otra que se conoce como *red de Petri* [23]. La red de Petri tiene cuatro elementos básicos: lugares, transiciones, arcos y testigos.

- Un *lugar* representa un estado en el cuál el sistema (o parte de él) puede encontrarse.
- Para indicar el lugar en que se encuentra el sistema, éste se define por la posición del *testigo* (un punto grueso).
- Una *transición* se indica mediante una barra horizontal o vertical.
- Cada transición tiene cero o más *arcos de entrada*, procedentes de lugares de entrada, y cero o más *arcos de salida* que se dirigen a sus lugares de salida.

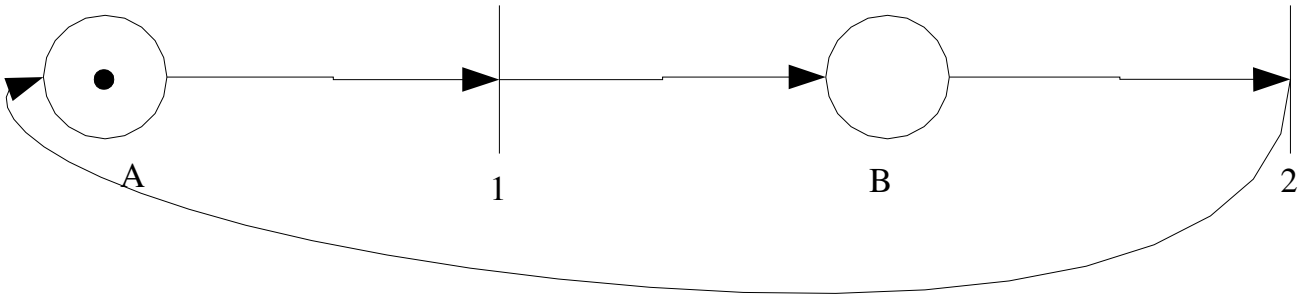

Figura 2-2 Ejemplo de una Red de Petri

En la Figura 2-2 se muestra un ejemplo de una red de Petri con dos lugares y dos transiciones, el *lugar A* es el que indica la posición del sistema con el *testigo*.

Una transición se *activa* si hay por lo menos, un testigo de entrada en cada uno de sus lugares de entrada. Cualquier transición activada puede dispararse a voluntad, eliminando un testigo de cada lugar de entrada y depositando un testigo en cada lugar de salida. Si el número de arcos de entrada y arcos de salida difieren, los testigos no se conservarán. Si se llegan a activar dos o más transiciones, cualquiera de ellas podrá dispararse. La elección para que una transición se dispare está indeterminada, razón por la cual las redes de Petri son útiles en el modelado de protocolos. La red que se muestra en la Figura 2-2 es determinística, y puede emplearse para modelar cualquier proceso que conste de dos fases, por ejemplo, el comportamiento de un bebé, definido por sus actividades: comer, dormir, comer, dormir, etc. Como sucede con todas las herramientas de modelado, los detalles que no son necesarios se eliminan.

Las redes de Petri pueden utilizarse para detectar fallos en los protocolos, de la misma manera como se emplea la máquina de estados finitos, donde el concepto de bloqueo mutuo en la red de Petri, es similar al modelo de máquina de estado finito; las redes de Petri pueden representarse en una forma algebraica conveniente parecida a una gramática. Cada transición contribuye a una regla de la gramática [23].

El protocolo *stop-and-wait* mostrado en el diagrama de estados será modelado mediante una red de Petri como un ejemplo más práctico.

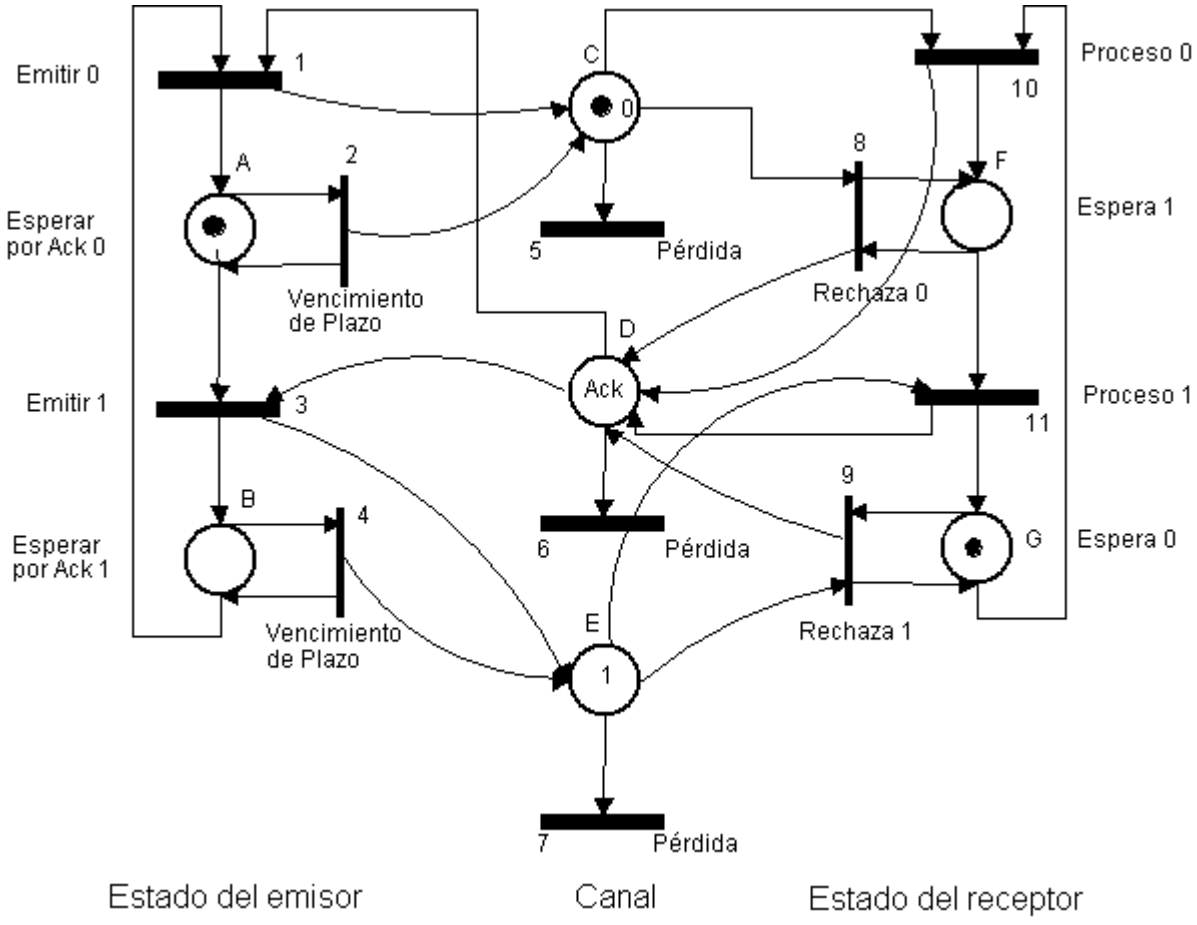

Figura 2-3 Red de Petri para el protocolo *Stop-and-Wait*

2.3.4.3 Lenguajes de especificación de protocolos

El problema principal con la máquina de estado finito y con las redes de Petri es que el número de estados requeridos, con cualquier protocolo realista, es sencillamente inmenso. Varios de estos estados son parecidos en términos de qué eventos espera la máquina de estado y lo que hará en el momento en que se presenten.

Esta observación ha llevado a la idea de máquina de estado extendido o ampliado, en las cuales los estados se pueden considerar variables. El resultado de introducir estados con variables tiene como propósito el llegar a fundir varios de los estados de la máquina de estado finito original en un solo estado parametrizado, reduciendo de esta manera, el tamaño del grafo de estados.

Una ventaja de la utilización de un lenguaje es que permite la modelación del protocolo en la manera más simple posible para poder estudiar bien la estructura del protocolo así como verificar su consistencia lógica y que se alcancen todos los estados definidos en el protocolo. Al realizar esta abstracción nos evitamos de manera deliberada de otros detalles propios del diseño de protocolos, por ejemplo del formato de mensajes.

Un modelo de validación define la interacción de los procesos en un entorno de redes, pero no resuelve los detalles de implementación ni se especifica como serán transmitidos, codificados o almacenados los mensajes en la red. Mediante la simplificación del problema del diseño de protocolos evitando todos estos detalles, nos podemos aislar y concentrar en la parte más difícil de la especificación de un protocolo: el diseño de un conjunto completo y sólido de reglas que gobiernan las interacciones en un entorno de red.

Uno de los lenguajes utilizados para la validación de un protocolo es PROMELA, éste es utilizado para especificar el comportamiento de un sistema en un modelo de validación formal.

2.3.4.3.1 PROMELA

PROMELA (*Protocol Meta Language*) es un lenguaje de validación de modelos para analizar protocolos que fue creado en 1990 por AT&T [29]. Los protocolos se modelan de manera sintetizada para estudiar su estructura y verificar la completitud y la consistencia lógica de estos protocolos. Cuando se dice sintetizada no se quiere decir que solo se implementará una parte del protocolo sino que se dejarán fuera detalles que son propios de la implementación del protocolo como el formato de los mensajes.

Realizar de esta manera el protocolo nos ayuda a aislar y enfocarnos en la parte más difícil al crear un nuevo protocolo: diseñar un conjunto de reglas completo y consistente que controle las interacciones en una comunicación en la red de computadoras.

Más información sobre la codificación de un protocolo para ser analizado en PROMELA se puede encontrar en la página 99.

Con las facilidades proporcionadas por PROMELA, se puede definir un protocolo de comunicaciones para su posterior validación por este lenguaje y obtener resultados de manera más rápida y sencilla de lo que se lograría mediante la validación de un diagrama de estados finitos de manera manual que puede ser muy larga y conducir a errores por la complejidad que se llega a alcanzar en un diagrama de estados finitos modelado a partir de un protocolo ya dado. Con base en la facilidad que proporcionan los lenguajes de especificación se utilizará PROMELA cumpliendo de esta manera el objetivo: "*Revisar y seleccionar las herramientas existentes para la validación del protocolo.*"

Las formas de validación de protocolos vistos en esta sección son útiles cuando se tratan de protocolos de comunicaciones pero el protocolo que se desarrollará en esta tesis además de ser un protocolo de comunicaciones será un protocolo de seguridad que autenticará a los usuarios. Por ser también un protocolo de seguridad se deben de realizar verificaciones adicionales para comprobar la seguridad. Estos análisis se detallan en la siguiente sección.

2.4 MÉTODOS FORMALES PARA EL ANÁLISIS DE PROTOCOLOS DE AUTENTICACIÓN

La autenticación en un red de computadoras grande es todo un reto debido a que la comunicación que se establece entre los principales en una red es vulnerable a varios ataques, anteriormente se mencionaron los tipos de ataques más comunes, en donde ahora se dará un breve repaso. Un intruso pasivo puede "observar" la comunicación que se transmite por una línea y obtener información sensible. Una situación más grave proviene de un intruso activo que puede modificar el tráfico de mensajes, ya sea bloqueando la transmisión de los paquetes de información e insertando sus propios paquetes modificados. Un intruso se puede hacer pasar por cualquier principal en el sistema y posiblemente interceptar sus permisos y privilegios sobre el sistema.

El cifrado puede frustrar los ataques de un intruso activo. Varios de los esquemas de cifrado preservan la propiedad de integridad, en donde cualquier modificación a alguna parte de los datos causa que el descifrado falle. Por lo que sin el conocimiento de la llave, la habilidad de un intruso activo se limita a evitar que los datos se bloqueen y no permitir que alcancen su destino.

La mayoría de los sistemas de autenticación usados en la práctica son simétricos, en donde la misma llave es usada para cifrar y descifrar. Se asume que cada principal comparte una llave secreta con un servidor de autenticación, y que esta llave es establecida por algún método seguro que no involucra una comunicación electrónica. Dos principales pueden comunicarse de manera segura mediante el envío de mensajes cifrados al servidor de autenticación, que puede cifrar de nuevo y enviar los mensajes al recipiente requerido. Sin embargo, cuestiones de escala hacen de ésta una manera impráctica.

Cuando dos principales desean comunicarse, establecen una llave secreta conocida solo entre ellos. Esta llave secreta sirve como un canal de comunicación seguro entre los dos principales porque un intruso activo que no conoce la llave no puede interferir exitosamente con la comunicación. Pero establecer una llave, conocida como llave de sesión, es un problema complejo.

El problema de establecer llaves de sesión seguras entre dos principales en un sistema de autenticación distribuido, estableció las bases para enfocar los esfuerzos para investigarlo. Esta investigación se enfoca en el desarrollo de protocolos de autenticación.

2.4.1 TIPOS DE ANÁLISIS

Meadows [2] define cuatro tipos de análisis de protocolos criptográficos:

Tipo I. El modelado y la verificación del protocolo mediante la utilización de lenguajes de especificación y herramientas de validación que no fueron desarrollados propiamente para el análisis de protocolos criptográficos.

Tipo II. El desarrollo de sistemas expertos que un diseñador de protocolos puede utilizar para desarrollar e investigarlos en diferentes escenarios.

Tipo III. Modelar los requerimientos de los protocolos empleando lógicas diseñadas para el análisis de *conocimientos* y *creencias*.

Tipo IV. El desarrollo de modelos formales basados en propiedades algebraicas de los sistemas criptográficos.

La utilización del Tipo I es el menos utilizado, mientras que el tipo III es el más común. Estos acercamientos comparten unas cuantas propiedades, al igual que en todos los casos las metodologías son independientes de los mecanismos de cifrado utilizados en el desarrollo del protocolo.

2.4.1.1 Tipo I

Este acercamiento al análisis de protocolos criptográficos es para modelar y verificar protocolos utilizando lenguajes de especificación así como herramientas de verificación que no fueron desarrollados propiamente para el análisis de los protocolos de autenticación. La idea principal es tratar un protocolo criptográfico como cualquier otro protocolo e intentar probar su correctitud. Pero el utilizar esta técnica para analizar un protocolo de autenticación no demuestra que el protocolo sea seguro.

Este acercamiento puede ser visto como un intento de utilizar técnicas de verificación auxiliadas por computadora. Las propiedades que el protocolo debe preservar son expresadas como estados invariantes, y los teoremas que deben ser probados garantizan que los medios criptográficos satisfacen las invariantes que son automáticamente generadas por el sistema de verificación.

Debemos de hacer notar que a pesar de que al principio el esfuerzo estaba dirigido a este tipo de acercamiento, la mayor parte del trabajo en esta área se redefinió mediante las lógicas del Tipo III que empezaron a ganar popularidad para la validación de protocolos criptográficos.

2.4.1.2 Tipo II

El acercamiento del Tipo II para el análisis de protocolos criptográficos es el desarrollo de sistemas expertos que un diseñador de protocolos puede utilizar para desarrollar e investigar en diferentes escenarios. Estos sistemas empiezan con un estado no deseado e intenta descubrir si este estado es alcanzable por algún estado inicial.

A pesar de que este acercamiento puede identificar mejor las fallas que el Tipo I, no garantiza la seguridad de un protocolo de autenticación, ni provee de una técnica automática para identificar posibles ataques sobre un protocolo. En otras palabras, el Tipo II puede descubrir cuando un protocolo dado contiene una falla conocida, pero son incapaces de descubrir tipos desconocidos de fallas en los protocolos.

Los sistemas expertos proveen de:

- Una nueva perspectiva en un protocolo de autenticación
- Una técnica para construir modelos capaces de un refinamiento continuo
- Un método de interacción con el modelo, que provee de una comprensión dentro de la operación del sistema
- Un modelo que responde a las preguntas de tipo *qué pasa si*
- Un método de examinar los efectos de las modificaciones propuestas sobre los sistemas

Los sistemas expertos pueden ser usados junto con otras técnicas de análisis, como las del tipo III y IV, pero nunca las llegarán a reemplazar.

2.4.1.3 Tipo III

Este acercamiento al análisis de protocolos usa los modelos de lógica formal desarrollados para el análisis de conocimiento y creencias. Burrows [4] junto con otros investigadores toman como punto principal la lógica BAN, iniciando así una investigación extensa usando este acercamiento.

Las lógicas de autenticación se utilizan en los protocolos de autenticación que se encargan del intercambio de llaves entre dos principales, que puedan estar utilizando una tercera entidad que actúe como generador de llaves recientes o que pueden ser generadas por uno de los principales y

enviada posteriormente. En estos casos la utilización de la lógica de autenticación para realizar la validación puede ser de gran ayuda y es donde mejores resultados dan, por lo que cualquier uso de estas lógicas más allá de este punto puede ser peligroso ya que no es para lo que la lógica fue diseñada, y por lo tanto, deben ser cuidadosamente utilizadas [6] [9].

2.4.1.4 Tipo IV

Este acercamiento al análisis de protocolos desarrolla un modelo formal basado en las propiedades algebraicas de los sistemas criptográficos. Generalmente involucra un análisis de accesibilidad de ciertos estados del sistema. Esta apreciación, es similar en partes al Tipo II que se mencionó antes, sin embargo, el acercamiento de Tipo IV intenta mostrar que un estado inseguro no puede ser alcanzado, mientras que el acercamiento del Tipo II comienza con un estado inseguro e intenta mostrar que no hay una ruta posible entre éste estado y el estado inicial en que se pudo haber originado.

2.4.2 LÓGICA BAN

La lógica BAN es el método utilizado de manera más amplia para el análisis de protocolos de autenticación. Aunque se han propuesto varias extensiones para ampliar la funcionalidad de la lógica BAN, como la lógica GNY [38], ésta sigue siendo la más utilizada para la validación de los protocolos de seguridad.

Según los autores Burrows et al. [4] la validación de un protocolo de seguridad mediante la lógica BAN trata de contestar las siguientes preguntas.

- ¿Qué trata de alcanzar este protocolo?

  Existen varios fines por los que se diseña un protocolo de seguridad: para autenticar un usuario, para transmitir llaves de sesión[7], etc.

- ¿Este protocolo necesita realizar más suposiciones que otro?

  Las suposiciones son los requerimientos que consideran los desarrolladores del protocolo para asegurar que mantendrá la confidencialidad. Estas suposiciones pueden ser que el equipo esté en un entorno protegido donde solo un usuario tenga acceso al equipo, que ya hubo un intercambio previo de las llaves que se utilizarán, etc.

- ¿Realiza pasos innecesarios que pueden eliminarse sin debilitar al protocolo?

  Un ejemplo aplicable a este paso es cuando el protocolo intercambia varios mensajes cifrados como prueba entre emisor y receptor para verificar que el intercambio de llaves fue correcto. Este intercambio de mensajes de prueba puede ser eliminado sin que afecte el servicio que proporciona el protocolo, que en este caso es el intercambio de llaves.

[7] Son llaves que se utilizan una sola vez para cifrar la información transmitida durante la ejecución del protocolo.

- ¿Cifra un mensaje que puede ser enviado en texto claro sin debilitar al protocolo? Esto ocurre cuando existe información que se envía de manera cifrada como el nombre de usuario. Este es un valor que puede ser enviado sin cifrar sin que afecte la seguridad que proporciona el protocolo. Lo único que sería necesario enviar cifrado sería la contraseña asociada al nombre del usuario.

La lógica BAN [4] hace notar los errores inherentes a la implementación de los protocolos, como son los interbloqueos o sistemas criptográficos no apropiados (que son fáciles de descifrar) no se consideran para el análisis; la lógica BAN trata con los protocolos de autenticación en un nivel abstracto indicando si un intercambio de llaves o una autenticación del usuario se lleva a cabo sin que haya problemas de seguridad, como puede ser que la llave sea descubierta por un usuario no autorizado.

Una descripción detallada de los elementos que se usan en la lógica BAN puede ser encontrada en la página 149 en el Anexo E.

Con la elección de la lógica BAN para realizar la validación del protocolo de autenticación en el aspecto de la seguridad, se cumple el objetivo especifico establecido en esta tesis: "*Seleccionar las técnicas para la validación de un algoritmo de autenticación*".

2.5 ALGORITMOS DE CIFRADO SIMÉTRICO

El protocolo de autenticación que se desarrollará utilizará el cifrado de llave secreta. Este cifrado es el de implementación más eficiente para la mayoría de las computadoras actuales, ya que las operaciones, permutaciones en su mayoría, que se realizan para cifrar la información se pueden llevar a cabo en pocos ciclos del microprocesador en comparación con el cifrado de llave pública en donde se utilizan operaciones aritméticas complejas, como factorización de números primos, que ocupan varios ciclos de procesador para llevarse a cabo.

De igual manera, el número de bits de los bloques de información y de las llaves que se ocupan en los algoritmos de cifrado son distintos dependiendo del tipo de algoritmo que se implementa. El número de bits es de 128 a 256, comúnmente, en los algoritmos de llave secreta tanto para los bloques de información como para la llave utilizada, mientras que en los algoritmos de llave pública el tamaño de la llave es de 1024 a 2048 para la llave utilizada.

El protocolo de autenticación propuesto utilizará bloques múltiplos de 128 ya que la información que se transmite no es mucha. Si se utilizaran los bloques de 1024 bits, como es el caso de la criptografía de llave pública, habrá un desperdicio de ancho de banda que no es recomendable ya que no se conoce el estado de las líneas de comunicaciones en donde serán ejecutados los programas cliente.

2.5.1 ALGORITMOS DE CIFRADO ACTUALES.

Entre los algoritmos simétricos más comunes actualmente se encuentran Blowfish [32], IDEA[8] [33] y Rijndael (AES[9]) [20].

2.5.1.1 Blowfish

Es un cifrado simétrico desarrollado por Bruce Schneier, que tiene las siguientes características [32].

- *Rapidez.* Cifra datos en procesadores de 32 bits a una velocidad de 18 ciclos de reloj por byte.
- *Compacto.* Puede ser ejecutado en un entorno con muy poca memoria (5K).
- *Simple.* Tiene una estructura simple y fácil de implementar lo que facilita la tarea de determinar la solidez del algoritmo.
- *Seguridad variable.* El tamaño de la llave es variable con un tamaño máximo de 448 bits. Esto permite que se pueda realizar una implementación del algoritmo en donde se complementen la velocidad y la seguridad en la transmisión.

Blowfish fue desarrollado para ser el sustituto de DES, el cuál se ha demostrado que puede ser descifrado aunque se necesita una máquina muy potente.

2.5.1.2 IDEA

IDEA es un algoritmo simétrico desarrollado por Xuejia y James Massey del *Swiss Federal Institute of Technology* [33]. Este es otro algoritmo que ha sido propuesto para reemplazar a DES como estándar, es una de las propuestas más viables que existen junto con Blowfish.

El diseño del algoritmo consiste de los siguientes elementos.

- *Tamaño del bloque.* El bloque de datos debe ser lo suficientemente grande para evitar el análisis estadístico (esto es, evitar que un intruso tenga la ventaja de que algunos bloques aparezcan más que otros). El uso de un bloque de 64 bits se considera lo suficientemente robusto para realizar el cifrado y evitar dificultar el criptoanálisis.
- *Tamaño de la llave.* Debe ser lo suficientemente largo para prevenir una búsqueda exhaustiva de la llave, como ocurre en el ataque de fuerza bruta. Por lo que se sugiere un tamaño de llave de 128 bits.
- *Complicación.* El texto cifrado debe depender del texto sin cifrar y la llave en una manera compleja. El objetivo es complicar la manera en que el texto cifrado es obtenido del texto sin cifrar sin importar la llave que se este utilizando, esto es, que no se pueda hacer un

[8] Internacional Data Encryption Algorithm

[9] *Advanced Encryption Standard*

análisis estadístico de cómo es obtenido el texto cifrado aplicando diferentes llaves al texto sin cifrar.
- *Difusión.* Cada uno de los bits en el texto sin cifrar debe de influir en cada bit del texto cifrado. El cambio de un bit en el texto en claro debe difundirse en varios bits en el texto cifrado, de esta manera oculta la estructura estadística del texto plano.

Este algoritmo ha tenido mucha aceptación debido a la facilidad de implementación y que no está sujeto a las leyes de exportación de EUA. El programa más conocido en el que se encuentra es en PGP (*Pretty Good Privacy*), programa utilizado para cifrar la información que se intercambia sin necesidad de un servidor central.

2.5.1.3 Rijndael

Es un algoritmo criptográfico desarrollado por Joan Daemen y Vicent Rijmen en el año 1999 [20]. Fue unos de los algoritmos propuestos y el que resultó ganador para ser el nuevo estándar de cifrado en sustitución del algoritmo DES que sería usado por la agencia norteamericana NIST (*National Institute of Standard and Technology*). El algoritmo al ser aprobado por la NIST se le llamó AES (*Advanced Encryption Standard*) El algoritmo cuenta con el siguiente diseño para su realización:

- *Tamaño de la llave.* La llave puede ser definida en bloques de 128, 192 y 256 bits. Siendo utilizado cualquiera de los tamaños como estándar AES.
- *Tamaño del bloque de la información.* El tamaño del bloque en el que se dividirá la información es de 128, 192 y 256 bits. La definición original del algoritmo hecho por Daemen y Rijmen indica que se puede usar cualquier tamaño del bloque de información con cualquier tamaño del bloque de la llave, pero la implementación propuesta por la agencia NIST indica que se debe de usar el bloque de 128 bits para la información que se cifrará.
- *Otras consideraciones.* El algoritmo fue diseñado para ser rápido y con un programa compilado que fue pequeño para poder ser almacenado en los ambientes donde operará el algoritmo criptográfico, como en las tarjetas inteligentes.

El algoritmo fue diseñado para trabajar en una amplia variedad de procesadores, tanto para PC como otros dispositivos, pero enfocado principalmente a los procesadores de 8 bits como los usados en las tarjetas inteligentes y en los procesadores de 32 bits que se utilizan en las PC's.

2.5.2 ELECCIÓN DEL ALGORITMO DE CIFRADO.

El algoritmo que se utilizará para cifrar parte de la información que se transmitirá durante la ejecución de protocolo de autenticación es el Rijndael. La elección de este algoritmo se basa en que es un estándar oficial propuesto por la NIST por lo que se puede suponer con la suficiente confiabilidad en que es lo suficientemente seguro y rápido para ser ejecutado en cualquier computadora en donde pueda ser implementado el protocolo de autenticación que se desarrollará

en esta tesis. Las validaciones de seguridad realizadas al protocolo por NIST se pueden verificar en el artículo *Hardware performance of the AES finalists – survey and analisys of results* [34], de esta manera cumplimos uno de los objetivos planteados al principio de la tesis: "2.2.2.3 *Seleccionar el algoritmo criptográfico a utilizar."*

3 ESTADO DEL ARTE

Iniciaremos este capítulo revisando algunas técnicas y algoritmos utilizados actualmente para realizar la autenticación de usuarios en una red de computadoras, en donde los usuarios que se autentican pueden estar ubicados dentro de la misma red de computadoras o en redes de computadoras distintas.

3.1 PROTOCOLO DE AUTENTICACIÓN DESAFÍO RESPUESTA (CHAP – *CHALLENGE HANDSHAKE AUTHENTICATION PROTOCOL*)

CHAP es utilizado para verificar la identidad de las entidades utilizando un protocolo de tres vías. Es ejecutado cuando se va a establecer la sesión, también el estándar sugiere de manera opcional que se repita la ejecución del protocolo en intervalos de tiempo [16].

Este protocolo está diseñado para ser utilizado principalmente sobre computadoras conectadas a través de líneas telefónicas pero también puede ser utilizado por entidades que utilicen líneas dedicadas.

La manera en cómo la comunicación es llevada a cabo se describe a continuación:

.

1. Una vez que se ha establecido la conexión, la entidad que va a autenticar (autenticador), envía un mensaje con un reto y un valor efímero a la entidad a ser autenticada.
2. La entidad a ser autenticada responde con un valor calculado a partir del reto usando una función *hash*.
3. El autenticador compara la respuesta contra el cálculo que ha realizado, si los valores son iguales el autenticador reconoce el mensaje (esto es, envía un mensaje en donde indica que tuvo éxito), de otra manera se termina la sesión.
4. En intervalos aleatorios el autenticador envía un nuevo reto a la entidad a ser autenticada y se repiten los pasos 1 al 3.

3.1.1 REQUISITOS DEL DISEÑO.

El algoritmo CHAP requiere que el tamaño del secreto sea de al menos un octeto (8 bits). El secreto debe ser al menos tan largo e impredecible como una contraseña bien escogida. Se recomienda que el secreto sea al menos del tamaño de un valor *hash* de la función *hash* que fue escogida, por ejemplo, si se escoge MD5 como función *hash* que maneja 16 bits como salida, el valor a escoger debe tener un tamaño mínimo de 16 bits.

La función *hash* se debe escoger de tal manera que sea computacionalmente indescifrable determinar el secreto cuando se conoce los valores de la pregunta y la respuesta.

Cada reto debe ser único, debido a que la utilización de un valor de reto junto con el mismo secreto permitirá a un atacante utilizar un mensaje que interceptó en una comunicación anterior. Al ser único el reto generado se evita intentos de penetración utilizando mensajes usados en una comunicación anterior.

Cada valor del reto debe ser impredecible, a menos que un atacante engañe a una entidad para responder con un valor predecible y utilice la respuesta para hacerse pasar como la entidad a ser autenticada ante el servidor.

El algoritmo CHAP puede evitar una amplia variedad de ataques activos mediante la utilización de valores no predecibles en el reto.

3.1.2 VENTAJAS

CHAP provee protección contra los ataques en donde se reutilicen los mensajes enviados con anterioridad, mediante la utilización de un número de secuencia incremental y una variable de reto. El uso de preguntas distintas es para reducir el tiempo en que se está utilizando el sistema, en caso de que se hayan expuesto a un ataque. El autenticador es el que tiene el control sobre la frecuencia y el control de tiempo de los retos.

El método de autenticación depende de un “secreto” conocido solo por el autenticador y la entidad a ser autenticada. El secreto no es enviado sobre el medio de comunicación.

A pesar de que la autenticación es llevada a cabo en una dirección únicamente, se puede ejecutar el protocolo en las dos entidades para realizar la autenticación entre ellas.

3.1.3 DESVENTAJAS

CHAP especifica que el secreto sea enviado en texto plano (sin cifrar), por lo que las contraseñas cifradas no pueden ser usadas.

No es conveniente utilizarlo en redes grandes debido a que cada posible secreto debe ser mantenido en los dos lados de la comunicación, en el autenticador y en la entidad a ser autenticada.

El protocolo no tiene manera de saber el número de veces que se han reintentado enviar un reto cuando ha fallado la autenticación, por lo que se recomienda la utilización de un contador para saber el número de veces que la autenticación ha fallado.

3.2 PROTOCOLO DE AUTENTICACIÓN MEDIANTE CONTRASEÑA (PAP – *PASSWORD AUTHENTICATION PROTOCOL*)

El protocolo de autenticación por contraseña proporciona un método sencillo para que los principales establezcan su identidad utilizando un saludo de dos pasos. Ésto hecho únicamente en el establecimiento del enlace inicial de la comunicación [19].

Después de que la fase del establecimiento del enlace esta completa, una tupla usuario/contraseña es enviado de manera repetida por el cliente al servidor hasta que la autenticación es aceptada o la conexión se termina.

PAP no es un método de autenticación robusto. Las contraseñas son enviadas sin cifrar, por lo que no hay protección para los ataques de repetición de mensajes o ataques de prueba y error. El cliente tiene el control de la frecuencia y sincronización de los intentos que se realicen.

A continuación se definen las opciones que se utilizan para la especificación del protocolo PAP.

3.2.1 TRAMAS DEL PROTOCOLO

Este protocolo cuenta con varias tramas que son utilizadas para enviar el identificador del usuario y su contraseña, para indicar al cliente si se reconoció o no su trama de identificación así como para indicar que puede enviar el identificador y la contraseña.

3.2.1.1 Petición de autenticación (Authenticate-*Request*)

La trama de petición de autenticación es utilizada para iniciar el Protocolo de Autenticación por Contraseña. El cliente debe de transmitir esta trama durante la fase de autenticación y debe ser repetido hasta que el servidor nos indique que ha recibido esta trama o el contador de reintentos termine.

El servidor debe de esperar a que el cliente envíe la *Petición de Autenticación* para que envíe el *Reconocimiento de Autenticación* indicando que está listo para recibir más datos.

En la *Petición de Autenticación* debe de ir el nombre del usuario y la contraseña para que el servidor pueda procesarla.

3.2.1.2 Reconocimiento o Desconocimiento de Autenticación (*Authenticate-Ack and Authenticate-Nak*)

Cuando se recibe el identificador y la contraseña en una *Petición de Autenticación* se debe de verificar que ambas sean válidas y aceptadas con el fin de que el servidor pueda enviar el *Reconocimiento de la Autenticación*.

Si el identificador y la contraseña no son válidos o no aceptados, el servidor debe de enviar al cliente un *Desconocimiento de Autenticación*, y el servidor debe de tomar las acciones necesarias para terminar la comunicación con el cliente o permitir otra oportunidad al usuario para autenticarse.

3.2.2 VENTAJAS

PAP es un protocolo que es fácil de implementar en cualquier equipo sin importar la capacidad de memoria o de procesador que pueda tener.

Es muy útil para probar comunicaciones que se establezcan durante el armado de una red.

3.2.3 DESVENTAJAS

No utiliza cifrado por lo que los mensajes que el cliente envíe al servidor pueden ser "vistos" mediante la intervención de la capa física.

Es vulnerable a los ataques de réplica de mensajes así como a los ataques de escucha pasiva, por lo que la contraseña no esta segura en un entorno que utilice el Protocolo de Autenticación por Contraseña.

3.3 SERVICIO DE AUTENTICACIÓN X.509

La recomendación ITU-T[10] X.509 [31] es parte de la serie X.500 de recomendaciones que definen un directorio de servicio. La definición de directorio dentro de la recomendación es la de un servidor o un conjunto de servidores distribuidos que mantienen la información de los usuarios en una base de datos distribuidas.

X.509 es un estándar importante debido a que la estructura del certificado y los protocolos de autenticación definidos en X.509 son usados en una amplia variedad de aplicaciones como en las compras por Internet y seguridad IP.

---

[10] International Telecommunication Union

X.509 se basa en el uso de criptografía de llave pública y firmas digitales. El estándar no especifica la utilización de un algoritmo en especial pero recomienda la utilización de RSA[11] [24] [26]. En el esquema de firma digital se asume el uso de una función *hash* pero en el estándar no se indica un algoritmo *hash* específico.

3.3.1 CERTIFICADOS

La parte principal del esquema X.509 es el certificado de llave pública asociado con cada usuario. El certificado del usuario es creado por alguna autoridad certificadora de confianza[12] (AC) y colocado en un servidor por la AC o por el usuario.

En la Figura 3-1 se muestra el formato general de un certificado, en donde se incluyen los siguientes elementos:

- **Versión.** Marca la diferencia entre las versiones progresivas del formato del certificado; la versión normal es 1. Si el Identificador Único del Emisor o el Identificador Único del Usuario están presentes, el número de la versión debe ser el 2. Si uno o más extensiones están presentes la versión debe de ser 3.
- **Número de serie del certificado.** Un valor entero único dentro de la AC emisora, que es asociado a un certificado para evitar ambigüedades.
- **Identificador del algoritmo.** El algoritmo que fue utilizado para firmar el certificado junto con cualquier parámetro asociado. Como esta información es repetida en el campo de firma que está al final del certificado, tiene poca o ninguna utilidad.
- **Entidad emisora.** Nombre X.500 de la AC que creó y firmó el certificado.
- **Periodo de validez.** Consiste en dos fechas, la fecha inicial y final del periodo en el que el certificado es válido.
- **Nombre del usuario.** El nombre del usuario a quien el certificado se refiere. El certificado legaliza que la llave pública pertenece a un usuario específico, y este usuario tiene la correspondiente llave privada.
- **Llave pública del usuario.** La llave pública del usuario más un identificador del algoritmo para el cuál esta llave es usada junto con cualquier parámetro asociado.
- **Identificador único del emisor.** Un campo opcional usado para identificar de manera única a la AC emisora en el caso de que su identificador en X.500 haya sido utilizado en otra ocasión por otras entidades emisoras.

Identificador único del usuario. Un campo opcional usado para identificar de manera única al usuario en caso de que su identificador en X.500 haya sido utilizado en otras ocasiones por otros usuarios.

---

[11] Rivest-Shamir-Adleman

[12] La autoridad certificadora debe ser reconocida mundialmente y fiable para que se pueda confiar en ella. La más famosa es la creada por los desarrolladores originales del criptosistema RSA.

- **Extensiones.** Un conjunto de uno o más campos de extensiones que fueron agregadas en la versión 3. Cada extensión consiste en un identificador de la extensión, la importancia y el valor de la extensión.
- **Firma.** Abarca los otros campos en el certificado; contiene el código *hash* de los campos y esta cifrado con la llave privada de la AC. En este campo se incluye el identificador del algoritmo de firmado.

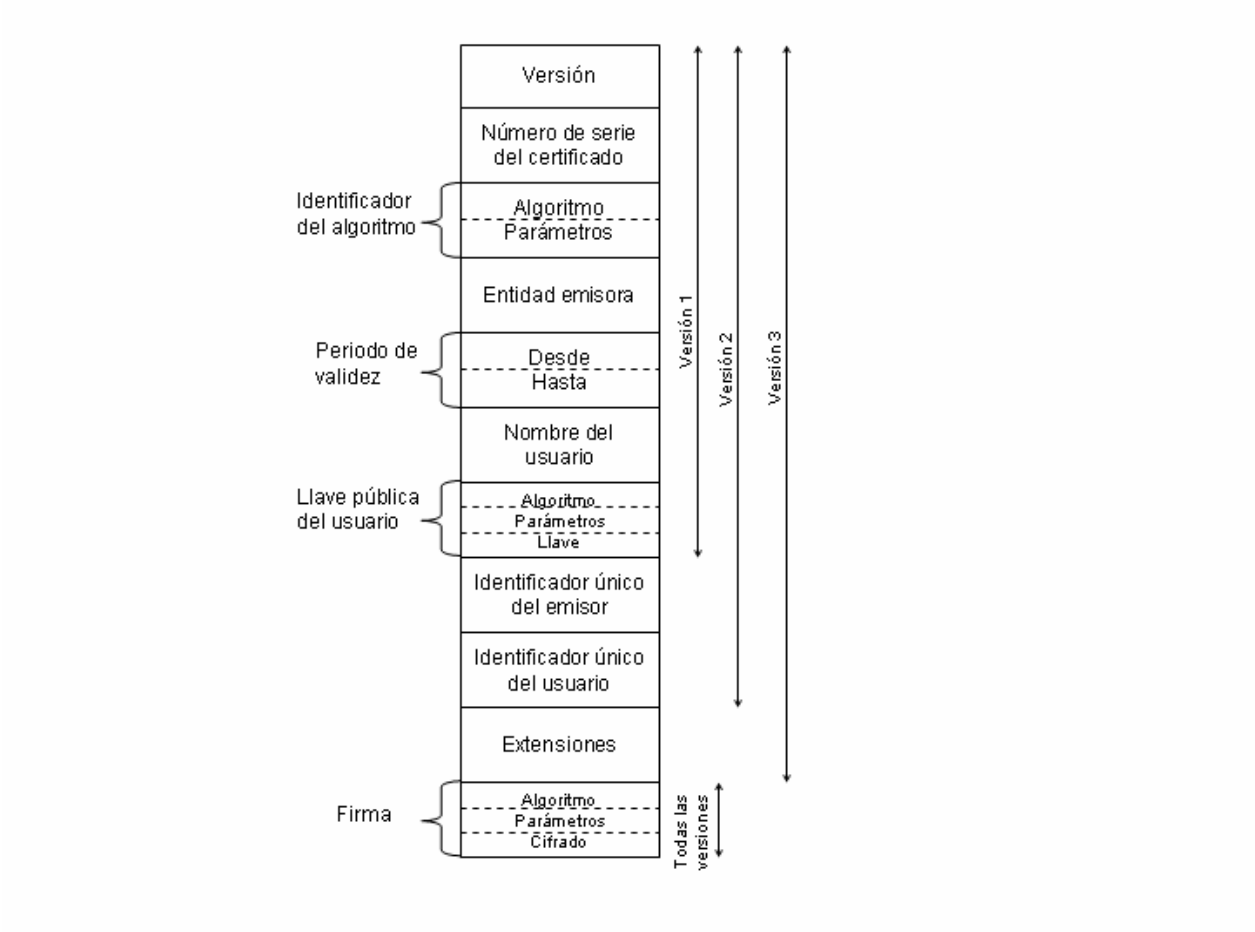

Figura 3-1 Formato del certificado X.509

3.3.2 FORMAS DE AUTENTICACIÓN EN CERTIFICADOS

En el estándar X.509 se incluyen tres alternativas para realizar la autenticación de las entidades que están diseñadas para ser utilizadas en una amplia variedad de aplicaciones. Todos los procedimientos se basan en la utilización de las firmas de llave pública. Se asume en el estándar que las dos entidades a autenticarse conocen la llave pública del otro, ya sea porque el certificado fue obtenido de un directorio, o porque el certificado fue incluido en el mensaje inicial que envió una entidad.

3.3.2.1 Autenticación en un paso

La autenticación presentada en la Figura 3-2, implica solo una transferencia de información de un usuario A hacia un usuario B en donde se indica lo siguiente:

1. La identidad de A y el mensaje que fue generado por A.
2. A quién va dirigido el mensaje, en este caso es dirigido a B.
3. La integridad y originalidad (esto es, que no ha sido enviado con anterioridad) del mensaje.

Hay que ver que solo la identidad de la entidad que quiere establecer la comunicación es confirmada en este proceso y la identidad de la entidad receptora no es establecida en este protocolo.

El mensaje debe incluir, al menos, una marca temporal t_A, un valor efímero r_A y la identidad de la entidad con la que se desea establecer comunicación, en este caso B y todo esto es firmado con la clave de A. El uso de la marca de tiempo y el valor efímero se utiliza para evitar que el mensaje sea reutilizado en el futuro por un intruso. En el mensaje se envía juntas la marca temporal t_A, el valor efímero r_A, el mensaje que se desea enviar (m) y la llave secreta (K_{ab}) que se utilizará para la comunicación entre los dos usuarios, la llave viene cifrada con la llave pública de B (K_b) y todos estos elementos viene firmados con la llave secreta de A para que B pueda comprobar que el mensaje solo pudo provenir de A.

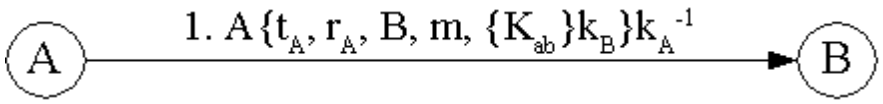

Figura 3-2 Autenticación en un paso en el certificado X.509

3.3.2.2 Autenticación en dos pasos.

Se añaden lo siguientes elementos a los puntos mencionados en la *autenticación de una dirección*, como se muestra en la Figura 3-3.

1. La identidad de B y un mensaje de réplica que fue generado por B.
2. El mensaje es dirigido a la entidad que originó el primer mensaje, en este ejemplo a la entidad A.
3. La integridad y originalidad de la réplica.

Este tipo de autenticación permite que las dos entidades se autentiquen entre sí.

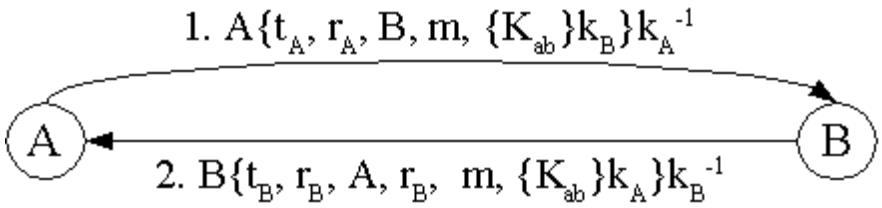

Figura 3-3 Autenticación en dos pasos en el certificado X.509

3.3.2.3 Autenticación en tres direcciones.

En la autenticación que se muestra en la Figura 3-4, se envía un mensaje final de A hacia B, que contiene una copia del valor efímero r_B. La marca temporal no es necesario verificarla, debido a que los valores efímeros que han sido utilizados fueron respondidos en el mensaje de réplica y de esta manera se pueden utilizar para detectar mensajes que ya fueron utilizados. Este enfoque es necesario cuando no se tienen los relojes sincronizados para establecer las marcas temporales en las dos entidades y no hay manera de sincronizarlos.

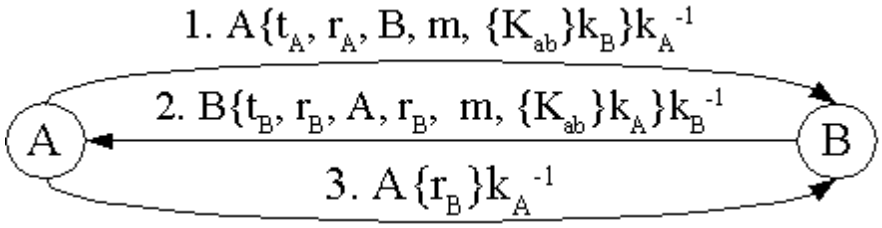

Figura 3-4 Autenticación en tres pasos en el certificado X.509

3.4 KERBEROS

El sistema de autenticación y de distribución de claves Kerberos fue desarrollado en el Instituto de Tecnología de Massachussets (MIT) para proteger los servicios de red que surgieron con el proyecto Athena[13][21]. El entorno del proyecto Athena estaba formado por los siguientes elementos:

- Estaciones de trabajo públicas y privadas.
  - Las estaciones públicas se localizan en áreas sin seguridad o con seguridad mínima.
  - Las estaciones privadas están bajo control físico y administrativo de individuos generalmente sin responsabilidad de administradores centrales de la red.

- Una red de campus, formada por redes de área local de varios tipos conectados a una red troncal. Las redes de área local están dispuestas en emplazamientos geográficamente

[13] Sistema de cómputo distribuido que proporciona servicios de comunicación a los usuarios dentro del MIT

dispersos y son vulnerables a diversos ataques. Por el contrario, los dispositivos de la red troncal están encerrados en armarios y funcionan por tanto en condiciones de seguridad física moderada.

- Servidores operados centralmente. La mayor parte están situados en habitaciones cerradas, y funcionan por tanto en condiciones de seguridad física moderada con software que no contiene código malicioso. Sólo unos pocos servidores funcionan en condiciones de considerable seguridad física y se pueden utilizar como servidores de seguridad.

En este entorno las amenazas principales a la seguridad surgen de la posibilidad de que el usuario de una estación de trabajo falsifique la identidad de otro usuario para obtener acceso no autorizado a los recursos del sistema. Las estaciones de trabajo están bajo el control completo de los usuarios que pueden intentar un engaño para suplantar a otro usuario o sistema. La privacidad de los datos transmitidos por la red es de baja prioridad, excepto donde es necesario evitar violaciones de la seguridad.

3.4.1 ARQUITECTURA

Los dominios de administración se denominan *reinos*. La compañía u organización que desee utilizar Kerberos establecerá un reino unívocamente determinado por un *nombre de reino*. En teoría, cada reino de Kerberos puede admitir hasta 100,000 usuarios.

Kerberos se basa en el modelo cliente/servidor. Los usuarios, clientes y los servicios de red ejecutándose en sistemas concretos se consideran generalmente principales. Cada principal se identifica mediante un único *identificador de principal*. El identificador esta compuesto por tres campos, cada uno de los cuales es una cadena de texto de hasta 40 caracteres, con diferenciación de mayúsculas y minúsculas, finalizada con el carácter nulo. Los tres campos componentes son:

- Un nombre de principal *NAME* que se utiliza para identificar un usuario en particular.
- Un nombre de realización *INSTANCE* que comúnmente se utiliza para identificar el sistema particular en el que se proporciona el servicio.
- Un nombre de reino *REALM* en donde es común utilizar el nombre del dominio de Internet de la empresa u organización.

El objetivo de Kerberos es permitir que un cliente ejecutándose en nombre de un usuario particular pruebe su identidad a un servicio o al correspondiente servidor de aplicaciones sin necesidad de enviar datos por la red que podrían facilitar el que un atacante suplantase posteriormente al usuario. El modelo de Kerberos requiere la existencia de una tercera entidad de confianza que actúe como centro de distribución de claves (KDC – *key distribution center*) en el reino de Kerberos, el cuál consta de dos componentes:

1. Un *servidor de autenticación* (AS – *authentication server*).
2. Un conjunto de *servidores de emisión de billetes* (TGS – *ticket granting servers*).

Estos son componentes separados solo de manera lógica lo cuál no quiere decir que no puedan estar en la misma máquina.

Kerberos utiliza un servidor de autenticación al igual que para la distribución de clave, pero debido al alcance de esta tesis solo nos limitaremos a la descripción del servidor de autenticación.

El KDC mantiene una base de datos con un registro por cada principal registrado en el reino. Para permitir que la seguridad de los datos sea la principal consideración cuando existen compromisos operacionales en la gestión de Kerberos, la información que Kerberos almacena es la mínima necesaria para realizar y gestionar las tareas de autenticación. La información que se almacena en el KDC de Kerberos para cada principal es:

- El identificador de principal de P.
- La clave maestra K_p (o la contraseña de P si es un usuario).
- Una fecha de expiración de la identidad de P.
- La fecha en la que el registro se modificó por última vez.
- La identidad del principal que modificó el registro por última vez.
- El tiempo de vida máximo de los billetes suministrados al principal.
- Algunos atributos.
- Algunos datos relacionados con la implementación que no son visibles externamente, como una versión de clave, una versión de clave maestra o apuntadores a los valores anteriores del registro.

Hay una parte de información de cada registro, concretamente la clave maestra K_P, que se debe mantener secreta. Por este motivo, Kerberos cifra todas las claves maestras de los principales con una clave maestra del KDC.

3.4.2 AUTENTICACIÓN

La autenticación de un usuario ante un servidor se realiza de la siguiente manera:

1) El usuario se conecta en una estación de trabajo y solicita acceso a un servicio. El módulo del cliente en la estación de trabajo le solicita al usuario su identificación y la contraseña, mismas que son enviadas al servidor de autenticación.
2) El servidor de autenticación busca en la base de datos la información que el usuario proporcionó y si ésta es válida para poder utilizar el servicio.
3) Si la información enviada por el usuario es correcta y tiene los permisos para utilizar el servicio, el servidor de autenticación crea un *ticket* que contiene el identificador del usuario y la dirección de red en donde está la estación de trabajo que usa el usuario. El *ticket* es cifrado con la llave secreta compartida entre el servidor de autenticación y el proveedor del servicio. El *ticket* es devuelto al cliente y como está cifrado no puede ser alterado por éste.
4) El cliente envía el *ticket* junto con su identificador de cliente al proveedor del servicio. El proveedor del servicio descifra el *ticket* y comprueba que el identificador del

usuario es el mismo que el identificador que llegó sin cifrar. Si los identificadores concuerdan, el servidor considera que el usuario esta autenticado y permite la utilización del servicio.

En la parte del cliente, el billete se guarda para uso futuro, para obtener otros billetes que se emplearán para autenticar servicios de red concretos. El principal propósito del billete inicial es proporcionar a los usuarios una utilidad de registro único. De esta manera, el usuario introduce su contraseña sólo una vez y no es preguntado continuamente por ella cada vez que solicita un servicio.

Los procedimientos de autenticación entre entidades que se han visto (X.509, CHAP, PAP y Kerberos) tienen sus ventajas y desventajas. Las aplicaciones basadas en X.509 y Kerberos requieren que las estaciones de trabajo estén bajo control del usuario (en X.509 ya que almacena su clave privada) o estén dentro de una red establecida para que el cliente sepa dónde se encuentran los servidores de autenticación (como en Kerberos).

Los otros dos protocolos están diseñados para funcionar entre dos usuarios, en donde las estaciones de trabajo no tienen que estar bajo control del usuario o de la red de una institución. Pero estos protocolos tienen una seguridad baja al no contar con el cifrado de la información lo que permite que cualquier intruso pueda conocer la clave utilizada. Por eso considero que sería una buena opción desarrollar un protocolo que tenga las ventajas de implementación de CHAP o PAP e la seguridad proporcionada por los algoritmos criptográficos para asegurar que los datos estan seguros mientras no se conozca la llave que cifro los datos, lo que es el objetivo principal de la presente tesis, desarrollar un protocolo de autenticación de usuarios con independencia de los medios de comunicación por donde se transmita la información.

3.5 DEFINICIÓN DEL PROBLEMA

El sistema de autenticación que se pretende desarrollar en la presente tesis funcionará sobre Internet, de esta manera se aprovecha la infraestructura que ya existe tanto en capa física como en la capa de enlace de datos para la información transmitida. La forma de operación del sistema es que una computadora cliente pueda comunicarse con una computadora servidor. La computadora cliente tiene como único requisito que debe tener una conexión a Internet mientras que la computadora servidor debe ser un equipo fijo con una conexión a Internet permanente y una dirección IP establecida, esto con el fin de que el cliente sepa la dirección a la que debe de comunicarse.

El protocolo tendrá como función establecer la identidad de un cliente y permitirle o denegarle el acceso al servidor. Se utilizarán los protocolos de Internet, TCP y UDP para realizar la comunicación entre el cliente y el servidor.

3.5.1 MOTIVACIÓN

La importancia de establecer un protocolo de autenticación en el servidor de un sistema de manufactura se debe a que se manejan recursos que son costosos y difíciles de reemplazar, costosos por ser equipos sofisticados y difíciles de reemplazar ya que se tienen que pedir con mucha anticipación por ser fabricados en muchas ocasioes a la medida del usuario; el protocolo de seguridad tendrá una función importante dentro de las comunicaciones entre el cliente y el servidor ya que evitará que un usuario mal intencionado pueda tener comunicación efectiva con el servidor.

Por estas causas, surge la necesidad de establecer un protocolo de autenticación adecuado a nuestras necesidades que actué junto con un protocolo de control entre el cliente y el servidor. Sin embargo el protocolo de seguridad no debe de afectar el desempeño del protocolo de control por lo que las operaciones realizadas dentro del protocolo de seguridad no deben de ocupar muchos ciclos del procesador, por este motivo se utilizará la criptografía de clave secreta ya que ésta tiene operaciones más sencillas que ocupan pocos ciclos de procesador para cifrar y descifrar la información.

Los protocolos, tanto el de seguridad como el de control, no tendrán una comunicación entre sí lo que facilita la administración ya que no se tendrá que manejar recursos compartidos. El único recurso compartido que tendrán es el puerto empleado para enviar y recibir la información, pero la administración de este puerto la realiza el sistema operativo.

Otra cosa que se evitará dentro del protocolo de seguridad es el uso de mensajes de confirmación, ya que si el usuario no es autenticado correctamente no se le permitirá realizar la conexión. En el caso de que el usuario haya sido autenticado correctamente se le permitirá establecer o continuar la comunicación por lo que no es necesario informarle que la conexión tuvo éxito. El intercambio de información bajo este esquema no sobrecarga los canales de comunicaciones optimizando el flujo de información de control entre las aplicaciones cliente y servidor.

3.5.2 CONOCIMIENTOS APLICADOS

Las herramientas utilizadas para el proyecto fueron las siguientes:

- **Sistema Operativo Linux.** Es el sistema sobre el que trabajará el proceso servidor, administrará los recursos que actuarán sobre el sistema de manufactura y al que los clientes podrán conectarse [25].
- **Sistema Operativo Windows.** Es el sistema sobre el que trabajará el proceso cliente, se utilizó por la facilidad de uso que tiene *Windows* disminuyendo de esta manera la curva de aprendizaje que habrá cuando el usuario aprenda el funcionamiento del sistema cliente.
- **El lenguaje C.** Fue utilizado para desarrollar el servidor que funcionará bajo el sistema operativo Linux, esto se hizo por el conocimiento adquirido durante la materia

Sistemas Distribuidos I de la maestría, lo que motivó su elección sobre otros lenguajes utilizados en Linux, uno de los cuáles podría ser Java.

- **Lenguaje Delphi.** Se utilizó para programar el sistema Cliente, la elección de este sistema se debió a la facilidad de uso, así como a una mayor disponibilidad de librerías que serán útiles para el desarrollo del Cliente, como son la utilización de librerías de cifrado para la transmisión de los mensajes.
- **Conocimientos de Diseño y Validación de Protocolos.** Los conocimientos adquiridos durante la materia de *Diseño y Validación de Protocolos*, serán útiles al momento de modelar el protocolo que se utilizará para comunicar la información entre el cliente y el servidor, entre los conocimientos están la utilización de una herramienta de diseño que sería *Promela*, así como una herramienta de validación vista durante la clase que es *Spin*, también los elementos para un correcto diseño de protocolos.
- **Conocimientos de validación de protocolos de seguridad.** Estos conocimientos fueron adquiridos durante la clase de *Seguridad Computacional*, en donde se vio la validación de un protocolo de seguridad empleando la lógica BAN para demostrar que el protocolo se puede considerar seguro desde un punto de vista formal.

3.5.3 PROPUESTAS DE SOLUCIÓN

Se proponen tres protocolos para ser validados de dos formas. La primera forma es la validación mediante la lógica BAN, en donde se comprueba la seguridad del protocolo. En la segunda forma, el protocolo que pase esta prueba de la lógica BAN de manera satisfactoria, se validará mediante una herramienta de validación de protocolos Spin en donde se comprueba la eficacia del protocolo. Los protocolos utilizarán cifrado simétrico dentro de algunos de los mensajes que se intercambian, como los mensajes que se utilizarán para realizar la identificación del usuario ante el servidor, la pregunta al usuario y la respuesta correspondiente a esa pregunta.

Antes de mencionar los protocolos que se propondrán se definirá la notación a utilizar durante la propuesta de los protocolos, la cuál es:

- ***A*.** Es el cliente que se conectará al servidor.
- ***S*.** El servidor que hará la autenticación del usuario *A*.
- N_a. Un número único para identificar que el mensaje no ha sido enviado en una comunicación anterior, este número es generado por el cliente.
- N_s. Un número único para identificar que el mensaje no ha sido enviado en una comunicación anterior, este número es generado por el servidor.
- K_{as}. La llave secreta que es compartida entre el servidor y el cliente para cifrar la información que se envién entre ellos.
- ***Pregunta*.** Representa la trama que el servidor envia al cliente para verificar que sea un usuario válido.
- ***Respuesta.*** Es la trama que representa lo que el cliente envia para confirmar que es un cliente válido.

Los protocolos propuestos para ser implementados son los que se muestran a continuación.

Primer Protocolo propuesto (P1).

P1.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P1.2 $S \rightarrow A : \{N_a, N_s, Pregunta\}_{K_{as}}$

P1.3 $A \rightarrow S : \{N_s, Respuesta\}_{K_{as}}$

Aplicando el principio 1 definido por Abadi [39] definiremos el primer protocolo propuesto.

En el primer paso, el cliente envía un mensaje al servidor, cuyo contenido es un identificador que es necesario indicarlo explícitamente de acuerdo con el principio 3 de [39] para que el servidor sepa que cliente es el que inicio la comunicación y también en el mensaje se incluye un *nounce* cifrado que el cliente sabe que es reciente debido a que fue generado por él mediante una función generadora de números aleatorios.

En el segundo paso, el servidor envía un mensaje al cliente, cuyo contenido es el *nounce* que recibió del cliente en el primer paso, un *nounce* generado por el servidor mediante una función generadora de números aleatorios y una *pregunta*. Todo el mensaje va cifrado en este paso.

En el tercer paso, el cliente envía un mensaje al servidor, cuyo contenido es el *nounce* enviado por el servidor en el segundo paso y una *respuesta* a la pregunta hecha en el segundo paso.

Segundo Protocolo propuesto (P2).

P2.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P2.2 $S \rightarrow A : \{N_a, Pregunta\}_{K_{as}}$

P2.3 $A \rightarrow S : \{Respuesta\}_{K_{as}}$

Aplicando el principio 1 de [39] definiremos el segundo protocolo propuesto.

En el primer paso, el cliente envía un mensaje al servidor, cuyo contenido es un identificador que es necesario indicarlo explícitamente de acuerdo con el principio 3 de [39] para que el servidor sepa que cliente es el que inicio la comunicación y también en el mensaje se incluye un *nounce* cifrado que el cliente sabe que es reciente debido a que fue generado por él mediante una función generadora de números aleatorios.

En el segundo paso, el servidor envía un mensaje al cliente, los elementos del mensaje incluyen un *nounce* que recibió del cliente en el primer paso y una *pregunta*. Todo el mensaje va cifrado en este paso.

En el tercer paso, el cliente envía un mensaje al servidor. Lo único que incluye el mensaje es una *Respuesta* que va cifrada.

Tercer protocolo propuesto (P3).

P3.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P3.2 $S \rightarrow A : \{N_s, N_a + 1\}_{K_{as}}, Pregunta$

P3.3 $A \rightarrow S : \{N_s + 1, Respuesta\}_{K_{as}}$

Aplicando el principio 1 de [39] definiremos el tercer protocolo propuesto.

En el primer paso, el cliente envía un mensaje al servidor, cuyo contenido es un identificador que es necesario indicarlo explícitamente de acuerdo con el principio 3 de [39] para que el servidor sepa que cliente es el que inicio la comunicación y también en el mensaje se incluye un *nounce* cifrado que el cliente sabe que es reciente debido a que fue generado por él mediante una función generadora de números aleatorios.

En el segundo paso, el servidor envía un mensaje al cliente. Este mensaje contiene un *nounce* generado por el servidor mediante una función generadora de números aleatorios, envía el *nounce* que recibió del cliente en el primer paso y una *pregunta* que debe ser respondida por el cliente. El mensaje va cifrado solo en la sección de los *nounces*, la pregunta es transmitida en claro, esto no es un problema de seguridad mientras la *respuesta* si este cifrada aplicando el principio 4 de [39].

En el tercer paso, el cliente envía al servidor un mensaje que incluye el *nounce* que recibió el cliente en el segundo paso y la *respuesta*, todo el mensaje va cifrado.

En cada uno de los protocolos anteriores se aplicaron los principios 6 y 7 de [39] para indicar que los *nounces* fueron utilizados para que las entidades sepan el orden de los mensajes y para garantizar la frescura de los mensajes. De acuerdo con la definición del principio 4 el cifrado fue utilizado para proteger la información transmitida, especialmente los *nounces* y que una entidad ajena a la comunicación no pueda conocer el contenido de los mensajes y por lo tanto no haga una réplica posterior de mensajes.

Aplicando el principio 10, la codificación de los mensajes llevaría el siguiente formato: los *nounces* serán números aleatorios de 4 bytes de longitud y la *pregunta/respuesta* serán cadenas de texto de 32 bytes de longitud.

Los principios restantes no fueron utilizados durante el diseño de los protocolos propuestos debido a que se refieren a funcionalidad que no es utilizada durante la ejecución del protocolo, por ejemplo el principio 5 de [39] se refiere a la firma de mensajes que es algo que no se emplea en los protocolos propuestos o como el principio 8 de [39] que se refiere a la utilización de tiempos absolutos que tampoco se utiliza en los protocolos propuestos.

En todos los protocolos propuestos anteriormente la idea básica es la misma: el cliente envía una identificación que comúnmente es el nombre de un usuario, el servidor para autenticar al usuario envía una pregunta específica a cada usuario y en el tercer paso el cliente contesta la pregunta. Si

contestó de manera correcta, se permite que establezca o continúe con el protocolo de control, en caso contrario se le corta el acceso. La diferencia entre cada uno de los protocolos son los elementos que se cifraran dentro de la comunicación que se establezca.

En los protocolos propuestos se realiza cifrado de algunos mensajes o parte de ellos, pero la forma en que se intercambia la contraseña que se utiliza para realizar el cifrado no es parte de los objetivos de esta tesis por lo que se puede utilizar cualquier método que los usuarios consideren necesarios, ya sea entregando personalmente la contraseña o por algún medio que ellos consideren seguro.

En el siguiente capítulo se procederá a la validación de estos protocolos para comprobar la efectividad que pueden tener, y la viabilidad de programar el protocolo elegido así como una descripción de la información que se intercambia en cada uno de los protocolos.

4 DESARROLLO DE LAS PROPUESTAS DE SOLUCIÓN

Como se vio al final del capítulo anterior, las propuestas de solución para la autenticación son las siguientes.

Primer Protocolo propuesto (P1).

P1.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P1.2 $S \rightarrow A : \{N_a, N_s, Pregunta\}_{K_{as}}$

P1.3 $A \rightarrow S : \{N_s, Respuesta\}_{K_{as}}$

Segundo Protocolo propuesto (P2).

P2.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P2.2 $S \rightarrow A : \{N_a, Pregunta\}_{K_{as}}$

P2.3 $A \rightarrow S : \{Respuesta\}_{K_{as}}$

Tercer protocolo propuesto (P3).

P3.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P3.2 $S \rightarrow A : \{N_s, N_a\}_{K_{as}}, Pregunta$

P3.3 $A \rightarrow S : \{N_s, Respuesta\}_{K_{as}}$

Con las tres propuestas de solución mostradas aquí se realizará la validación de cada uno de los protocolos, primero por la lógica BAN para establecer cuál se considera mas efectivo considerando los puntos que manejan los autores, que a manera de recordatorio mencionamos de nuevo:

- ¿Qué trata de alcanzar este protocolo?
- ¿Este protocolo necesita realizar más suposiciones que otro?
- ¿Realiza pasos innecesarios, pasos que pueden eliminarse sin debilitar al protocolo?
- ¿Cifra un mensaje que puede ser enviado en texto claro sin debilitar al protocolo?

Hay que recordar también que el objetivo que se desea alcanzar es que el servidor confíe en el cliente con el que se estableció la comunicación, esto quiere decir, que el servidor sepa con quien esta hablando, por lo que se utilizará un nombre de usuario para identificar a un usuario específico.

Segundo, mediante una validación del funcionamiento del protocolo como si fuera un protocolo sin técnicas de seguridad. Esta validación será útil para saber que el protocolo seleccionado no tiene errores en la lógica que podría llevar a un interbloqueo durante el intercambio de la información.

4.1 VALIDACIÓN MEDIANTE LA LÓGICA BAN

En la lógica BAN el postulado que queremos alcanzar es:

$$S \models A \models Respuesta$$

El cuál nos indica que el servidor confía en que el cliente confía en la respuesta, o de manera más sencilla, el servidor sabe que solo el cliente (o alguien de su confianza) pudo enviar la respuesta.

Hay que indicar que la validación hace algunas suposiciones para desarrollar la validación de los protocolos; un ejemplo de estas suposiciones es que considera que no hay errores en los mensajes enviados, como puede ser que las preguntas y respuestas recibidas siempre son correctas.

Antes de empezar con las validaciones mediante la lógica BAN de los protocolos propuestos, se mencionará de nuevo la notación utilizada para representar a los protocolos.

- ***A*.** Es el cliente que se conectará al servidor.
- ***S*.** El servidor que hará la autenticación del usuario *A*.
- N_a**.** Un número único para identificar que el mensaje no ha sido enviado en una comunicación anterior, este número es generado por el cliente.
- N_s**.** Un número único para identificar que el mensaje no ha sido enviado en una comunicación anterior, este número es generado por el servidor.
- K_{as}**.** La llave secreta que es compartida entre el servidor y el cliente para cifrar la información que se envién entre ellos.
- ***Pregunta*.** El servidor realiza la pregunta al cliente para verificar que sea un usuario válido.
- ***Respuesta.*** Lo que el cliente contesta al servidor para confirmarle que es un cliente válido.

4.1.1 PRIMER PROTOCOLO PROPUESTO (P1)

Para la validación, necesitamos los postulados y suposiciones iniciales con los que trabaja la lógica BAN que es la idealización siguiente:

P1.1 $A \rightarrow S : \{N_a\}_{K_{as}}$
P1.2 $S \rightarrow A : \{N_a, N_s, Pregunta\}_{K_{as}}$
P1.3 $A \rightarrow S : \{N_s, Respuesta\}_{K_{as}}$

Para conocer más detalles del protocolo uno propuesto remítase en la página 51.

Se definen las suposiciones iniciales que nos servirán para comenzar a derivar más postulados para probar si es conveniente este protocolo.

1.1 $A \models A \overset{K_{as}}{\leftrightarrow} S$ 1.2 $S \models A \overset{K_{as}}{\leftrightarrow} S$
1.3 $A \models S \Rightarrow Pregunta$ 1.4 $S \models A \Rightarrow Respuesta$
1.5 $A \models \sharp(N_a)$ 1.6 $S \models \sharp(N_s)$

Estos postulados son demasiado generales, por lo que pueden ser definidos en casi todos los protocolos que se validen. A continuación, se explicará cuál es el significado que tiene cada uno de los postulados.

Los postulados 1.1 y 1.2 establecen que el cliente sabe que comparte una llave de cifrado con el servidor; el siguiente postulado establece que el servidor sabe que comparte una llave de cifrado con el cliente.

Los postulados 1.3 y 1.4 afirman que el cliente sabe que solo el servidor tiene control sobre las Preguntas hechas, esto es, que solo el servidor sabe cuáles son las preguntas y solo él puede generar preguntas válidas. El siguiente postulado es similar, quiere decir que el servidor sabe que solo el cliente puede generar las respuestas válidas.

Los postulados 1.5 y 1.6 significan que el cliente sabe que el valor N_a es reciente porque él mismo lo acaba de enviar; lo mismo para el servidor, sabe que N_s es reciente porque lo acaba de enviar.

4.1.1.1 Validación.

Paso 1. El mensaje P1.1 nos indica que el servidor recibe un mensaje cifrado, y de las suposiciones iniciales 1.1 y 1.2 sabemos que es una llave que comparte con el cliente por lo que se obtiene como conclusión que el mensaje fue enviado por el cliente en algún momento. La

suposición es que el mensaje fue enviado en la ejecución actual del protocolo o en una ejecución anterior como se indica en el postulado 1.7.

$$1.7 \quad \frac{S \triangleleft \{N_a\}_{K_{as}}, S \models A \overset{K_{as}}{\longleftrightarrow} S}{S \models A \mid\sim N_a}$$

Paso 2. En el mensaje P1.2, el cliente recibe un mensaje cifrado bajo una clave que comparte con el servidor de acuerdo con el postulado 1.1 inicial. El cliente, al descifrar el mensaje, ve que los elementos incluidos en el mensaje son una pregunta, el valor (N_a) y el valor (N_s) y sabe que este mensaje pudo haber sido enviado por el servidor en algún momento dentro de la ejecución actual o una ejecución anterior del protocolo como se indica en el postulado 1.8. Se supone que el servidor es quien envió el mensaje porque la clave con la que se cifró el mensaje solo es conocida por el cliente y el servidor.

En el postulado 1.9 se demuestra que el cliente sabe que el mensaje fue enviado en la ejecución actual del protocolo, debido a que el cliente sabe que el valor N_a es reciente como se indica en el postulado inicial 1.5.

$$1.8 \quad \frac{A \triangleleft \{N_a, N_s, Pregunta\}_{K_{as}}, A \models A \overset{K_{as}}{\longleftrightarrow} S}{A \models S \mid\sim (N_a, N_s, Pregunta)}$$

$$1.9 \quad \frac{A \triangleleft (N_a, N_s, Pregunta), A \models \sharp(N_a)}{A \models \sharp(N_a, N_s, Pregunta)}$$

Paso 3. En el mensaje P1.3, el servidor recibe un mensaje cifrado bajo la clave que comparte con el cliente de acuerdo con el postulado 1.2 inicial. El servidor, al descifrar el mensaje, ve que los elementos incluidos en el mensaje son la respuesta y el valor N_s y sabe que este mensaje pudo haber sido enviado por el cliente en algún momento dentro de la ejecución actual o una ejecución anterior del protocolo como se indica en el postulado 1.10. El servidor supone que el cliente es quién envió el mensaje porque la clave con la que se cifró el mensaje solo es conocida por el cliente y el servidor.

En el postulado 1.11 se demuestra que el servidor sabe que el mensaje fue enviado en la ejecución actual del protocolo, debido a que el servidor sabe que el valor N_s es reciente como se específico en el postulado inicial 1.6.

$$1.10 \quad \frac{S \triangleleft \{N_s, Respuesta\}_{K_{as}}, S \models A \overset{K_{as}}{\longleftrightarrow} S}{S \models A \mid\sim (N_s, Respuesta)}$$

$$1.11 \quad \frac{S \triangleleft (N_s, Respuesta), S \models \sharp(N_s)}{S \models \sharp(N_s, Respuesta)}$$

Paso 4. Solo falta obtener el postulado que indique que el servidor confía en la respuesta que le envío el cliente, esto se logra con el postulado 1.11 obtenido en el paso tres junto con la suposición inicial 1.4 como se demuestra en el postulado 1.12. En este postulado la lógica seguida para el servidor es que al ser reciente el mensaje y como sabe que solo el cliente tiene autoridad sobre la *Respuesta* ya que solo el las conoce las respuestas válidas, entonces el servidor puede creer que el cliente cree en la *Respuesta* para esta ejecución del protocolo.

$$1.12 \quad \frac{S \models \sharp(N_s, Respuesta), S \models A \Rightarrow Respuesta}{S \models A \models Respuesta}$$

Paso 5. Como se puede observar llegamos a la conclusión que se esperaba. El servidor confía en la respuesta enviada por el cliente, como se confirmó en el postulado 1.12.

4.1.2 SEGUNDO PROTOCOLO PROPUESTO (P2)

Definimos los postulados a utilizarse dentro de la lógica BAN, empezando por los postulados del protocolo que se validará.

P2.1 $A \rightarrow S : \{N_a\}_{K_{as}}$
P2.2 $S \rightarrow A : \{N_a, Pregunta\}_{K_{as}}$
P2.3 $A \rightarrow S : \{Respuesta\}_{K_{as}}$

Para conocer más detalles del protocolo dos propuesto remítase a la página 51.

También se definen los postulados iniciales, que son similares a los del primer protocolo.

2.1 $A \models A \overset{K_{as}}{\leftrightarrow} S$ 2.2 $S \models A \overset{K_{as}}{\leftrightarrow} S$
2.3 $A \models S \Rightarrow Pregunta$ 2.4 $S \models A \Rightarrow Respuesta$
2.5 $A \models \sharp(N_a)$

4.1.2.1 Validación

Paso 1. El mensaje P2.1 nos indica que el servidor recibe un mensaje cifrado y de las suposiciones iniciales sabemos que es una llave que comparte con el cliente, por lo que se obtendrá como conclusión de estas dos premisas que el mensaje fue enviado por el cliente en algún momento dentro de la ejecución actual del protocolo o en una ejecución anterior, como se muestra en el postulado 2.6.

$$2.6 \quad \frac{S \triangleleft \{N_a\}_{K_{as}}, S \models A \overset{K_{as}}{\leftrightarrow} S}{S \models A \mid\sim N_a}$$

Paso 2. El mensaje P2.2 indica que el cliente recibió una pregunta que solo el servidor pudo enviar. Dentro del mismo procesamiento de este mensaje se comprueba el valor reciente que el cliente envío anteriormente (N_a) lo que le hace conocer al cliente A que el mensaje es reciente, como se muestra en los postulados 2.7 y 2.8.

$$2.7 \quad \frac{A \triangleleft \{N_a, Pregunta\}_{K_{as}}, A \models A \overset{K_{as}}{\leftrightarrow} S}{A \models S \mid\sim N_a, Pregunta}$$

$$2.8 \quad \frac{A \models \sharp(N_a), A \triangleleft (N_a, Pregunta)}{A \models \sharp(N_a, Pregunta)}$$

Paso 3. El servidor recibe la respuesta que envió el cliente y se comprueba con el postulado 2.9.

$$2.9 \quad \frac{S \triangleleft \{Respuesta\}_{K_{as}}, S \models A \overset{K_{as}}{\leftrightarrow} S}{S \models A \mid\sim Respuesta}$$

Paso 4. De aquí ya no podemos continuar para comprobar el postulado que se desea alcanzar ya que no existe alguna forma de que el servidor S compruebe que el mensaje que recibió es reciente y no fue recibido mediante un ataque de réplica de algún mensaje que se haya dado en una ejecución anterior. Por lo que no se puede asegurar que el protocolo propuesto P2 sea seguro.

4.1.3 TERCER PROTOCOLO PROPUESTO (P3)

Transformando los mensajes a lógica BAN resulta el siguiente protocolo idealizado para la validación en la lógica.

P3.1 $A \rightarrow S : \{N_a\}_{K_{as}}$

P3.2 $S \rightarrow A : \{N_s, N_a\}_{K_{as}}$

P3.3 $A \rightarrow S : \{N_s, Respuesta\}_{K_{as}}$

Para conocer más detalles en la definición del protocolo tres propuesto remítase a la página 52.

También se definen las suposiciones iniciales.

3.1 $A \models A \overset{K_{as}}{\leftrightarrow} S$ 3.2 $S \models A \overset{K_{as}}{\leftrightarrow} S$

3.3 $A \models \sharp(N_a)$

3.4 $S \models A \Rightarrow Respuesta$

3.5 $S \models \sharp(N_s)$

Aquí no existe el postulado de que el cliente confía en que el servidor es el único que puede realizar las preguntas debido a que al no estar cifradas las preguntas sino se cifran solamente los *nounces*, cuando se realice un ataque pasivo el atacante será capaz de saber cuál es la pregunta que se realice lo que le da la posibilidad de realizar esta misma pregunta en otra ejecución del protocolo. Aunque no afecta a la seguridad ya que el elemento importante dentro de este protocolo propuesto P3 es que la respuesta siga manteniéndose secreta para que no pueda ser utilizada posteriormente por alguien no autorizado [7] [8] y aunque un atacante enviara la misma pregunta también tendría que conocer el valor numérico N_a que el cliente envió en el paso 1 al servidor.

4.1.3.1 Validación

La validación de este protocolo se desarrolla de la siguiente manera.

Paso 1. En el mensaje P3.1 el cliente envía un valor efímero (N_a) cifrado, que el servidor sabe por el postulado inicial 3.2 que comparte una llave con el cliente, de esta manera el servidor sabe que el mensaje fue enviado en algún momento por el cliente lo que se demuestra en el postulado 3.6.

3.6 $$\frac{S \triangleleft \{N_a\}_{K_{as}}, S \models A \overset{K_{as}}{\longleftrightarrow} S}{S \models A \mid\sim N_a}$$

Paso 2. En el mensaje P3.2 del protocolo idealizado el servidor envía un mensaje al cliente, en este mensaje se incluyen los siguientes elementos: el valor enviado por el cliente (N_a), un valor generado por el servidor (N_s). Este mensaje es recibido por el cliente y sabe que fue enviado por el servidor en algún momento dentro de la ejecución actual del protocolo o de una ejecución anterior ya que el mensaje viene cifrado con la llave que comparte con el servidor. Después de descifrar el mensaje, el cliente sabe que el mensaje es reciente porque en el mensaje viene el valor N_a que envió en el mensaje P3.1 como se ve en los postulados 3.6 y 3.7.

3.7 $$\frac{A \triangleleft \{N_a, N_s\}_{K_{as}} A \models A \overset{K_{as}}{\longleftrightarrow} S}{A \models S \mid\sim (N_a, N_s)}$$

3.8 $$\frac{A \triangleleft (N_a, N_s), A \models \sharp(N_a)}{A \models \sharp(N_a, N_s)}$$

Paso 3. El servidor recibe un mensaje envíado por el cliente, este mensaje contiene el valor N_s y la *Respuesta* cifrados con la llave que comparte con el cliente, por lo que el servidor asume que el mensaje es reciente por el valor N_s como se ve en los postulados 3.9 y 3.10.

3.9 $$\frac{S \triangleleft \{N_s, Respuesta\}_{K_{as}}, S \models A \overset{K_{as}}{\longleftrightarrow} S}{S \models A \mid\sim (N_s, Respuesta)}$$

3.10 $$\frac{S \triangleleft (N_s, Respuesta), S \models \sharp(N_s)}{S \models \sharp(N_s, Respuesta)}$$

Paso 4. En este paso, el servidor comprueba que solo el cliente pudo haber dicho la *Respuesta*, ya que solo el cliente tiene autoridad sobre la *Respuesta* por lo que se obtiene el postulado que queremos alcanzar como comprobante de que el servidor confía en que solo el cliente pudo enviar la respuesta correcta como se indica en el postulado 3.8.

3.11 $$\frac{S \models \sharp(N_s, Respuesta), S \models A \Rightarrow Respuesta}{S \models A \models Respuesta}$$

Con este último paso comprobamos que este protocolo es válido ya que se alcanzó el postulado que se había propuesto para aceptar un protocolo de seguridad.

4.1.4 CONCLUSIONES

Como se puede observar con las validaciones que se le hicieron a los protocolos, solo el primer y el tercer protocolo alcanzaron el resultado que se esperaba, el cuál fue la siguiente ecuación.

$$S \models A \models Respuesta$$

La diferencia entre los protocolos uno y tres es que el protocolo uno realiza el cifrado sobre la *Pregunta* enviada en el paso dos, esta operación no es realizada en el paso dos del protocolo tres ya que no se considera una falla en la seguridad que alguien pueda "ver" la *Pregunta* mientras la *Respuesta* siga protegida, algo que se consigue con el cifrado del mensaje en donde se envía la *Respuesta* y se considera más eficiente ya que al necesitar cifrar menos elementos el trabajo del procesador disminuye. [39]

El servidor sabe que el cliente fue el que envió la respuesta; esto se consiguió con las llaves compartidas, valores que solo son conocidos entre el servidor y el cliente, hay que notar que en esta lógica no se consideran aspectos como el algoritmo de cifrado que se utiliza, el tamaño de las llaves que se estén usando o la capacidad que tenga el usuario para mantener seguros los datos confidenciales, ya que la lógica considera que esos aspectos han sido cubiertos para ofrecer la mayor seguridad posible, y lo demuestra validando que el intercambio de la información que se da, asegure al diseñador del protocolo que el intercambio de mensajes para comprobar la autenticación se lleva a cabo de manera segura.

El protocolo que se propone desarrollar es el tercero, el protocolo P3, ya que atendiendo a los principios expuestos al principio de este capítulo, es el protocolo que necesita hacer menos suposiciones, alcanza el objetivo propuesto y envía elementos dentro del mensaje sin cifrar, en el

caso concreto del protocolo dos es la *Pregunta*, y no afecta la seguridad del protocolo que la *Pregunta* no vaya cifrada. El primer protocolo propuesto también alcanza los objetivos, pero necesita más datos cifrados y realiza más suposiciones para poder desarrollarlo, por lo que se vuelve ineficiente para los objetivos de esta tesis.

Esta lógica no se encarga de la validación de problemas que son inherentes a los protocolos, como los interbloqueos que pueden llegar a ocurrir por el uso del protocolo, por lo que se debe de realizar una validación extra para comprobar que además de ser seguro también es correcto, lo que se hará en la siguiente sección.

4.2 VALIDACIÓN DE CORRECTITUD CON PROMELA

PROMELA es un lenguaje que nos permite realizar una validación automatizada de un protocolo [5] [13] [29]; para realizar la validación debemos de especificar qué es lo que hará el protocolo. El protocolo que se analizará es el protocolo propuesto P3, ya que este fue escogido por las ventajas que tiene sobre los demás, varios de los siguientes puntos han sido especificados antes, pero para comodidad del lector se hará un recordatorio de algunos.

- **Especificación del servicio que se proporcionará.**

  El servicio que proporcionará el protocolo es realizar la autenticación de un usuario ante un servidor, el usuario debe demostrar su identidad para poder utilizar los recursos que administra el servidor. Se utilizará cifrado para proporcionar seguridad a la transmisión de mensajes.

  Así mismo, como pasos a realizar fuera de la ejecución del protocolo, se creará un archivo de configuración para el programa cliente, el archivo de configuración estará en formato INI y contendrá valores para saber la dirección del servidor que utilizará, la llave que será utilizada, así como una serie de preguntas-respuestas codificadas mediante un algoritmo *hash* para que tengan el mismo tamaño de caracteres todos los valores, sin importar de qué pregunta o respuesta fue generado.

- **Suposiciones acerca del medio ambiente en que será ejecutado el protocolo.**

  El medio ambiente en el que se desarrollará la ejecución del protocolo es Internet, por lo que detalles correspondientes al control de flujo, direccionamiento de los mensajes, medio físico que será utilizado serán administrados por el protocolo TCP/IP; de lo único que el protocolo desarrollado en esta tesis se encargará es del control de flujo de sus propios mensajes, de establecer, administrar y cerrar una conexión.

  Tampoco son considerados errores en la transmisión. Se consideran pérdidas de los mensajes solo al inicio, cuando se envía el nombre del usuario, debido a la pérdida de los mensajes que pueden ocurrir se hace necesaria la retransmisión de estos mensajes.

- **El vocabulario de los mensajes que se utilizarán.**

Los mensajes comprendidos dentro del vocabulario son los siguientes.

- *Seg*. Indica que la trama que se transmite o recibe es del elemento de seguridad.
- *Id*. Es el nombre de usuario del cliente.
- *Pregunta*. Indica que se transmite o recibe una pregunta.
- *Respuesta*. Indica que se transmite o recibe una respuesta.

No hay mensajes dentro del protocolo para indicar éxito o error dentro de la transmisión de una trama debido a que esto es innecesario e inseguro, ya que podría proporcionar a un atacante pasivo información de cuando hubo una respuesta incorrecta. Lo que ocurrirá en caso de error es que la comunicación se cerrará; habrá opciones para reintentos que serán administrados por el servidor.

- **El formato de los mensajes.**

La trama que será transmitida lleva el siguiente orden, comenzando por la izquierda.

TIPO	*SUBTIPO*	*VAL1*	*VAL2*	*VALn*

El campo TIPO se refiere a qué trama será transmitida. Este protocolo siempre será de seguridad, pero se incluye para propósitos de mejoras en un futuro. El tamaño del campo es de un byte.

El campo SUBTIPO se refiere a los valores que se transmiten dentro de la trama. Los valores posibles son identificación, pregunta y respuesta. El tamaño del campo es de un byte.

Los campos VAL1, VAL2 y VALn llevarán distintos tipos de valores, que dependerán del SUBTIPO de trama que se envíe. Esto será explicado con más detalle cuando se desarrolle la implementación del protocolo.

- **Las reglas de intercambio de los mensajes.**

La regla de intercambio de mensajes se describe mejor con la especificación que realizamos antes.

P3.1 $A \rightarrow S : A, \{N_a\}_{K_{as}}$

P3.2 $S \rightarrow A : \{N_s, N_a\}_{K_{as}}, Pregunta$

P3.3 $A \rightarrow S : \{N_s, Respuesta\}_{K_{as}}$

En el mensaje P3.1 el cliente envía una identificación, este mensaje solo será transmitido al iniciar la ejecución del protocolo. Los mensajes P3.2 y P3.3 son para realizar la pregunta y contestarla respectivamente, los valores únicos tendrán como propósito la identificación de la trama, aunque no serán consecutivos, así también se podra probar que se tienen las llaves apropiadas para este usuario. Los mensajes P3.2 y P3.3 serán los que se realicen durante la ejecución del protocolo, el que inicia la comunicación en este caso es el servidor, que enviará la pregunta en un tiempo aleatorio, mientras que el usuario solo tendrá que contestarlas.

4.2.1 ARCHIVO DE ESPECIFICACIÓN DEL LENGUAJE

Para realizar la validación se realizó un programa mediante un archivo de definiciones usando el lenguaje PROMELA, este archivo es el que será validado en SPIN, el archivo completo se muestra en la parte del Anexo B.

El protocolo fue validado dividiendo la simulación en tres componentes para facilitar la comprensión: la simulación del paso de mensajes por la red, los mensajes enviados por el cliente y los mensajes enviados por el servidor.

4.2.2 SIMULACIÓN DEL PASO DE MENSAJES EN LA RED

La programación de la simulación de la red se realizó para representar el funcionamiento de la transmisión de mensajes y una facilidad en la modificación en caso de que se quiera cambiar la manera en como se intercambian los mensajes.

```
proctype mc()
{
    byte msj, msj2;
endmc: do
        /* En la transmisión por udp puede llegar el mensaje
          o no llegar, no se considera que el mensaje llegue
          con errores
        */
        :: cli_udp?msj,msj2 -> udp_ser!msj,msj2;
        :: cli_udp?msj,msj2 -> skip;
        /* En la transmisión por tcp no se consideran errores en los
          mensajes, ni pérdida de estos.
        */
        :: cli_tcp?msj,msj2 -> tcp_ser!msj,msj2;
        :: ser_tcp?msj,msj2 -> tcp_cli!msj,msj2;
    od;
}
```

Los canales de transmisión que existen son los siguientes:

Usu_udp	Es el canal que va del usuario a la red (usando el protocolo UDP)
Udp_usu	Es el canal que va de la red al servidor (usando el protocolo UDP)
Usu_tcp	Es el canal que va del usuario a la red (usando el protocolo TCP)
Ser_tcp	Es el canal que va del servidor a la red (usando el protocolo TCP)
Tcp_usu	Es el canal que va de la red al usuario (usando el protocolo TCP)
Tcp_ser	Es el canal que va de la red al servidor (usando el protocolo TCP)

El protocolo UDP se usa al iniciar la comunicación, este protocolo se utiliza solamente para que el cliente envíe el nombre de usuario al servidor, que también se toma como una petición de servicio, ya que el protocolo UDP contiene tanto la dirección IP de la máquina del cliente así como del puerto que desea utilizar para establecer la comunicación. La utilización del protocolo UDP es óptimo para cuando se desea enviar un mensaje sin establecer una conexión directa con el servidor, lo que ocurre en este caso, ya que quien establece la conexión es el servidor una vez que se ha recibido una petición de servicio. La desventaja de este protocolo es que no se garantiza la entrega del mensaje, esto es, puede que el mensaje no llegue, por este motivo se deben de implementar métodos para realizar la repetición de mensajes en caso de pérdidas en el lado del servidor.

El protocolo TCP se utiliza durante el resto de la comunicación, ya que el servidor es el que establece una conexión, se debe de asegurar de que todos los mensajes vengan de la misma máquina, esto se logra con una conexión TCP. La ventaja de este protocolo es que la entrega de mensajes sí se garantiza, ya que en caso de pérdida de algún mensaje el propio protocolo se encarga del reenvío del mismo sin que el programa intervenga, también permite una cantidad de datos de envío más grande de la que se da con UDP. Entre las desventajas esta la lentitud del protocolo en comparación con UDP, ya que al garantizar la entrega se necesita más información para la trama del paquete TCP de la que se necesitaría para una trama UDP. En el protocolo TCP puede ocurrir también que los mensajes se alteren en algunos bits durante la transmisión, por lo que se hace necesario un reenvío de los mensajes.

La programación del medio de comunicación solo es para simular la pérdida de mensajes en caso del protocolo UDP, y de la alteración de los mensajes en el protocolo TCP.

4.2.3 VALIDACIÓN DEL CLIENTE

El cliente es un protocolo sencillo, ya que solo se encarga de responder a las preguntas que le hace el servidor, sin enviar confirmación de mensajes recibidos así como tampoco esperarlos, el diagrama de transiciones de estado que simula esto, se muestra en la Figura 4-1 y la descripción en la Tabla 4-1.

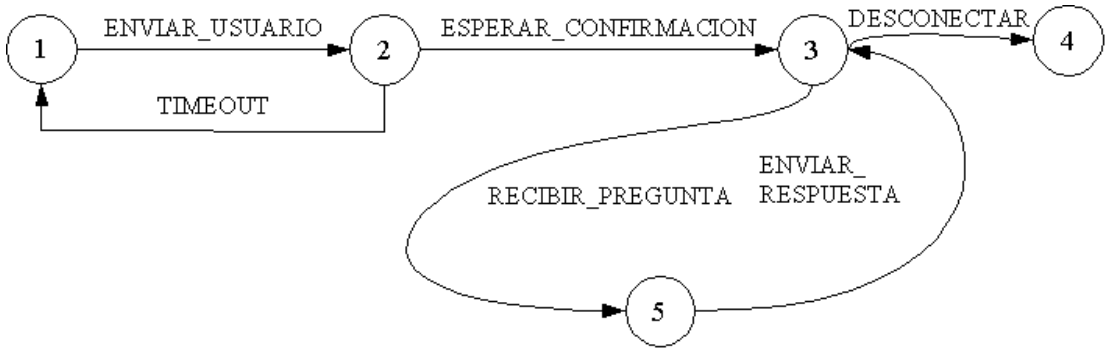

Figura 4-1 Diagrama de transiciones del cliente

ESTADO	DEFINICIÓN
Estado 1	Es el estado inicial, el cliente va a iniciar la comunicación enviando la identificación del usuario.
Estado 2	Es un estado en el que se queda en espera, ya sea para reenviar la identificación del usuario en caso de un *timeout* o cuando se establece una conexión desde el servidor.
Estado 3	Es un estado de espera también, aunque en este estado se la pasa la mayor parte de la ejecución del protocolo, aquí recibimos la pregunta, así como también se puede realizar una desconexión, ya sea de nuestra parte, o que nos desconecte el servidor
Estado 4	Es el estado final, en donde el servidor o el cliente realizó la desconexión, para iniciar otra sesión debe de ejecutarse el protocolo desde el estado uno.
Estado 5	Es un estado de transición interna, en donde se recibe la pregunta, y se envía la respuesta a esa pregunta, siempre se envía alguna respuesta, aunque puede ser una incorrecta en caso de que no se conozca la pregunta.

Tabla 4-1 Estados de transición del cliente

La programación del cliente se dividió en tres secciones que se pueden consultar en el Anexo B, una sección es un módulo que se encarga de recibir todas las tramas que se reciben, para que a partir de ahí se envíe a la sección correspondiente, otra sección es un módulo de seguridad que se encarga de recibir todas las tramas que son de tipo SEGURIDAD, que fue realizado con base en los diagramas de estados del protocolo del cliente, y la última sección es un módulo ficticio, que se utiliza para los casos en que las tramas recibidas no pertenezcan al tipo de SEGURIDAD, esto se realiza para ampliar el protocolo a otros mensajes, pero sin importarnos que se realice dentro de esos mensajes.

4.2.4 VALIDACIÓN DEL SERVIDOR

El servidor es el elemento más complejo de todo el protocolo, esto se debe a que el servidor se encarga de autenticar al cliente, de saber si un mensaje se ha alterado, de reenviar mensajes en caso de que ocurriera la alteración de los mensajes, del contador de los mensajes recibidos, así como realizar preguntas en tiempos aleatorios. El diagrama de estados del protocolo del servidor se muestra en la Figura 4-2 Diagrama de transiciones del servidor y su descripción en la Tabla 4-2.

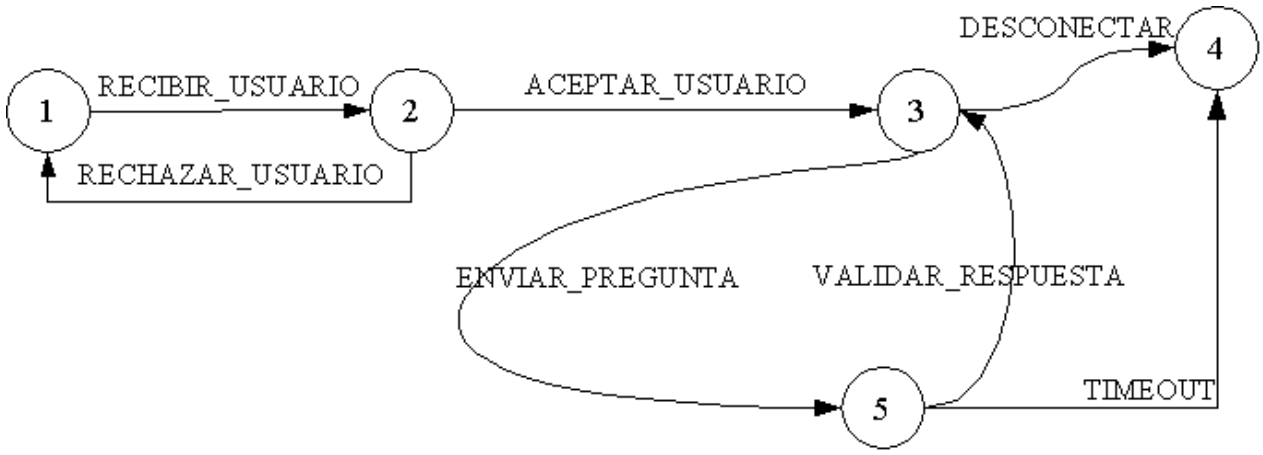

Figura 4-2 Diagrama de transiciones del servidor

La definición de los estados es la siguiente.

ESTADO	SIGNIFICADO
Estado 1	Es el estado inicial, el servidor siempre estará en este estado cuando espere la petición de conexión de un cliente.
Estado 2	Es un estado de transición, en el que ya se recibió un usuario, y se realiza la verificación de éste, en caso de aceptación pasa al estado tres, en caso de rechazo al estado uno.
Estado 3	Una vez que se ha aceptado un usuario, se estable la conexión con el usuario y se le envía la pregunta.
Estado 4	Es el estado final, al que se llega por una desconexión ocurrida en el proceso del usuario, mediante un timeout o por el proceso del servidor.
Estado 5	Es el estado al que se llega cuando se envía la pregunta, y en el que está cuando se espera la respuesta.

Tabla 4-2 Estados de transición del Servidor

La figura 4-2 es un diagrama de estados que muestra un modelo general del funcionamiento del servidor, ya que no se indica que se debe de hacer en caso de que la respuesta recibida sea incorrecta, o cuando por algún motivo el cliente no quiera responder la pregunta.

Por eso se definieron las siguientes subsecciones del servidor, para mostrar que otros procesos deben de correr dentro del proceso servidor.

4.2.4.1 Sección de Preguntas

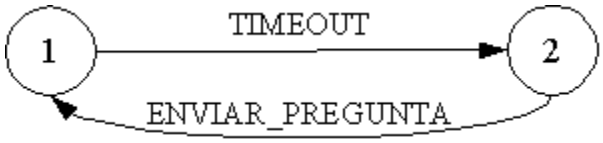

Figura 4-3 Sección de preguntas en el servidor

ESTADO	SIGNIFICADO
Estado 1	Se puede considerar un estado inicial, en este estado se espera que ocurra un *timeout*, que se utiliza para simular el tiempo aleatorio que debe de esperarse antes de enviar la pregunta
Estado 2	Una vez que ocurre el *timeout* enviamos una pregunta, esta pregunta es tomada de un archivo de configuración especial para este usuario.

Tabla 4-3 Estados para la sección de preguntas en el servidor

Esta sección se encuentra dentro del estado tres del diagrama general del servidor en la Figura 4-2, porque ahí es donde se envía una pregunta al cliente.

El *timeout* es para simular un tiempo aleatorio en segundos, lo que quiere decir que cada *n* segundos es enviada una pregunta, donde *n* es un número aleatorio que se calcula durante la ejecución del protocolo.

4.2.4.2 Sección de respuestas

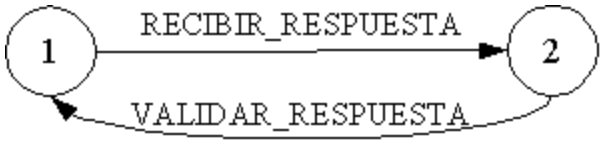

Figura 4-4 Sección de respuestas en el servidor

Esta sección se encarga de revisar las respuestas que se reciben del cliente, y tomar la acción necesaria ya sea si la respuesta es válida o inválida.

ESTADO	DEFINICIÓN
Estado uno	Se puede considerar un estado inicial, se encarga de recibir las respuestas del cliente.
Estado dos	Es el estado que se encarga de validar las respuestas que recibe.

Tabla 4-4 Estados para la sección de respuestas en el servidor

En la Figura 4-4 se considera que las respuestas recibidas son válidas siempre, aunque como también está el caso en que la respuesta que se reciba del lado del servidor llegue alterada o sea errónea, cuando ocurra dependerá del servidor saber que acción tomar. Para la implementación práctica en esta tesis la acción es que se permita un número de reintentos, en la sección de reenvío, cuando no se reciba la pregunta correcta y si el número de reintentos es superado que el servidor termine la comunicación con el cliente.

4.2.4.3 Sección de reenvío

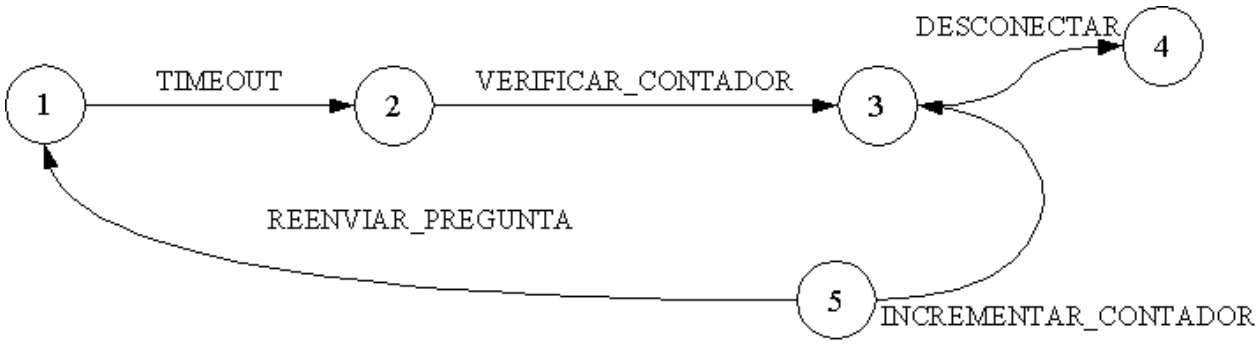

Figura 4-5 Sección de reenvío de preguntas en el servidor

El diagrama mostrado en la Figura 4-5 es para revisar cada una de las preguntas que se han enviado y no han sido contestadas satisfactoriamente, se verifica un contador de cuántas veces se han hecho las preguntas, y en caso de que hayan excedido el contador se desconecta al usuario, si no lo excedió se reenvía la pregunta.

Los estados se definen de la siguiente manera.

ESTADO	DEFINICIÓN
Estado 1	Es un estado inicial, es donde se espera el *timeout*, éste es un valor fijo.
Estado 2	Se revisa el contador para cada una de las preguntas hechas.
Estado 3	Una vez que se revisó el contador de cada una de las preguntas, se comprueba si el contador es mayor de los reintentos válidos y si es mayor se desconecta el servidor, sino se incrementa el contador
Estado 4	Es un estado de terminación, cuando no se pudo contestar una pregunta de manera satisfactoria, en *n* números de intentos, se realiza la desconexión.
Estado 5	Una vez que se ha realizado el incremento de los intentos, se reenvía la pregunta al cliente.

Tabla 4-5 Estados para la sección de reenvios en el servidor

Este diagrama tiene que ser representado por un proceso que se ejecute en un tiempo determinado para comprobar periódicamente que preguntas no han sido contestadas y reenvíar la pregunta o

cortar la comunicación en caso de que haya excedido un tiempo determinado o haya ocurrido un error.

4.2.4.4 Resultado de la validación con SPIN

Para realizar la validación del protocolo escogido P3 se creó un archivo del lenguaje del protocolo en una especificación formal utilizando PROMELA para este fin, realizándose la validación con SPIN.

Para comenzar la demostración del protocolo, se utilizará la simulación aleatoria utilizando como entrada el archivo que contiene la especificación del protocolo, que se llamará *validacionProtocoloAuntenticacion.spn*, mediante la instrucción *spin validacionProtocoloAutenticacion.spn*.

```
$ spin validacionProtocoloAutenticacion.spn
```

El resultado obtenido con esta instrucción es:

```
    timeout
    timeout
#processes: 5
            queue 1 (cli_udp):
            queue 4 (udp_ser):
            queue 2 (cli_tcp):
            queue 5 (tcp_ser):
            queue 3 (ser_tcp):
            queue 6 (tcp_cli):
 16: proc  4 (servidor_seguridad) line 136 "validacionTesisCorregida.spin" (state 6) <valid end
state>
 16: proc  3 (cliente) line  56 "validacionTesisCorregida.spin" (state 5) <valid end state>
 16: proc  2 (servidor) line 115 "validacionTesisCorregida.spin" (state 7) <valid end state>
 16: proc  1 (mc) line  35 "validacionTesisCorregida.spin" (state 9) <valid end state>
 16 proc  0 (:init:) line 198 "validacionTesisCorregida.spin" (state 6) <valid end state>
6 processes created
```

Como se aprecia en el resultado, no se muestra que haya errores durante la validación ni quedan mensajes en los canales de comunicación, pero esto no es certeza de que el protocolo esté definido de manera correcta, por lo que se procede a realizar una validación en vez de una simulación.

Las opciones que se manejan dentro de la validación del programa SPIN son:

- -a Genera un analizador de protocolos específico a la definición del protocolo que se desea validar. El resultado es un conjunto de archivos en C llamados pan.[cbhmt] que se compilan para generar el analizador, que a su vez se ejecuta para realizar el análisis.

- Para realizar la compilación se requiere del lenguaje C ANSI o en su defecto uno que sea compatible con éste; las opciones de compilación son:
    - $ gcc –o pan pan.c Lo que garantiza una exploración exhaustiva.
    - $ gcc –DBITSTATE -o pan pan.c. Que realiza un uso eficiente de la memoria.
    - $ gcc –DSAFETY -o pan pan.c Que realiza una búsqueda de estados finales inválidos, esto es, que el protocolo no termina en donde debe de terminar.

- Para realizar la validación, una vez obtenido el programa *pan*, se realizan las siguientes acciones.
    - $ pan –cN, se detiene cuando ocurre el error número N. De manera predeterminada es uno.
    - $ pan –l encuentra ciclos no progresivos (non-progress cycles) (compilado con –DNP)
    - $ pan –a encuentra ciclos de aceptación (acceptance cycles) (No compilado con –DNP)
    - $ pan –mN profundidad N (predeterminado son 10k)
    - $ pan –wN tabla hash de 2^N entrada (predeterminado son 18)
    - $ pan –m Permite cambiar la semántica por omisión de las acciones de envío. Es decir, una operación de envío es posible si el canal no está lleno, mientras que con esta opción, siempre son ejecutables. Los mensajes enviados a un canal lleno son desechados.
    - Combinando la opción anterior (–m) con –a permite analizar el caso de pérdida de mensajes.
    - $ pan –t permite llevar un seguimiento del análisis y almacenar los problemas (*deadlocks*, errores, etc.) en el archivo pan.trail.
    - spin con la opción –t permite analizar este archivo. Se puede combinar con las opciones pglrs para el seguimiento.

Para la creación del programa de validación para este protocolo específico usando SPIN, se realiza de la siguiente manera:

```
$ spin –a validacionProtocoloAutenticacion.spn
```

Con este comando se generan los archivos pan.[bchmt] que serán utilizados para la creación del programa de validación.

La creación del archivo pan.[bchmt] se refiere a que se crean los siguientes archivos.

- pan.b
- pan.c
- pan.h

- pan.m
- pan.t

El siguiente comando para crear el programa de validación es:

```
$ gcc -o -DSAFETY pan pan.c
```

Con este comando se crea un archivo ejecutable, que se ejecuta sin parámetros, para ver si encuentra algún error dentro del protocolo.

```
$ pan
```

El resultado de la ejecución del programa *pan* es el siguiente:

```
Spin Version 4.2.3 -- 5 February 2005)
        + Partial Order Reduction
        + Compression

Full statespace search for:
        never claim         - (not selected)
        assertion violations       +
        cycle checks        - (disabled by -DSAFETY)
        invalid end states +

State-vector 84 byte, depth reached 80, errors: 0
    110 states, stored
     10 states, matched
    120 transitions (= stored+matched)
      0 atomic steps
hash conflicts: 0 (resolved)

unreached in proctype mc
        line 48, "pan_in", state 12, "-end-"
        (1 of 12 states)
unreached in proctype cliente
        line 59, "pan_in", state 6, "-end-"
        (1 of 6 states)
unreached in proctype cliente_seguridad
```

```
        (0 of 22 states)
unreached in proctype servidor
        line 118, "pan_in", state 8, "-end-"
        (1 of 8 states)
unreached in proctype servidor_seguridad
        (0 of 36 states)
unreached in proctype :init:
        (0 of 6 states)
```

A continuación analizaremos el contenido del archivo (salida) del programa pan para ver qué resultado fue el que se obtuvo con la validación.

```
Full statespace search for:
        never claim         - (not selected)
        assertion violations       +
        cycle checks        - (disabled by -DSAFETY)
        invalid end states +
```

Esta sección indica que se realizó una búsqueda total en el espacio de los estados que se pueden alcanzar.

La verificación de la validez del protocolo se realizó buscando estados finales inválidos, así como violaciones a las aserciones.

```
State-vector 84 byte, depth reached 80, errors: 0
    110 states, stored
     10 states, matched
    120 transitions (= stored+matched)
      0 atomic steps
hash conflicts: 0 (resolved)
```

Las líneas anteriores indican que para realización la validación del protocolo se utilizó un vector de estados de 84 bytes, con 120 estados utilizados para validar todos las posibles variantes que podían ocurrir durante la ejecución del protocolo y en ninguno de estas variantes se encontró un error que pudiera afectar la ejecución del protocolo.

```
unreached in proctype mc
```

```
        line 48, "pan_in", state 12, "-end-"
        (1 of 12 states)
unreached in proctype cliente
        line 59, "pan_in", state 6, "-end-"
        (1 of 6 states)
unreached in proctype cliente_seguridad
        (0 of 22 states)
unreached in proctype servidor
        line 118, "pan_in", state 8, "-end-"
        (1 of 8 states)
unreached in proctype servidor_seguridad
        (0 of 36 states)
unreached in proctype :init:
        (0 of 6 states)
```

Esta última parte indica los estados que no se alcanzaron dentro de los procesos creados por la validación del protocolo, es normal que no se alcancen algunos estados, como ocurre en este caso, en donde no se alcanzan los estados finales en el proceso *mc* (el medio de comunicación), en el proceso *cliente* ni en el proceso *servidor* pero este es un error aceptable debido a que son procesos que siempre estarían corriendo en una ejecución normal.

La validación del protocolo sugerido, tanto en la seguridad que proporciona como en la ejecución del protocolo demostró que el protocolo que conviene implementar de manera práctica es el *protocolo Tres*, cuya implementación se hará en la siguiente sección.

4.3 REALIZACIÓN PRÁCTICA DEL PROTOCOLO

Para la realización del protocolo nos basamos en la validación de SPIN, por lo que se separa el protocolo en dos partes, el cliente y el servidor; para realizar el desarrollo se deben definir ciertos elementos en común a estos dos procesos que es el formato de la trama.

4.3.1 FORMATO DE LAS TRAMAS

El formato de la trama debe ser adaptable, para poder añadirle nuevos mensajes que se puedan requerir en un futuro, así como de fácil implementación en la mayoría de los lenguajes.

El formato general de la trama es:

ID	SID	TAM	DATOS
1 byte	1 byte	1 byte	n bytes

Cuyos campos se definen como:

- **ID.** Es de un byte, se utiliza para indicar el tipo de mensaje que se envía o recibe por el cliente o servidor respectivamente y será el número hexadecimal 3C que indicará que se trata de un mensaje de seguridad, por lo que cualquier mensaje intercambiado que contenga este número será tratado por el protocolo desarrollado en esta tesis.
- **SID.** Es de un byte también, es un identificador secundario, que será dependiente del identificador que es enviado en el primer byte, se usarán tres identificadores secundarios para este protocolo, que serán los números hexadecimales C3, F0, 0F y serán utilizados para identificar un mensaje de identificación, pregunta o respuesta respectivamente.
- **TAM.** Indica el tamaño de la trama, contando todos los bytes que están dentro de la trama, esto es, un byte del identificador + un byte del identificador secundario + un byte del tamaño + n bytes de los datos.
- **DATOS.** Estos son los datos que lleva el mensaje, dependiendo del tipo de mensaje que se transmita.

A continuación, se define el formato del mensaje específico a cada paso de la transmisión, que se distinguen entre sí por el identificador secundario que se transmita en la trama.

El primer mensaje que se manejará es el de identificación del usuario, que permitirá a un cliente identificarse ante el servidor por medio de un nombre único para cada usuario, esta identificación le es útil al servidor porque le indica qué llave de cifrado es la que se debe de utilizar para comunicarse con el cliente.

El formato de este mensaje es:

ID	SID	TAM	Valor efímero	Relleno	USUARIO
1 byte	1 byte	1 byte	2 bytes	14 bytes	N bytes

Los campos a utilizar se definen como:

- **SID.** Es de un byte también, es un identificador secundario, que será dependiente del identificador que es enviado en el primer byte, esto quiere decir que son mensajes específicos al parámetro que contenga el primer campo. El identificador de este mensaje para indicar que se trata de una identificación del usuario es el número hexadecimal C3.
- **Valor efímero.** Es un valor que genera el cliente para poder identificar en mensajes posteriores que los mensajes recibidos son recientes al contener este valor.
- **Relleno.** Es un campo necesario para realizar el cifrado del valor efímero, ya que el algoritmo de cifrado trabaja en bloques de 16 bytes para esta implementación.
- **USUARIO.** Es un nombre de usuario que indica quién es el cliente que quiere solicitar un servicio.

Este tipo de mensaje solo se utiliza cuando el usuario intenta establecer la conexión con el servidor, siendo innecesario enviarlo cada vez que se envía una pregunta porque, excepto en la

conexión, el cliente nunca toma la iniciativa del intercambio de información. El servidor es el encargado de establecer la comunicación que siempre será para establecer una pregunta.

Otro de los mensajes utilizados es la realización de la pregunta que siempre será enviada por el servidor al cliente en cualquier momento durante la ejecución del protocolo.

El formato de este mensaje es:

ID	SID	TAM	Valor efímero 1	Valor efímero 2	Relleno	DATOS
1 byte	1 byte	1 byte	2 bytes	2 bytes	12 bytes	32 bytes

Los tres campos iniciales ya se explicaron anteriormente, por lo que solo se explicarán los restantes.

- **Valor efímero 1.** Es un número único que será utilizado por el servidor que le ayuda a identificar al servidor de que el mensaje es reciente, y también lo utiliza para saber a qué pregunta pertenece una respuesta. Se utilizan 2 bytes para representar números largos, de esta manera los números tienen rangos más amplios para ser escogidos. Este campo también es utilizado para identificar a una trama, actúa como el número de la trama, pero no se asegura que sean números consecutivos, lo único que asegura es que cada trama es única dentro de la ejecución del protocolo.
- **Valor efímero 2.** Es el campo en donde ira el valor efímero que envío el cliente cuando realizo la identificación. En ejecuciones posteriores tendrá el valor de cero.
- **Relleno.** Es un campo necesario para realizar el cifrado del valor efímero, ya que el algoritmo de cifrado trabaja en bloques de 16 bytes para esta implementación.
- **DATOS.** Es la pregunta que se realizará, es de 32 bytes porque siempre serán éstos los que se envían dentro de la trama, esto es, porque lo que se envía es un valor *hash* de la pregunta real, utilizando la función *MD5*[14].

El servidor puede reenviar dos veces más una pregunta, en caso de que una respuesta haya sido incorrecta o no se tenga una respuesta en un periodo de tiempo. El que lleva el control del reenvío de una pregunta es el servidor, por lo que éste decide si se debe cortar una comunicación cuando no se obtenga una respuesta satisfactoria.

El último mensaje a revisar es de respuesta, este mensaje solo puede ser enviado por un cliente y tiene que corresponder a una de las preguntas que se reciban, así como también ser correcta para que el servidor permita que la comunicación continúe. El cliente no tiene la capacidad de reenviar una respuesta si ésta no fue recibida por el servidor, esto se debe a que como no existe un

[14] Esta es un función que siempre devuelve 16 bytes sin importar el número de bytes de los datos a los que se les aplique la función, aunque para efecto de mostrar el resultado, se convierten los 16 bytes a su representación hexadecimal por lo que se transforman en 32 bytes.

mensaje de comprobación de los mensajes enviados, no sabe si una pregunta llegó de manera correcta a su destino. En el protocolo propuesto no se requiere recibir reconocimientos en la parte del cliente, lo que hace eficiente al protocolo ya que se reduce el número de mensajes a intercambiar sin afectar la ejecución del protocolo.

El formato del mensaje es.

ID	SID	TAM	Valor efímero	RESPUESTA	Relleno
1 byte	1 byte	1 byte	2 bytes	32 bytes	14 bytes

- **Valor efímero.** Este es un número único que tiene relación con el número que recibimos del servidor, y solo puede ser modificado por el cliente con alguna función que reciba como entrada. El número único enviado por el servidor, es un valor que va cifrado por una llave compartida con el servidor, también es utilizado para identificar a una trama en especifico, actúa como el número de la trama, pero no se asegura que sean consecutivo, lo único que se asegura es que cada trama es única dentro de la ejecución del protocolo.
- **Respuesta.** Es la respuesta que enviamos al servidor, contestando la pregunta que realizó éste al cliente, es de 32 bytes debido a que la respuesta se codificará con la función MD5 para que los campos tengan una cantidad de caracteres constante ya que siempre se devuelve 16 bytes, es un valor que va cifrado con un llave compartida con el servidor.
- **Relleno.** Este campo se utiliza como relleno, son valores 0 (cero) antes de realizar el cifrado, y la función de este campo es que al realizar el cifrado, se haga en bloques múltiplos de 16 caracteres, que es el estándar para el algoritmo de cifrado que se implementó en este protocolo, así la suma de los bytes del número único, la respuesta y el relleno son 48 bytes que es un múltiplo de 16 bytes.

4.3.2 LIBRERÍA DE CIFRADO

Se utilizó el algoritmo estándar AES (*Advanced Encryption Standar*) que se basa en el algoritmo Rijndael; éste es un algoritmo de cifrado simétrico por bloques que puede cifrar y descifrar información. El cifrado convierte los datos a una forma no legible que se llama texto cifrado; el descifrado toma el texto cifrado y lo convierte de nuevo a su forma original, que se llama texto plano.

El algoritmo es capaz de trabaja con llaves de tamaño de 128, 192 y 256 bits, y tanto el cifrado como el descifrado se realiza en bloques de 128 bits; aunque se diseñó para manejar tamaños de bloque y tamaños de llave adicionales, solo las mencionadas anteriormente están reconocidas dentro del estándar AES; el tamaño de la llave utilizada dentro del protocolo desarrollado en la presente tesis es de 128 bits.

Debido a que no era el interés principal de la tesis desarrollar un programa de cifrado, se utilizó una librería que se obtuvo de Internet, denominada *libmcrypt*[15] que se desarrolla para el ambiente Linux y puede ser utilizada por el lenguaje C para realizar el servidor, esta es una librería que implementa varios algoritmos criptográficos. La librería utilizada en el ambiente Windows es la librería *Eldos AES Implementation for Delphi*[16], también se puede obtener de Internet de manera gratuita.

La elección de estos algoritmos se debe a que implementan el algoritmo AES, son compatibles entre sí, y la licencia bajo la que están sujetos permite su utilización de manera libre, por lo que se puede utilizar dentro del proyecto de la tesis sin necesidad de pagar por el derecho de usar estas librerías.

4.3.3 LIBRERÍA DE *HASH*

Se utilizó el algoritmo MD5 para realizar la codificación de las preguntas que se hacen al cliente, también se aplico sobre las respuestas que éste tendrá que contestar y de esta manera organizar las preguntas mediante el *hash* obtenido.

El algoritmo toma un mensaje de entrada de tamaño variable y produce una "huella" o "firma" de 128-bits de la entrada. Se conjetura que es computacionalmente imposible producir dos mensajes que contengan la misma firma, o que, dada alguna firma de un mensaje, se pueda producir el mensaje de entrada con que fue generada esta firma. El algoritmo MD5 se pretende que sea utilizado para aplicaciones de firma digital, donde un archivo debe ser "comprimido" de alguna manera segura antes de ser cifrada con una llave privada bajo un criptosistema de llave pública.

El algoritmo MD5 es diseñado para ser rápido en máquinas de 32-bits. Además, el algoritmo no requiere una tabla de sustituciones grande, por lo que el algoritmo puede ser implementado eficientemente sin ocupar mucho espacio en memoria.

Se utilizó una librería conseguida en Internet. Para este caso, sólo es necesaria la librería para el ambiente Linux, ya que es parte de las herramientas del servidor convertir la información que se va a utilizar para realizar la autenticación, por lo que en el servidor se tomarán las preguntas a realizar y las respuestas esperadas para producir las huellas de estas preguntas y respuestas, lo que proporciona dos ventajas a la comunicación, que son:

- Las preguntas y respuestas, no necesitan ir en texto claro, por lo que será una protección adicional al cifrado que tienen los mensajes, al enviar la huella de la pregunta en lugar de la pregunta en sí.
- Como el algoritmo toma un mensaje de entrada variable y produce siempre 128-bits a la salida del algoritmo MD5, los mensajes siempre van a ser del mismo tamaño, sin importar cuán larga sea la pregunta que produjo la huella.

---

[15] Al momento de realizar esta tesis, la dirección es http://mcrypt.sourceforge.net/

[16] Al momento de realizar esta tesis, la dirección es http://www.eldos.org/elaes/elaes.html

De igual manera que con la librería criptográfica, la librería *hash* para obtener el MD5[17] se puede usar sin restricciones.

4.3.4 PROGRAMACIÓN DEL SERVIDOR

Para realizar la programación del servidor, se utilizó el lenguaje C como lenguaje de desarrollo, se diseñó un programa principal, que es la parte que actúa como servidor, así como varios programas más; estos programas fueron diseñados para facilitar la tarea al usuario de realizar las conversiones de los archivos.

Por ejemplo, el formato del archivo de configuración tanto para el usuario como para el servidor, usa el formato de los archivos INI, como se muestra en el siguiente ejemplo.

*** Inicio de archivo de configuración ***
[Usuario]

[Retos]

*** Fin de archivo de configuración ***

En la sección de Usuario van los datos relevantes al cliente con el que se establecerá la comunicación, entre estos datos se encuentra el nombre del usuario que utilizará, la llave compartida que se utilizará para el cifrado y descifrado de los mensajes; también incluye los datos del servidor con el que se comunicará el cliente, tal como la dirección IP y el puerto que utilizarán.

En la sección de retos se incluirán las preguntas y respuestas que deberán ser utilizadas dentro de la comunicación, esta es la sección en la que se aplicará el algoritmo MD5 para obtener la huella de la pregunta así como de la respuesta correspondiente, un ejemplo de cómo se transforman la pregunta y la respuesta en los valores que serán utilizados tanto por el servidor como por el cliente sería.

```
Primer día de la semana=Lunes
```

Este es el valor que estará escrito en la sección de retos; el valor devuelto será:

eb9cbe3ac445c4c3869912cc7f9403f4 para el valor “Primer día de la semana”
4b1ddd643fea47e16092ebf093f63004 para el valor “Lunes”

Lo que da como resultado en el archivo de configuración a utilizar algo como esto, que esta separado como Pregunta (32 bytes)=Respuesta (32 bytes).

---

[17] Al momento de realizar esta tesis, la dirección para obtenerla es: http://256.com/sources/md5/

```
[Retos]
eb9cbe3ac445c4c3869912cc7f9403f4=4b1ddd643fea47e16092ebf093f63004
.........................
```

Otra de las herramientas dentro del servidor es el de poder cifrar un archivo; esta herramienta es útil para cifrar el archivo de configuración que será enviado al cliente para ser utilizado por su programa, el cifrado del archivo de configuración es necesario ya que cada archivo de configuración es personal, para evitar de esta manera que algún usuario válido dentro del sistema pueda usar el archivo de configuración de otro usuario válido, también se debe de cifrar para evitar que si algún usuario malintencionado obtiene el archivo no pueda utilizarlo aunque también tenga el programa, ya que no posee la clave para descifrar el archivo.

El cifrado se basa en el algoritmo AES (*Rijndael*), por lo que el programa cliente tendrá implementado un módulo que utilizará para descifrar el archivo que se le haya enviado; hay que notar que la clave de cifrado del archivo, no es necesariamente igual a la clave de cifrado que se utilizará para la comunicación, esto quiere decir que las llaves son independientes entre sí.

El objetivo de hacer las llaves independientes es para que el usuario pueda escoger una llave de cifrado para el archivo de configuración que sea fácil de recordar para él, y permitirle al administrador del servidor que escoja la llave que se usará para el intercambio de información, en donde el administrador seleccionará una que la hará difícil de adivinar.

Una vez que se han cubierto estos programas y librerias necesarias para permitir que el archivo de preguntas y respuestas genere un archivo de configuración que pueda ser utilizado por el servidor, se procede a detallar la programación del servidor.

El servidor utilizará dos puertos de manera interna, un puerto utilizará el protocolo UDP, este puerto UDP será definido por el administrador del servidor, la función de éste es esperar peticiones de servicio, este puerto es el que recibirá el mensaje de identificación, una vez recibido este mensaje el servidor verifica que el usuario exista dentro del sistema, pero no se encarga de autenticarlo; el otro puerto utiliza el protocolo TCP, este puerto será único para cada uno de los clientes que se conecten, es en donde se llevará a cabo toda la comunicación entre el servidor y el cliente; este puerto será escogido por el cliente, y le será enviado al servidor dentro de la trama UDP en donde va la identificación del usuario que se realiza la conexión.

El servidor se realizó en módulos que ejecutarán una función específica, cada uno de estos módulos se ejecutará dentro de un hilo (*thread*), por lo que serán concurrentes entre sí, utilizando variables globales para administrarse; los módulos de los que consta el servidor son de PREGUNTA, RESPUESTA, REENVÍO, que son utilizados dentro de la autenticación del protocolo, junto con un administrador que dirige la información que recibimos al módulo que corresponda.

4.3.4.1 Módulo de Preguntas

Este módulo se encarga de realizar las preguntas en un tiempo aleatorio, el tiempo aleatorio está programado para ser entre 1 y 900 segundos (15 minutos), siendo el tiempo mínimo entre una pregunta y la siguiente de un segundo, este segundo de intervalo que se maneja es necesario debido a que una función dependiente del tiempo es utilizado como identificador de la trama, por lo que si se enviaran dos preguntas en el mismo segundo el servidor se confundirá ya que estarán marcados con el mismo identificador de la trama, y cuando reciba la respuesta, no sabrá que pregunta estará siendo contestada.

Este módulo se ejecutará dentro de un hilo en un ciclo infinito, mantendrá recursos compartidos con los otros módulos dentro del servidor; como parámetros recibirá el puerto que se está utilizando para la comunicación y el nombre del usuario que estableció la comunicación, el nombre solo es útil para saber qué preguntas se realizarán dependiendo del usuario.

Todas la preguntas que se realicen dentro de este módulo se incluirán en una lista que será compartida con los otros módulos, esta lista contendrá los detalles de cada una de las preguntas realizadas aunque puede haber preguntas repetidas dentro de esta lista pero cada pregunta será única por el identificador de la trama con que fue enviado. Esta lista será útil para el módulo de “Respuestas” y el módulo de “Reenvíos”.

4.3.4.2 Módulo de Respuestas

Este módulo se encarga de verificar las respuestas que recibe el servidor; este módulo se ejecuta dentro de un hilo y también usa la lista compartida de las preguntas que se han realizado, pero a diferencia del módulo de preguntas, no se ejecuta en un ciclo infinito sino que es llamado cuando se recibe una respuesta; aquí el servidor se encarga de descifrar la respuesta que se recibe, una vez que se ha descifrado se revisa contra la lista de las preguntas realizadas que se tiene y que el módulo de preguntas va actualizando con las preguntas que haga, si se encuentra una pregunta que corresponda a esta respuesta utilizando para este fin el número de trama que recibimos con la respuesta, se elimina la respuesta de la lista; en caso de que la respuesta no corresponda a una pregunta, o que el número de trama no se encuentre, no se realiza ninguna acción sobre la lista.

No se utilizan reconocimientos a las respuestas, indicando si esta bien o mal, por lo que en caso de que una respuesta se considere incorrecta no se le informa al programa cliente que hubo un error con la respuesta o que ésta no se pudo utilizar, no se modifica nada en las preguntas enviadas para que otro modulo en el servidor sea quien tome la decisión de reenviar la pregunta o cortar la comunicación como se sugiere en el artículo de Li Gong [11].

4.3.4.3 Módulo de reenvíos

Este módulo es un auxiliar dentro del servidor, se ejecuta en un ciclo cada cinco segundos, utiliza la lista compartida de las preguntas realizadas.

Se encarga de verificar qué preguntas no han sido contestadas en un límite de tiempo establecido, en caso de que una pregunta no haya sido contestada, revisa el contador de cuántas veces se ha enviado éste dentro del límite de reenvío de preguntas, si está dentro del límite se reenvía la pregunta, en caso de que el límite se haya excedido se considera que el cliente no tiene las credenciales necesarias para continuar dentro de los servicios proporcionados, por lo que desconecta al cliente.

El reenvío de las preguntas es necesario, ya que al no manejarse reconocimientos de los mensajes que se han intercambiado, no se sabe si una pregunta llegó bien al cliente, o si una respuesta fue recibida de manera correcta por el servidor, por lo que si en este módulo se encuentra una pregunta que no ha sido contestada, se asume que hubo un error ya sea cuando el cliente recibió la pregunta y por este motivo no pudo contestarla, o cuando el servidor recibió una respuesta que se pudo corromper durante la comunicación, y se reenvía la pregunta de nuevo, esperando que ahora la comunicación se realice de manera adecuada y sin errores.

4.3.5 PROGRAMACIÓN DEL CLIENTE

El cliente es un sistema muy sencillo comparado con la programación del servidor, debido a que el cliente no tiene que revisar si el mensaje es duplicado, si llega de manera incorrecta no se necesita enviar un reconocimiento negativo de que no se recibió bien el mensaje.

Se ejecuta en Windows, fue programado utilizando el lenguaje Delphi en su versión 6.0, la elección de este lenguaje se debe a su facilidad de uso, así como la existencia de una cantidad de paquetes considerable, que se encargan del cifrado y descifrado en varios algoritmos incluyendo AES que se utiliza dentro del protocolo.

El módulo del protocolo de seguridad que es implementado dentro del cliente es sencillo en su funcionamiento, pero tiene el control tanto de la autenticación del usuario dentro del programa cliente, como de su autenticación ante el servidor.

El programa cliente utiliza el archivo de configuración que se genera en el servidor, que contiene el nombre del usuario, la clave que se utilizará para cifrar la comunicación que tendrá el cliente con el servidor, la dirección IP en la que da servicio el servidor, el puerto que utiliza el servidor, así como una sección de respuestas válidas a las preguntas que haga el servidor.

El archivo de configuración que use el programa cliente estará cifrado, y será descifrado siempre que se utilice el programa cliente, por lo que será una manera de autenticar al cliente ante el programa ya que si no conoce la palabra clave para descifrar el archivo el programa cliente no podra utilizarlo. Como cada archivo de configuración es único, un usuario no podrá usar el archivo de configuración de otro usuario, a menos que tenga la clave para descifrar ese archivo de configuración, esto es verificado por el programa cliente.

Los elementos necesarios para autenticarse ante el programa cliente son, el nombre del usuario, que será el mismo que se utilizará para realizar la autenticación ante el servidor, así como una

clave que se utilizará para descifrar el archivo, esta clave puede no ser la misma que se utilizará para cifrar la comunicación con el servidor.

Una vez que se estableció una autenticación correcta ante el programa cliente, el usuario ya no necesita autenticarse más, por lo que permanece invisible al usuario la comunicación del protocolo de seguridad entre el servidor y el cliente.

En el cliente, el protocolo de seguridad no necesita enviar reconocimientos negativos ni positivos, así como tampoco necesita guardar una lista de las preguntas que han sido contestadas, como ocurre con el servidor; cada pregunta que recibe es contestada en el momento en que fue recibida.

Como el tamaño de la trama es muy pequeño, en las pruebas es de 37 bytes para las preguntas, esto sin considerar el tamaño en el número de bytes que se agregan por la utilización del protocolo TCP, se considera que la trama no llegará fragmentada, por lo que cada trama se toma como una pregunta, por este motivo, no se realizan comparaciones con el número de la trama para verificar si llegaron en el orden adecuado las preguntas, tampoco los números de las tramas son consecutivos, solo se asegura que las tramas tendrán número distinto, sin embargo, debido a que el cliente no guarda un listado de las preguntas recibidas y que han sido contestadas, si le llega la misma pregunta no tendrá manera de saber si una pregunta específica, con un número de trama recibido anteriormente, ya ha sido contestada.

El cliente es el que inicia la comunicación con el servidor usando el protocolo UDP, al iniciar la comunicación se envía la identificación del usuario, la misma que utilizó para identificarse al iniciar el programa cliente. La dirección y el puerto que se utilizará para iniciar la comunicación con el servidor vienen definidos en el archivo de configuración: El cliente envía una dirección IP, al igual que un puerto que indicará al servidor donde será establecida la comunicación con el cliente; una vez que se ha enviado la identificación al servidor, se inicia un servicio TCP en el cliente que corresponde a la dirección y al puerto que envió al servidor, este servicio es utilizado por el servidor para establecer la comunicación con el cliente.

El cliente contará con una sección de cifrado y descifrado de los mensajes ya que se intercambiará mensajes cifrados entre el cliente y el servidor.

4.3.6 INSTALACIÓN DE LOS PROGRAMAS.

La instalación del servidor requiere de ciertos conocimientos en Linux así como de permisos para poder abrir y terminar conexiones de red, de preferencia debe ser ejecutado por el usuario *root* o un usuario con permisos especiales. Por otra parte, la instalación por parte del cliente no requiere de permisos especiales.

4.3.6.1 Instalación del servidor

El servidor necesita la instalación de una librería antes de poder ser ejecutado; la librería a instalar es libmcrypt que puede ser encontrado como software libre en Internet.

Para instalar esta librería se realizan los siguientes pasos:
Primero se crea un directorio en linux que sea exclusivo para los archivos desempacados, el comando es:
mkdir libmcrypt

Luego se copia el archivo a este directorio y se procede a desempacarlo.
tar –zxvf libmcrypt-<versión>.tar.gz

Después se realiza la compilación de la librería así como su instalación, con los siguientes comandos.
./configure
make
make install

Comúnmente no se requieren opciones adicionales para estos comandos por lo que no se detallarán aquí.

Una vez instalada la librería se compila el servidor.

Se desempaca el archivo en el que esta el servidor, como se muestra en el siguiente ejemplo.
tar –jxvf miTesis.tar.bz2

Una vez realizado esto, se ejecuta la sentencia make, para que se compilen los archivos necesarios para crear el servidor así como las herramientas necesarias para compilarlo.

Los ejecutables creados son.

ServTesis	Ejecuta el programa que actúa como servidor
Cifrado	Se utiliza para cifrar un archivo
Convertir	Se utiliza para convertir un archivo de configuración que escribe el usuario a un formato que los programas, cliente y servidor, puedan “comprender”.

4.3.6.2 Instalación del cliente.

El cliente no necesita realizar instalación especial, solo es necesario descomprimir el archivo que incluye el cliente y el archivo de configuración en la misma carpeta, y ejecutar el programa cliente.

4.3.7 EJEMPLO DE EJECUCIÓN DEL SERVIDOR

Se presentan a continuación ejemplos de una corrida del programa servidor; el sistema operativo que se utiliza es el Linux Mandrake 9.1.

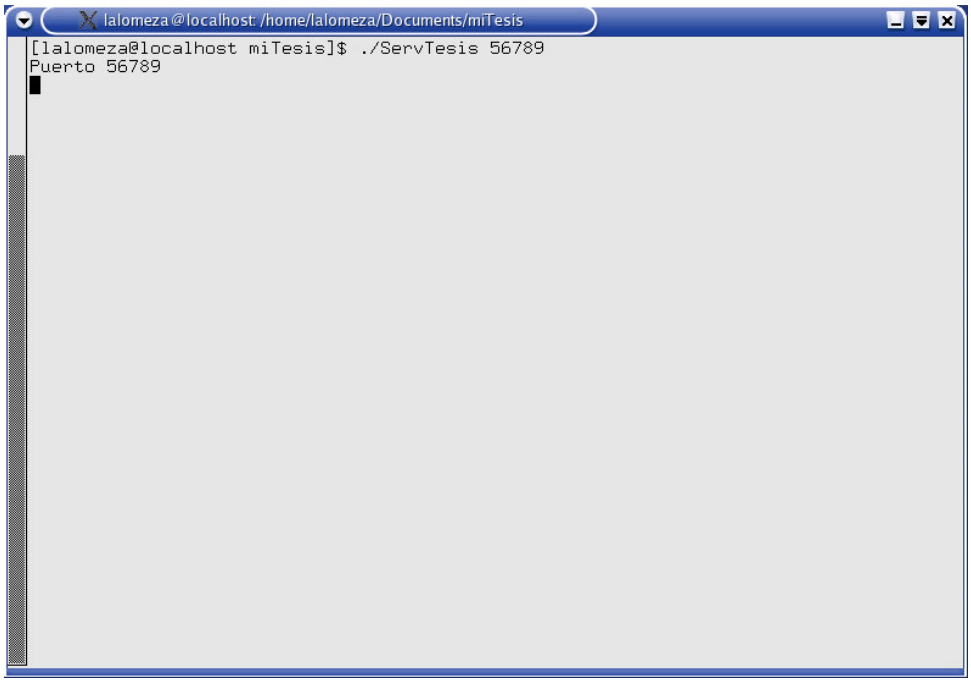

Figura 4-6 Ejecución inicial del servidor

En la Figura 4-6 se muestra al servidor con un parámetro que indica el puerto por el que escuchará las peticiones que le sean realizadas mediante el protocolo UDP, el puerto para la comunicación por el protocolo TCP es escogido por el programa cliente.

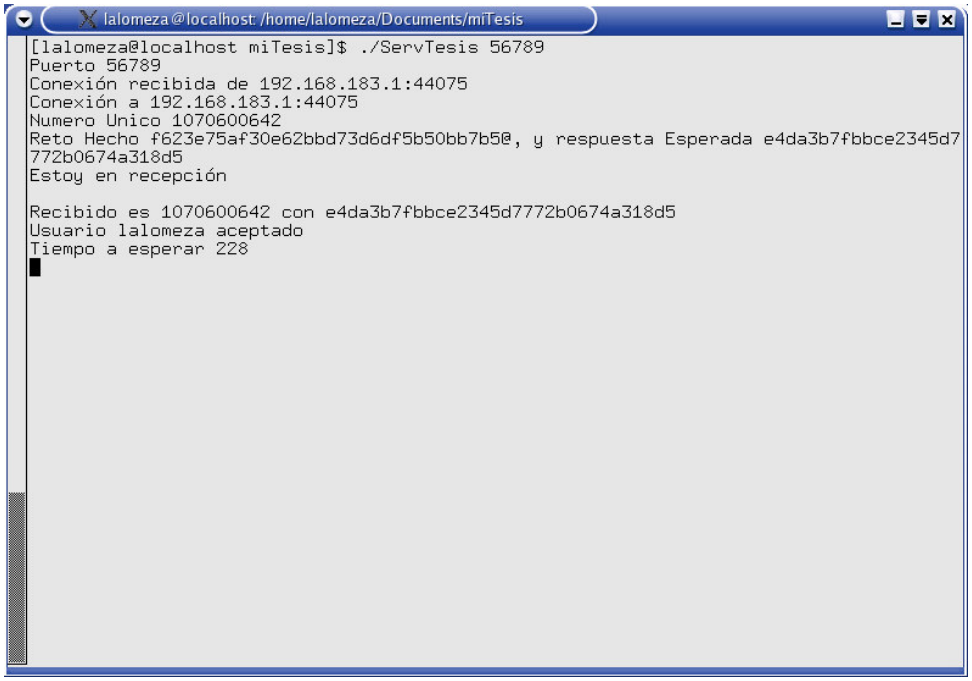

Figura 4-7 Conexión de un usuario

En la Figura 4-7 se muestra lo que ocurre cuando un usuario se ha conectado, indica la dirección IP y el puerto que se utilizará en el cliente para que el servidor establezca la comunicación, e inmediatamente viene la primera pregunta que es realizada por el servidor, después de que el cliente, cuya identificación es *lalomeza* contesta la pregunta, se indica qué éste ha sido aceptado; el tiempo a esperar es el tiempo en segundos que pasará antes de que le sea hecha otra pregunta a este usuario, el tiempo es variable.

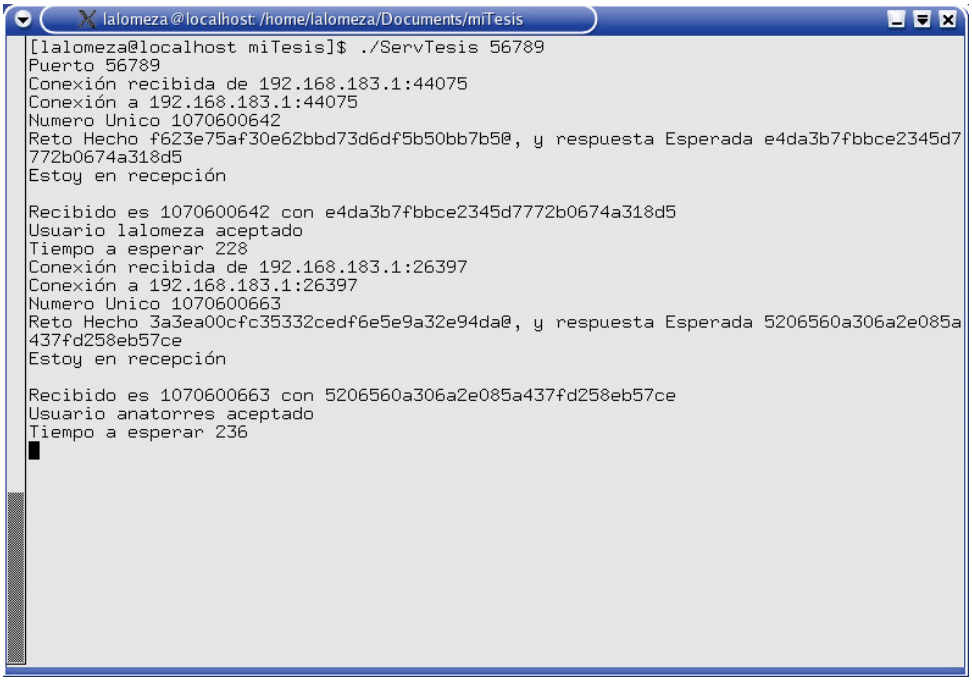

Figura 4-8 Conexión de un usuario distinto

En la Figura 4-8 se muestra como realiza la conexión otro usuario; al igual que con el primer usuario, se muestra la dirección IP desde la que fue ejecutado el programa, en este caso los dos clientes tienen la misma dirección debido a que se encuentran en la misma máquina pero usan distinto puerto para la comunicación que establece el servidor.

Después de que se realiza la conexión, se le envía una pregunta al usuario que éste contesta correctamente, una vez que se han ejecutado estos pasos se muestra el tiempo a esperar por parte del servidor antes de enviar la siguiente pregunta para este usuario, cuya identificación es *anatorres*.

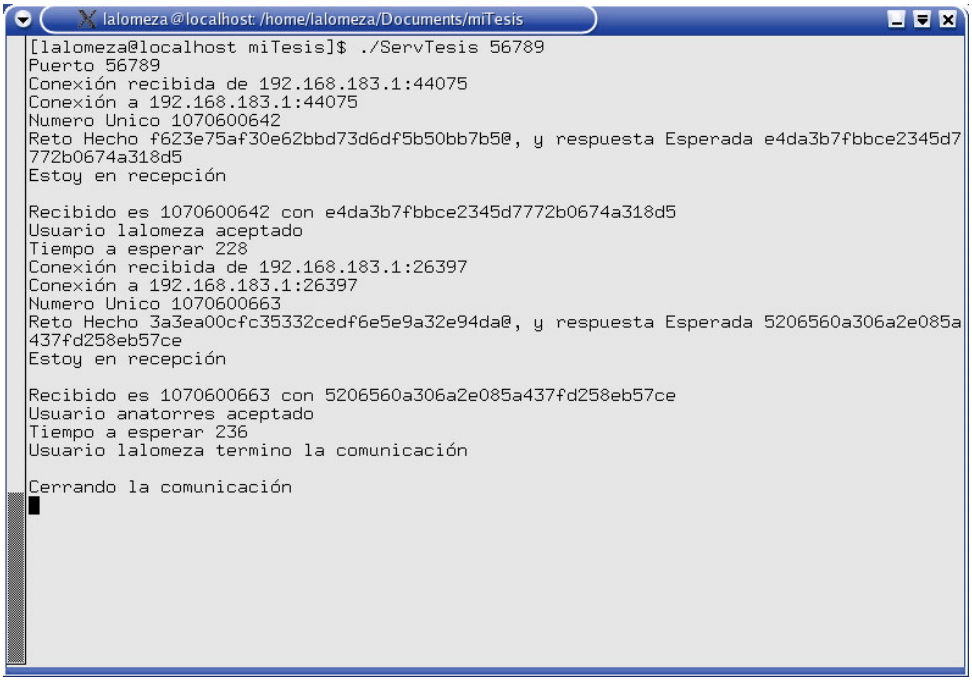

Figura 4-9 Un usuario termina la conexión

En la Figura 4-9 se demuestra la terminación de la conexión cuando es realizada por un usuario, indicando cuál es el usuario que terminó la comunicación.

4.3.8 EJEMPLO DE EJECUCIÓN EN EL CLIENTE

Los ejemplos que se muestran fueron hechos en un sistema operativo Windows 2000 profesional, se muestra la ejecución de dos clientes en la misma máquina de manera simultánea. El programa que se maneja en el cliente es una demostración de cómo funcionaría el protocolo del lado del cliente.

Figura 4-10 Programa cliente

En la Figura 4-10 se muestra la pantalla del programa cliente, solo se necesita el nombre de usuario y su contraseña, esta contraseña no es la que se utilizará para cifrar la comunicación que se realice con el servidor sino que se utiliza para cifrar y descifrar el archivo de configuración en que vienen guardadas las respuestas.

Se muestra el puerto TCP que será utilizado para que el servidor pueda establecer la comunicación con el cliente, se indica para información únicamente ya que el puerto no puede ser modificado por el usuario.

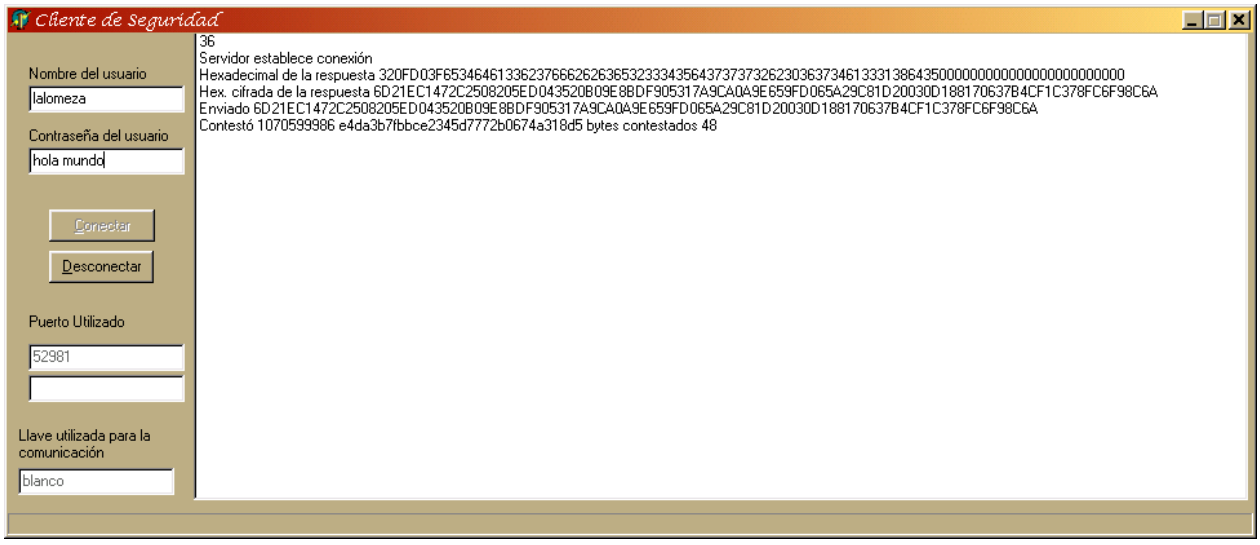

Figura 4-11 Conexión del programa cliente

En la Figura 4-11 mostramos el nombre de usuario utilizado para realizar la conexión, así como su contraseña, el nombre de usuario debe ser una identificación sin espacios en blanco, la contraseña puede contener espacios, aunque el tamaño de la contraseña no debe ser mayor a 16 caracteres incluyendo los espacios.

También en la Figura 4-11 se muestra qué contraseña es utilizada para cifrar la comunicación por los medios digitales, en este caso la contraseña es *blanco*. En la parte derecha en el cuadro de texto se muestra información relevante a la comunicación por parte del cliente, se indica qué valores son los que se están enviando al servidor, en la línea *Enviado* dentro del cuadro de texto de la Figura 4-11 se muestra la trama en formato hexadecimal que es enviada al servidor, la siguiente línea muestra los datos que fueron enviados al servidor sin la cabecera de control, se muestra el número único (que en la Figura 4-11 es 1070599986) y la respuesta *hash* que fue enviada (que en la Figura 4-11 es e4da3b7fbbce2345d7772b0674a318d5), así como el número de bytes con que cuenta la trama enviada. Estos datos solo son para describir que elementos fueron enviados.

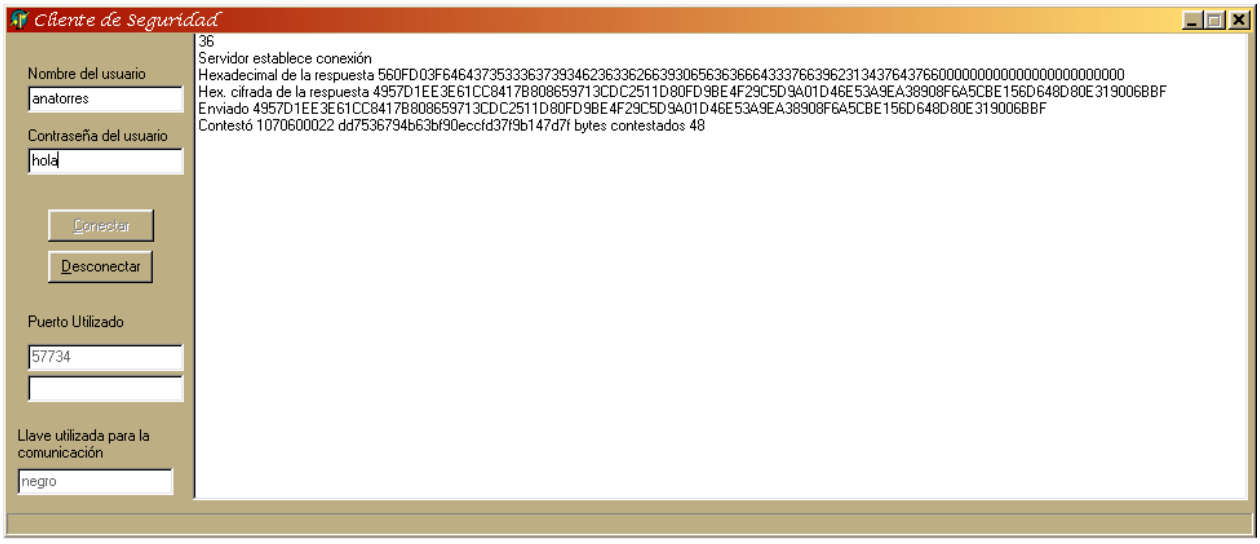

Figura 4-12 Conexión de un usuario distinto

En la Figura 4-12 la conexión de otro usuario, en este caso el usuario es *anatorres* y su contraseña es *hola*. El puerto utilizado es distinto al primer usuario para no interferir con la ejecución del otro programa.

Figura 4-13 Ejecución del cliente

En la Figura 4-13 se muestra la ejecución cuando se han realizado varias preguntas a un usuario, sólo es para demostrar como se desarrolla la ejecución del protocolo, en este caso para el usuario *anatorres*.

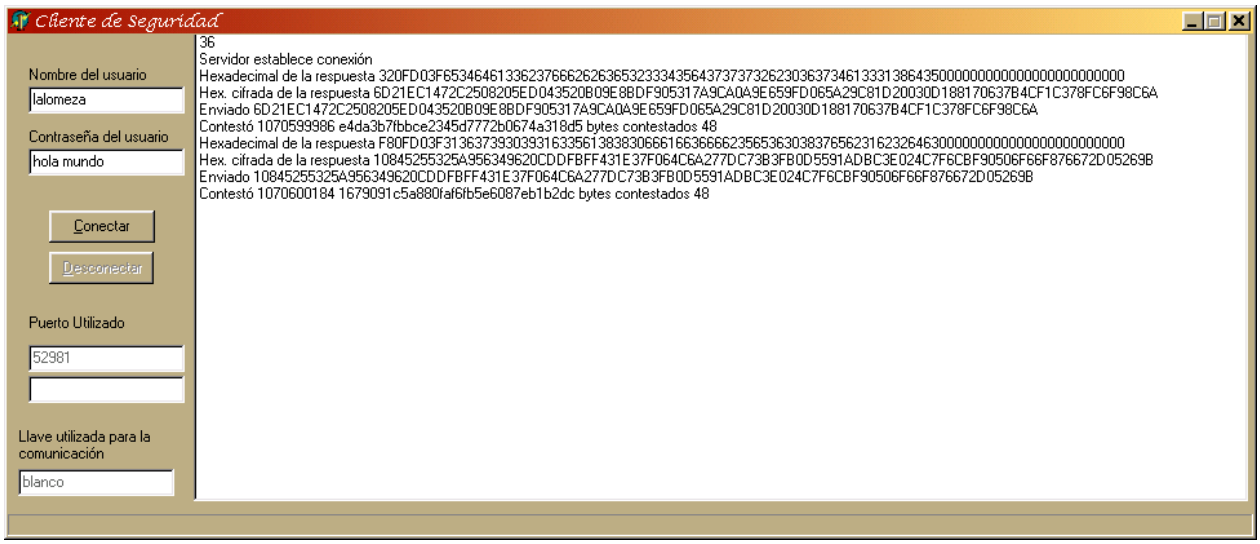

Figura 4-14 Desconexión por parte del usuario

En la Figura 4-14 muestra la desconexión que se realiza por parte de un usuario, el botón "Desconectar" se deshabilita para evitar que sea presionado varias veces, la contraparte en la ejecución del servidor se muestra en la Figura 4-9.

5 CONCLUSIONES Y TRABAJO FUTURO

Por los resultados obtenidos mediante las demostraciones realizadas en los capítulos anteriores, se comprueba que se alcanzó el objetivo planteado al principio de esta tesis. Recordemos que el objetivo planteado establece lo siguiente: "Diseño, validación e implementación de un protocolo de autenticación para un servidor de manufactura con independencia de los medios de transmisión de información que se utilicen", que con la metodología seguida se pudo alcanzar.

El desarrollo del trabajo de tesis constó de dos partes principales, el primero es el desarrollo teórico de los protocolos donde se consideran el diseño y la validación que analizaron la confiabilidad de cada uno de los protocolos para ver cuál era el más factible de implementar; a los tres protocolos propuestos en esta tesis [pag 60], se analizó cada uno de ellos mediante una metodología formal, se utilizó la lógica BAN, para comprobar la confiabilidad de cada uno de los protocolos con respecto a la seguridad que ofrecen; después de la validación mediante la lógica BAN se utilizo el lenguaje PROMELA para demostrar la correctitud del protocolo con respecto a la forma de ejecución, con esto se confirma que el protocolo que se escogió [protocolo P3, pag. 65] es el ideal para ser desarrollado de manera práctica.

La implementación práctica del protocolo sugerido demostró la factibilidad de que pueda ser desarrollado y aplicado a un sistema, en el caso de la presente tesis bajo ambiente Linux; la elección de este sistema operativo se debe al mayor control que proporciona sobre la programación de sockets, también a que existe un conocimiento previo de programación en lenguaje C para sockets en este sistema operativo.

La ventaja que se obtuvo al utilizar el sistema operativo Linux sobre el Sistema Operativo Windows es un mayor control sobre los puertos que son abiertos por el programa servidor, así como una facilidad mayor acerca de la creación y administración de procesos *hijos*[18] junto con la creación y administración de procesos con hilos.

La creación de procesos *hijos*, nos permite administrar un cliente de manera más eficiente, es como iniciar el mismo programa para cada uno de los clientes que se conecten, con la ventaja de que cada uno de los clientes tiene su propio espacio de memoria en el sistema operativo y puede

[18] Son procesos creados por otro proceso que es llamado padre.

manejar sus propias variables. En el caso del programa servidor las variables que se compartían entre los procesos, había un proceso que realizaba los retos a cada uno de los programas cliente, otro proceso se encargaba del tiempo que transcurría entre un reto y el siguiente, por lo que estos procesos de comunicación con cada uno de los programas clientes que están conectados se maneja de manera independiente. La desventaja de utilizar la creación de procesos hijos es el espacio de memoria que se consume, que si no se administra de manera correcta puede llegar a saturar la memoria de la máquina en la que este ejecutándose el servidor.

También ser crearon procesos usando hilos en el servidor que permitían compartir información entre las distintas funciones que se utilizan para administrar la comunicación establecida por el programa cliente, estos eran útiles para administrar las preguntas realizadas, en donde cada pregunta que el programa servidor le hace al programa cliente, es almacenada en una lista que se comparte entre distintas funciones del mismo proceso hijo creado, de esta manera se podía hacer una pregunta al cliente y al mismo tiempo estar recibiendo la respuesta a una pregunta hecha con anterioridad, por lo que había concurrencia en el funcionamiento del protocolo, la desventaja al utilizar los procesos hilos es la administración que se realiza para manejar las variables compartidas, por lo que había que asegurar que cuando una función estuviera agregando una pregunta que se acababa de realizar, la función que revisa las respuestas que el programa cliente contesta no borra alguna pregunta de la lista, ya que esto podía dar el caso de la corrupción de datos, y por ende, una falla en el servidor.

5.1 TRABAJO FUTURO

El protocolo desarrollado es suficiente para los objetivos planteados en esta tesis, aunque puede haber algunas mejoras para proporcionar mayor seguridad al protocolo; entre las mejoras propuestas, tanto para el desarrollo como para el diseño del protocolo sugerido, se pueden citar:

- **Una mejor generación de la llave de cifrado.**
  La llave que se utiliza para cifrar es escogida por el administrador del servidor por lo que se puede crear algún procedimiento que permita generar una clave mediante alguna función *hash* a partir de una palabra creando de esta manera una clave que mezcla números y letras lo que la hace difícil de adivinar.

- **Utilización de Base de Datos**
  Para administrar a los usuarios en el servidor se utilizan archivos de configuración específicos a un usuario, esto es conveniente cuando no se cuentan con una gran cantidad de usuarios, más de 20 por ejemplo, pero cuando se sobrepasa esa cantidad la administración de los archivos de configuración se puede volver confusa y tediosa; por lo que sería necesario implementar una base de datos que pueda guardar los usuarios y los retos que le sean hechos. Así todo se encuentra en un mismo lugar, de esta manera sería más sencillo recuperar la información específica a un usuario sin necesidad de estar abriendo o cerrando archivos.

- **Preguntas interactivas**

  Para el desarrollo de esta tesis, las preguntas y respuestas van dentro de un archivo de configuración por lo que cuando el servidor realiza una pregunta es el programa cliente quien contesta de manera automática al servidor y transparente al usuario, esto es una ventaja ya que no interrumpe las actividades que esté realizando un usuario en el programa cliente, pero puede ser una desventaja y un fallo en la seguridad si el archivo de configuración es obtenido por un usuario malintencionado. Por este motivo se pueden implementar preguntas interactivas en donde el usuario tenga que contestar la pregunta durante la ejecución del programa para comprobar que no esté utilizando el programa cliente algún usuario no autorizado que haya tomado el archivo de claves de un usuario que sí está autorizado.

6 BIBLIOGRAFÍA

[1] Abadi M. *Two facets of authentication.* In Proceedings of the 11th IEEE Computer Security Foundations Workshop, pages 27--32, 1998.

[2] C. Meadows. *Formal verification of cryptographic protocols: A survey.* In Advances in Cryptology - Asiacrypt 94, vol. 917 of LNCS, pp. 133--150.

[3] Anderson, Bergadano, Crispo, Lee, Manifavas, and Needham. *A new family of authentication protocols*. ACMOSR: ACM Operating Systems Review, 32, 1998.

[4] Burrows, M., Abadi, M., and Needham, R. M. *A Logic of Authentication*, ACM Transactions on Computer Systems, Vol. 8, No. 1, Feb 1990, pp. 18-36. A Formal Semantics for Evaluating Cryptographic Protocols p 14

[5] G. J. Holzmann. *The model checker SPIN*. IEEE Trans. on Softw. Eng., 23(5):279--295, May 1997.

[6] Lawrence C. Paulson. *The inductive approach to verifying cryptographic protocols*. Journal of Computer Security, 6:85--128, 1998.

[7] Buttyán L., Staamann S., Wilhelm U., *A Simple Logic for Authentication Protocol Design, Proceedings of the IEEE Computer Security Foundations Workshop XI*, (1998) 153-162, IEEE Computer Society Press

[8] Vanderwauver M., Govaerts R, Vandewalle J., *Overview of Authentication Protocols*. Dept. Elektrotechniek, ESAT-COSIC.

[9] Aviel D. Rubin and Peter Honeyman, *Formal methods for the analysis of authentication protocols*, Technical Report 93--7, Center for Information Technology Integration, Department of Electrical Engineering and Computer Science, University of Michigan, 8. November 1993.

[10] T. Wu, *The Secure Remote Password Protocol*, in Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium, San Diego, CA, Mar 1998, pp. 97-111.

[11] Li Gong and Paul Syverson. *Fail-stop protocols: An approach to designing secure protocols*. In R. K. Iyer, M. Morganti, Fuchs W. K, and V. Gligor, editors, Dependable Computing for Critical Applications 5, pages 79--100. IEEE Computer Society, 1998.

[12] W. Mao and C. Boyd. *Development of authentication protocols: Some misconceptions and a new approach*. In Proceedings of Computer Security Foundations Workshop VII. IEEE Computer Society Press, 1994.

[13] G.J. Holzmann, *Designing Bug-Free Protocols with SPIN*, The Computer Comm. J., to appear 1997.

[14] Li Gong, *Using One-Way Functions for Authentication*. ACM Computer Communications Review. October 1989.

[15] Walton Sean, *Programación de Socket Linux*. Pearson Educación 2001. 616 p.

[16] Simpson W. *PPP Challenge Handshake Authentication Protocol (CHAP)*. RFC 1994. August 1996

[17] Eastlake, D. and Jones P. *US Secure Hash Algorithm 1*. RFC 3174. September 2001.

[18] Rivest R. *The MD5 Message-Digest Algorithm.* RFC 1321. April 1992

[19] Lloyd B. and Simpson W. *PPP Authentication Protocols.* RFC 1334. October 1992

[20] Daemen J., Rijmen V. *AES Proposal: Rijndael* AES Algorithm Submission. September 1999

[21] Vázquez, Jesús. *Apuntes de Seguridad Computacional I*. CEM-ITESM.

[22] Oppliger Rolf, *Sistemas de Autenticación para Seguridad en Redes.* Ed. Computec RA-MA. 1998

[23] Tanenbaum, Andrew S., *Redes de Ordenadores*. Prentice Hall. 2a Edición. 1998

[24] Stallings, William, *Cryptography and Network Security. Principles and Practices.* Prentice Hall. 2ª Edición. 2000

[25] Meghabghab, G. *Introducción a UNIX.* Prentice Hall. 1998.

[26] Rivest, R., Shamir A. y Adleman, L. *A method for obtaining digital signatures and Public Key Criptosystems.* Communications of the ACM. 1978.

[27] Fúster, A. et al. *Técnicas criptográficas de protección de datos.* Computec Ra–Ma. 1998.

[28] Singh, Simon. *The Code Book.* Fourth Estate. 1999.

[29] Holzmann, Gerard J. *Design and Validation of Computer Protocols.* Prentice Hall. 1991.

[30] Keiser, Gerd. *Local Area Networks.* Mc Graw Hill. 1989.

[31] *The Directory: Public-Key and attribute certificate frameworks.* ITU-T. 2000.

[32] Schneier, Bruce. *Blowfish Encryption Algortihm*. Dr. Dobb's Journal, Abril 1994.

[33] Lai, X., and Massey, J. *A proposal for a new block encryption standard.* Proceedings, EUROCRYPT '90. 1990.

[34] Gas Kris and Chodowiec Pawel. *Hardware performance of the AES finalists – survey and analisys of results.* George Mason University. 1999.

[35] Black, Uyless. *Redes de Computadores Protocolos, normas e interfaces.* Computec RA-MA. 2ª Edición. 1997.

[36] Ferreira Giozza, William et al. *Redes locales de computadoras Protocolos de alto nivel y evaluación de prestaciones.* Mc Graw Hill. 1ª Edición. 1992.

[37] Raya, Jose Luis y Rodrígo Víctor. *Domine TCP/IP.* Computec-Rama. 1998.

[38] Gong Li, Needham Roger and Yahalom Raphael. *Reasoning about belief in cryptographic protocols.* Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy.

[39] Abadi Martin, Needham Roger. *Prudent Engineering Practices for Cryptographic Protocols.* IEEE Trans. Software Eng. 1995.

ANEXOS

ANEXO A. PROMELA

Una de las formas más sencillas de modelar y validar un protocolo es mediante un lenguaje de especificación de protocolos. En esta tesis se utilizara el lenguaje PROMELA para validar el protocolo de autenticación, en la faceta de cómo se lleva a cabo la comunicación [5] [13].

PROMELA fue creado por el AT&T en 1990[29], para realizar la modelación de un protocolo para su validación posterior se basa en procesos, canales y variables.

A.1 PROCESOS, CANALES Y VARIABLES

Como se menciono anteriormente, uno de los métodos para validar un protocolo es la utilización de una máquina de estados finita. La máquina de estados puede modelar todos los objetos que nos interesan dentro del diseño del protocolo, incluir variables finitas y canales de mensajes, aunque esté modelo es suficiente para un protocolo, no es muy viable trabajar con la máquina de estados, debido a que un protocolo complejo puede llegar a tener varios estados que lo vuelve muy complejo para seguir la ejecución de un protocolo.

Para validarlo en el lenguaje se definirán tres tipos de objetos:

- *Procesos.*
- *Canales de mensajes*
- *Variables de estado*

Para el propósito del análisis, cada uno de estos objetos puede ser trasladado a una máquina de estados finita; por definición todos los procesos son objetos globales, las variables y canales representan datos que pueden ser globales o locales a un proceso.

A.2 EJECUCIÓN DE LAS SENTENCIAS

En PROMELA no existen diferencias entre condiciones y sentencias. Las condiciones lógicas pueden ser utilizadas como sentencias. La ejecución de una sentencia esta condicionada a su

ejecución; todas las sentencias son entre ejecutables o bloqueadas, dependiendo de los valores actuales de las variables o el contenido de los canales de mensaje. La ejecución es el punto principal de la sincronización. Un proceso puede esperar a que ocurra un evento, esperando a que una sentencia se vuelva ejecutable. Un ejemplo sería el siguiente.

En lugar de escribir un ciclo de espera de la siguiente manera.
while (a ¡= b) skip

En PROMELA se puede lograr el mismo resultado con la siguiente sentencia
(a = = b)

La condición solo puede ser ejecutada si son iguales en su valor las variables *a* y *b*. Si las variables no son iguales, la ejecución de la sentencia se bloquea hasta que lo sean.

A.3 VARIABLES Y TIPOS DE DATOS

Las variables son utilizadas ya sea para almacenar información que es global al sistema o información que es local a un proceso específico, esto depende del lugar en que la variable fue declarada. Una variable puede tener cualquiera de los siguientes tipos de datos predefinidos.

Bit, bool, byte, short, int, chan.

Los primeros cinco tipos de datos en la lista se consideran los tipos de datos básicos. Son usados para especificar objetos que pueden contener un solo valor a la vez. El sexto tipo de dato especifica los canales de mensajes. Los canales de mensajes es un objeto que puede almacenar varios valores, agrupados en estructuras definidas por el usuario.

La declaración de una variable y el tipo de dato que almacenara la variable es la siguiente:

bool bandera;
int estado;

Aquí se definen variables que pueden almacenar valores enteros en dos rangos diferentes. El alcance de la variable es global si esta fue declarada fuera de la declaración de todos los procesos, y local si es declarada dentro de un proceso. El rango de los datos se define en la **Error! Reference source not found.**.

Nombre	Tamaño (bits)	Tipo	Rango
Bit	1	Sin signo	0..1
Bool	1	Sin signo	0..1
Byte	8	Sin signo	0..255
Short	16	Con Signo	$-2^{15}..2^{15}-1$
Int	32	Con Signo	$-2^{31}..2^{31}-1$

Tabla A-1 Tipos de datos básicos en Promela

A.4 MATRICES

Las variables pueden ser declaradas como matrices. Por ejemplo:

```
byte state[N]
```

declara una matriz de N bytes que pueden ser escogidos por una sentencia como la siguiente:

```
state[0] = state[3] + 5 * state[3*2/n]
```

donde n es una constante o variable que ha sido declarada en otro lugar dentro del programa. El índice de la matriz puede ser cualquier expresión que sea determinado por un valor entero. El rango de los valores válidos para el índice de la matriz va de 0 (cero) a N – 1, donde N determina el tamaño de la matriz. La utilización de un valor de índice fuera del rango es indefinida, aunque la mayoría de las veces causará un error en tiempo de ejecución.

Para terminar este punto hay que indicar que las declaraciones las variables y las asignaciones de valores a estas siempre son *ejecutables.*

A.5 TIPOS DE PROCESOS

Para ejecutar un proceso tenemos que nombrarlo, definir su tipo e instanciarlo. Primero veamos como se define y se nombran los procesos. Todos los tipos de procesos que puedan ser instanciados son definidos en declaraciones mediante la palabra *proctype*. El siguiente, por ejemplo, declara un proceso con una variable local llamada estado.

```
proctype A() {byte estado; estado = 3 }
```

El proceso es nombrado *A*. El cuerpo e la declaración, que va entre llaves, consiste en variables locales, declaración de canales y sentencias. La declaración de arriba contiene una declaración de una variable local y una sentencia, que es una asignación del valor 3 a la variable *estado*.

El signo de puntuación punto y coma (;) es un separador de sentencia, (no es una terminación de sentencia ya que no hay un punto y coma después de la última sentencia). En PROMELA se definen dos tipos de separadores de sentencias, una flecha (->) y un punto y coma (;). Los dos separadores son equivalentes, aunque la flecha es usada muchas veces para indicar una manera informal de indicar una relación casual entre dos sentencias. Considere el siguiente ejemplo:

```
byte state = 2;
proctype A() { (state == 1) -> state = 3 }
proctype B() { state = state – 1 }
```

En este ejemplo declaramos dos tipos de procesos, *A* y *B*. La variable *state* es una variable global, con un valor inicial de 2. El proceso A contiene dos sentencias, separadas por una flecha. El proceso B contiene una sentencia que disminuye el valor de la variable *state* en 1. Desde que la asignación es siempre ejecutable, el proceso B siempre terminara sin retraso, mientras que el proceso A, por otro lado, puede tener un retraso, hasta que la variable *state* contenga el valor esperado.

A.6 EL PROCESO INICIAL

Una definición mediante *proctype* solo declara el comportamiento del proceso, no lo ejecuta. Al principio solo un proceso es ejecutado: un proceso de tipo *init* que debe ser declarado explícitamente en cada especificación de PROMELA. El proceso *init* es comparable a la función *main()* de un programa estándar en C.

Lo más importante del proceso inicial es que se puede utilizar para asignar valores iniciales a las variables globales, crear canales de mensajes e iniciar procesos. Una declaración *init* para los dos procesos del sistema definidos más atrás, puede lucir de la siguiente manera:

```
init { run A(); run B() }
```

En este caso el proceso *init* inicia dos subprocesos, que se ejecutarán de manera concurrente a partir de que son llamados por el proceso *init*; el proceso *init* termina después que se inicia el segundo proceso (*B*). *Run* es un operador unitario que ejecuta una copia del proceso dado, y no se detiene a esperar a que el proceso termine. La sentencia *run* es ejecutable y devuelve un valor positivo si el proceso inicio su ejecución de manera correcta. Si no se puede ejecutar se devuelve un valor de cero indicando que ocurrió un problema al ejecutar el proceso.

El operador *run* puede enviar valores de parámetros a los nuevos procesos, definiéndolos de la siguiente manera:

```
proctype A(byte state; short set)
{
      (state == 1) -> state = set
}

init { run A(1, 3) }
```

Solo los canales y los tipos de datos básicos pueden ser enviados como parámetros. Matrices y procesos no pueden ser utilizados para este fin.

El operador *run* puede ser usado en cualquier proceso para crear otros procesos, no solo en el proceso inicial. Un proceso que se esta ejecutando desaparece cuando termina (por ejemplo, que

alcanzo el fin de su código de ejecución), pero no termina antes que todos los procesos que creó utilizando el operador *run*(sus *hijos*) han terminado primero.

A.7 CANALES

Los canales son utilizados para modelar la transferencia de datos de un proceso a otro. Son declaradas tanto de manera local o global de la misma manera en que se declararía una variable de cualquiera de los tipos de datos básicos, usando la palabra reservada *chan*. Por ejemplo:

```
chan a, b; chan c[3]
```

declara los nombres *a*¸ *b* y *c* como identificadores del canal, en donde el último es una matriz de canales. La declaración del canal puede tener un campo iniciador tal cómo:

```
chan a = [16] of {short}
```

en donde se declara el canal *a*. El campo iniciador indica que el canal puede almacenar hasta 16 mensajes del tipo *short* (definido en **Error! Reference source not found.**).

Si el mensaje que será transmitido por el canal contiene más de un campo, la declaración se hace de la siguiente manera:

```
chan qname = [16] of { byte, int, chan, byte }
```

En donde definimos un canal de datos que puede almacenar 16 mensajes en donde cada uno consiste en 4 campos distintos: uno con tamaño de 8 bits, el segundo de 32 bits, un nombre de canal y el último con 8 bits.

El envío de un valor a través de un canal, se hace de la siguiente forma:

```
qname!expr
```

en donde enviamos el valor de la variable *expr* a través del canal que acabamos de crear, lo que sería que el valor de la variable sea agregado al final de la cola del canal.

Para recuperar un valor del canal se usa el operador *?*, como en el siguiente ejemplo:

```
qname?msg
```

toma el valor de la parte inicial de la cola del canal, y lo almacena en la variable *msg*. El paso de los mensajes se hace de la manera *FIFO (First In – First Out, o* Primero en Entrar – Primero en Salir*)*, en donde el primer dato que se introdujo a la cola utilizando el operador *!* es el primer dato que se leerá con el operador *?*.

Si el canal cuenta con campos múltiples en cada mensaje, se pueden enviar y recibir varios valores en una sola expresión, mediante la siguiente sintaxis:

```
qname!expr1,expr2,expr3
qname?var1,var2,var3
```

Si se envían por el canal mas parámetros de los que esté puede almacenar, los parámetros sobrantes se pierden, si se envían menos parámetros, los campos sobrantes tienen un valor indefinido.

En la recepción ocurre de la siguiente manera, si la operación de recepción trata de recibir más parámetros de los que están disponibles, el valor de los parámetros extra en indefinido, si el receptor trata de recibir menos parámetros de los que fueron enviados, la información extra se pierde.

La ejecución de estas operaciones es dependiente del estado del canal, la operación de envío solo es ejecutable cuando el canal no esta lleno completamente. La operación de recepción, es solo ejecutable cuando el canal no está vacío.

A.8 COMUNICACIÓN *RENDEZVOUS*

Un tipo especial de canal es cuando se declara de tamaño 0 en la matriz, por ejemplo:

```
chan port = [0] of {byte}
```

En donde este tipo de canal solo puede transmitir, no puede almacenar, se utilizan mensajes con declaración de tipo *byte*.

Este tipo de canal es utilizado únicamente por dos procesos, un proceso emisor y otro receptor, que lo utilizan para sincronizarse.

En este tipo de canal no se produce la perdida de mensajes, si un mensaje no se puede transmitir se bloquea el proceso que envía el mensaje, al contrario de lo que ocurre en un canal que es definido con un buffer mayor que cero.

A.9 CONTROL DE FLUJO

Hasta el momento, los programas se han ejecutado de manera secuencial, pero la mayoría de los programas necesitan un control para evaluar varias condiciones y actuar en base a una de estas condiciones, para esto, PROMELA define los siguientes controles de flujo:

- Selección *Case.*

- Repetición.
- Saltos condicionales

A.10 SELECCIÓN *CASE*

El más simple de construir es la estructura de selección. Usando los valores de dos variables *a* y *b* para escoger entre estas opciones, por ejemplo, se puede escribir:

```
if
:: (a != b) -> opcion1
:: (a == b) -> opcion2
fi
```

La estructura de selección anterior contiene dos sentencias de ejecución, cada una precedida por cuatro puntos (dos puntos dobles), donde solo una secuencia de la lista será la que se ejecutará. Una secuencia puede ser selecciona solamente si la primer sentencia es ejecutable.

En el ejemplo, las condiciones son excluyentes entre sí, por lo que solo una puede ser ejecutable en un estado de tiempo, pero no necesariamente va a ser siempre así, por lo que en dado caso que dos o más sentencias son ejecutables, se selecciona una de ellas de manera aleatoria. Si todas las condiciones no son ejecutables, el proceso se bloquea hasta que al menos una de ellas puede ser seleccionada. No hay restricciones para el tipo de sentencias que pueden ser utilizadas como condicionantes.

A.11 REPETICIÓN

Una extensión de la estructura de selección es la estructura de repetición. Se puede realizar un programa que contenga código cíclico que al azar incremente o disminuya una variable dada.

```
byte count;

proctype counter()
{
      do
      :: count = count + 1
      :: count = count – 1
      :: (count == 0) -> break
      od
}
```

Al igual que en el caso de selección, solo una opción puede ser ejecutada en un momento determinado, cuando la opción complete su ejecución la estructura es repetida para escoger otra

opción. La manera normal para terminar una estructura de repetición es con la sentencia *break*; en el ejemplo, la estructura de repetición es terminada cuando el contador tiene un valor de cero.

A.12 SALTOS

Otra manera de terminar un ciclo es con un salto no condicional, una sentencia *goto* como la utilizada en otros lenguajes comunes (tal como Visual Basic, Basic, C).

La mejor manera de ilustrar el funcionamiento de esta sentencia *goto* es con un ejemplo. El siguiente es el algoritmo de Euclides que se utiliza para encontrar el máximo común divisor de dos posibles números:

```
proctype Euclid(int x, y)
{
      do
      :: (x > y) -> x = x – y
      :: (x < y) -> y = y – x
      :: (x == y) -> goto done
      od

done:
      skip
}
```

La sentencia *goto* en el ejemplo anterior salta a una etiqueta definida como *done*. Una etiqueta puede aparecer antes de una sentencia únicamente; en el ejemplo lo que se busca es saltar al final del programa. En este caso la sentencia *skip* no es útil, esta es una sentencia que es siempre ejecutable y no tiene efecto sobre el desarrollo del programa. La sentencia *goto* en sí es siempre ejecutable.

A.13 DEFINICIÓN DE TIPOS DE MENSAJES

En algunos lenguajes se podrá observar el uso de constantes en el código de un programa. El lenguaje PROMELA también permite la definición de constantes como en el siguiente ejemplo:

mtype = { ack, nak, err, next, accept }

Que es equivalente a la siguiente definición:

```
#define ack         1
#define nak         2
#define err         3
#define next        4
```

```
#define accept          5
```

La definición de tipos de mensajes formales es la manera preferida para especificar los tipos de mensajes, ya que se pueden utilizar los nombres de las constantes en lugar de sus valores en una implementación dentro de PROMELA, y así mejorar los reportes de error cuando se este modelando el protocolo. Solo puede haber una definición de mensajes con *mtype* por especificación.

A.14 SIMULACIÓN DE TIEMPOS DE ESPERA

Se han definido dos tipos de sentencias que tienen un significado predeterminado en PROMELA: *skip* y *break*, ahora se verá otra sentencia predefinida que es *timeout*. La sentencia *timeout* permite a un proceso abortar la espera por una condición que no puede ser verdad, como por ejemplo, la espera de un canal que está vacío. El *timeout* nos permite salir de un estado inconcluso. Puede ser considerada una condición artificial y predefinida que se vuelve verdad solo cuando ninguna otra sentencia en la modelación del protocolo de red es ejecutable. Está sentencia no lleva ningún valor, no se especifica un intervalo de espera para que se cumpla, solo modela la posibilidad de que ocurra el *timeout*; de esta manera nos abstraemos de manera deliberada de consideraciones de espera, que es muy importante en el trabajo de validación, y no se especifica como debe ser implementado el *timeout*.

A.15 INTERBLOQUEOS

En un sistema de estados finita, todas las secuencias de ejecución terminan ya sea después de un número finito de transiciones de estado, o cuando el ciclo regresa a un estado visitado previamente. No todas las secuencias de terminación se consideran interbloqueos. Para definir que es un interbloqueo en un modelo de PROMELA se debe distinguir entre los estados de terminación esperados o propios de los inesperados. Los estados finales inesperados son los que no incluyen solamente estados de interbloqueo, sino que también conllevan errores que son el resultado de una no completitud lógica de la especificación del protocolo. Un ejemplo común es la recepción de un mensaje inesperado.

El estado final en una secuencia de ejecución debe satisfacer los siguientes criterios para ser consideran estados finales correctos:

- Cada proceso que fue ejecutado ha terminado.
- Todos los canales están vacíos.

Pero no todos los procesos necesariamente terminan. Puede ser perfectamente válido, por ejemplo, para un proceso servidor permanecer vivo después de que el proceso usuario ha terminado. Se debe ser capaz de identificar estados de procesos individuales en las definiciones *proctype* como estados de terminación válidos. En el lenguaje PROMELA esto puede ser hecho con etiquetas de fin de estados. Un ejemplo de una definición la siguiente:

```
proctype dijkstra()
{
end: do
     :: sema!p -> sema?v
     od
}
```

En donde definimos que cualquier proceso ejecutado de este código estará en un estado de terminación válido porque esta etiquetado con la palabra *end*.

Si hay más de un estado de terminación válido dentro de la misma definición *proctype*, todas las etiquetas deben ser únicas. Una etiqueta de terminación válida se definirá como cualquier etiqueta que tenga un prefijo de tres caracteres *end*, como las etiquetas siguientes: *end0*, *end1*, *endn* en donde todas son consideradas como etiquetas de terminación válidas por contar con el prefijo *end*.

ANEXO B. CÓDIGO DE LA VALIDACIÓN CON SPIN

```
/* Mensajes utilizados durante la transmisión de datos. */
mtype = {seguridad, noseguridad, datos, identificacion, pregunta, respuesta, cnx,
dcnx};

/* Canales para simular la transmisión por UDP, el tamaño del canal es
cero porque no se simulará una cola de mensajes ya que no entra en los
objetivos de la tesis desarrollada.
*/
chan cli_udp = [1] of {byte, byte};
chan udp_ser= [1] of {byte, byte};

/* Canales para simular la transmisión por TCP, el tamaño del canal es
cero porque no se simulará una cola de mensajes ya que no entra en los
objetivos de la tesis desarrollada.
*/
chan cli_tcp=[1] of {byte, byte};
chan tcp_ser=[1] of {byte, byte};
chan ser_tcp=[1] of {byte, byte};
chan tcp_cli=[1] of {byte, byte};

/* Canales que se utilizaran para el paso de mensaje entre procesos internos
   para el servidor y para el cliente.
*/
chan cli_cls=[0] of {byte};
chan cli_cln=[0] of {byte};
chan ser_srs=[0] of {byte};
chan ser_srn=[0] of {byte};
```

```
/* Procedimiento para simular el medio de comunicación.
No se esta validando el medio, solo se simula en un proceso aparte
para facilitar la modificación en como se transmiten los mensajes
*/
proctype mc()
{
    byte msj, msj2;
endmc: do
        /* En la transmisión por udp puede llegar el mensaje
           o no llegar, no se considera que el mensaje llegue
           con errores
        */
        :: cli_udp?msj,msj2 -> udp_ser!msj,msj2;
        :: cli_udp?msj,msj2 -> skip;
        /* En la transmisión por tcp no se consideran errores en los
           mensajes, ni pérdida de estos.
        */
        :: cli_tcp?msj,msj2 -> tcp_ser!msj,msj2;
        :: ser_tcp?msj,msj2 -> tcp_cli!msj,msj2;
    od;
}

/* Módulo que se encarga de enviar los mensajes al proceso correspondiente
  ya sea al cliente normal o al cliente de seguridad.
*/
proctype cliente()
{
    byte msj;
endcliente: do
        :: tcp_cli?noseguridad,msj -> cli_cln!msj;
        :: tcp_cli?seguridad,msj -> cli_cls!msj;
    od;
}

/* Simulamos el cliente en su proceso de seguridad
*/
proctype cliente_seguridad()
{
    /* El primer paso es enviar la identificación al servidor
```

```
      usando el canal de UDP.
    */
    cli_udp!seguridad,identificacion;

    /* El canal UDP no garantiza entrega, por lo que las condiciones
      que pueden ocurrir son un timeout (se perdió el mensaje) o que
      llegue la pregunta por el canal UDP
     */
    if
        :: timeout -> goto salida_cls;
        :: cli_cls?pregunta ->
progress_cls1: cli_tcp!seguridad,respuesta;
    fi;

    /* Después de contestar la pregunta inicial pueden ocurrir dos cosas:
      1. Que el servidor desconecte al cliente por que la respuesta es inválida.
        (Mensaje dcnx).
      2. Que el servidor acepte al cliente, que en este caso se simulará con
        un mensaje de conexión, pero en el mundo real no habrá ningún mensaje
        de confirmación. (Mensaje cnx).
    */
    if
        :: cli_cls?cnx -> skip;
        :: cli_cls?dcnx -> goto salida_cls;
    fi;

/* Cuando la conexión este establecida, el proceso estará en un ciclo
  en donde recibe la pregunta y la contesta o puede ocurrir una
  desconexión (dcnx).
*/

end_cls1: do
        :: cli_cls?pregunta ->
progress_cls2:    cli_tcp!seguridad,respuesta;
        :: cli_cls?dcnx -> goto salida_cls;
    od;

salida_cls:
    skip;
```

```
}


/* Módulo que se encarga de enviar los mensajes al proceso correspondiente
   ya sea al servidor normal o al servidor de seguridad.
*/
proctype servidor()
{
    byte msj;
end_servidor: do
        :: udp_ser?seguridad,msj -> ser_srs!msj;
        :: tcp_ser?noseguridad,msj -> ser_srn!msj;
        :: tcp_ser?seguridad,msj -> ser_srs!msj;
    od;
}

/* Simulamos al servidor en su proceso de seguridad.
*/
proctype servidor_seguridad()
{
    byte contador;
    /* El primer paso que esperamos es una identificación del usuario,
       aunque es transmitido por el protocolo UDP, en el servidor no
       se considerá que hay una pérdida de mensaje porque no se sabe
       que envió un mensaje.

    Cuando se recibe la identificación, se consideran dos posibilidades
       1. Que el usuario sea conocido, en este caso se envia una pregunta.
       2. Que el usuario se desconozca, en este caso no se hace nada.
    */
end_srs1: do
        :: ser_srs?identificacion -> ser_tcp!seguridad,pregunta;
                break;
        :: ser_srs?identificacion -> skip;
    od;

    /* Se espera una respuesta, cuando se recibe pueden ocurrir dos cosas.
       1. Que la respuesta sea válida. En este caso se envía un mensaje de
          conexión, algo que no ocurrirá en la ejecución real ya que en la
```

```
        ejecución real ya estará conectado, pero este mensaje es útil en
        la simulación ya que no existe ningún mensaje de confirmación.
    2. Que la respuesta sea inválida. En este caso se envía un mensaje
      de desconexión, algo que también ocurre en el mundo real.
    */
    if
        :: ser_srs?respuesta -> ser_tcp!seguridad,cnx;
        :: ser_srs?respuesta -> ser_tcp!seguridad,dcnx;
                goto salida_srs;
    fi;

    /* Cuando se haya contestado la pregunta inicial, el servidor de seguridad
      estará en un ciclo en el que en tiempos aleatorios se enviara una pregunta
      al cliente. También se inicia el proceso del servidor normal.
    */

end_preguntas: ser_tcp!seguridad,pregunta;
    contador = 1;
    assert(contador==1);

    /* Hay que esperar la respuesta */
    do
        :: ser_srs?respuesta ->
            /* Mientras el contador no sea igual tres, se reenviará la pregunta
                si la respuesta recibida fue incorrecta.
            */
            if
                :: contador < 3 ->
                    assert(contador<3);
                    contador = contador + 1;
                    ser_tcp!seguridad,pregunta;
                :: contador == 3 ->
                    assert(contador==3);
                    ser_tcp!seguridad,dcnx;
                    goto salida_srs;
            fi;
        /* La respuesta fue válida, se regresa a la etiqueta de preguntas, para
          realizar otra.
        */
```

```
        :: ser_srs?respuesta -> goto end_preguntas;
    od;

salida_srs:
    skip;
}

init
{
    run mc();
    run servidor();
    run cliente();
    run servidor_seguridad();
    run cliente_seguridad();
}
```

ANEXO C. CODIGO FUENTE DEL SERVIDOR

6.1 ARCHIVO *SERVTESIS-VFINAL.C*

```
/*************************************************************************
Hecho por: Eduardo Meza Carmona  Para obtener el grado de maestría en
ciencias de la computación.
Tesis: Diseño y validación de un protocolo de autenticación para un sistema
de manufactura.

Código fuente del sistema servidor para realizar la autenticación de usuarios.

Compilación
cc ServTesis.c libusuario2.c md5.o -o ServTesis -lmcrypt

Ejecucion
ServTesis.c <puerto>

Donde:
<puerto>: Es el puerto que se utiliza en el servidor para esperar la
comunicación.
*************************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
```

```
#include <mcrypt.h>
#include <pthread.h>

//Estas constantes son utilizadas para armar la
//trama del protocolo de seguridad.
#define SEGURIDAD 0x3C
#define IDENTIFICACION    0x3C
#define RETO          0xF0
#define RESPUESTA  0x0F
//Esta constante indica el tiempo de espera en el servidor antes de reenviar o
//terminar la comunicación.
#define TIEMPO       60.0

//Puerto que usuara el servidor para esperar la conexión.
#define PUERTO       56789

#define VERDAD      1
#define FALSO        0

#define min(a,b) (a>b)?b:a
/*
 *   Definiciones de variables
 */

// Estructura utilizada para convertir de una cadena de 4 bytes a un número long
typedef union {
  unsigned long numero;
  unsigned char cadena[4];
} convNumCad;

//Estructura para manejar la trama de identificación.

typedef union {
  struct {
    unsigned char numUnico[4];
    unsigned char relleno[12];
    unsigned char usuario[12];
  };
  struct {
```

```
    unsigned char datoscifrados[16];
    unsigned char usuario2[12];
  };
  struct {
    unsigned char cadena[28];
  };
} TIdentificacion;

/*
  Estructura para utilizar la trama de comunicaciones de una manera sencilla
  y lo que se utilizará para enviarla a traves de la red es la cadena.
*/
typedef union {
  struct {
    unsigned char identificacion;
    unsigned char subidentificacion;
    unsigned char tam;
    unsigned char numUnicoS[4];
    unsigned char numUnicoA[4];
    unsigned char relleno[8];
    unsigned char reto[32];
  };
  struct {
    unsigned char cabecera[3];
    unsigned char datoscifrados[16];
    unsigned char pregunta[32];
  };
  struct {
    unsigned char cadena[51];
  };
} TReto;

/* Unión para manejar de manera más fácil la respuesta que fue
   recibida, para poder descifrarla utilizando la variable cadena
   y manejarla de manera fácil con las otras variables, así como
   también servira de paso para el hilo que verifique las respuestas
*/
typedef union {
  struct {
```

```
    unsigned char numUnicoS[4];
    unsigned char respuesta[32];
    unsigned char relleno[12];
  };
  struct{
    unsigned char cadena[48];
  };
} TRespuesta;


// Esta estructura es para controlar los retos enviados, el timeout y
// el número de reintentos que se han hecho
struct sRetosHechos{
  //Numero utilizado para evitar los mensajes duplicados y comprobar
  //que es reciente.
  unsigned char numUnico[4];
  //El reto que fue hecho, útil cuando se va a hacer el reintento
  unsigned char reto[32];
  //La respuesta que esperamos para este reto
  unsigned char respuesta[32];
  //Para saber si ha caducado algún tiempo y reenviar este reto
  unsigned long tiempo;
  //Numero de intentos que se han hecho
  int intentos;
  //Puntero para crear una lista
  struct sRetosHechos *sig;
};

/* ****************************** */
/* Variables para manejo de hilos */
/* ****************************** */
typedef struct {
  unsigned char nombre[17];
  int sck;
} datos_hilo_pregunta;

typedef struct {
  unsigned char cadena[48];
  unsigned char llave[16];
```

```
} datos_hilo_respuesta;

// Será para la lista de los retos hechos
// en donde se agregarán las preguntas y
// se verificarán las respuestas
struct sRetosHechos *retosHechos;

// Variable utilizada en la actualización de las preguntas
// ya sea para agregar una pregunta recien hecha o para
// comprobar una respuesta recibida
pthread_mutex_t mutexRespuestas;

//Variable global que contendrá el socket por el que se realiza
//la comunicación y la otra variable indica cuando un cliente
//se rechazó por algún motivo.
int socketCliente;
int Rechazado;

// Variables globales para manejar datos informativos.
// Por el momento solo la dirección IP y el puerto utilizado.
// La dirección IP en formato XXX.XXX.XXX.XXX
char direccionIP[16];
int puertoIP;
char gUsuario[50];

// Función que será utilizada para realizar las preguntas
// Sera ejecutada en un ciclo infinito, en sus argumentos
// incluira el socket a utilizar y la llave a utilizar.
void *realizarPreguntas(void *arg);


// Función que será utilizada para revisar las respuestas
// Sera ejecutada solo cuando reciba una respuesta en los
// argumentos estará la llave a utilizar así como TRespuesta
void *revisarRespuestas(void *arg);

// Función para revisar de manera periodica si hay necesidad
// de reenviar una pregunta
void *reenviarPreguntas(void *arg);
```

```
//Funcion para liberar la memoria de los retos cuando el usuario
//cierre la conexión
void liberarMemoriaRetos(void);

/* Funcion utilizada para descifrar un texto, será utilizada
   para descifrar el mensaje que recibamos como respuesta de
   la pregunta que hayamos hecho, los parametros recibidos son:

   unsigned char *    es el bloque a descifrar, debe de ir en multiplos de 16
   bytes
   int        es el tamaño del bloque, debe de ser un multiplo de 16 bytes
*/
void descifrar(unsigned char *, unsigned char *, int);

/* Función utilizada para cifrar un texto, será utilizado
   para cifrar los números únicos que se envien en la pregunta.
   Los parametros utilizados son:
   unsigned char * es el bloque a cifrar, debe de ser un multiplo de 16 bytes
   int el tamaño del bloque, debe se un número múltiplo de 16.
*/
void cifrar(unsigned char *, unsigned char *, int);

/* Función para saber si existe un usuario en la BD
   Recibe como parametros.
   Unsigned char *    nombre del usuario que se verificara.

   Devuelve:
   0 cuando el usuario no existe.
   1 cuando el usuario existe.
*/
int existeUsuario(unsigned char *);

/* Función para establecer las conexiones a los usuarios y administrar la
   conexion    que se ha establecido


   Parametros:
   struct sockaddr_in         nos permite saber la dirección y el puerto que utilizaremos
```

```
   para conectarnos al cliente
   unsigned char *            para saber el nombre del archivo.

   Devuelve:
   Un valor 0 (cero) si hubo problemas durante la conexión.

*/
int administraUsuario(struct sockaddr_in, TIdentificacion);

/* Funcion para terminar los hijos creados con fork
 */
void matar();


void descifrar(unsigned char *bloque, unsigned char *llave, int tam)
    /* Funcion utilizada para descifrar un texto, será utilizada
    para descifrar el mensaje que recibamos como respuesta de
    la pregunta que hayamos hecho.

    Los parametros recibidos son:

    bloque: es el bloque a descifrar, debe de ir en multiplos de 16
    bytes
    tam: es el tamaño del bloque, debe de ser un multiplo de 16 bytes
    */

{
  //Inicializamos las variables del cifrado.
  MCRYPT td;
  td = mcrypt_module_open("rijndael-128",NULL,"ecb",NULL);
  mcrypt_generic_init(td,llave,16,NULL);

  //Desciframos el bloque.
  mdecrypt_generic(td, bloque, tam);

  //Limpiamos las variables.
  mcrypt_generic_deinit(td);
  mcrypt_module_close(td);
  return;
```

```
};

void cifrar(unsigned char *bloque, unsigned char *llave, int tam)
{
  MCRYPT td;
  td = mcrypt_module_open("rijndael-128",NULL,"ecb",NULL);
  mcrypt_generic_init(td,llave,16,NULL);
  mcrypt_generic(td, bloque, tam);
  mcrypt_generic_deinit(td);
  mcrypt_module_close(td);
  return;
};


void matar()
{
  // Para liberar el espacio utilizado por un hijo que
  // administra la conexión con un cliente
  printf("Terminando el proceso hijo\n");
  wait(0);
  signal(SIGCHLD,matar);
}

void liberarMemoriaRetos(void)
    /*
      Se utiliza para liberar la memoria de las preguntas hechas al cliente.
      Se ejecuta cuando se termina la conexión con el cliente.
    */
{
  struct sRetosHechos *p;

  p = retosHechos;
  //Se recorre la lista de las preguntas hechas y se va liberando la memoria.
  while (retosHechos)
    {
      p = p->sig;
      retosHechos=NULL;
      free(retosHechos);
      retosHechos = p;
```

```
    };

  return;
};

// Función para revisar de manera periodica si hay necesidad
// de reenviar una pregunta
void *reenviarPreguntas(void *arg)
{
  int sck, i;
  unsigned char clave[16], *llave, *nombre, *archivo;
  convNumCad numUnico;
  datos_hilo_pregunta *argumentos;
  unsigned long tiempo;
  TReto retoAEnviar;
  struct sRetosHechos *p;

  argumentos = (datos_hilo_pregunta *) arg;
  archivo = (unsigned char *) calloc(strlen(argumentos->nombre)+5,sizeof(char));
  nombre = (unsigned char *) calloc(strlen(argumentos->nombre)+1,sizeof(char));
  strcpy(nombre, argumentos->nombre);
  strcpy(archivo, argumentos->nombre);
  strcat(archivo, ".ser");
  sck = argumentos->sck;

  // Leemos la llave a utilizar con este usuario
  leerValor(archivo, "[Usuario]", "clave", &llave);
  memset(clave, 0, sizeof(clave));
  strncpy(clave, llave, strlen(llave));
  free(llave);

  //Iniciando la función reenviarPreguntas
  while (!Rechazado)
    {
      sleep(5);
      if (Rechazado)
     break;
      p = retosHechos;
      //Falta revisar los retos, para saber si existe
```

```
      //algún reto que haya que reenviar
      tiempo = time(NULL);

      while (p)
     {
       if (((tiempo - (p->tiempo)) >= 5) && (p->intentos<=3))
         //Cuando se cumple la condición se reenvia el reto al que apunta p.
         {
           memset(retoAEnviar.cadena,sizeof(retoAEnviar.cadena),0);
           //Se construye la trama tomando como datos los que se enviaron
originalmente.
           bcopy(p->reto,retoAEnviar.reto, sizeof(retoAEnviar.reto));
           bcopy(p->numUnico,retoAEnviar.numUnicoS, sizeof(p->numUnico));
           retoAEnviar.identificacion = SEGURIDAD;
           retoAEnviar.subidentificacion = RETO;
           retoAEnviar.tam = 51;
           bcopy(p->numUnico, numUnico.cadena, sizeof(numUnico.cadena));
           cifrar(&(retoAEnviar.datoscifrados[0]),clave, 16);

           //Se bloquean los hilos de ejecución para actualizar
           //la lista de las preguntas.
           pthread_mutex_lock(&mutexRespuestas);
           (p->intentos)++;
           p->tiempo = tiempo;
           pthread_mutex_unlock(&mutexRespuestas);

           if (p->intentos > 3)
           //Ya son varios intentos, cerramos la conexion
           //si no pudo responder
           {
             printf("Se cierra la conexión por máximo de reintentos para %s en %s:%d\n",\
                 gUsuario, direccionIP, puertoIP);
             Rechazado=VERDAD;
             exit(0);
           }

           //Se muestra los intentos que se han hecho (informativo) y
           //se reenvia la pregunta.
           printf("\n--- REENVIO DE RETO ---\n");
```

```
          printf("Núm. de intento %d\nNúmero único %u\n",p->intentos, \
             numUnico.numero);
          printf("Pregunta enviada %.32s\nRespuesta %.32s\nPara %s en %s:%d", \
             retoAEnviar.reto, p->respuesta, nombre, direccionIP, \
             puertoIP);
          printf("\n--- FIN DE REENVIO DE RETO ---\n");
          send(sck, retoAEnviar.cadena, sizeof(retoAEnviar), 0);
        }
       p = p->sig;
     }; // Fin del while que recorre las preguntas hechas.
    };    // Fin del while(!Rechazado)
  free(nombre);
  free(archivo);
  return;
};

void *realizarPreguntas(void *arg)
     /*
       Función que será utilizada para realizar las preguntas
       Sera ejecutada en un ciclo infinito, en sus argumentos
       incluira el socket a utilizar y la llave a utilizar.
       Los parametros:
       arg: Se convierte (cast) en una estructura que contiene
       el nombre del archivo y el puerto utilizado para comunicarse
       con el cliente.
     */
{
  int sck, retoEscogido, numRetos, tam, tiempo, tiempoInicial, tiempoFinal;
  unsigned char *nombre, *archivo, clave[16], *llave, *retoAHacer, *respuestaEsperada;
  TReto retoAEnviar;
  convNumCad numUnico;
  datos_hilo_pregunta *argumentos;
  pthread_t administraReenvios;
  struct sRetosHechos *p;
  int pvez;
  argumentos = (datos_hilo_pregunta *) arg;

  // Pasamos los datos de manera que se puedan manejar mejor
  archivo = (unsigned char *) calloc(strlen(argumentos->nombre)+5, sizeof(char));
```

```
nombre = (unsigned char *) calloc(strlen(argumentos->nombre)+1, sizeof(char));
strcpy(nombre, argumentos->nombre);
strcpy(archivo, argumentos->nombre);
strcat(archivo, ".ser");
sck = argumentos->sck;

// Leemos la llave a utilizar con este usuario
leerValor(archivo, "[Usuario]", "clave", &llave);
memset(clave, 0, sizeof(clave));
strncpy(clave, llave, strlen(llave));

// Liberamos la memoria de la llave
free(llave);

// Contamos los retos que hay para este usuario
numRetos = contarValores(nombre, "[Reto]");

srand(time(NULL));
numUnico.numero = (unsigned long) (4294967294.0 * rand() / (RAND_MAX+1.0));
while (!Rechazado)
 {
   //Dormimos este proceso por un tiempo
   tiempo = 1 + (int) (TIEMPO * rand() / (RAND_MAX+1.0));
   printf("Tiempo a esperar %d segs. para el cliente %s:%d\n",\
      tiempo, direccionIP, puertoIP);

   sleep(tiempo);

   if (Rechazado)
  break;

   // Escogemos una de las preguntas de manera aleatoria
   retoEscogido = 1 + (int) ((float)numRetos * rand() / (RAND_MAX+1.0));

   // Pasos necesarios para realizar la pregunta inicial al usuario
   // Inicializamos la estructura del reto a cero para evitar
   // caracteres no contemplados
   memset(retoAEnviar.cadena, 0, sizeof(retoAEnviar.cadena));
```

```
// Inicializamos el formato de nuestra trama, con nuestros codigos
retoAEnviar.identificacion = SEGURIDAD;
retoAEnviar.subidentificacion = RETO;

// Leemos una de las preguntas propias para este usuario,
// y se le envia.
printf("Leemos reto y respuesta\n");
leerClave(archivo, "[Reto]", &retoAHacer, retoEscogido);
leerValor(archivo, "[Reto]", retoAHacer, &respuestaEsperada);
printf("Terminamos de leer el reto y respuesta\n");
strncpy(retoAEnviar.reto, retoAHacer, strlen(retoAHacer));

// Se le asigna un número único para comprobar que la respuesta
// recibida es reciente
numUnico.numero++;
strncpy(retoAEnviar.numUnicoS,numUnico.cadena, sizeof(numUnico.cadena));
cifrar(&(retoAEnviar.datoscifrados[0]),clave, 16);

//Se asigna el formato de la trama, en este caso siempre será 51,
//ya que es una trama fija.
retoAEnviar.tam = 51;

printf("\n--- ENVIO DE RETO ---\n");
printf("Número único %u\nReto Hecho %.32s\nRespuesta Esperada %.32s" \
   ,numUnico.numero, retoAEnviar.reto, respuestaEsperada);
printf("\nPara el usuario %s en la dirección %s:%d", nombre, \
   direccionIP, puertoIP);
printf("\n--- FIN DE ENVIO DE RETO ---\n");

pthread_mutex_lock(&mutexRespuestas);
//Ahora pasamos el reto a la lista de espera
if (retosHechos == NULL)
{ //Es la primera pregunta que hacemos
  retosHechos = (struct sRetosHechos *) malloc(sizeof(struct sRetosHechos));
  p = retosHechos;
} else {
  p=retosHechos;

  // Buscamos el final
```

```
      while (p->sig)
        p = p->sig;

      // Creamos el nodo
      p->sig = (struct sRetosHechos *) malloc(sizeof(struct sRetosHechos));
      p = p->sig;
    };

    //Estamos al final de la lista, agregamos los datos
    bcopy(numUnico.cadena, p->numUnico, sizeof(p->numUnico));
    bcopy(retoAHacer, p->reto, sizeof(p->reto));
    bcopy(respuestaEsperada, p->respuesta, sizeof(p->respuesta));

    //Para saber en que momento se mando, no tiene relación con el numUnico
    p->tiempo = time(NULL);
    p->intentos = 1;
    p->sig = NULL;

    pthread_mutex_unlock(&mutexRespuestas);

    //Enviamos el reto, no hacemos ninguna comprobación de error por el momento
    send(sck, retoAEnviar.cadena, sizeof(retoAEnviar.cadena), 0);

    //Liberamos la memoria utilizada para leer los retos y las respuestas.
    free(retoAHacer);
    free(respuestaEsperada);
  };
 free(archivo);
 free(nombre);
 return;
};

void *revisarRespuestas(void *arg)
    /*
     Función utilizada para revisar las respuestas.
     Será ejecutada cuando se reciba una respuesta.
     El argumento es:
     arg: Se convierte (cast) en una estructura datos_hilo_respuesta.
     Incluye la trama recibida y la llave utilizada
```

```
    */
{
  datos_hilo_respuesta *argumentos;
  TRespuesta respuestaActual;
  convNumCad numUnico;
  unsigned char *llave, *bloque;
  struct sRetosHechos *p, *q;

  argumentos = (datos_hilo_respuesta *) arg;
  bcopy(argumentos->cadena, respuestaActual.cadena,
sizeof(respuestaActual.cadena));
  bcopy(respuestaActual.numUnicoS, numUnico.cadena, sizeof(numUnico.cadena));
  printf("\n--- RESPUESTA RECIBIDA ---\n");
  printf("Número único: %u\nValor: %.32s del usuario %s",
     numUnico.numero, respuestaActual.respuesta, gUsuario);
  printf("\n--- FIN DE RESPUESTA RECIBIDA ---\n");

  //Una vez que se descifra se compara con las respuestas que
  //se esta esperando.
  pthread_mutex_lock(&mutexRespuestas);
  p = retosHechos;
  //Buscamos que el número de trama (numUnico) exista en nuestra lista
  while (bcmp(p->numUnico, respuestaActual.numUnicoS,
sizeof(respuestaActual.numUnicoS)) != 0 \
     && p)
   {
     q = p;
     p = p->sig;
   }

  //Se encontro el número único que también se utiliza como el número de trama,
  //ahora comparamos lo contestado contra lo esperado.
  if (p && bcmp(p->respuesta, respuestaActual.respuesta,sizeof(p->respuesta))==0)
   {
     // Si se encontro el reto estara en el nodo p, se elimina ese de la lista
     if (p==retosHechos)
    { // No nos hemos movido del padre
      retosHechos = retosHechos->sig;
      free(p);
```

```
      p = NULL;
    } else
      {
        q->sig = p->sig;
        free(p);
        p = NULL;
      };
   } else if(p && p->intentos>=3)
    {   //Ya son varios intentos, cerramos la conexion
    //si no pudo responder
    printf("\nSe cierra la conexión por máximo de reintentos para %s en %s:%d\n",\
          gUsuario, direccionIP, puertoIP);
    Rechazado=VERDAD;
    pthread_mutex_unlock(&mutexRespuestas);
    exit(0);
    }else if(p) {
    printf("No se encontro la respuesta\n");
    } else {
    printf("No se encontro la trama\n");
    }
  pthread_mutex_unlock(&mutexRespuestas);
};

int existeUsuario(unsigned char *nombre)
    /* Función para saber si existe un usuario en la BD
    El parametro:
    nombre: usuario que se verificara.

    Devuelve:
    0 cuando el usuario no existe.
    1 cuando el usuario existe.
    */
{
  FILE *comprobacion;
  char *nombrearchivo;
  int tam;

  tam = strlen(nombre);
  tam += 5;
```

```
  nombrearchivo = (unsigned char *) malloc(sizeof(unsigned char) * tam);
  strcpy(nombrearchivo, nombre);
  strcat(nombrearchivo, ".ser");

  if ((comprobacion = fopen(nombrearchivo, "r")) == NULL)
    {
      free(nombrearchivo);
      return FALSO;
    };

  fclose(comprobacion);
  free(nombrearchivo);
  return VERDAD;
};

int administraUsuario(struct sockaddr_in direccion, TIdentificacion nombre)
     /*
       Función para establecer las conexiones a los usuarios y administrar la
       conexion que se ha establecido con el usuario.

       Parametros:
       direccion: Indica la dirección y el puerto que utilizaremos
       para conectarnos a la computadora cliente
       nombre: para saber el nombre del archivo.

       Devuelve:
       Un valor 0 (cero) si hubo problemas durante la conexión.
     */
{
  fd_set leerfd;
  int fd, numbytes;
  unsigned char *temporal, *nombrearchivo, buffer[255], *retoAHacer, \
    *llave, *clave;
  unsigned char *respuestaEsperada;
  unsigned int tiempo;
  int blocksize, tam, numRetos, retoEscogido;
  TReto retoAEnviar;
  TRespuesta respuesta;
```

```
convNumCad numUnico;
datos_hilo_pregunta argsPreguntas;
datos_hilo_respuesta argsRespuesta;

pthread_t administraPreguntas, administraRespuestas, administraReenvios;

nombrearchivo = NULL;
retoAHacer = NULL;

tam = strlen(nombre.usuario);
tam += 5;

nombrearchivo = (unsigned char *) calloc(tam, sizeof(unsigned char));

strcpy(nombrearchivo, nombre.usuario);
strcat(nombrearchivo, ".ser");

// Leemos la llave que se utiliza para este usuario.
leerValor(nombrearchivo, "[Usuario]", "clave", &llave);
memset(clave, 0, 16);
strncpy(clave, llave, strlen(llave));

// Liberamos la memoria de la llave
free(llave);

// Contamos los retos que hay para este usuario
numRetos = contarValores(nombrearchivo, "[Reto]");

puertoIP = ntohs(direccion.sin_port);
strcpy(direccionIP, (char *)inet_ntoa(direccion.sin_addr));
fd = socket(AF_INET, SOCK_STREAM, 0);

//Desciframos el número único que nos envio el cliente
descifrar(&(nombre.datoscifrados[0]), clave, 16);
strncpy(numUnico.cadena,nombre.numUnico, 4);

if (connect(fd, (struct sockaddr *)&direccion, sizeof(direccion)) == -1) {
  perror("connect");
  return 1;
```

```
};

// Escogemos una de las preguntas de manera aleatoria
srand(time(NULL));
retoEscogido = 1 + (int) ((float)numRetos * rand() / (RAND_MAX+1.0));

// Pasos necesarios para realizar la pregunta inicial al usuario
// Inicializamos la estructura del reto a cero para evitar caracteres
// no contemplados
memset(retoAEnviar.cadena, '\0', sizeof(TReto));

// Inicializamos el formato de nuestra trama, con nuestros codigos
retoAEnviar.identificacion = SEGURIDAD;
retoAEnviar.subidentificacion = RETO;
strncpy(retoAEnviar.numUnicoA, nombre.numUnico, 4);

// Leemos una de las preguntas propias para este usuario, y
// lo asignamos al reto
leerClave(nombrearchivo, "[Reto]", &retoAHacer, retoEscogido);
leerValor(nombrearchivo, "[Reto]", retoAHacer, &respuestaEsperada);
strncpy(retoAEnviar.reto, retoAHacer, strlen(retoAHacer));

// Le asignamos un número único para comprobar que la respuesta
//recibida es reciente
numUnico.numero = (unsigned long)(4294967295.0 * rand() / RAND_MAX+1.0);

strncpy(retoAEnviar.numUnicoS, numUnico.cadena, sizeof(numUnico.cadena));

// Asignamos el formato de la trama, en este caso siempre será 51
// ya que es una trama fija, 32 bytes del código hash, 4 del número
// único enviado por el cliente, 4 del número único del servidor
// y 8 del relleno para poder cifrar los números únicos.
retoAEnviar.tam = 51;

printf("Número único S %u\nReto Hecho %.32s\n", \
   numUnico.numero, retoAEnviar.reto);
printf("Respuesta esperada %.32s\n", respuestaEsperada);
printf("Para el usuario %s en la dirección %s:%d\n", nombre.usuario, \
   direccionIP, puertoIP);
```

```
cifrar(&(retoAEnviar.datoscifrados[0]), clave, 16);

//Enviamos el reto, no hacemos ninguna comprobación de error por el momento
send(fd, retoAEnviar.cadena, sizeof(retoAEnviar), 0);

memset(buffer,0,255);

// Se espera una respuesta
numbytes = recv(fd, buffer, 255,0);
if (numbytes <= 0)
 {
   if (numbytes == 0) {
  printf("Socket %d cerrado\n",fd);
  exit(0);
   } else {
  perror("recv");
  exit(1);
   };
 } else {
   if ((buffer[0] == SEGURIDAD) && (buffer[1] == RESPUESTA))
  {
    if (numbytes == (buffer[2]) + 3)
     {
       temporal = (unsigned char *) malloc(sizeof(unsigned char) * buffer[2]);
       printf("\n");
       memset(temporal,0,sizeof(temporal));
       bcopy((void *)&(buffer[3]), (void *)temporal, buffer[2]);
       descifrar(temporal, clave, buffer[2]);
       strncpy(respuesta.cadena, temporal, buffer[2]);

       // Liberamos la memoria temporal
       free(temporal);

       //Debemos de comparar contra lo que esperamos
       if (memcmp(respuesta.numUnicoS, numUnico.cadena, sizeof(long)) == 0
        && (strncmp(respuesta.respuesta, respuestaEsperada, \
                strlen(respuestaEsperada)) == 0))
      {
```

```
        printf("Usuario %s contestó correctamente aceptado en %s:%d\n", \
           nombre.usuario, direccionIP, puertoIP);
      } else
        {
          printf("Usuario %s incorrecto en %s:%d\n", nombre.usuario, direccionIP, \
             puertoIP);
          free(retoAHacer);
          free(respuestaEsperada);
          exit(1);
        };
      };
  } else
    //Llego algo no contemplado dentro del protocolo, cerramos la comunicación
    {
      exit(0);
    };
 };

free(retoAHacer);
free(respuestaEsperada);

//Como el usuario fue válido se indica el socket que se utilizara
//para establecer la comunicación con el usuario.
Rechazado=FALSO;
argsPreguntas.sck = fd;
strcpy(argsPreguntas.nombre, nombre.usuario);
//Se crea un hilo de ejecución para la función administraPreguntas.
if (pthread_create(&administraPreguntas, NULL, realizarPreguntas, \
          (void *) &argsPreguntas) != 0)
  printf("Error al ejecutar la función administraPreguntas");
if (pthread_create(&administraReenvios, NULL, reenviarPreguntas,
          (void *) &argsPreguntas) != 0)
  printf("Error al administrar los reenvíos\n");

strcpy(gUsuario, nombre.usuario);
retosHechos = NULL;

FD_ZERO(&leerfd);
FD_SET(fd, &leerfd);
```

```
//Se mantiene un ciclo infinito durante la duración de la
//comunicación con el servidor.
numbytes=1;
while(1)
 {
   //Aquí va el select, para evitar un socket bloqueante
   if (select(fd+1, &leerfd, NULL, NULL, NULL) == -1)
  {
    perror("select");
    break;
  }

   //Se administra el socket de conexión.
   if (FD_ISSET(fd, &leerfd)) {
  if ((numbytes = recv(fd, buffer, 255, 0)) <= 0)
   {
     if (numbytes == 0) {
       printf("Usuario %s termino la comunicación desde %s:%d\n", \
          nombre.usuario, direccionIP, puertoIP);
       Rechazado=VERDAD;
       pthread_mutex_lock(&mutexRespuestas);
       liberarMemoriaRetos();
       pthread_mutex_unlock(&mutexRespuestas);
       pthread_mutex_destroy(&mutexRespuestas);
     } else {
       perror("recv");
     };
     break;
   } else
     {
       //Llego un mensaje
       if (buffer[0] == SEGURIDAD)
      {
        if (buffer[1] == RESPUESTA)
          // Si no llega una respuesta hubo un error en el mensaje
          {
            temporal = (unsigned char *) malloc(sizeof(unsigned char) * buffer[2]);
            memset(temporal,0,sizeof(temporal));
```

```
                bcopy((void *)&(buffer[3]), (void *)temporal, buffer[2]);
                descifrar(temporal, clave, buffer[2]);
                strncpy(respuesta.cadena, temporal, buffer[2]);
                free(temporal);
                strncpy(argsRespuesta.cadena, respuesta.cadena,
sizeof(argsRespuesta.cadena));
                strncpy(argsRespuesta.llave, clave, 16);
                revisarRespuestas((void *)&argsRespuesta);
              };
          };
         };
      };
    };
  Rechazado=VERDAD;
  printf("\nSe cerró la comunicación con %s\n", nombre.usuario);
  close(fd);
  pthread_join(administraPreguntas, NULL);
  pthread_join(administraReenvios, NULL);
  exit(0);
};

int main(int argc, char *argv[])
    /*
      Función de inicio al sistema servidor.
      Aquí se espera la comunicación que inicia el cliente
      utilizando el protocolo UDP.
    */
{
  struct sockaddr_in my_addr;
  struct sockaddr_in their_addr;
  TIdentificacion identificacion;
  int addr_len, numbytes;
  unsigned char buf[255];
  unsigned char *usuario;
  int puerto;
  int escucha;

  int error, conectado, tam;
  signal(SIGCHLD, matar);
```

```
puerto = PUERTO;
printf("Puerto %d\n",puerto);
if ((escucha = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
  perror("socket");
  exit(1);
}

my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(puerto);
my_addr.sin_addr.s_addr = INADDR_ANY;

memset(&(my_addr.sin_zero), '\0', 8);

if (bind(escucha, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1)
  {
    perror("bind");
    exit(1);
  }

while (1)
  {
    memset(buf,'\0',255);
    addr_len = sizeof(struct sockaddr);
    if ((numbytes = recvfrom(escucha, buf, 255, 0, \
                  (struct sockaddr *)&their_addr, &addr_len)) == -1)
   {
     perror("recvfrom");
     exit(1);
   }

    //El único paquete que puede venir en esta parte es de SEGURIDAD
    //y subcampo IDENTIFICACION por lo que comprobamos eso
    if ((buf[0] == SEGURIDAD) && (buf[1] == IDENTIFICACION))
   {
     int hijo, retorno;

     //Se copia el mensaje recibido en la variable de identificación.
     tam = (buf[2] > 31? 31: buf[2]);
```

```
      bcopy((void *)&buf[3],(void *)identificacion.cadena, tam);
      identificacion.cadena[tam]='\0';

      //Si fue, lo agregamos a la lista
      if (!existeUsuario(identificacion.usuario))
       {
         //Como este usuario no existe en la BD de datos no
         //realizamos la conexion.
         printf("El usuario %s no existe\n", identificacion.usuario);
         continue;
       };

      printf("\n\nConexión de %s recibida de %s:%d\n", identificacion.usuario, \
          inet_ntoa(their_addr.sin_addr), ntohs(their_addr.sin_port));
      //lo enviamos a un proceso hijo, para que realice los detalles relacionados
      //con esta conexion
      if ((hijo = fork()) == 0)
       {
         //Si el usuario es válido se crea un proceso hijo para que
         //administre la comunicación. El envío de las preguntas y
         //la recepción de las respuestas.
         administraUsuario(their_addr, identificacion);
       } else if (hijo < 0)
         {
         perror("Error en fork");
         };
     };
      //Cualquier otro paquete que venga no pasa nada, solo se borra el buffer
      memset(buf, '\0',255);
    };
  return 0;
};
```

ANEXO D. CODIGO FUENTE DEL CLIENTE

```
unit UClienteTesis;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, WSocket, WSocketS, ExtCtrls, uFunciones, IniFiles;

type
    TTcpCliente = Class(TWSocketClient)
    public
        RecvBuffer : Array [0..255] of byte;
    end;

  TfrmCliente = class(TForm)
     lblUsuario: TLabel;
     lblClave: TLabel;
     txtUsuario: TEdit;
     txtClave: TEdit;
     btnConectar: TButton;
     btnDesconectar: TButton;
     txtPuertoLocal: TEdit;
     lblPuertoLocal: TLabel;
     mmoMensajes: TMemo;
     tmrTiempo: TTimer;
     WssServidor: TWSocketServer;
     WsUDP: TWSocket;
     procedure btnConectarClick(Sender: TObject);
```

```
    procedure txtUsuarioExit(Sender: TObject);
    procedure WssServidorClientConnect(Sender: TObject;
      Client: TWSocketClient; Error: Word);
    procedure ClienteDataAvailable(Sender: TObject; Error: Word);
    procedure WssServidorClientDisconnect(Sender: TObject;
      Client: TWSocketClient; Error: Word);
    procedure btnDesconectarClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
    procedure MostrarInformacionServidor(Reto:TMensajeRecibido);
    procedure MostrarInformacionCliente(Respuesta:TRespuesta);
  public
    { Public declarations }
  end;

var
  frmCliente: TfrmCliente;
  gStrPuerto : String;
  NumeroUnicoA: Cardinal;
  PVez: Boolean;

implementation

{$R *.dfm}

procedure TfrmCliente.btnConectarClick(Sender: TObject);
var
    usuario, archivoe, archivos: String;
    buffer:Array[0..31] of Char;
    Ini: TIniFile;
    tam: Integer;
  IdentificacionCliente: TIdentificacion;
  Llave: String;
begin
    Randomize;
    repeat
        gstrPuerto := IntToStr(Random(60000) + 1025);
        WssServidor.Addr:='0.0.0.0';
```

```
        WssServidor.Port:=gstrPuerto;
        wssServidor.Proto:='tcp';
        WssServidor.ClientClass:=TTcpCliente;
        WssServidor.Listen;
    until wssServidor.State= WSocket.wsListening;
    WssServidor.Close;
    wsUDP.LocalPort := gStrPuerto;
    wsUDP.Proto := 'udp';
    wsUDP.LocalAddr := '0.0.0.0';
    txtPuertoLocal.Text:=gStrPuerto;
    //Obtenemos los datos del servidor.
    if (Length(txtUsuario.Text)=0) or (Length(txtClave.Text)=0) Then
    begin
        ShowMessage('Falta el usuario o la contraseña');
        exit;
    end;
    mmoMensajes.Clear;
if (Length(txtUsuario.Text)>12) then
begin
  ShowMessage('El usuario no puede tener más de 12 carácteres');
  exit;
end;
    archivoe:=txtUsuario.Text+'.aes';
    archivos:=txtUsuario.Text+'.cli';
    Descifrar(archivoe, archivos, txtClave.Text);
    Ini:=TIniFile.Create(ExtractFilePath(Application.ExeName)+archivos);
    try
        Usuario:=Ini.ReadString('Usuario','Nombre','Desconocido');
   Llave:=Ini.ReadString('Usuario','Clave','Estandar');
        if strLComp(PChar(usuario),Pchar(txtUsuario.Text),
            Length(txtUsuario.Text))=0 Then
        begin
     PVez := True;
            wsUDP.Addr:=Ini.ReadString('Usuario','servidor','localhost');
            wsUDP.Port:=Ini.ReadString('Usuario','Puerto','56789');
    FillChar(IdentificacionCliente.Cadena,
      sizeof(IdentificacionCliente.Cadena), 0);
    FillChar(Buffer,sizeof(Buffer),0);
            Move(Pchar(txtUsuario.Text)^, IdentificacionCliente.Usuario,
```

```
    Length(txtUSuario.Text));
    IdentificacionCliente.NumUnicoA:=Random(2147483647);
    NumeroUnicoA := IdentificacionCliente.NumUnicoA;
    CifrarBuffer(IdentificacionCliente.DatosCifrados, Llave,
      sizeof(IdentificacionCliente.DatosCifrados));
          tam := 3 + Length(txtUsuario.Text);
          buffer[0] := Chr(SEGURIDAD);
          buffer[1] := Chr(IDENTIFICACION);
    tam := tam + 16;
          buffer[2] := Chr(tam);
    Move(IdentificacionCliente.Cadena, buffer[3],tam-3);
          //Se asignan los valores al socket que estara en escucha.
          wssServidor.Port := gStrPuerto;
          wssServidor.Addr := '0.0.0.0';
          wssServidor.Proto:='tcp';
          wssServidor.Listen;
          wsUDP.Connect;
          wsUDP.Send(Addr(buffer), tam);
          wsUDP.Close;
       end
       else
          ShowMessage('Hubo un error al realizar la conexión');
    Finally
       Ini.Free;
    end;
end;

procedure TfrmCliente.txtUsuarioExit(Sender: TObject);
begin
    txtUsuario.Text:=Trim(txtUsuario.Text);
end;

procedure TfrmCliente.WssServidorClientConnect(Sender: TObject;
  Client: TWSocketClient; Error: Word);
begin
    With TTcpCliente(Client) do
    begin
       mmoMensajes.Lines.Add('Se establecio la conexión');
       OnDataAvailable:=ClienteDataAvailable;
```

```
        Banner := '';
        LineMode:=FALSE;
        LineEdit:=FALSE;
    end;
    btnConectar.Enabled:=FALSE;
    btnDesconectar.Enabled:=TRUE;
    txtUsuario.ReadOnly:=TRUE;
    txtClave.ReadOnly:=TRUE;
    tmrTiempo.Enabled:=FALSE;
end;

procedure TfrmCliente.ClienteDataAvailable(Sender: TObject;
  Error: Word);
var
    tam, rec: Integer;
    buffer: Array[0..255] of byte;
    bufferAux: Array[0..100] of char;
    miReto: TMensajeRecibido;
    miRespuesta: TRespuesta;
    nombre: String;
    Ini: TIniFile;
    Llave, queResponder : String;
begin
    with TTcpCliente(Sender) do
    begin
        fillchar(recvBuffer, sizeof(RecvBuffer),0);
        Rec := Receive(@recvBuffer,sizeof(recvBuffer));
    end;
    if Rec <= 0 Then
        exit;
    //Se espera que la trama recibida sea un reto.
    if (TTcpCliente(Sender).recvBuffer[0]=SEGURIDAD) and
        (TTcpCliente(Sender).recvBuffer[1]=RETO) then
    begin
        tam := TTcpCliente(Sender).recvBuffer[2];
        if tam=51 then   //Es una trama fija.
        begin
            with TTcpCliente(Sender) do
                CopiarnBytes(miReto.Cadena, recvBuffer, 3,
```

```
            sizeof(miReto.Cadena));
    //Se lee el archivo INI para saber que contestar
    nombre := ExtractFilePath(Application.ExeName)+txtUsuario.Text+'.cli';
    Ini:=TIniFile.Create(nombre);
    try
        fillchar(bufferAux,sizeof(bufferAux),0);
        Bytes2Char(bufferAux, miReto.valor, sizeof(miReto.valor));
        queResponder:=Ini.ReadString('Reto',bufferAux, 'ninguno');
        if queResponder<>'ninguno' then
        begin
            llave := Ini.ReadString('Usuario','clave','estandar');
DescifrarBuffer(miReto.DatosCifrados, llave,
 sizeof(miReto.DatosCifrados));
MostrarInformacionServidor(miReto);
if Pvez Then
begin
  if miReto.NumUnicoA <> NumeroUnicoA then
  begin
    ShowMessage('La respuesta del servidor no es válida');
    btnDesconectar.Click;
    exit;
  end;
  PVez := False;
end;
            fillchar(bufferAux,sizeof(bufferAux),0);
            strLCopy(bufferAux,PChar(queResponder),sizeof(bufferAux));
            Char2Bytes(miRespuesta.valor, bufferAux,
                sizeof(miRespuesta.valor));
miRespuesta.NumUnicoS:=miReto.NumUnicoS;
            MostrarInformacionCliente(miRespuesta);
            Fillchar(miRespuesta.relleno, sizeof(miRespuesta.relleno),0);
            CifrarBuffer(miRespuesta.Cadena,llave,
                sizeof(miRespuesta.Cadena));
            buffer[0]:=SEGURIDAD;
            buffer[1]:=RESPUESTA;
            buffer[2]:=sizeof(miRespuesta.Cadena);
            Move(miRespuesta.Cadena, buffer[3],sizeof(miRespuesta.Cadena));
            TTcpCliente(Sender).Send(@buffer,buffer[2]+3);
        end
```

```
                else
                begin
                    with mmoMensajes.Lines do
                    begin
                        add('******************************************************');
                        add('Mensajes internos (no se envian al servidor)');
                        add('No se encontro una respuesta a la pregunta hecha');
                        add('******************************************************');
                    end;
                end;
            finally
                Ini.Free;
            end;
        end;
    end;
end;

procedure TfrmCliente.WssServidorClientDisconnect(Sender: TObject;
  Client: TWSocketClient; Error: Word);
var
    nombre : String;
    archivo : File;
begin
    mmoMensajes.Lines.Add('Se cerró la conexión con el servidor');
    nombre := ExtractFilePath(Application.ExeName) + txtUsuario.Text + '.cli';
    AssignFile(archivo,nombre);
    try
        Erase(archivo);
    except
        on EInOutError do nombre:='A';
    end;
    btnDesconectar.Enabled:=FALSE;
    btnConectar.Enabled:=TRUE;
    wssServidor.Close;
    txtUsuario.ReadOnly:=FALSE;
    txtClave.ReadOnly:=FALSE;
end;

procedure TfrmCliente.btnDesconectarClick(Sender: TObject);
```

```
var
    nombre: String;
    archivo: File;
    i : Integer;
begin
    nombre := ExtractFilePath(Application.ExeName) + txtUsuario.Text + '.cli';
    AssignFile(archivo,nombre);
    try
        Erase(archivo);
    except
        on EInOutError do nombre:='A';
    end;
    btnDesconectar.Enabled:=FALSE;
    btnConectar.Enabled:=TRUE;
    for i:= 0 to wssServidor.ClientCount-1 do
    begin
        wssServidor.Client[i].Close;
    end;

    wssServidor.Close;
    txtUsuario.ReadOnly:=FALSE;
    txtClave.ReadOnly:=FALSE;
end;

procedure TFrmCliente.MostrarInformacionServidor(Reto: TMensajeRecibido);
var
    valor: Array[0..60] of char;
begin
  fillchar(valor,sizeof(valor),0);
    Bytes2Char(valor,Reto.valor,sizeof(Reto.Valor));
    with mmoMensajes.Lines do
    begin
        add('*************** El servidor envia al cliente ***************');
        add('Id del mensaje:    SEGURIDAD');
        add('SubId del mensaje: PREGUNTA');
        add('Id de la trama:    ' + IntToStr(Reto.NumUnicoS));
        add('Valor respuesta:   ' + valor);
        add('************************************************************');
        add(' ');
```

```
    end;
end;

procedure TfrmCliente.MostrarInformacionCliente(Respuesta: TRespuesta);
var
    valor: Array[0..60] of Char;
begin
  fillchar(valor, sizeof(valor),0);
    Bytes2Char(valor,respuesta.valor,sizeof(Respuesta.valor));
    with mmoMensajes.Lines do
    begin
        add('*************** El cliente envia al servidor ***************');
        add('Id del mensaje:    SEGURIDAD');
        add('SubId del mensaje: RESPUESTA');
        add('Id de la trama:    ' + IntToStr(Respuesta.NumUnicoS));
        add('Valor pregunta:    ' + valor);
        add('*****************************************************************');
        add(' ');
    end;
end;
procedure TfrmCliente.FormClose(Sender: TObject; var Action: TCloseAction);
var
    nombre: String;
    archivo: File;
begin
    nombre := ExtractFilePath(Application.ExeName) + txtUsuario.Text + '.cli';
    AssignFile(archivo,nombre);
    try
        Erase(archivo);
    except
        on EInOutError do nombre:='A';
    end;
    if wssServidor.State = WSocket.wsListening Then
        wssServidor.Close;
end;

end.
```

ANEXO E. LÓGICA BAN

Los elementos que componen la lógica BAN fueron tomados del trabajo de Burrows, Abadi y Needham [4] por lo que si se desean más detalles de esta lógica se puede consultar ese escrito.

E.1 ELEMENTOS BÁSICOS DE LA LÓGICA

La única asociación permitida es la conjunción, que es denotada por una coma. Los elementos para definir sentencias lógicas para analizarlas son:

$P \equiv X$ P cree en X. Quiere decir que el principal *P*, actúa como si *X* fuera verdad.

$P \triangleleft X$ Alguien envío un mensaje conteniendo *X* a *P*, que puede ser leído y repetido (puede ser después de un descifrado).

$P \mid\sim X$ En algún momento, el principal *P* envía un mensaje que incluye la sentencia X. No se sabe el momento cuando el mensaje fue enviado, o incluso, si fue enviado durante la corrida actual del protocolo. Solo se sabe que *P* confiaba en *X* cuando lo envío.

$P \Rightarrow X$ El principal *P* es una autoridad sobre *X* y debe ser confiable en esta materia. Esta sentencia es usada principalmente cuando un principal ha delegado autoridad sobre alguna sentencia *X*.

$\sharp(X)$ La fórmula *X* es *actual*. Quiere decir, que *X* no ha sido enviado en ningún momento anterior a la corrida actual del protocolo. Esto es definido para ser verdad por *números únicos*, que son, expresiones generadas para el propósito de ser actuales.

$P \overset{K}{\leftrightarrow} Q$ *P* y *Q* están usando la llave compartida *K* para comunicarse. La llave *K* es buena, en el sentido que nunca será descubierta por ningún principal, excepto *P* o *Q*, o un principal en el que confíen ya sea *P* o *Q*.

$\overset{K}{\mapsto} P$ *P* tiene una llave pública *K*. La llave secreta que es pareja de esta llave K^{-1} que nunca será descubierta por ningún principal diferente a *P*, o alguno en el que *P* confíe.

$P \overset{X}{\leftrightarrow} Q$ La fórmula *X* es un secreto conocido solo a *P* y *Q* y aquellos principales a quienes ellos revelen el secreto.

$\{X\}_K \ de\ P$ Representa la fórmula *X* cifrada bajo la llave *K* del principal *P*. La parte *de P* es omitida comúnmente, y se asume que cada principal es capaz de reconocer e ignorar sus propios mensajes.

Los postulados lógicas son formados a partir de estas sentencias básicas. Un protocolo de seguridad es idealizado, de acuerdo a las reglas definidas por los autores, en términos de estos postulados. Cada protocolo debe ser idealizado antes de aplicar la lógica BAN.

E.2 REGLAS DE INFERENCIA

En el estudio de la autenticación, hay que hacer una distinción entre dos tiempos: *pasado* y *presente*. El tiempo presente se cuenta a partir de una ejecución particular del protocolo que se desee analizar. Todos los mensajes enviados antes de esta ejecución son considerados en el pasado, y el protocolo de autenticación debe de ser cuidadoso para prevenir que cualquier mensaje enviado con anterioridad sea aceptado como un mensaje reciente. Todas las suposiciones se basan en el *presente* y son válidos para la duración de la corrida del protocolo; se asume que cuando un principal *P* dice *X* entonces él actualmente cree en *X*.

Un mensaje cifrado es representado como una sentencia lógica cifrada con una llave. Se asume que el cifrado es realizado de tal manera que sabemos que el mensaje completo fue enviado a la vez. Si se reciben dos secciones cifradas distintas en un mismo mensaje, lo tratamos como si se hubieran recibido en distintos mensajes. Un mensaje no puede ser *entendido* por un principal que no conozca la llave, y la llave no puede ser deducida a partir del mensaje cifrado. Cada mensaje contiene suficiente redundancia para permitir al principal que descifra la información verificar que ha usado la llave correcta. Hay que hacer notar, que los mensajes contienen suficiente información para que un principal detecte (e ignore) sus propios mensajes.

Postulados lógicos

La primera regla que se revisará es el *significado de los mensajes* que trata sobre la interpretación de los mensajes. Existen tres reglas en este postulado, de los cuáles dos son para la interpretación de los mensajes cifrados, y el tercero corresponde a la interpretación de mensajes con secretos. Se explica como derivar otros postulados a partir del origen de los mensajes.
Para llaves compartidas, se postula:

$$\frac{P \models Q \overset{K}{\leftrightarrow} P,\ P \triangleleft \{X\}_K}{P \models Q \mid\sim X}$$

Quiere decir, si *P* cree que comparte una llave *K* con *Q* y ve un mensaje *K* cifrado con la llave *X*, entonces *P* cree que *Q* una vez dijo *X*. Para esta regla sea válida, se debe de garantizar que *P* no se envío *X* a si mismo.

De igual manera, para la utilización de llaves públicas se postula:

$$\frac{P \models \overset{Q}{\mapsto} K, P \triangleleft \{X\}_{K^{-1}}}{P \models Q \mid\sim X}$$

Otro punto de vista para cifrar es mediante secretos compartidos, que se postula de la siguiente forma:

$$\frac{P \models P \overset{Y}{\leftrightarrow} Q, P \triangleleft \langle X \rangle_Y}{P \models Q \mid\sim X}$$

Esto es, si *P* cree que el secreto *Y* es compartido con *Q* y ve $\langle X \rangle_Y$, entonces *P* cree que *Q* una vez dijo *X*. Este regla es posible porque la regla $\triangleleft$, dada antes, garantizamos que $\langle X \rangle_Y$ no fue enviada por *P* a si mismo.

La regla verificación de *números únicos* expresa que un mensaje es reciente, y que el emisor todavía confía en lo que dijo:

$$\frac{P \models \sharp(X), P \models Q \mid\sim X}{P \models Q \models X}$$

Lo anterior se entiende como, si *P* cree que *X* fue enviado recientemente y que *Q* una vez dijo *X*, entonces *P* cree que *Q* cree en *X*. *X* debe estar en texto claro.

La siguiente regla es la de *jurisdicción* establece que si *P* cree que *Q* tiene jurisdicción sobre X entonces *P* confía en que *Q* confía en *X*:

$$\frac{P \models Q \Rightarrow X, P \models Q \models X}{P \models X}$$

Una propiedad necesaria de los operadores lógicos es que *P* cree que en un conjunto de sentencias, si y solo si, *P* cree en cada sentencia individual de manera separada. De esta manera se justifican las siguientes reglas:

$$\frac{P \models X, P \models Y}{P \models (X,Y)} \quad \frac{P \models (X,Y)}{P \models X} \quad \frac{P \models Q \models (X,Y)}{P \models Q \models X}$$

Una regla similar se puede aplicar de la siguiente manera:

$$\frac{P \models Q \mid\sim (X, Y)}{P \models Q \mid\sim X}$$

Hay que notar, que si $P \models Q \mid\sim X$ y $P \models Q \mid\sim Y$ no se puede derivar en que $P \models Q \mid\sim (X, Y)$, ya que se estaría mencionando que las dos partes fueron enviadas al mismo tiempo y no tenemos la seguridad de que así sea.

Otro postulado útil es que si una parte de la fórmula se conoce que es *actual*, entonces la fórmula en su totalidad también debe ser *actual*:

$$\frac{P \models \sharp(X)}{P \models \sharp(X, Y)}$$

Un elemento dentro de los protocolos de seguridad es los retos (o preguntas) que son realizados a quien deseamos autenticar, los retos siempre se consideran *actuales*, y no es necesario que se envíen los mensajes de forma cifrada, pero la respuesta correspondiente a ese reto si debe de estar cifrado.

Existen varios postulados que son utilizados en sentencias más complejas, pero no se mencionan en la tesis debido a que no son elementos necesarios para realizar la validación de los protocolos propuestos.