UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO

DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

"Autenticación entre componentes en un prototipo de aplicación empresarial distribuida".

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A

L.C.C. Johanna Gema Espinosa Lazo.

Asesores

*M en C. Hugo Moncayo López, Dr. Héctor Benítez Pérez *,*
Dr. Ricardo López Bautista.

* Asesor Externo, Departamento de Investigación en Sistemas Computacionales y Automatización del Instituto de Investigaciones Matemáticas Aplicadas y en Sistemas de la Universidad Nacional Autónoma de México.

México, D. F. Noviembre 2004

Esta tesis corresponde a los estudios realizados con una beca otorgada por la Secretaría de Relaciones Exteriores del Gobierno de México.

DEDICATORIA

A quienes creyeron en mí, su confianza me ayudó a perseverar.
A quienes no creyeron en mí, me motivaron a buscar la fortaleza interna.

AGRADECIMIENTOS

Agradezco especialmente al pueblo de México que a través de la beca otorgada por la Secretaría de Relaciones Exteriores de México me permitió realizar mis estudios de postgrado en este hermoso país.

Al M en C. Hugo Moncayo López quien depositó su confianza en mi persona y me apoyó en uno de los momentos más difíciles de mi maestría y aceptó ser mi asesor y dirigir ésta tesis.

Al Dr. Héctor Benítez Pérez por haber aceptado compartir esta aventura como mi asesor, por su apoyo, orientación y sobretodo su paciencia durante el tiempo que duró esta investigación.

Al Dr. Ricardo López Bautista por su entusiasmo con el tema y su aportación oportuna en lo referente a la criptografía y sus bases matemáticas.

A la Dra. Ana Lilia Laureano Cruces quien desde su posición de coordinadora de la Maestría en Ciencias de la Computación me apoyó para que concluyera exitosamente mis estudios.

Al Dr. Felipe Monroy Pérez y la Dra. Marisela Gómez Guzmán por haberme proporcionado el toque cálido a mi estadía en México.

A mi familia por animarme en todo momento a seguir adelante.

A mis amigos por el cariño y apoyo que siempre me han brindado.

RESUMEN

Las aplicaciones empresariales basadas en Web requieren que el midleware implente los mecanismos de seguridad necesarios para satisfacer los requerimientos de: confidencialidad, autenticación, integridad y no repudio. Estos mecanismos, debido a que se incorporan de manera vertical, generan una sobrecarga en la funcionalidad del midleware. En esta tesis se propone incorporar un mecanismo de autenticación entre componentes a nivel de la aplicación, incorporándolo en el midleware en forma horizontal y que garantice un canal seguro de comunicación entre los componentes distribuidos del sistema.

El mecanismo de autenticación aquí propuesto se basa en un esquema híbrido que combina las ventajas del algoritmo simétrico RC2 y el algoritmo asimétrico RSA. El funcionamiento del esquema se prueba por medio de la implementación de un prototipo para el caso de estudio basado en el modelo aerodinámico de un avión de dos plazas denominado Admire.

ABSTRACT

WEB Business applications require a middleware that implements security mechanisms to satisfy the requirements of confidentiality, authentication, integrity and non-repudiation. The middleware functionality is usually overloaded since these mechanisms are vertically implemented. In this project work, an authentication mechanism for components is proposed, which works at the application layer and it is horizontally incorporated into the middleware to provide a security communication channel between the components of the distributed system.

The authentication mechanism proposed in this work is based on a hybrid scheme, combining the advantages of the symmetric RC2 and asymmetric RSA algorithms. The functionality of this approach is tested in a prototype for an aerodynamic model of an airplane, called Admire.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

I. INTRODUCCIÓN

Uno de los principales problemas que enfrentan los desarrolladores de software es el análisis de requerimientos de los sistemas empresariales que cada vez son más grandes y complejos debido a la diversidad de los servicios que ofrecen y a los volúmenes de transacciones que manejan. A esto se debe sumar que las grandes organizaciones tienen sus recursos distribuidos geográficamente y el acceso a sus sistemas se realiza por usuarios dispersos en diferentes puntos. Por esta razón, los sistemas deben diseñarse para satisfacer los requerimientos de transparencia, confiabilidad, desempeño, escalabilidad y seguridad requeridos parar operar adecuadamente en un ambiente hostil como el de la Web.

La búsqueda de soluciones ha colocado a la arquitectura de software como la disciplina que en los últimos años ha permitido crear modelos para las estructuras de los sistemas de gran escala. Un diseño de arquitectura se visualiza como una composición de módulos o componentes que interactúan entre si. El sistema tiende a ser dividido en partes pequeñas y la unión de estas satisface los requerimientos del sistema.

Los requerimientos del sistema se dividen en funcionales y no funcionales; los requerimientos funcionales describen al sistema en términos de entradas, salidas y procesamiento de la información, mientras que los no funcionales, describen las cualidades deseables del sistema como tiempo de respuesta, escalabilidad, robustez y seguridad. Al diseñar la arquitectura de un sistema se busca cumplir con los requerimientos no funcionales, esto conlleva a que el arquitecto de software tenga que decidir la organización del sistema; identificar los elementos estructurales e interfaces que componen el sistema y su comportamiento definido en las colaboraciones; analizar la composición de elementos estructurales y de su comportamiento en subsistemas más grandes; modelar el estilo arquitectónico que guía a la organización; además debe considerar los principios de usabilidad, funcionalidad, desempeño, reutilización, comprensibilidad, estética y limitaciones tecnológicas o económicas.

En este proyecto se trabaja el requerimiento de seguridad de aplicaciones empresariales, específicamente el problema de autenticación entre componentes distribuidos basados en una arquitectura Web. Es necesario hacer mención que la hipótesis de este proyecto es el poder garantizar la seguridad en cualquier relación de comunicación entre componentes.

El estudio se inicia con una arquitectura de referencia para aplicaciones Web, a continuación se analiza el mecanismo de seguridad y la arquitectura de seguridad multi-capa lo que permite ubicar el punto donde se realiza el proceso de autenticación y su incidencia en los demás procesos de seguridad. Una vez alcanzada esta etapa inicia el estudio de los conceptos de que permitan elaborar el modelo de autenticación expresando la conformación de componentes y la constitución de los conectores personalizados que permitan que los componentes realicen el proceso de reconocimiento de unos con otros en aplicaciones específicas de un dominio común. Posteriormente se realiza la implementación el modelo de autenticación en el prototipo del caso de estudio Admire y finalmente se prueba el funcionamiento del prototipo.

I.1. Motivación.

En los últimos años se han realizado todo tipo de trabajos teóricos sobre la seguridad en las aplicaciones computacionales y la criptografía, pero aun se implementan muchos sistemas de forma insegura. Es importante que cualquier solución de seguridad contemple los riesgos internos y externos de la organización. En un sistema distribuido muchas de sus partes no están sólo afuera del centro de protección de datos sino que a menudo están completamente afuera de las fronteras del negocio en términos de seguridad.

La mayor parte de los elementos técnicos requeridos para diseñar un sistema distribuido ya han sido definidos. Sin embargo, la seguridad no es fácilmente dirigida en forma incremental, hay muy pocos resultados en estándares, arquitecturas abiertas e ínter operación de productos que permitan una solución completa. [HREF 1]

Antes de implementar soluciones técnicas se deben resolver dos áreas del problema de administración de la seguridad. Una es en la etapa de diseño e implementación y la otra es durante la operación.

En este trabajo se realiza el estudio del requerimiento de seguridad, específicamente el problema de la autenticación entre componentes. Por lo que se identifican los posibles mecanismos de autenticación y criptografía factibles para dicha tarea. Así mismo, se diseña el proceso ad-hoc para garantizar la autenticación entre componentes. Todo esto bajo la filosofía de la seguridad a través de prácticas de arquitectura de software.

I.2. Objetivos.

El objetivo de esta tesis es definir un procedimiento para acceso seguro (autenticación) entre componentes distribuidos, además, construir un prototipo que soporte la autenticación con base a una arquitectura de software del tipo distribuida.

Para alcanzar el objetivo planteado se completarán tres etapas. La primera etapa se inicia con el estudio del estado del arte de la autenticación de sistemas distribuidos y se continúa con la identificación de aquellos componentes que requieran establecer un protocolo de autenticación para su comunicación. Con las bases teóricas obtenidas de la etapa anterior procedemos con la segunda etapa donde se define y desarrolla un proceso de autenticación que tenga como característica la respuesta predecible en los estados de comunicación, tanto temporal como eventual. Por último se realiza la implementación de un prototipo para la evaluación del proceso.

I.3. Aportaciones.

- Los resultados que se obtengan en esta tesis pretende mostrar que los problemas de seguridad pueden ser resueltos desde una perspectiva de arquitectura de software.

- Este proyecto plantea la necesidad de seguridad en la comunicación entre componentes y se propone un esquema de autenticación ad-hoc tomando en cuenta un tipo de arquitectura distribuida con comunicación remota.

- El esquema de autenticación propuesto está dirigido a que componentes autónomos puedan realizar el proceso de identificación y que establezcan un canal seguro de comunicación.

- La implementación del esquema de autenticación por medio del caso de estudio Admire pretende ser una guía genérica para facilitar el trabajo del diseño de arquitecturas de aplicaciones Web seguras. Por otra parte constituye un acercamiento a la valoración de la implementación del diseño de la arquitectura segura en la plataforma .NET.

I.4. Organización del documento.

Capítulo I. Se describe la motivación de ésta tesis, el objetivo y pasos a seguir, así como las contribuciones de este trabajo.

Capítulo II. Se presentan las investigaciones que anteceden a este trabajo. Se inicia con la definición de Arquitectura de Software, seguida del estudio de una arquitectura Web de referencia que es punto de partida de éste proyecto de investigación, además se explica a manera general los elementos que constituyen el desarrollo de sistemas de información Web basados en componentes.

Posteriormente se estudia cada uno de los niveles en que se divide el mecanismo de seguridad de la arquitectura de seguridad multi-capa, este estudio nos permite determinar el punto en que se lleva acabo el proceso de autenticación que constituye el problema de estudio del presente trabajo.

Por último se describen los componentes del Framework de .NET y las generalidades de ASP.NET debido a que la implementación del caso de estudio Admire se realiza en esta plataforma.

Capítulo III. Se desarrolla ampliamente el tema de criptografía detallando los diferentes mecanismos que se han empleado para alcanzar los objetivos de seguridad poniendo especial atención en aquellos que se utilizan en este trabajo de tesis. Además se describe el funcionamiento de los algoritmos RSA y RC2 que se utilizan en el esquema de autenticación que proponemos. En éste capítulo también se incluye una sección donde se explica como los

esquemas simétricos basados en llaves privadas y los esquemas asimétricos basados en llaves públicas se complementan en un criptosistema híbrido.

Capítulo IV. Se presenta el problema de la comunicación entre componentes desde un punto de vista de arquitectura Web y se detallan los alcances del presente trabajo de tesis. Además se describe el caso de estudio que se utiliza en este proyecto para implementar el esquema de autenticación que proponemos; así mismo se especifica la estructura de los datos que se utilizan para validar la comunicación entre los componentes de este caso de estudio.

Capítulo V. Constituye el capítulo principal de la tesis ya que está compuesto por el esquema de autenticación propuesto, su implementación en el caso de estudio Admire y los resultados de las pruebas realizadas. Inicialmente se presenta el esquema de autenticación que se propone y que permite que las entidades participantes en una comunicación se identifiquen y acuerden un canal seguro de comunicación. Se describen a detalle las tres fases que conforman el esquema; fase de intercambio de llaves públicas, fase de identificación de entidades y la fase de acuerdo de llave de sesión.

Posteriormente, se describe el proceso que se siguió para implementar el esquema de autenticación en el caso de estudio Admire y se proporcionan los diagramas de clases y de secuencias del prototipo desarrollado.

Por último, se describen las pruebas efectuadas así como sus correspondientes resultados, y al final se realiza un análisis de los mismos.

Capítulo VI. Se presentan las conclusiones finales de esta tesis, así como algunas propuestas para investigaciones futuras.

Capítulo VII. Se presentan las referencias bibliográficas utilizadas en el desarrollo de éste trabajo de tesis en el orden que se citan. Primero se enumeran las referencias tomadas de Internet [HREF #] y luego las referencias encontradas en artículos y libros [#].

II. ANTECEDENTES Y REVISIÓN BIBLIOGRÁFICA.

II.1. Arquitectura de Software.

No existe un estándar, o una definición universal del término "Arquitectura de Software". Una de las primeras definiciones se encuentra en los trabajos de Monroe y Garlan (1996) que proponen que la arquitectura de software puede ser definida en términos de *componentes* que representan las entidades computacionales a nivel de la aplicación, *conectores* que representan la interacción entre componentes y *configuraciones* que representan el ensamblaje entre componentes y conectores [1].

Una definición más descriptiva la proporciona Booch, Rumbaugh, y Jacobson (1999) quienes definen la arquitectura como el conjunto de decisiones significativas acerca de la organización de un sistema de software, la selección de los elementos estructurales y de las interfaces que componen el sistema, junto con su comportamiento según lo especificado en la colaboración entre esos elementos, la composición de estos elementos estructurales y del comportamiento de los elementos en subsistemas progresivamente más grandes así como el estilo arquitectónico que guía esta organización, estos elementos, sus interfaces, su colaboración y su composición [2].

Recientemente Bass, Clements y Kazman (2003) definen la arquitectura de software de un programa o un sistema computarizado como la estructura o estructuras del sistema, que comprende elementos de software, las características visibles de estos y la relación entre ellos [3][1].

Las definiciones de arquitectura de software que se presentan generan la necesidad de definir los elementos que la constituyen; componentes, conectores y configuraciones. En general un componente se puede definir por medio de sus características básicas [4]:

- Un componente es una estructura de software auto-contenida: una pieza de unidad de software auto-contenida es una que puede empacarse, instalarse y distribuirse independientemente (autónomamente).
- Un socket de componente proporciona una interfase bien definida y bien conocida en tiempo de ejecución. Lo que permite que los componentes puedan combinarse con otros componentes de forma sencilla para proveer funcionalidades útiles.
- Un componente se construye para combinarse y colaborar con otros componentes.

Además, un conector esta definido como un mecanismo que media la comunicación, la coordinación, o la cooperación entre componentes. Las implementaciones de los conectores

[1] Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison-Wesley Professional, 1998, pág.23.

pueden ser distribuidas entre varios componentes, a menudo no corresponden a elementos discretos del sistema en ejecución [3][2].

La relación entre los componentes y conectores está descrita por la configuración que es la colección de componentes estructurados con la identificación de los límites de sus interfases, descrita como conectores. Una configuración va más allá de la especificación de los componentes individuales; es el panorama de la estructura de la colección [5].

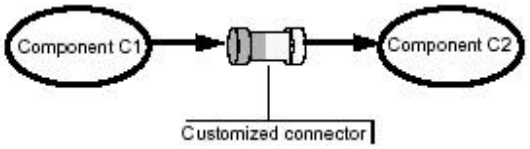

Fig. 1 Conector personalizado para la interacción de componentes.

II.2. Arquitectura Web.

Una vez establecida la definición de arquitectura de software se puede estudiar las particularidades de una arquitectura Web la cual constituye el punto de partida de este proyecto de investigación. Antes se establece que una arquitectura de referencia es una arquitectura usada para definir referencias contra la cual las implementaciones pueden ser comprobadas para evaluar la conformidad.

A continuación se detalla la arquitectura Web de referencia que se ha tomado como base para este proyecto, se inicia con la definición de algunos de los elementos que la conforman para facilitar su comprensión.

Una arquitectura Web tiene tres elementos básicos, el primero es el WebClient que es una aplicación de software que permite acceso a Internet y a sus páginas de hipervínculos, el segundo es un WebServer que es una aplicación de software que utiliza http. Un WebServer se ejecuta en una computadora que esta conectada a Internet. Un WebServer puede recibir o proveer acceso a contenido y responder a las solicitudes recibidas de un WebClient. Todo Web Server tiene una dirección IP y usualmente un nombre de dominio, aunque algunos son servidores virtuales. El tercero es el WebService, que es un sistema de software identificado por un URI (*Uniform Resource Identifiers*), sus interfases públicas están definidas y descritas utilizando XML. Sus definiciones pueden ser accedidas por otros sistemas. Estos sistemas interactúan con el WebService como su definición lo establece, utilizando mensajes basados en XML y que son transportados por protocolos de Internet.

Una vez que se han establecido los tres elementos básicos de una arquitectura Web se puede retomar del trabajo de Stolle, Rossak y Kirova (2000) la arquitectura Web de referencia

[2] Bass, L., Clements, P., Kazman, R., "Software Architecture in Practice", Addison-Wesley Professional, 1998, pág.105.

para sistemas de información en Internet con reactividad directa [6]; la arquitectura propuesta esta compuesta de cinco capas y es una extensión de la arquitectura típica de tres capas [7].

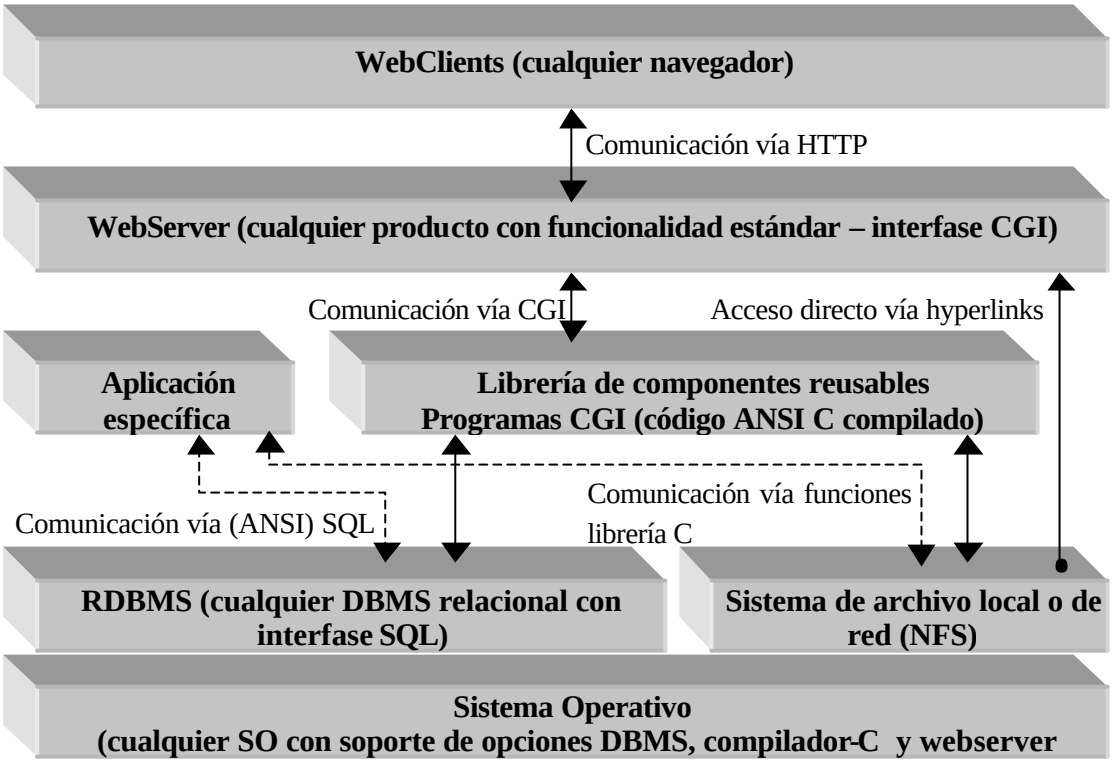

Fig. 2 Arquitectura de referencia de 5 capas.

La capa de WebClients asegura la comunicación con el usuario final e intercambia información con el WebServer vía el estándar http. Las capas dos y tres están orientadas a la aplicación, en estas se establece la lógica de la aplicación. La capa tres contiene la librería de componentes reutilizable que implementa las funcionalidades específicas que típicamente están soportadas por el DBMS para sistemas reactivos. Las capas cuatro y cinco forman la infraestructura. La comunicación con DBMS en la capa cuatro se realiza por medio del estándar SQL.

Por otra parte los componentes de la arquitectura de aplicaciones Web pueden clasificarse por el papel que desempeñan en el funcionamiento de la aplicación [8]:

Un agente de usuario utiliza un conector cliente para iniciar una solicitud y se convierte en el recipiente de la respuesta. El ejemplo más común es un Webbrowser quien provee acceso a los servicios de información y rinde servicios de respuesta de acuerdo a las necesidades de la aplicación.

Un servidor origen utiliza un conector de servicio para gobernar el dominio de un recurso solicitado. Es la fuente definitiva para la representación de los recursos y debe ser el último

recipiente de cualquier respuesta que intente modificar el valor de sus recursos. Cada servidor origen provee una interfase genérica de sus servicios como una jerarquía de recursos. Los detalles de implementación de los recursos son ocultados por la interfase.

Los componentes intermedios actúan como cliente y servidor para remitir con posibles traducciones, solicitudes y respuestas. Un componente proxy es un intermediario seleccionado por el cliente para proveer encapsulación de interfase de otros servicios, conversión de datos, aumentar el desempeño, reforzar la seguridad.

Las aplicaciones empresariales de interés en el presente trabajo son aquellas que se desarrollan con base a una arquitectura Web y para las cuales se siguen los lineamientos del desarrollo de software basados en componentes por ello el siguiente acápite se abordan las generalidades de este tema.

II.3. Desarrollo de Software Basado en Componentes.

Hasta mediados de los años 80's la mayoría de las aplicaciones de software se construían de forma monolítica, los programadores iniciaban la construcción de la aplicación desde cero, muy poco código era reutilizado de una aplicación a la siguiente. En los años recientes se ha popularizado un nuevo estilo de desarrollo de aplicaciones basadas en componentes. En este estilo la nueva aplicación se construye a partir del reutilización de componentes y aplicaciones existentes; los programadores escriben solo el código necesario para conectar los componentes y proveer facilidades adicionales que aun no están disponibles en estos

El desarrollo de software basado en componentes permite que el código existente sea reutilizado, lo que conlleva al desarrollo de aplicaciones de forma más rápida, a menor costo y con un producto final de mayor calidad.

El desarrollo de software basado en componentes es de especial utilidad en el desarrollo de sistemas de información Web debido a que se requiere una separación de la presentación y los aspectos funcionales. Lee y Shirani (2002) expresan esta división de la siguiente manera; las aplicaciones Web generalmente consisten principalmente de dos tipos de contenidos descriptivo y preceptivo. El contenido descriptivo provee hechos y descripciones acerca de un sujeto, el contenido preceptivo requiere de la implementación de procedimientos y funciones que reciben alguna entrada y producen una salida. Además expresan que los aspectos funcionales se vean más beneficiados por los componentes que la presentación. Sin embargo independientemente del tipo de componentes estos juegan un papel importante en el despliegue de páginas Web y en la ejecución de funciones necesarias para la aplicación Web [9].

El desarrollo de sistemas de información Web basa su metodología en la identificación de elementos significativos para el negocio y la aplicación. A continuación se proporciona una lista de estos elementos.

Los componentes del negocio (business components) representan una implementación en software de un concepto autónomo del negocio o de un proceso del negocio. Compuesto de todos los artefactos de software necesarios para expresar, implementar y distribuir un

componente de negocio como uno autónomo y como un elemento reutilizable de un sistema de información.

Los objetos del negocio (business objects) son objetos con límites bien definidos y una identidad que encapsula un estado del negocio y el comportamiento. Un estado del negocio es una característica estructural representada por atributos o variables de instancia mientras el comportamiento es una propiedad representada por métodos que operan sobre los atributos. Los objetos de negocio representan recursos del negocio en un dominio de negocio.

El proceso del negocio (business process) consiste en un grupo de actividades del negocio (businnes activities) asumidas por una organización con la finalidad de alcanzar un objetivo. La ejecución de una serie de actividades que conlleva al alcance de un resultado de negocio medible. El resultado puede ser la creación de un producto o servicio, de forma directa o indirecta. Un proceso de negocio es ejecutado por la participación/cooperación de un número de recursos del negocio llamados objetos de negocio. Estos objetos de negocio pueden ser *activadores* que representan actores quienes inician un proceso de negocio cuando un evento de proceso de negocio ocurre. *Estados de negocio* (dato del negocio u objeto del negocio) son creados y/o usados por un proceso de negocio mientras se ejecuta el proceso.

Arch-int y Batanov (2002) proponen una metodología para el desarrollo de componentes de software de negocio como el elemento básico de construcción de los sistemas de información industriales implementados y distribuidos en un ambiente basado en la Web [4].

La metodología esta compuesta de dos tareas principales; el modelado de los componentes de negocio y el modelado de la implementación Web. La primera tarea describe métodos y técnicas para identificar y generar componentes de negocio; el resultado es el modelo de componentes de negocio que se utiliza para representar componentes de negocio. La segunda tarea consiste en la implementación del modelo de componentes de negocio en una infraestructura específica de sistemas de información basados en la Web.

II.4. Arquitectura de Seguridad Multi-Capa.

Una vez que se ha expuesto la arquitectura Web de referencia que servirá de guía en este trabajo y que se ha explicado a manera general los elementos que constituyen el desarrollo de sistemas de información Web basado en componentes se inicia con el tema de seguridad desde una perspectiva de arquitectura lo cual constituye el área de interés de este trabajo.

Este acápite se inicia proporcionando una definición de seguridad. Seguridad significa protección en contra de los intentos de acceder la información de manera no autorizada. Esta relacionada con la *confidencialidad* (la información esta disponible solo a los usuarios autorizados a accederla), *integridad* (la información es modificada solo por los usuarios que tienen el derecho y solo en forma autorizada) y la *disponibilidad* (el uso del sistema no puede ser negado de forma maliciosa al acceso autorizado).

La seguridad es un reto en todo sistema distribuido y se implementa a través de funcionalidades de seguridad como encriptación, autenticación y autorización (control de acceso) que constituyen el mecanismo de seguridad

A continuación se identifican los niveles de seguridad en que se divide el mecanismo de seguridad lo que permite ubicar el momento en que se realiza el proceso de autenticación que constituye el problema estudio del presente trabajo.

Como se presenta en la fig. 3 el mecanismo de seguridad puede expresarse como un esquema de 4 niveles. Para las capas inferiores de seguridad, criptografía y comunicación segura se dispone de componentes de seguridad ya elaborados que permiten el desarrollo de aplicaciones seguras. En cambio para las capas superiores, modelos de autorización, control de acceso, autenticación y supervisión, los componentes adecuados requieren una plataforma específica y/o arquitectura (ej. .NET, J2EE) y la mayoría del tiempo no son lo suficientemente expresivos por lo que no pueden ser adaptados a los requerimientos de las aplicaciones complejas. Es por ello que la identificación de los requerimientos de seguridad es un paso crítico en el ciclo de desarrollo de los sistemas de información [10].

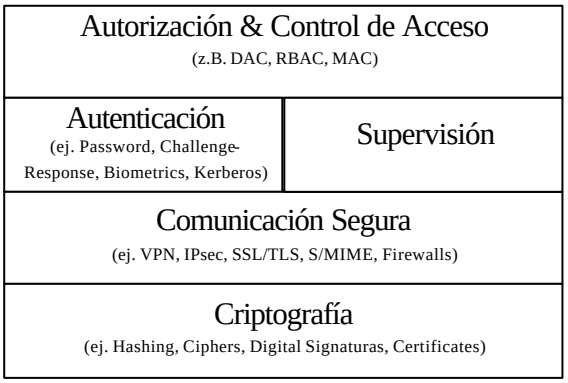

Fig. 3 Niveles de mecanismos de seguridad.

En el nivel inferior se encuentra la encriptación o criptografía que consiste en hacer que la información sea ilegible (encriptación para asegurar la confidencialidad) o inalterable (firmas digitales para asegurar la integridad de los datos) o ambas cosas. La encriptación se utiliza para proteger la información almacenada o el intercambio de ésta en contra de lectura o modificación no autorizada [11].

La Autenticación es el proceso de identificar positivamente a los clientes de la aplicación; los clientes pueden referirse a los usuarios finales, servicios, procesos o computadoras. La autenticación permite asegurar la identidad de la entidad, verificar que la entidad es quien afirma ser. Siendo más específicos el protocolo de autenticación permite asociar cada operación del sistema con un único usuario real, permitiendo comprobar si la operación esta autorizada o no. La autenticación se considera uno de los mecanismos fundamentales de seguridad, si no se provee, una entidad puede afirmar ser cualquier entidad y por lo tanto acceder a cualquier información [11].

El proceso de autorización determina cuales recursos y operaciones tiene autorizado el cliente autenticado. Recursos incluyen archivos, bases de datos, tablas, filas, etc., también los recursos a nivel del sistema como registro de llaves y configuración de la información [12]. De forma más general, en un sistema distribuido, la autorización verifica la interacción entre las entidades, además controla el flujo de información entre las entidades. Se debe remarcar que la autorización depende de la autenticación, la identidad de las entidades debe ser única e infalsificables [11].

Como parte de los trabajos realizados sobre el tema de seguridad enfocados desde una perspectiva de arquitectura se puede retomar el de Popescu, Steen y Tanenbaum (2002) quienes exponen los pasos para diseñar una arquitectura global de seguridad. El primer paso, analizar los requerimientos globales de seguridad e identificar todos los mecanismos posibles que pueden utilizarse para satisfacer estos requerimientos. El segundo paso, seleccionar un subconjunto de funciones que pueden ser implementadas como parte del middleware (no todas, algunas funciones pueden ser mejor manejadas en el nivel de la aplicación). Finalmente, en el tercer paso, seleccionar métricas para evaluar [13].

Una ejecución de estos pasos para el diseño de una arquitectura global de seguridad se encuentra en el trabjo de Probst, Essmayr y Weippl quienes presentan el diseño y la implementación en Java de un framework denominado GAMMA (Generic Authorization Mechanisms for Multi-Tier Applications). El framework contiene un conjunto de componentes que ofrecen mecanismos de seguridad de alto nivel que incluye control de acceso discreto (DAC), control de acceso por roles (RBAC), la posibilidad de utilizar múltiples modelos de autorización de forma concurrente, soporte de autorizaciones negativas, o reglas arbitrarias. El diseño se enfoca especialmente en la independencia de plataforma y arquitectura, lo que permite transferir la implementación a otra plataforma y lenguajes de programación en el futuro [14].

En la fig. 4 se observa que el software se divide entre varias capas de acuerdo a la funcionalidad. Cada capa puede comunicarse con la capa inferior o superior por medio de una interfase bien definida.

Se establece una capa de seguridad entre la capa de negocio y la capa de recursos de la arquitectura. Se ofrece un componente especial (Ej. conector de seguridad) el cual se utilizará como punto de entrada de desde la capa de negocio o aplicación a la capa de seguridad. La capa de seguridad consiste de un conjunto de clases con una finalidad común (Ej. seguridad) por lo que puede ser llamada framework.

De esta forma se combina en este esquema (ver fig. 4) la arquitectura de referencia Web (ver fig. 2) con los niveles de mecanismo de seguridad (ver fig. 3).

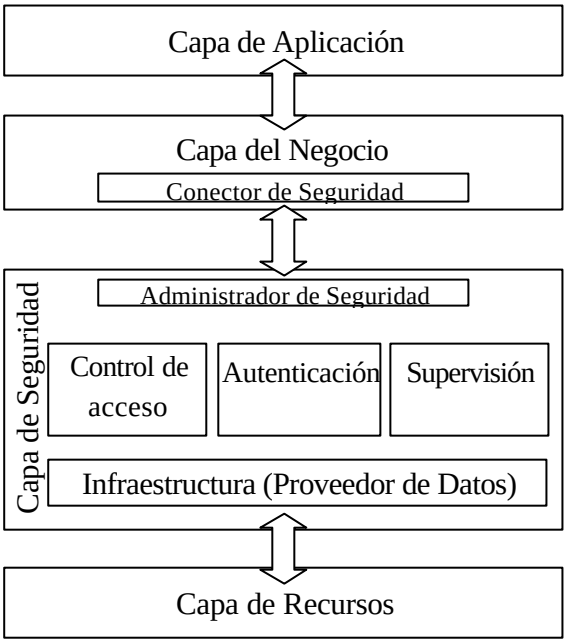

Fig. 4 Arquitectura para seguridad multi-capa.

Como ya se ha mencionado el problema que se estudia en este trabajo de tesis es la autenticación entre componentes distribuidos. Para que dos componentes lleven a cabo el proceso de autenticación requieren intercambiar la llave con que cifraran los mensajes (key agreement) y para realizar este intercambio de llaves requieren establece un canal seguro de comunicación. Debido a esto en el capítulo III se estudia el tema de encriptación por algoritmos simétricos y asimétricos.

II.5. .NET Framework.

La implementación del caso de estudio de este proyecto es en la plataforma .NET, específicamente los componentes se desarrollan haciendo uso del Framework de .NET y el desarrollo de páginas Web se realiza con ASP.NET por lo que en esta sección se inicia con las descripción del Framework de .NET y en la siguiente sección con las generalidades de ASP.NET.

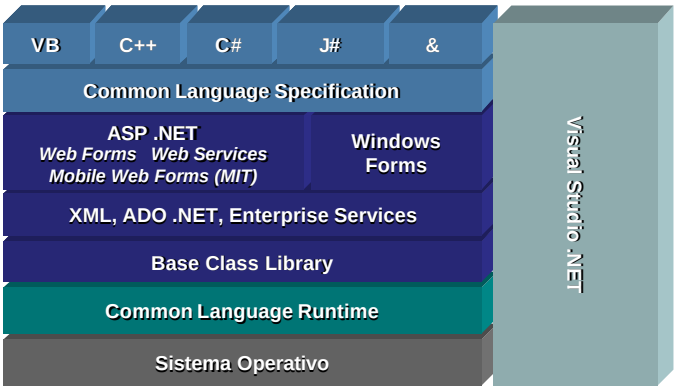

Fig. 5 .NET Framework y Visual Studio .NET.

.NET representa una evolución de frameworks (marco de trabajo) orientados a componentes. Sus predecesores COM/DCOM/COM+ de los cuales COM fue el primer esfuerzo de Microsofts en interoperatibilidad a nivel empresarial.

.NET esta orientado a brindar soluciones a los problemas comunes de desarrollo empresarial por medio de un framework que asiste a los desarrolladores que trabajan con acceso a bases de datos, servicio de mensajes, transacciones u otras tareas de tipo empresarial. Así mismo, .NET simplifica el desarrollo de aplicaciones Web distribuidas ya que se cuenta con herramientas y tecnologías como Internet Information Server (IIS). La plataforma .NET soporta la infraestructura de Internet existente, incluyendo HTTP, XML y SOAP.

La arquitectura de .NET permite el desarrollo de aplicaciones Web distribuidas con la separación clásica de las capas de presentación, lógica del negocio y acceso a datos o almacenamiento. Además, .NET agrega el enfoque basado en servicios a la arquitectura tradicional de N-capas.

Un modelo simple de aplicación distribuida consiste en un cliente que se comunica con la capa intermedia que se compone del servidor de aplicaciones y una aplicación que contiene la lógica del negocio. La aplicación de forma alterna se comunica con la base de datos que suple y almacena los datos.

Servicios de Presentación (Presentation Services).

La capa de presentación incluye una interfaz cliente-pesado (rich-client) o una interfaz cliente-ligero (thin-client) de una aplicación. El cliente pesado, ya sea directamente con Microsoft Win 32 API o indirectamente con de Windows Forms, proporciona una interfaz de programación completa a las capacidades del sistema operativo y utiliza extensivamente componentes. El cliente ligero (Web browser) se ha convertido rápidamente en la interfaz de elección de los desarrolladores. Un desarrollador puede construir la lógica del negocio que puede ser ejecutada en cualquiera de las tres capas de la aplicación. Con las aplicaciones Web en ASP.NET y XML Web services, el cliente ligero es capaz de proveer a las aplicaciones una interfaz de usuario que sea visualmente rica, flexible e interactiva. Los clientes ligeros asimismo tienen la ventaja de proporcionar un alto grado de portabilidad a través de plataformas.

Lógica del Negocio/Servicios de Aplicación (Business Logic/Application Services).

Esta capa esta dividida en servidores y servicios de aplicación, que están habilitados para recibir solicitudes de los clientes. Las aplicaciones Web pueden ser escrita de forma que tomen ventaja de los servicios COM+ del .NET Framework, cola de mensajes (MSMQ), servicio de directorio y servicios de seguridad. Los servicios de aplicación pueden a su vez interactuar con varios servicios de datos en la capa de acceso a datos.

Acceso a Datos y Servicios de almacenamiento (Data Access and Storage Services).

Los servicios de datos que soportan el acceso a datos y el almacenamiento consisten de: ADO.NET, que provee acceso programático simplificado a los datos al utilizar lenguajes scripting o de programación. OLE DB, el cual se ha establecido como el proveedor de datos universal. XML, que es el estándar de marcación para especificar estructuras de datos.

Servicios de Sistema (System Services).

Los elementos dentro de cada segmento de este modelo son soportados completamente por el .NET Framework y el Sistema Operativo de Windows. Entre los muchos servicios están directorio, seguridad, administración y servicios de comunicaciones que trabajan a través de las tres capas. Las herramientas de programación que constituyen el sistema de desarrollo del Visual Studio .NET permiten a los desarrolladores construir componentes de aplicación a través de las capas.

En términos de las opciones de comunicación entre las capas de la aplicación, .NET suporta:

DCOM (comunicación de los objetos COM)
.NET Remoting (usando TCP o HTTP)
ASP.NET (usando SOAP)

II.5.1. Componentes Básicos de .NET Framework.

La plataforma .NET consiste de dos partes principales: el common language runtime (CLR) y la librería de clases del .NET Framework.

- **Common language runtime (CLR).** Es el núcleo de la plataforma .NET. Proporciona los servicios comunes para las aplicaciones desarrolladas en el Framework de .NET. El CRL determina para las aplicaciones .NET un conjunto de tipos de datos CTS, un lenguaje intermedio CIL y un empaquetado de código Assembly. El código que ejecuta el CLR se llama código gestionado (manged code).

- **Librería de clases del .NET Framework** [HREF 3]**:** La librería incluye un conjunto de funcionalidades pre-empacadas que los desarrolladores pueden utilizar para extender de forma más rápida las capacidades de su propio software. La biblioteca de clases de .Net Framework incluye, entre otros, tres componentes clave:
  - ASP.NET, que ayuda a construir aplicaciones y servicios Web.
  - Windows Forms, que facilita el desarrollo de interfases de usuario de clientes inteligentes.
  - ADO.NET, para conectar las aplicaciones con las bases de datos.

- **Common Language Specification (CLS)** [HREF 3]**:** Para que los objetos puedan interactuar completamente, a pesar del lenguaje en el que fueron implementados, estos deben exponer a los objetos solicitantes solo aquellas propiedades que son comunes a todos los lenguajes con los que ínter operan. Por esta razón, se ha definido el CLS el cual es un conjunto de características de lenguaje básico necesario para muchas aplicaciones; este conjunto define el estándar común al cual los lenguajes y desarrolladores deben adherirse de manera que los componentes y aplicaciones desarrolladas puedan ser ampliamente usados por otros lenguajes compatibles con .NET.

II.5.2. ASP.NET

ASP.NET es un Framework de programación construido sobre el comman language runtime (CLR) que puede ser utilizado en un servidor para construir aplicaciones Web. ASP.NET toma lo mejor de ASP (Active Server Pages) así como de los servicios y características proporcionadas por el CLR, además agrega varias funcionalidades nuevas [HREF4].

Los Web Forms son la base de ASP.NET, la interfaz de usuario (UI) similar a los Windows Forms en que proporcionan propiedades, métodos y eventos a los controles ubicados en ellos. Sin embargo, estos elementos UI se traducen en el lenguaje de marcación apropiado requerido por los clientes, eje. HTML [HREF 3].

Los Web Forms están compuestos por dos componentes: la porción visual (el archivo ASPX) y el código atrás de la forma que reside en un archivo de clase separado.

Los Web Forms y ASP.NET fueron creados para superar algunas de las limitaciones de ASP. Las nuevas ventajas incluyen:

- Separación de la interfaz HTML de la lógica de la aplicación.
- Un amplio conjunto de controles del servidor que pueden se identificados por el navegador y enviar el lenguaje de marcación apropiado como por ejemplo HTML.
- Menos código al escribir debido a las capacidades de data biding de los controles .NET del servidor.
- Modelo de programación basado en eventos.
- Código compilado y soporte de múltiples lenguajes, opuesto a ASP que era interpretado como Microsoft Visual Basic Scripting (VBScript) o Microsoft Jscript.
- Permitir a terceras partes crear controles que provean funcionalidad adicional.

Como se mencionó al inicio de esta sección esta arquitectura de software será nuestro marco de referencia para desarrollar el caso de estudio y la implementación del procedimiento de seguridad entre componentes.

III. MARCO TEÓRICO.

III.1. Criptografía.

Notación

El conjunto vacío se denota por $\emptyset$. La cardinalidad del conjunto A se denota por $\#(A)$. Diremos que el conjunto A es finito si $|A| < \infty$, en caso contrario A se dice conjunto infinito. El conjunto de números naturales se denota por $\mathbb{N}$, donde $\mathbb{N} = \{1;2;3;4;\ldots\}$. El conjunto de números enteros se denota por $\mathbb{Z}$, donde $\mathbb{Z} = \{\ldots,-4;-3;-2;-1;0;1;2;3;4;\ldots\}$. $\mathbb{Q}$: Denotará el conjunto de números racionales. $\mathbb{R}$: Denotará el conjunto de números reales. $\mathbb{R}^+$ denota el conjunto de reales positivos. $\mathbb{C}$: Denotará el conjunto de números complejos. $A \times B$: Denota el producto cartesiano de los conjuntos A y B. A^n: Denota el producto cartesiano $A \times \ldots^{n} \ldots \times A$.

Criptografía es una de las disciplinas con más aplicaciones dentro de Ciencias de la Computación. Entre sus aplicaciones están la encriptación de información, manejo de llaves, firmas digitales, autenticación de usuarios, sistemas de pagos digitales y una gran variedad de protocolos para realizar tareas tales como votación electrónica. Por otro lado, criptografía constituye una de las técnicas básicas para construir la emergente infraestructura para información internacional, en particular para soportar comercio electrónico y otros servicios importantes de seguridad.

Un campo particularmente importante dentro de criptografía es la criptografía de llave pública. Esto, en contraste a la criptografía simétrica, donde las llaves secretas son guardadas en al menos dos lugares diferentes (por ejemplo para encriptación y desencriptación). Esta es la principal razón por la cual la criptografía de llave pública es de gran importancia en el manejo de llaves y otras aplicaciones. El reto de la criptografía de llave pública es desarrollar un sistema en el cual sea imposible determinar la llave privada, esto se logra por medio del uso de funciones hash de un sentido, pues si f es función en un sentido es fácil calcular $f(x)$ sin embargo es computacionalmente difícil encontrar x si sólo es conocido $f(x)$.

Sólo es necesario aplicar un pequeño número de estructuras matemáticas para construir mecanismos de llave pública, la mayoría de estos mecanismos están basados en teoría de números y álgebra, en particular sobre el grupo multiplicativo de enteros módulo un entero grande. Una operación criptográfica fundamental es la exponenciación en un grupo finito. Como consecuencia, la teoría de números computacional juega un papel importante en criptografía.

La seguridad de sistemas criptográficos no ha podido ser probada, lo más cercano a pruebas de seguridad que se puede lograr es que cierto criptosistema en la práctica ha probado que es tan difícil de romper como la solución de algún problema computacional, dichos problemas computacionales han sido conjeturados difíciles de resolver, por ejemplo la factorización de enteros grandes o el de logaritmos discretos en un grupo finito.

Los de encriptación asimétricos en comparación con los métodos de encriptación de llaves simétricas requieren de cálculos más complejos y menos eficientes. Por lo que los métodos simétricos se utilizan para la encriptación de cantidades grandes de datos.

Consideremos la siguiente situación: Un usuario *A* envía un mensaje *m* a un usuario *B* por medio de una línea de transmisión *L* que se supone insegura y en el que un enemigo *E* puede interceptar el mensaje. Surgen los siguientes problemas:

Confidencialidad: Si *E* intercepta el mensaje *m*, que hacer para que *E* no conozca el verdadero contenido de *m*? Ya que la información no debe ser entendida por un adversario.

Autenticación: El usuario *B* al recibir el mensaje *m*, como puede asegurarse de que efectivamente el usuario *A* lo envió? Para lograr que el emisor y el receptor pueda confirmar la identidad del otro y el origen/destino de la información.

Integridad: El usuario *B* al recibir el mensaje *m* como puede asegurarse de que *m* no ha sido modificado? La información no debe ser alterada en el almacenamiento o en el tránsito entre el emisor y el receptor previsto sin que la alteración se detecte.

No repudio: Que puede hacer *B* para que el usuario *A* no pueda negar que él fue quién envió el mensaje *m*? El autor/emisor de la información no puede negar en una etapa posterior sus intenciones en la creación o transmisión de la información.

El conjunto de técnicas que dan solución a estos problemas se le conoce como criptografía. Podemos decir en una primera instancia que criptografía (Kriptos = ocultar, Graphos = escritura) es el proceso asociado con la transformación de un texto simple (encriptación) en uno ininteligible (texto encriptado, criptograma) por medio de algoritmos matemáticos. El proceso inverso se conoce como desencriptación y en principio solo las personas autorizadas pueden realizar la desencriptación.

En criptografía se usan funciones especiales que permiten realizar la encriptación y desencriptación con eficiencia y seguridad.

Definición: Una función *h* se dice función criptográfica hash si
$h: [\ n \in \mathbb{N} \{0, 1\}^n \to \{0, 1\}^r$, para algún $r \in \mathbb{N}$ fijo y *h* es función en un sentido.

Por simplicidad llamaremos función hash a una función criptográfica hash.
Para cada $x \in [_{n \in \mathbb{N}} \{0, 1\}^n$ la imagen $f(x)$ se llamará valor hash. Adicionalmente una función hash *h*, puede tener las siguientes propiedades:

[i] La función hash *h* se dice débilmente resistente a colisiones si dado $x \in [_{n \in \mathbb{N}} f0;1g^n$ es computacionalmente difícil encontrar
$x' \in [_{n \in \mathbb{N}} f0;1g^n$ con $x \neq x'$ tal que $h(x) = h(x')$.

[ii] La función hash *h* se dice fuertemente resistente a colisiones si es computacionalmente difícil encontrar parejas

$x, x' \in \bigcup_{n \in \mathbb{N}} \{0,1\}^n$ con $x \neq x'$ tal que $h(x) = h(x')$.

Con objeto de formalizar el concepto de criptografía definiremos los elementos que constituyen, lo que será llamado un esquema de encriptación.

Definición: Un conjunto finito A será llamado *alfabeto de Definición*.
Ejemplo: El conjunto $A := \{0, 1\}$ es un alfabeto de Definición. Es llamado alfabeto binario.

Definición: Un conjunto M consistente de cadenas finitas de símbolos del alfabeto de Definición será llamado *espacio de mensajes*. Un elemento $m \in M$ será llamado *mensaje en texto simple*.
Ejemplo: 10101, 000000111001 son mensajes en texto simple, donde el alfabeto de Definición es el alfabeto binario.

Definición: Un conjunto C consistente de cadenas finitas de símbolos de un alfabeto de Definición A' será llamado *espacio de textos encriptados*. Un elemento $c \in C$ será llamado *texto encriptado*.

El alfabeto de Definición A' para el espacio C puede diferir del alfabeto de Definición A para el espacio de mensajes M.

Definición: Un conjunto de parámetros K será llamado el *espacio de llaves*. Un elemento $k \in K$ será llamada una llave.

En criptografía se usan funciones biyectivas f para encriptar mensajes y la inversa f^{-1} para desencriptar mensajes.

Definición: Para cada llave $e \in K$ se asocia una función biyectiva $E_e : M \to C$ llamada *función de encriptación.*

Definición: Para cada llave $d \in K$ se asocia una función biyectiva $D_d : C \to M$ llamada *función de desencriptación* . En este caso la función D_d es la inversa de la función de encriptación E_e, esto es $D_d = E_e^{-1}$.
Definición: Sea $m \in M$ y $k \in K$ y E_k la biyección asociada a k. A la imagen $E_k(m)$ se le llama *encriptación del mensaje m*.

Definición: Sea $c \in C$ y $k \in K$ y D_k la biyección asociada a k. A la imagen $D_k(c)$ se le llama *desencriptación del texto encriptado c*.

Definición: Un *esquema de encriptación* será una estructura $(M; C; K; E; D)$ donde M es el espacio de textos planos, C es el espacio de textos encriptados, K el espacio de llaves,
$E = \{E_e | e \in K\}$ un conjunto de funciones de encriptación y $D = \{D_d | d \in K\}$ un conjunto de funciones de desencriptación . Tal que para cada $e \in K$ existe una única $d \in K$ tal que
$D_d = E_e^{-1}$, esto es $D_d(E_e(m)) = m$ para todo $m \in M$. A la pareja $(e;d)$ se le llama pareja de llaves. Los elementos $E_e \in E$ serán funciones tales que $E_e : M \to C$. Los elementos $D_d \in D$ son funciones

con $Dd : C ! M$. Para cada $E_e \; 2 \; E$, existe su función inversa $E_d \; 2 \; D$. Así que $D_d \pm E_e(x) = x$ y $E_e \pm D_d(y) = y$

Definición: Participantes en un proceso de comunicación.

[a] *Entidad:* Es alguien o algo el cual envía, recibe o manipula información.
[b] *Emisor*. Es una entidad en una comunicación bipartita, el cual es el legítimo transmisor de información.
[c] *Receptor:* Es una entidad en una comunicación bipartita, el cual es el legítimo destinatario de la información.
[d] *Adversario:* Es una entidad en una comunicación bipartita, el cual no es ni el receptor ni el emisor de la información.

El adversario trata de invalidar la seguridad en el intercambio de información entre el emisor y el receptor.

Definición: *Canal de comunicación*

[a] *Canal:* Un medio para enviar información de una entidad a otra.
[b] *Canal seguro:* Es un canal que no es físicamente accesible al adversario.
[c] *Canal inseguro:* Es un canal en el que un adversario puede leer, reordernar, insertar, borrar la información.
[d] *Canal seguro:* Es un canal en el que un adversario no puede leer, reordernar, insertar, borrar la información.

III.1.1. Autenticación del mensaje.

En criptografia, SHA (Secure Hashing Algorithm) and MD5 (Message Digest Version 5) son algoritmos hash usados para garantizar que un mensaje no ha sido alterado al ser enviado por el canal de transmisión. Notemos que un adversario aún cuando no sea capaz de leer un mensaje, si podría ser capaz de alterarlo, ya sea borrando una parte del mensaje, modificándolo o aleatoriamente dañar el mensaje. SHA y MD5 garantizan que el mensaje llegará al receptor en la forma exacta en que fue enviado. Un valor hash del mensaje realiza una operación matemática sobre el mensaje antes de enviarlo. Cuando llega al receptor, la misma operación es realizada. Si la respuesta que se obtiene no es la misma entonces el mensaje será rechazado.

Un buen algoritmo hash debe ser libre de colisiones. Esto significa que dos diferentes mensajes no deberían accidentalmente producir el mismo valor hash.

III.1.2. Esquemas de encriptación.

Existen dos tipos de esquemas de encriptación, el clásico de llaves simétricas y el de encriptación de llaves asimétricas.

En la encriptación simétrica, se usa una única llave para encriptar y desencriptar. Esto es, cada pareja de usuarios (emisor/receptor) comparten una llave privada común.

La encriptación asimétrica, recibe este nombre puesto que una llave encripta y otra llave desencripta, esto es así pues las llaves están basadas en una función TOW. Cada receptor proporciona una llave pública a sus emisores para la encriptación de los mensajes y él conserva una llave privada (secreta) para desencriptar los mensajes que recibe. La idea de criptografía de llave pública esta íntimamente relacionada con la idea de función en un sentido.

III.1.3. Esquema de encriptación de llave pública (llave asimétrica).

Definición: Un esquema de encriptación $(M; C; K; E; D)$ donde $E = \{E_e | e \in K\}$ y $D = \{D_d | d \in K\}$ se dice esquema de encriptación de llave pública si para cada pareja $(e;d)$ de llaves de encriptación/desencriptación , la llave e es hecha pública mientras que la llave d se mantiene en secreto. Este esquema de encriptación será seguro si en computacionalmente difícil calcular la llave secreta d a partir de la llave e.

En un esquema de llave pública cada entidad tiene una llave pública e y una llave privada d. En sistemas seguros, dada una llave pública e es prácticamente imposible encontrar la llave privada.

En un esquema de encriptación de llave pública, el emisor y receptor no comparten una llave secreta, esto es, tanto el emisor como el receptor poseen un par de llaves: una llave secreta que es conocida solo por el receptor y una llave pública conocida por todos los emisores. Las llaves públicas parametrizan un conjunto de funciones de encriptación. Este conjunto de funciones de encriptación es también público.

Generalizando los requerimientos de la encriptación por llaves públicas se puede decir que se busca una familia de funciones F tal que cada función $f \in F$ se calcula por un algoritmo eficiente. De hecho, no debe ser computacionalmente fácil calcular una pre-imagen de la función f, esto es dado $b \in Im(f)$ deberá ser difícil encontrar $a \in Dom(f)$ tal que $f(a) = b$. Estas funciones se conocen como funciones de un sentido. Para cada función $f \in F$ en la familia existirá información que permanece secreta la cual permite un cálculo eficiente de la función inversa f^{-1}. Las funciones f de un sentido con esta propiedad se conocen como funciones TOW.

La encriptación por llaves públicas se utiliza para la distribución segura de llaves privadas. Algunas formas de autenticación y no repudio requieren de la utilización de métodos proporcionados por este tipo de encriptación, ejemplo de ello son las firmas digitales [16].

Existen numerosos esquemas de llave pública. Solo unos pocos algoritmos son seguros y prácticos. La razón del porque estos algoritmos no son prácticos es que o bien tienen llaves muy grandes o el texto cifrado que se produce es más grande que el texto simple. Para usos prácticos los esquemas de llave pública necesitan ser rápidos y eficientes. Los esquemas de encriptación de llave pública son mucho más lentos que los métodos de encriptación de llave simétrica tales como DES (Data Encryption Standards). Los algoritmos de llave pública son comúnmente usados para encriptar pequeñas cantidades de datos, tales como passwords, números de tarjetas de crédito y números PIN. Estos esquemas no son útiles para encriptar grandes volúmenes de datos, de hecho son usados para transportar la llave, la cual es finalmente usada para encriptar los grandes volúmenes de datos usando los métodos de encriptación de llave simétrica.

El uso de llaves asimétricas para este proyecto es con el objeto de que los componentes se autentiquen unos a otros. Además, acuerden la llave privada a utilizar para la transferencia de los mensajes. La llave privada se genera con el algoritmo simétrico RC2. El algoritmo asimétrico a utilizar es RSA.

III.1.4. Esquema de encriptación de llave privada (llave simétrica).

Definición: Un esquema de encriptación $(M;C;K;E;D)$ donde $E = \{E_e | e \in K\}$ y $D = \{D_d | d \in K\}$ se dice esquema de encriptación de llave privada si para cada pareja $(e;d)$ de llaves de encriptación/desencriptación, es computacionalmente fácil calcular d conociendo solo e y recíprocamente es computacionalmente fácil calcular e conociendo solo d.

Mientras que un esquema de encriptación de llave simétrica puede ser matemáticamente fuerte, todos estos esquemas comparten una debilidad común, a saber el manejo de la llave. Si el usuario A y el usuario B desean mandar mensajes encriptados con este esquema, la pregunta es: ¿Como coinciden ellos en una llave común para encriptar y desencriptar los mensajes? Un esquema de encriptación de llave pública no sustituye a un esquema de encriptación de llave privada. En la práctica, un esquema de encriptación de llave pública no es utilizado para encriptar mensajes, esencialmente su uso es para encriptar llaves. Esto se debe esencialmente a que los algoritmos de llave pública son lentos y a que los algoritmos simétricos son generalmente, al menos, mil veces más rápidos que los algoritmos de llave pública. Los algoritmos de llave pública son vulnerables a ataques de texto plano seleccionado.

Existen dos clases de esquemas de encriptación de llave privada:

- **Esquema de encriptación por bloques**.
  Es un esquema de encriptación el cual rompe el texto simple $m \in M$ en cadenas de longitud fija r sobre el alfabeto A, estas cadenas se llamaran bloque de longitud r y encripta un bloque a la vez.

- **Esquema de encriptación por cadenas**.
  Sea K un espacio de llaves para un criptosistema y $k_1; k_2; \ldots \in K$. Considere la cadena de llaves $k1k2 \ldots$ El criptosistema es llamado criptosistema por cadenas si la encriptación sobre cadenas de texto plano $m_1m_2 \ldots$ se realiza por aplicaciones repetidas de las aplicaciones de encriptación sobre las unidades de texto plano, $E_{k_j}(m_j) = c_j$ y si d_j es la función inversa de k_j entonces la desencriptación es realizada por $D_{d_j}(c_j) = m_j$ para j , 1.

Los métodos simétricos clásicos requieren una llave secreta k la cual se comparte por el emisor y el receptor.

Para cada llave $k \in K$, un esquema de encriptación de llave simétrica consiste de una función de encriptación biyectiva $E_k : M \times K \to C$ con D_k La función inversa se conoce como la función de desencriptación . Deberá existir un algoritmo eficiente para calcular las funciones E_k y D_k.

Un requerimiento básico de seguridad para la función E_k es que si no se conoce la llave k debe ser imposible calcular la función de D_k.

Los algoritmos de encriptación de llaves simétricas tienen la implementación más rápida en hardware y software por lo que son convenientes para la encriptación de cantidades grandes de datos.

La encriptación simétrica clásica provee un canal de comunicación segura a cada pareja de usuarios una vez que han acordado en una llave secreta común. Habiendo establecido el canal seguro de comunicación, se puede garantizar que la transferencia de mensajes permanece secreta. Además, la encriptación simétrica incluye métodos para detectar modificaciones en los mensajes y para la verificación del origen de un mensaje, de forma que se alcancen los objetivos de confidencialidad e integridad.

Aún cuando existen distintos algoritmos de llaves simétricas que tienen su impacto tanto en el protocolo de comunicación como en la especificación propia del sistema a asegurar, se propone en esta tesis el uso de RC2 por su robustez y sencillez.

Cabe señalar que el simple hecho de la selección de este algoritmo es un trabajo formal que requiere un tratamiento propio, el cual, sale de los objetivos de esta tesis.

El uso de llaves simétricas en este proyecto es propuesto con el objeto de establecer el canal de comunicación entre los componentes y encriptar los mensajes que se transmiten entre ellos.

III.1.5. Distribución de llaves.

Como en todos los esquemas de encriptación un punto delicado es la distribución de llaves, o bien la implementación para el acuerdo de llaves.

Por ejemplo, es importante como la llave pública en RSA es distribuida. En este caso, la distribución debe ser segura en contra de ataques tales como el llamado ataque del hombre de en medio. Supongamos que en una comunicación bipartita entre *A* y *B*, un adversario *C* tiene la capacidad de proporcionarle llaves públicas a *B* y hacerle creer que pertenece al usuario *A*. Supóngase además que *C* puede interceptar mensajes entre *A* y *C*. *C* envía a *B* su propia llave pública y este supone que pertenece a *A*. *C* puede entonces interceptar cualquier mensaje encriptado enviado por *B* y *C* entonces lo desencripta con su llave, manteniendo una copia del mensaje, encripta el mensaje con la llave publica de *A* y lo envía a *A*, En principio ni *A* ni *B* serán capaces de detectar la presencia de *C*. Defensas contra tales ataques frecuentemente se basan en certificados digitales o otros componentes de la infraestructura en los esquemas de llave pública.

III.1.6. Firma digital.

Una importante aplicación de criptografía de llave pública lo constituye la firma digital. Una firma digital de un documento es cierta información basada en la llave privada del emisor y el documento mismo. Las funciones de una firma digital son similares a las tradicionales firmas manuscritas, teniendo como analogías:

[a] Autenticación: Una firma digital garantiza que el signatario es el creador del item.
[b] Garantía de firmado: Una firma digital es una prueba de que el signatario deliberadamente firmó el documento.
[c] No reusable: Una firma digital no puede ser movida a otro item.
[d] Inalterable: Después que el item es firmado, no puede ser alterado.
[e] No repudio: El signatario no puede negar que él haya firmado el documento.

El método por el cual un mensaje es firmado es completamente simple y es aplicable para cualquier criptosistema de llave pública. Sea $E_A; D_A$ las funciones de encriptación y desencriptación del usuario A y sean $E_B; D_B$ las correspondientes para el usuario B. Sea a un bloque del mensaje que el usuario A desea enviar al usuario B. Sabemos que para asegurar que el mensaje no pueda ser leído por un adversario, el usuario A debe mandar el bloque encriptado $E_B(a)$, para asegurarse que el mensaje es también firmado el usuario A mandara también el bloque $E_B(D_A(a))$. En otras palabras, el mensaje es primeramente encriptado usando la llave privada de A y entonces el resultado es encriptado nuevamente, esta vez con la llave pública de B.

Habiendo recibido el bloque $E_B(D_A(a))$, el usuario B aplicara su función de desencriptación para obtener $D_A(a)$ y a este bloque entonces le aplicara la función de encriptación del usuario A, para conseguir a el cual es el bloque del mensaje original. Notemos que E_A es pública y de esta forma es conocida para el usuario B.

Definición: Considere dos usuarios $A;B$ en una comunicación bipartita. Una *firma digital para el usuario* A es una estructura $(S;F; f_A;U_A)$, donde S es el conjunto de mensajes que serán firmados, $F;$ es un conjunto de elementos llamados firmas.

$f_A : S ! F;$ es función llamada *función de firma* para el usuario A y será mantenida en secreto. $U_A : S £ F !\{0;1\};$ es función llamada *función de verificación* para las firmas del usuario A y será de dominio público.

El conjunto $S £ F$ es llamado espacio de mensajes signados y $(m; f_A(m))$ es llamado mensaje signado por A.

Existen varios esquemas de firma digital, algunos de ellos son.

[a] Firma digital usando un esquema de encriptación de llave privada y un árbitro.
[b] Firma digital usando un esquema de encriptación de llave pública.
[c] Firma digital usando un esquema de encriptación de llave pública y una función hash.

En implementaciones prácticas, los algoritmos de llave pública son demasiado ineficientes para firmar documentos grandes. Para salvar tiempo, se implementan esquemas de firma digital con una función hash, en vez de firmar todo el documento, el usuario A firma el valor hash del documento. En este esquema, la función hash y el algoritmo de firma digital son convenidos de antemano por los participantes en la comunicación.
Es común utilizar como firmas, cadenas binarias de longitud fija.

[a] El usuario *A* produce un valor hash del documento
[b] El usuario *A* encripta el valor hash con su llave privada, por tanto firmando el documento.
[c] El usuario *A* envía dos cosas al usuario *B*: El documento y el valor hash signado.
[d] El usuario *B* produce un valor hash del documento que el usuario *A* ya envió. El entonces usando el algoritmo de firma digital, desencripta el valor hash signado, usando para ello la llave pública del usuario *A*. Si el valor hash signado coincide con el valor hash que el usuario *B* ha generado entonces la firma es válida.

III.1.7. Firma y verificación de un mensaje.

Considere una firma digital $(S;F;f_A;U_A)$ para el usuario *A*.

[a] Sea $m \in S$. El usuario *A* crea el mensaje signado $(m; f_A(m))$ el cual envía a *B*.
[b] El usuario *B* obtiene la función de verificación del usuario *A*: U_A y calcula

$$U_A(m; f_A(m)) = \begin{cases} 1, & \text{si } f_A(m) = m \\ 0, & \text{si } f_A(m) \neq m \end{cases}$$

Si $U_A(m; f_A(m)) = 1$, el usuario *B* acepta el mensaje.
El usuario *B* rechaza el mensaje si $U_A(m; fA(m)) = 0$.

III.2. RSA.

La técnica de encriptación RSA debe su nombre a sus inventores R. Rivest, A. Shamir y L. Adleman. Es un esquema de encriptación de llave pública que no incrementa el tamaño del mensaje. El esquema RSA se caracteriza por brindarnos dos funciones importantes, a saber, el intercambio seguro de llaves y la firma digital. Una operación fundamental en este esquema de encriptación es la exponenciación en grupos finitos, también conocida como exponenciación modular. Los procesos de encriptación y desencriptación en RSA, ambos usan exponenciación modular. Esta operación consume mucho tiempo. La mayoría de esquemas de llave pública basados en exponenciación, por ejemplo ElGamal y Diffie-Hellman presentan la misma problemática. Una "operación RSA" sin importar que esta sea para encriptar o desencriptar, para firmar o autentificar es esencialmente exponenciación modular, la cual se realiza por una serie de multiplicaciones modulares.

En RSA se usan dos números primos grandes para construir la llave privada y el producto de estos primos para construir la llave pública. En este esquema asimétrico, es difícil (presumiblemente) obtener la llave privada *d* a partir de la llave pública $(e;n)$. Por otro lado, si uno fuera capaz de factorizar el número *n* en sus factores primos *p* y *q* entonces se podría calcular el exponente privado *d*. El sistema RSA requiere un proceso de multiplicación modular de enteros los cuales tiene cientos de bits. En RSA, la función de un sentido usada es multiplicación de números primos grandes. Es computacionalmente fácil multiplicar dos números primos grandes, pero para la gran mayoría de números primos grandes es

computacionalmente difícil encontrar los factores primos $p;q$ conociendo solo el valor $p \cdot q$. Este problema inverso es la clave del sistema de seguridad electrónica, RSA. Con respecto a velocidad, RSA es mucho más lento que los algoritmos de llave simétrica. Es aproximadamente 1000 veces más lento que DES, es por esta razón por lo que la mayoría de sistemas prácticos usan RSA solo para el intercambio de llaves DES y entonces usan DES para encriptar. Para encriptar mensajes, usando RSA se procede primeramente encriptando la información con una llave aleatoria DES y entonces antes de ser enviado sobre un canal inseguro, la llave DES es encriptada con RSA. Juntos, el mensaje encriptado usando DES y la llave DES encriptada con RSA son enviados. Este protocolo es conocido como una envoltura digital RSA. Por otro lado, RSA ha sido extendido a otras estructuras, por ejemplo sucesiones de Lucas y en curvas elípticas.

Problema RSA: Sea n un entero positivo tal que $n = p \cdot q$ es el producto de dos primos impares $p;q$, un entero positivo e tal que $(e;q-1) = 1 = (e;p-1)$ y un entero c. El problema RSA es: Encuentre un entero m tal que $m^e \equiv c \pmod{n}$.

Conjetura RSA: El problema RSA y el problema de factorización de enteros son computacionalmente equivalentes.

Se tiene que si los factores de n son conocidos, entonces el problema RSA es trivial. Sin embargo, no ha sido probado que el conocimiento de los factores de n sea la única forma de resolver el problema RSA.

III.2.1. Algoritmo de generación de llaves RSA.

Cada uno de los usuarios A, B generará su llave pública y su llave privada.

Por medio del siguiente algoritmo, el usuario A genera su llave pública y su llave privada. (El usuario B realiza lo propio).

[a] El usuario A selecciona dos números primos grandes p_A y q_A al azar, con $p_A \neq q_A$, de aproximadamente el mismo tamaño. Estos primos $p_A; q_A$ deben mantenerse secretos.

[b] Calcule el módulo n_A para el esquema de encriptación, esto es calcula $n_A := p_A q_A$ como el producto de los dos primos grandes.

[c] Calcule el valor de la función ϕ de Euler en n_A, esto es $\phi(n_A) = \phi(p_A)\phi(q_A) = (p_A-1)(q_A-1)$.

[d] Selecciona al azar un entero e_A con $1 < e_A < \phi(n_A)$ y tal que $\gcd(e_A;\phi(n_A)) = 1$. El número e_A se llama exponente de encriptación RSA para el usuario A.

[e] Como $\gcd(e_A;\phi(n_A)) = 1$, construimos por algoritmo euclideano extendido los enteros únicos $d_A; f$ tales que $1 = e_A d_A + \phi(n_A) f$. Esto significa encontrar el inverso multiplicativo d_A de $e_A \pmod{\phi(n_A)}$, esto es, resuelve la congruencia

$d_A e_A \equiv 1 \pmod{\phi(n_A)}$. El número d_A se llamará exponente de desencriptación RSA para el usuario *A*. La pareja $(e_A; n_A)$ se llamará llave pública de *A*, el número d_A se llamará llave privada de *A*.

[f] Habiendo realizado el usuario *B* los pasos análogos entonces los usuarios *A*, *B* publican sus respectivas llaves públicas $E_A = (e_A; n_A)$, $E_B = (e_B; n_B)$

[g] Los usuarios mantienen en secreto sus respectivas llaves privadas $D_A = d_A$, $D_B = d_B$.

III.2.2. Algoritmo de encriptación/desencriptación RSA.

El usuario *B* envia un mensaje encriptado *c* al usuario *A* y este a su vez desencripta *c*.

Encriptación:

[a] El usuario *B* obtiene la llave pública $(e_A; n_A)$ perteneciente al usuario *A*.
[b] Representa el texto plano como un entero *m* tal que $0 \cdot m \cdot n_A i 1$.
[c] Calcula $c := m^{e_A} \pmod{n_A}$.
[d] Envía el texto encriptado *c* al usuario *A*.

Desencriptación:

[a] El usuario *A* desencripta el mensaje *c* usando su llave privada d_A y calculando $c^{d_A} = (m^{e_A})^{d_A} = m \pmod{n_A}$

III.2.3. Esquema de firma digital RSA.

Aparte de que el esquema RSA es usado para la confidencialidad de la información , sirve también como un esquema de firma digital debido a que las operaciones de encriptación y desencriptación son conmutativas. Esto es, un emisor con llave pública $(e; n)$ puede crear una firma digital *S* para un mensaje *m* usando su llave privada *d* y calculando $S = m^d \pmod{n}$ lo cual puede ser verificado por cualquiera usando la llave pública del emisor $(e; n)$ y calculando $S^e = m \pmod{n}$.

Considere las firmas digitales $(M; F; f_A; U_A)$ y $(M; F; f_B; U_B)$ para los usuarios *A*, *B* respectivamente.

Sean $p_A; q_A$ números primos diferentes de alrededor de 200 dígitos y $n_A := p_A q_A$. Sean $e_A; d_A \in Z$ tal que $e_A d_A \equiv 1 \pmod{\phi(n_A)}$. La llave pública $(e_A; n_A)$ y d_A llave privada de *A*. Considere la firma digital $(M_{nA}; F; f_A; U_A)$ del usuario *A*, donde $M_{nA} = Z_{nA}$ es el espacio de mensajes a firmar, $F = Z_{nA}$ es el espacio de firmas. La función de firma de *A* es f_A,
donde:

$$f_A : Z_{nA} \to Z_{nA}$$
$$m \to m^{d_A}$$

Por tanto el mensaje m firmado por A será: $f_A(m) = m^{d_A}$

La función de verificación del usuario A es: $U_A : Z_{nA} \times Z_{nA} \to \{0;1\}$ donde

$$U_B(x,y) = \begin{cases} 1 \text{ si y sólo si } x \equiv y^{e_A} \pmod{n_A} \\ 0 \text{ si y sólo si } x \not\equiv y^{e_A} \pmod{n_A} \end{cases}$$

La estructura $(M_{nB}; f_B; f_B; U_B)$ será la firma digital del usuario B, donde $M_{nB} = Z_{nB}$ es el espacio de mensajes a firmar, $f_B = Z_{nB}$ es el espacio de firmas. La función de firma de B es: $f_B : Z_{nB} \to Z_{nB}$, donde $f_B(m) = m^{d_B} \pmod{n_B}$.

Análogamente, la función de verificación del usuario B es $U_B : Z_{nB} \times Z_{nB} \to \{0;1\}$ donde

$$U_B(x,y) = \begin{cases} 1 \text{ si y sólo si } x \equiv y^{e_B} \pmod{n_B} \\ 0 \text{ si y sólo si } x \neq y^{e_B} \pmod{n_B} \end{cases}$$

El usuario A manda al usuario B la pareja $(m; f_A(m))$.
El usuario B obtiene $(e_A; U_A)$ y calcula $U_A(m; f_A(m)) = U_A(m; m^{dA}) = 1$.
Pues $(m^{dA})^{eA} \equiv m \pmod{n_A}$.

III.2.4. Seguridad de RSA

Fundamental para la seguridad del algoritmo RSA es la exponenciación modular de enteros grandes de hasta 200 dígitos decimales (o 600 bits) de longitud. Las llaves pública y privada son funciones de dos primos aleatorios (de más de 200 dígitos). Estos números grandes son necesarios para garantizar la seguridad del esquema RSA, debido a que el esfuerzo necesario para romper el esquema por la vía de factorizar el módulo n es una función exponencial de n.

La seguridad entera de RSA es basada en la suposición de que factorizar es difícil. Por tanto si alguien encontrara un algoritmo fácil de factorización, esto invalidaría definitivamente a RSA. Por otro lado, no ha sido demostrado que la seguridad de RSA sea tan difícil como factorizar el módulo n.

La seguridad de RSA depende esencialmente en la dificultad de factorizar números grandes. Pero nadie ha probado o refutado que la factorización sea la única forma de romper la seguridad de RSA Sin embargo, es creencia generalizada que no existe otra forma de romper RSA.

Por otro lado, factorizar números pequeños no es problema, pero factorizar números grandes, actualmente constituye un verdadero reto, pues en la actualidad no existe algoritmo que en tiempo polinomial resuelva el problema de factorización.

Para tener un nivel de seguridad aceptable en RSA, se deben dar condiciones sobre los primos $p; q$ para minimizar los ataques sobre RSA. Además es recomendable la aplicación de algoritmos para generar primos $p; q$ que sean "seguros" contra ataque sobre RSA.

De todos los algoritmos de llave pública propuestos, RSA es el algoritmo más fácil de entender y de implementar. Desde el momento de su introducción ha sido sometido a significativos criptoanálisis. Sin embargo, esto no prueba ni desaprueba la seguridad de RSA.

Aún cuando RSA no es una solución perfecta, este esquema brinda razonable seguridad en Internet con cientos de bits como estándar de encriptación. Uno de los tópicos de mayor importancia a optimizar dentro de la tecnología RSA es la búsqueda de procesos más rápidos de exponenciación modular. A medida que la demanda de volúmenes más altos de información circulan en la red se hace imperativo el lanzamiento de procesadores y de algoritmos de encriptación cada vez más rápidos.

Los problemas de seguridad en la red debido al incremento en el tráfico de transacciones en Internet han hecho su aparición y han evidenciado la necesidad de mayor investigación en algoritmos criptográficos. RSA actualmente es ampliamente usado.

Ver apéndice 2 para un ejemplo detallado de la encriptación con RSA.

III.3. RC2.

Usaremos el término ”palabra” para denotar una cantidad de 16 bits.

[a] RC2 es un esquema de encriptación por bloques, el cual fue diseñado en 1989 por Ron Rivest para RSA Data Security, Inc. Inicialmente fue mantenido como un algoritmo confidencial, bajo patente y marca registrada por RSA Data Security Inc. RC2 fue publicado en Internet en 1997 [20]. Uno de los propósitos iniciales de RC2, fue el de tratar de ser eficiente sobre procesadores de 16 bits y con bloques de tamaño de 64 bits se intentó desplazar a DES. Característica importante de RC2 es la flexibilidad al usuario en términos del tamaño efectivo de la llave.

[b] RC2 es un método de encriptación de llave privada.

[c] RC2 trabaja sobre bloques de 64 bits, los cuales son divididos en 4 palabras cada una de 16 bits. Es un método iterado de encriptación por bloques, donde el texto cifrado se calcula como una función del texto plano y la llave secreta se calcula en un número de rounds. El método de encriptación RC2 encripta bloques de 64 bits, con llaves de longitud variable.

Hay dos clases de rounds en RC2. Llamados rounds ”Mixing” y ”Mashing”. Hay en total 16 rounds Mixing y 2 rounds Mashing. En cada round, cada una de las cuatro palabras (cada una de 16 bits) se actualiza, en una encriptación intermedia, como una función de las otras palabras. Cada uno de los rounds Mixing toma una sub-llave de 16 bits. Las 64 sub-llaves se obtienen a partir de la llave seleccionada por el usuario, esta llave puede ser de hasta 128 bytes. Hay un parámetro adicional del algoritmo, llamado longitud efectiva de la llave. Se observa que la operación de desencriptación no es igual a la operación de encriptación, esto podría ser contraproducente en la implementación.

III.3.1. Descripción del algoritmo de encriptación RC2

La suma de palabras de 16 bits será denotada por + y esta suma se reducira módulo 2. El simbolo & denota el bitwise lógico equivalente a la conjunción. Por © denotaremos el bitwise logico equivalente al "o exclusivo". El símbolo : denota el bitwise lógico equivalente a la negación. La expresión "x <<<< k" denota la palabra de 16-bit x rotada a la izquierda por k bits.

El esquema de encriptación RC2 involucra tres algoritmos:

III.3.1.1 Expansión de la llave.

En el proceso de expansión de la llave se usaran operaciones de byte y operaciones sobre palabras. El arreglo $K[:]$ que guarda las 64 rounds de llaves de 16 bits, serán denotados en dos formas:

[a] Para operaciones de palabra las posiciones del buffer serán denotadas por $K[0];K[1];:::;K[63]$, donde cada $K[i]$ es una palabra de 16 bits.

[b] Para operaciones de byte el arreglo de los round de llave será denotado por $L[0];L[1];:::;L[127]$ donde cada $L[i]$ es un byte de 8 bits. Tendremos que el byte de orden inferior será dado primero, esto es $K[i] = L[2i]+256£L[2i+1]$.

Este algoritmo toma una llave de longitud de 1 hasta 128 bytes, proporcionada por el usuario y produce una llave expandida consistente de 64 sub-llaves, estas sub-llaves son las $K[i]$.

Supongamos que la llave del usuario consiste de T bytes, donde $1 \cdot T \cdot 128$. El proceso de expansión de la llave coloca la llave de T bytes en $L[0];L[1];:::;L[T ¡1]$ del buffer de llave.

Sin importar la longitud de la llave proporcionada por el usuario, el algoritmo tiene una longitud de llave efectiva, la cual es denotada por $T1$.

En bytes, la longitud efectiva de llave se define por $T8 := [T1=8]$ y definimos una máscara por $TM := 255 \pmod{2^{8(1¡T8)+T1}}$. Notemos que $TM = 2^{T1 \pmod 8¡1}$.

El objetivo del algoritmo de expansión de llave es modificar el buffer de llave de tal forma que cada bit de la llave expandida dependa en alguna forma complicada sobre cada bit de la llave dada.

La expansión de llave y la encriptación hacen uso de una tabla de sustitución, que denotaremos por Π. Π es una tabla consistente de una permutación aleatoria de los números $0;1;2;\ldots;127$ y fue derivada de la expansión de $\pi = 3{:}14159\ldots$

En el apéndice 3, detallamos Π en notación hexadecimal.

La operación de expansión de llave consiste de las siguientes dos iteraciones.

[a] for $i = T;T+1;:::;127$ do

$L[i] = \Pi[L[i-1]+L[i-T]]$; donde la suma es módulo 256
[b] $L[128-T8] = \Pi[L[128-T8]\&TM]$;
[c] for $i = 127-T8; : : : ;0$ do
$L[i] = \Pi[L[i+1]\perp L[i+T8]]$;

Cada una de las 64 sub-llaves, vienen dadas por:

$$K[i] = L[2i]+256\ L[2i+1]; \ i = 0;1;2; : : : ;63:$$

El algoritmo de expansión de llave inicia colocando el T-byte dado de la llave en bytes $L[0]; : : : ;L[T-1]$ del buffer de la llave.

Por ejemplo, con una llave efectiva de longitud de 64 bits, $T1 = 64$, $T8 = 8$ y $TM = 0xff$. Con una llave efectiva de longitud de 63 bits, $T1 = 63$, $T8 = 8$ y $TM = 0x7f$.

La "llave efectiva" consiste de los valores $L[128-T8]; :::;L[127]$. El paso intermedio correspondiente a la operación "and" reduce el espacio de búsqueda para $L[128-T8]$ de tal forma que el número efectivo de bits de la llave es $T1$.

Ver apéndice 3 para tabla de sustitución.

III.3.1.2 Algoritmo de encriptación.

Dado un bloque de texto plano a encriptar de 64 bits, se divide en cuatro palabras, cada una de 16 bits denotadas por $R[0]; : : : ;R[3]$.

La operación de encriptación es definida en términos de operaciones primitivas, que llamaremos operaciones "Mix" y operaciones "Mash".

Mix up $R[i]$.

La operación "Mix up $R[i]$"se define como sigue, donde $s[0] = 1$, $s[1] = 2$, $s[2] = 3$, y $s[3] = 5$,

$R[i] = R[i]+K[j]+(R[i-1]\&R[i-2])+((:R[i-1])\&R[i-3])$;
$j = j+1$;
$R[i] = R[i] <<<< s[i]$;

Donde los índices del arreglo R siempre se considerarán módulo 4 y j es una variable "global" de tal forma que $K[j]$ es siempre la primera palabra llave en la llave expandida la cual aún no ha sido usada en una operación "Mix".

Round mixing.

Un round mixing consiste de Mix(R[0]), Mix(R[1]), Mix(R[2]), Mix(R[3]).

Mash $R[i]$

La operación primitiva "Mash $R[i]$" se define por:

$R[i] = R[i] + K[R[i-1] \& 003 fx]$;
Así, se aplica la operación Mash a $R[i]$ al sumarla a una de las palabras de la llave expandida. La palabra llave que será usada es determinada fijándose en los seis bits de menor orden en $R[i-1]$, y usándolo como índices en el arreglo de la llave K.

Round mashing.

Un round mashing consiste de: Mash($R[0]$), Mash($R[1]$), Mash($R[2]$), Mash($R[3]$).

III.3.1.3 Operación de encriptación con RC2.

La operación de encriptación ahora puede ser descrita como sigue. Aquí, j es una variable entera global la cual es afectada por las operaciones Mixing.

[a] Inicialice las palabras $R[0]; : : : ;R[3]$ de tal forma que contengan el bloque de texto plano de 64 bits,
[b] Expanda la llave, tal que las palabras $K[0]; : : : ;K[63]$ se logren definir.
[c] Inicialice $j = 0$.
[d] Realice cinco Mixing rounds.
[e] Realice un Mashing round.
[f] Realice seis Mixing rounds.
[g] Realice un Mashing round.
[h] Realice cinco Mixing rounds.
[i] El texto cifrado es definido como el valor resultante de $R[0];R[1];R[2];R[3]$.

Note que cada round mixing usa 4 palabras llave, y que hay 16 round mixing juntos, así que cada palabra llave es usada exactamente una vez en un Mixing round.

III.3.1.4 Algoritmo de desencriptación.

La operación de desencriptación se define en términos de operaciones primitivas que anularán las operaciones "Mix" y "Mash" del algoritmo de encriptación. Estas operaciones serán llamadas "*I*-Mix" y "*I*-Mash" (*I*- denota la operación inversa).

Aquí, la expresión $x >>>> k$ denota la palabra x de 16-bits rotada hacia la derecha por k bits.

I-**Mix** $R[i]$

La operación primitiva "*I*-Mix $R[i]$" se define como sigue, donde $s[0] = 1; s[1] = 2; s[2] = 3$, y $s[3] = 5$,
$R[i] = R[i] >>>> s[i]$;
$R[i] = R[i] - K[j] - (R[i-1] \& R[i-2]) - ((:R[i-1]) \& R[i-3])$;
$j = j - 1$;
Donde los índices del arreglo R siempre se consideraran módulo 4, de tal forma que $R[i-1]$ se referirá a $R[3]$ si i es 0 (estos valores son "enrrollados" de tal forma quet R siempre tendrá subindíces en el rango 0; 1; 2; 3):

Round *I*-Mix
Un round *I*-mix consiste de: *I*¡mix(*R*[3]), *I*¡mix(*R*[2]), *I*¡mix(*R*[1]), *I*¡mix(*R*[0]).
***I*-mash *R*[*i*]**
La operación primitiva "*I*-Mash *R*[*i*]" se define por:

R[*i*] = *R*[*i*]¡*K*[*R*[*i*¡1]&003 *fx*];
round *I*-mash
Un round *I*-mash consiste de: *I*¡mash(*R*[3]), *I*¡mash(*R*[2]), *I*¡mash(*R*[1]), *I*¡mash(*R*[0]).
Operación de desencriptación

Aquí *j* es una variable global que es afectada por las operaciones Mix. El proceso de desencriptación se realiza por:

[a] Inicialice las palabras *R*[0]*; : : : ;R*[3] que contienen el texto cifrado de 64-bits.
[b] Expanda la llave, tal que las palabras *K*[0]*; : : : ;K*[63] sean definidas.
[c] Inicialize *j* a 63.
[d] Realice cinco round *I*-Mix.
[e] Realice un round *I*-mash.
[f] Realice seis round *I*-Mix.
[g] Realice un round *I*-mash.
[h] Realice cinco round *I*-Mix.
[i] El texto plano se define como los valores resultantes *R*[0]*;R*[1]*;R*[2]*;R*[3]

Ver apéndice 3 para vectores de prueba para la encriptación con RC2.

III.4. Esquema de encriptación híbrido.

Puesto que los esquemas de encriptación simétrica tienen su desventaja principalmente en el manejo de la llave, se tiene que en contraparte los esquemas de encriptación asimétrica tienen su desventaja en los requerimientos computacionales de aquí que los esquemas asimétricos sean más lentos. Afortunadamente, las dos tecnologías pueden ser usadas juntas en forma efectiva. Debido a la utilización de esquemas públicos y privados, estos criptosistemas son llamados *criptosistemas híbridos*.

Por otro lado, en la mayor parte de las implementaciones prácticas los esquemas de encriptación de llave pública son usados para distribuir de forma segura llaves privadas y entonces con estas llaves se usan algoritmos simétricos para el tráfico seguro de mensajes.

En este proyecto usamos un criptosistema híbrido consistente de los esquemas RSA y RC2.

IV. PLANTEAMIENTO DEL PROBLEMA.

IV.1. Alcances del Proyecto.

En esta investigación se plantea la necesidad de proveer seguridad en la comunicación entre componentes por lo que se desarrolla un procedimiento que cubra de manera inicial los requerimientos de comunicación horizontal entre componentes fig. 6.

De hecho el desarrollo de componentes, su comunicación y su incorporación de seguridad son presentados en la fig. 6. En este caso, la incorporación de la seguridad se hace de manera horizontal (ver fig. 7) a diferencia de las estrategias de seguridad clásica las cuales se realizan de manera vertical. En la fig. 7 se resalta el lugar donde la comunicación entre componentes toma lugar, la cual es independiente y transparente a los usuarios de los servicios de navegación.

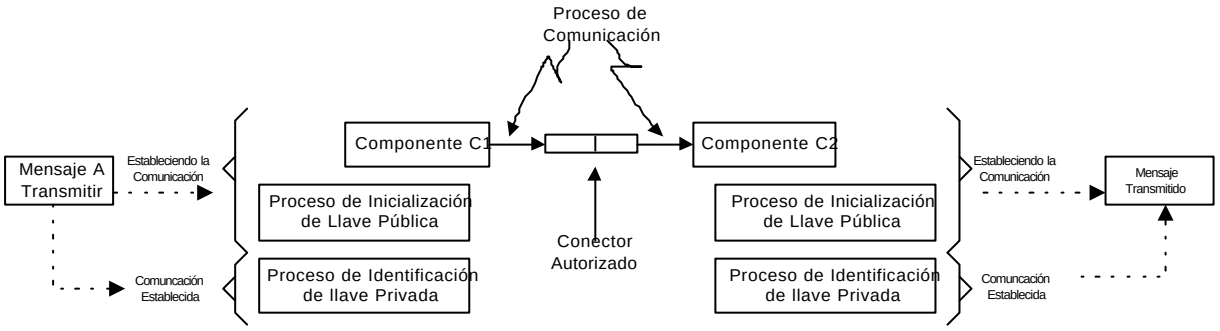

Fig. 6 Esquema Preliminar.

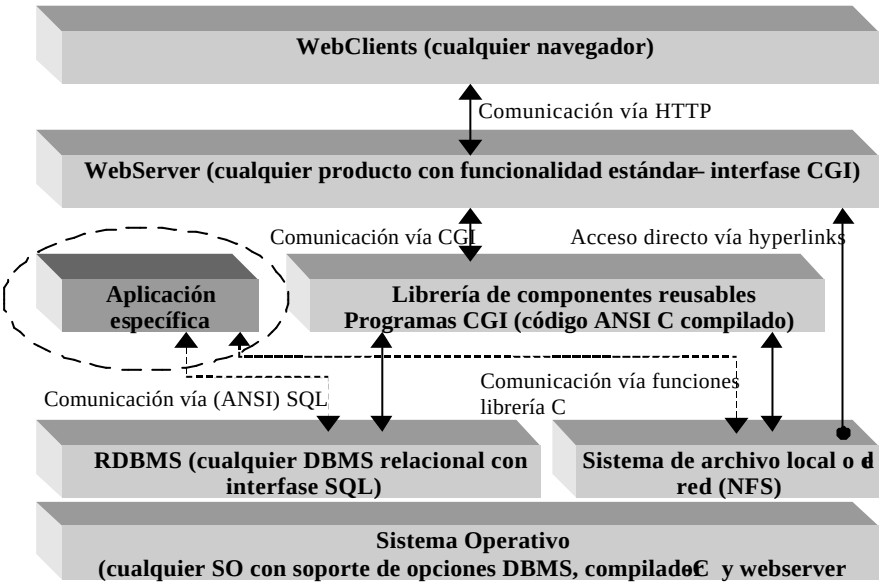

Fig. 7 Ubicación Esquemática del Proceso.

El objetivo principal de este proyecto es definir un procedimiento para acceso seguro (autenticación) entre componentes distribuidos. Los alcances de este proyecto son:

- Formalización del proceso mostrado en la fig. 6 como un esquema de autenticación híbrido.
- Su implementación en una arquitectura similar a la mostrada en la fig. 7.
- Su validación a través de un caso de estudio, el cual será explicado en la siguiente sección.

Se eligió implementar el esquema de autenticación a través de un caso de estudio en la plataforma .NET porque su arquitectura Web es similar a la presentada en la fig. 7 y además proporciona un framework orientado a componentes que facilita el desarrollo de soluciones empresariales en un tiempo razonable.

Por otra parte esta plataforma no soporta el desarrollo de sistemas en tiempo real por lo que esta tesis no tiene como objetivo evaluar el esquema de autenticación con respecto a las restricciones de tiempo lo cual no se contradice con el objetivo principal planteado.

IV.2. Caso de Estudio.

Actualmente, un problema básico en el esquema de comunicación entre componentes es la recurrencia de accesos inseguros. Por ejemplo, en sistemas de navegación autónoma como lo son los robots móviles o algún tipo de avión para monitoreo, la transmisión de datos en línea que tenga la característica primordial de ser en tiempo real requiere ser acotada con respecto de los tiempos de acceso y comunicación entre componentes para llevar a cabo una evaluación fidedigna del proceso. Así mismo se requiere asegurar que un adversario no lea la transmisión de datos sin haber sido validado por la comunidad de componentes. Por lo que el presente proyecto de tesis se enfoca a proponer un proceso de autenticación entre componentes acotando su efecto en el caso de estudio.

El uso de este prototipo será enfocado a la transmisión de datos seguros en Web para aplicaciones con alto requerimiento de seguridad (Safety Critical System) tales como sistemas autónomos, en específico en procesos relacionados con la fusión de datos en línea. Para tal efecto se plantea la transmisión de información vía Web de un modelo de avión hacia bases terrestres con el objeto de monitorear en línea su desempeño. Se propone el modelo de avión llamado ADMIRE (ver fig. 8) [HREF 5] el cual transmite los datos provenientes de los sensores de manera empaquetada.

Estos sensores transmiten información variada y con distintos valores de prioridad, como pueden ser la relación del consumo de combustible de la turbina de gas, la temperatura que guardan ciertos sectores del avión, la estabilidad geométrica del avión, su posición global, sus aceleraciones axiales, sus momentos correspondientes, así como otras relacionadas con la presión y altitud.

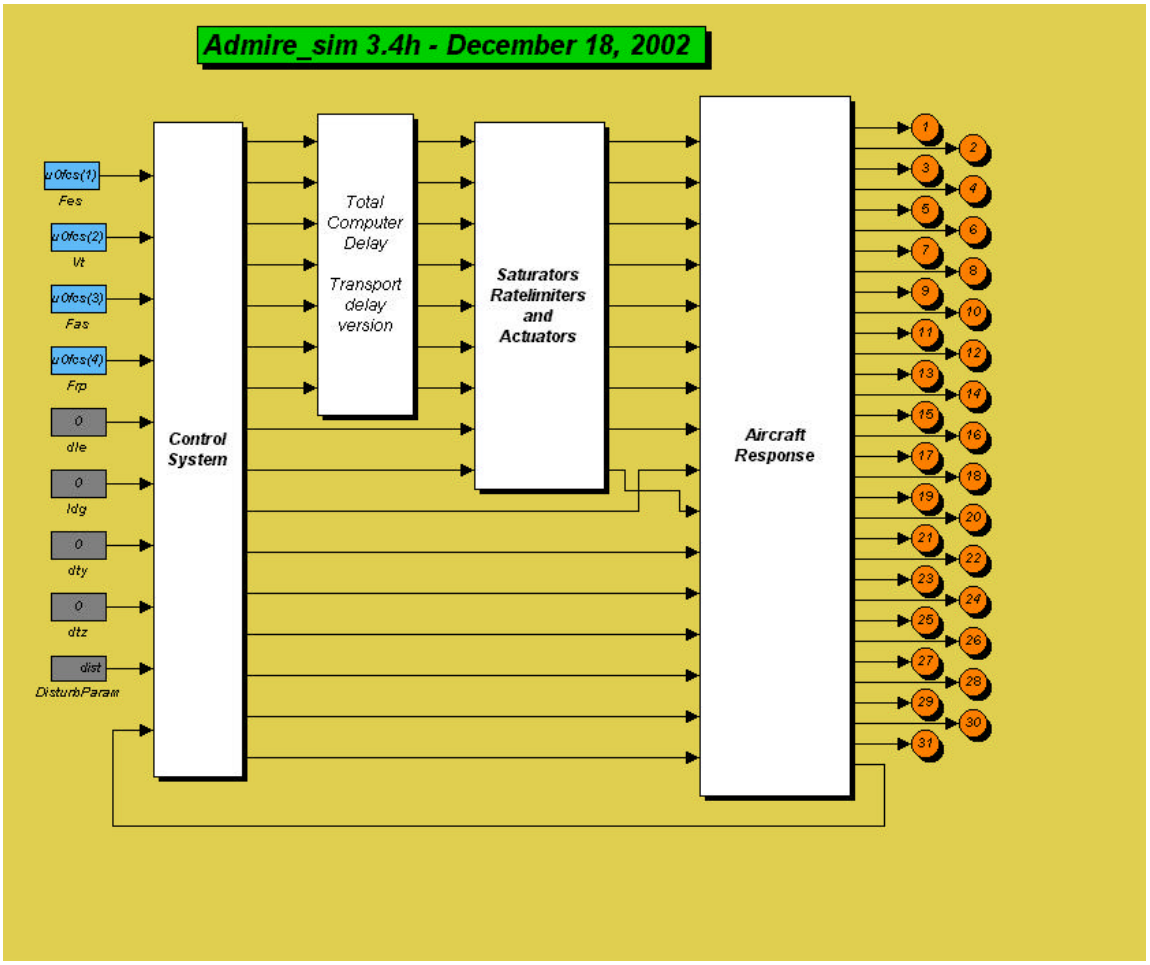

Fig. 8 Caso de Estudio.

Dicha información está disponible como respuesta del monitoreo del modelo aereo-termodinámico llamando "Aircraft Response" cada cierto módulo de tiempo. Esta información es transmitida al sistema de monitoreo y almacenamiento terrestre. Es importante señalar que esta información es sensible a la seguridad que representa desde el punto de vista de su valor industrial hasta la seguridad del grupo humano que se transporta. Además, esta comunicación es transparente entre aplicaciones finales que pueden dar una sobre carga en el protocolo. Dado esto, se requiere establecer un tipo de comunicación del tipo "horizontal" la cual se lleva a cabo entre componentes. Es por esto, que el desarrollo planteado en esta tesis define una alternativa a la definición de seguridad en este caso de estudio.

Como se mencionó en la sección de objetivos, se pretende definir un procedimiento que asegure la no violación en la comunicación entre componentes como resultado de la transmisión de datos entre un sistema autónomo y su sistema de análisis remoto (ver fig. 9).

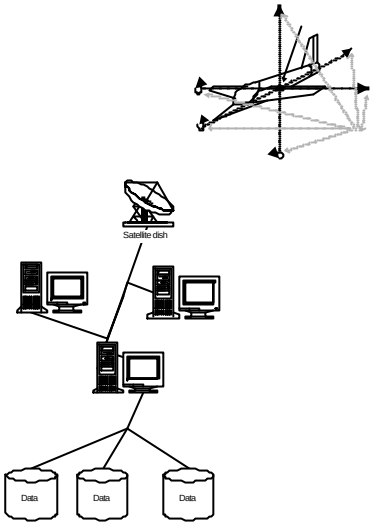

Fig. 9 Diagrama esquemático del caso de estudio.

El modelo del avión admire reporta las mediciones de los sensores en tres puntos de operación ó escenarios, los escenarios no reportan transicio nes debido a que las transiciones son no lineales. Para la implementación del caso de estudio se eligió registrar la información del sensor de velocidad, el sensor de ángulo de ataque y sensor de de rotación del eje vertical.

Sensores	**Alias**	**Simbología**	**Unidad de Medida**
Sensor de velocidad	alt1	d_{Mach}	Mach
Sensor de ángulo de ataque	alt2	da	radianes
Sensor de rotación del eje vertical	alt4	d Alt	metros

Tabla 1 Lista de sensores utilizados en el caso de estudio Admire.

En el siguiente capítulo se presenta el esquema de autenticación como fue concebido, inicialmente se define cada una de las variables utilizadas y continuación se especifica cada una de las etapas previstas en base a diagramas y descripción de los pasos contemplados.

V. TRABAJO DESARROLLADO.

V.1. Esquema de autenticación.

El esquema de autenticación que se propone está orientado a una comunicación cliente/servidor donde el componente *A* es quien requiere el servicio y eventualmente inicia la comunicación; el componente *B* es el servidor.

En éste esquema se utiliza el algoritmo RSA para la generación de la pareja de llaves asimétricas y el algoritmo RC2 para la generación de llaves simétricas, además firmaremos digitalmente las llaves públicas usando el esquema de encriptación de llave pública RSA y la función hash SHA.

A continuación se define cada uno de los elementos que conforman en el esquema de autenticación:

[i] $(e_A;n_A)$: Llave pública generada por el algoritmo asimétrico RSA que *A* que publica a sus emisores para que cifren los mensajes que le dirigen.

[ii] d_A: Llave privada generada por el algoritmo asimétrico RSA que *A* utiliza para descifrar los mensajes recibidos de sus emisores.

[iii] $(e_B;n_B)$: Llave pública generada por el algoritmo asimétrico RSA que *B* publica a sus emisores para que cifren los mensajes que le dirigen.

[iv] d_B: Llave privada generada por el algoritmo asimétrico RSA que *B* utiliza para descifrar los mensajes recibidos de sus emisores.

[v] ?: Llave privada generada por un algoritmo simétrico RC2 que utilizarán el emisor y receptor para cifrar y descifrar respectivamente los mensajes.

Con el objetivo que *A* y *B* se autentiquen y acuerden una llave de sesión (llave privada) a utilizar se deben completar tres fases; a continuación se describe cada uno de los pasos a seguir.

V.1.1. Fase intercambio de llaves públicas.

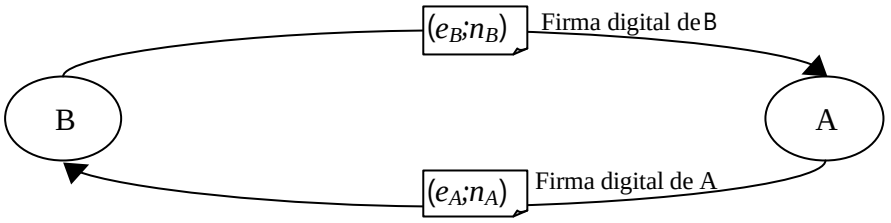

Fig. 10 Fase intercambio llaves públicas.

[i] *A* solicita la llave pública de *B* $(e_B;n_B)$.

[ii] *B* genera su pareja de llaves asimétricas $(e_B;n_B)$ y d_B por medio del algoritmo RSA, para proporcionar de una manera segura la llave pública $(e_B;n_B)$ a A, calcula el valor hash de $(e_B;n_B)$ y firma digitalmente este valor.

[iii] A recibe el mensaje que contiene la llave pública $(e_B;n_B)$ de B y la firma digital calculada para el valor hash de esta llave, A nuevamente recalcula el valor hash de $(e_B;n_B)$ y verifica la firma digital.

[iv] A genera su pareja de llaves asimétricas $(e_A;n_A)$ y d_A por el medio del algoritmo RSA, para proporcionar de una manera segura la llave pública $(e_A;n_A)$ a *B*, calcula el valor hash de $(e_A;n_A)$ y firma digitalmente este valor.

[v] *B* recibe el mensaje que contiene la llave pública de A $(e_A;n_A)$ y la firma digital calculada para el valor hash de esta llave, *B* recalcula el valor hash de $(e_A;n_A)$ para verifica la firma digital.

V.1.2. Fase identificación de entidades.

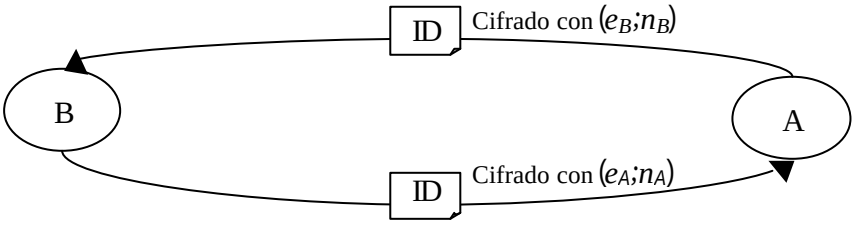

Fig. 11 Fase identificación de entidades.

[i] A cifra su ID haciendo uso de la llave pública de B $(e_B;n_B)$. A auténtica a *B* bajo la idea de que solo *B* podrá descifrar el mensaje con su llave privada d_B que sólo *B* debe conocer.

[ii] *B* descifra el mensaje haciendo uso de su llave privada d_B y lo regresa cifrado con la llave pública de *A* (e_A;n_A) como muestra de su autenticidad.

V.1.3. Fase acuerdo de llave de sesión.

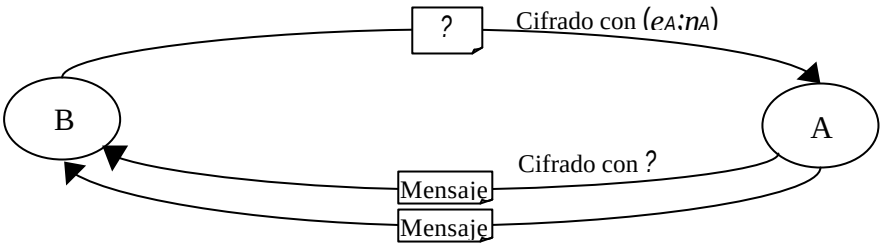

Fig. 12 Fase acuerdo de llave de sesión.

[i] *B* genera una llave privada *?* por medio del algoritmo simétrico RC2 y la envía cifrada con la llave pública de *A* (e_A;n_A). *B* auténtica a *A* bajo la idea que solo *A* podrá descifrar el mensaje y obtener la llave privada del algoritmo simétrico con la llave privada d_A que sólo *A* conoce.

[ii] *A* cifra los mensajes que enviará a *B* con la llave privada *?*.

[iii] *B* descifra los mensajes que recibe de *A* con la misma llave privada *?*.

V.2. Esquema de Autenticación Aplicado al Caso de Estudio Admire.

El caso de estudio Admire está basado en una comunicación cliente/servidor, donde el cliente lo constituye el componente que recopila los datos obtenidos de las mediciones de los sensores del avión y los transmite por medio de un canal seguro al servidor para que sean almacenados. El servidor lo constituyen los componentes encargados de establecer el canal seguro de comunicación para recibir la transmisión de los datos de los sensores proveniente del componente ubicado en los aviones y almacenarlos en la base de datos local de manera que posteriormente puedan ser divulgados en formato HTML.

Las clases en que se implementan los algoritmos RSA, RC2 y SHA se encuentran en la librería Crypto.dll. Las clases que contienen la funcionalidad del cliente se encuentran en el componente RemotingClient; la funcionalidad del servidor se encuentran en las clases de los componentes RemotingServerHost, RemotingServer, las clases que permiten la divulgación de la información en formato HTML se encuentran en el componente WebAdmire ver fig 13.

En el componente RemotingClient sólo se encuentran los llamados remotos a los objetos del servidor y la lógica de éstos llamados, un objeto cliente realiza la invocación remota de un objeto servidor por medio de la interfaz IServer para comunicarse con el representante (proxi) para obtener los servicios remotos de los objetos servidor.

La libería Crypto.dll se encuentra instalada tanto en el servidor como en el cliente, de forma que el componente cliente y el componente servidor puedan generar su pareja de llaves asimétricas RSA y realizar la respectiva encriptación y desencriptación por medio de la invocación de los métodos de la clase PublicCrypto de la librería Crypto.dll. El servidor es el encargado de generar la llave privada del algoritmo simétrico RC2 por medio de la invocación de los métodos de la clase PrivateCrypto de la libería Crypto.dll y de proporcionarla al cliente. Además, tanto el cliente como el servidor realizan la encriptación y desencriptación de los mensajes haciendo uso de la clase PrivateCrypto. Ver fig. 14 para mayor detalle de las clases de la implementación del caso de estudio Admire.

La comunicación la inicia el cliente (componente en el avión) debido a que es el que requiere los servicios proporcionados por el servidor (componentes en la estación terrestre). El esquema de autenticación presentado en la sección anterior se implementa en el caso de estudio Admire por medio de una secuencia de llamados del cliente a los objetos remotos del servidor en conjunto con las respectivas respuestas de éstos objetos. Para mayor detalle de la secuencia de llamados y respuestas entre el cliente el servidor ver figuras 15, 16 y 17.

En las siguientes dos secciones de este documento se presenta la explicación detallada del diagrama de clases y los diagramas de secuencia productos de la implementación del caso de estudio Admire.

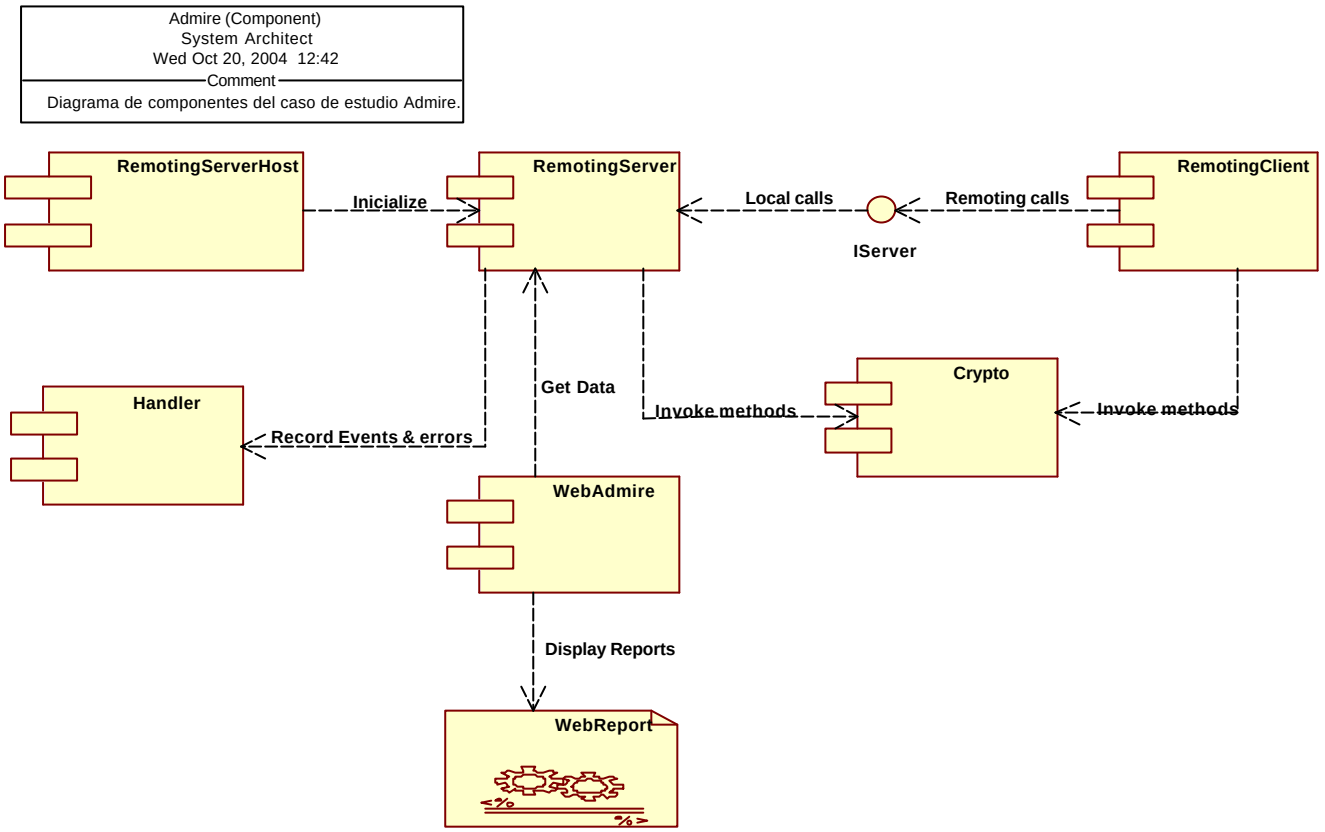

Fig. 13 Diagrama de componentes del caso de estudio Admire.

V.3. Diagrama de clases.

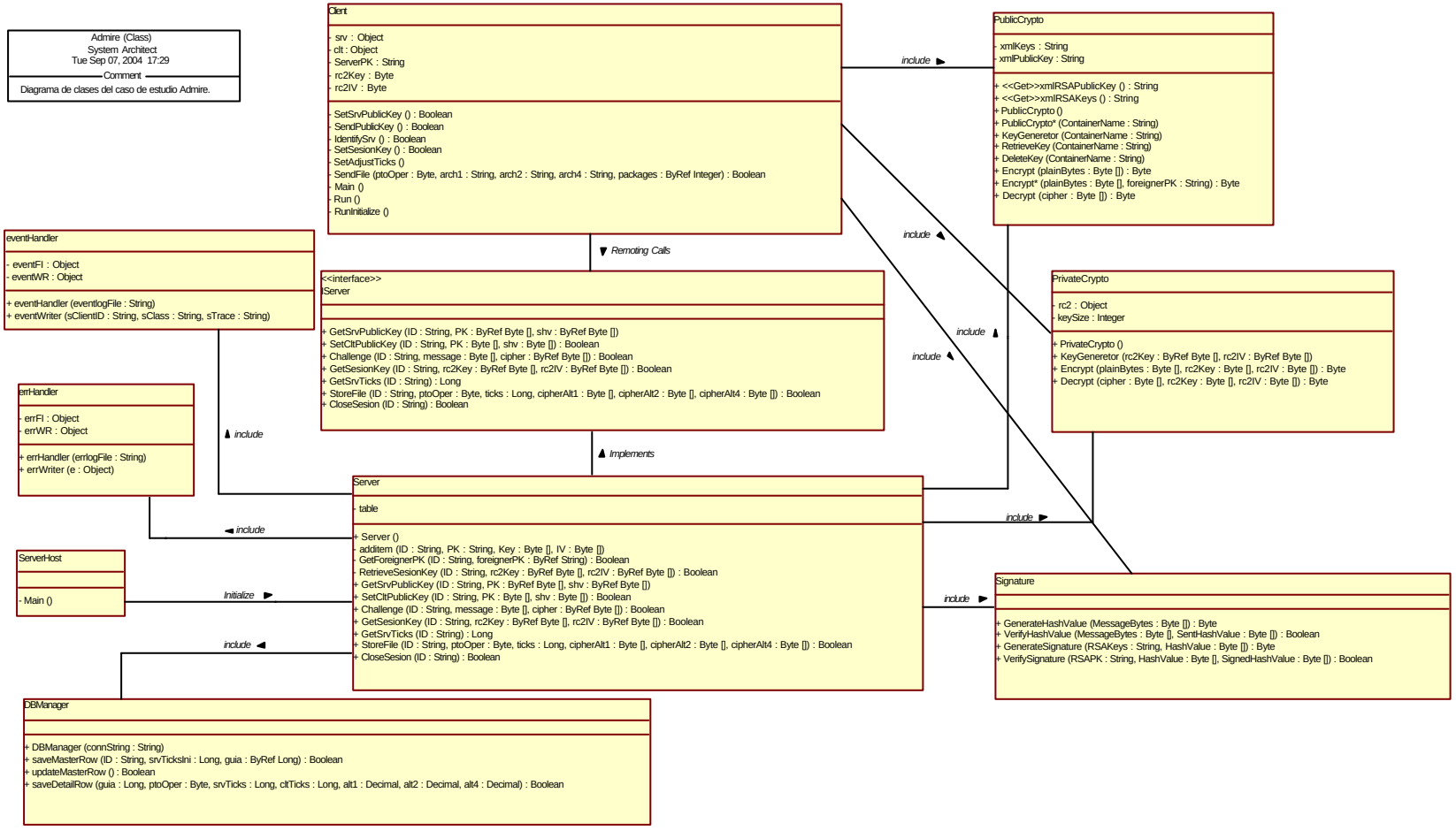

Fig. 14 Diagrama de clase del caso de estudio.

Utilizamos el diagrama de clases UML (Unified Modeling Language) para describir la estructura de las clases, sus identidades, relaciones entre clases y operaciones. A continuación se presenta y describe el diagrama de clase de la implementación del caso de estudio Admire.

Public Crypto

Contiene los métodos necesarios para hacer uso de la clase RSACryptoServiceProvider de la plataforma .NET la cual implementa el algoritmo asimétrico RSA. El acceso a estos métodos es concurrente en el lado del servidor y secuencial en el lado del cliente.

Propiedades.

+xmlRSAPublicKey (): String → Devuelve la llave pública local en formato XML generada con el algoritmo RSA.

+xmlRSAKeys (): String → Devuelve la pareja de llave pública y privada local en formato XML generada con el algoritmo RSA.

Métodos.

+PublicCrypto () → Constructora por defecto

+PublicCrypto(ContainerName: String) → Constructora sobrecargada, invoca al método RetrieveKey.

+KeyGeneretor (ContainerName: String) → Genera localmente la pareja de llaves asimétricas del algoritmo RSA y las almacena en el contenedor con el nombre especificado.

+RetrieveKey (ContainerName: String) → Recupera del contenedor local especificado la pareja de llaves asimétricas del algoritmo RSA; si el contenedor existe recupera la pareja de llaves sino crea una nueva.

+DeleteKey (ContainerName: String) → Elimina del contenedor local especificado la pareja de llaves asimétricas del algoritmo RSA

+Encrypt (plainBytes: Byte[]): Byte[] → Cifra con el algoritmo RSA el mensaje proporcionado en bytes planos haciendo uso de la llave pública local.

+Encrypt (plainBytes: Byte[], foreignerPK: String): Byte[] → Método sobrecargado que cifra con el algoritmo RSA el mensaje proporcionado en bytes planos haciendo uso de la llave pública foránea proporcionada.

+Decrypt (cipher: Byte[]): Byte[] → Descifra el mensaje proporcionado con el algoritmo RSA haciendo uso de la llave privada local.

PrivateCrypto

Contiene los métodos necesarios para hacer uso de la clase RC2CryptoServiceProvider de la plataforma .NET la cual implementa el algoritmo simétrico RC2. El acceso a estos métodos es concurrente en el lado del servidor y secuencial en el lado del cliente.

Métodos.

+PrivateCrypto() → Constructura de la Clase PrivateCrypto. Crea la instancia del objeto rc2 e invoca la constructora de la clase RC2CryptoServiceProvider. Además se establece la longitud de la llave secreta utilizada por el algoritmo RC2 a 128 bits.

+KeyGeneretor (rc2Key: ByRef Byte[], rc2IV: ByRef Byte[]) → Genera localmente la llave simétrica del algoritmo RC2. Invoca a GenerateKey de la clase RC2CryptoServiceProvider que genera una llave aleatoria que la utiliza el algoritmo RC2. Invoca a GenerateIV de la clase RC2CryptoServiceProvider que genera un vector de inicialización aleatorio que lo utiliza el algoritmo RC2.

+Encrypt (plainBytes: Byte[], rc2Key: Byte[], rc2IV: Byte[]): Byte[] → Cifra el mensaje proporcionado en bytes planos con el algoritmo RC2 haciendo uso de la llave y vector de inicialización proporcionados.

+Decrypt (cipher: Byte[], rc2Key: Byte[], rc2IV: Byte[]): Byte[] → Descifra el mensaje proporcionado con el algoritmo RC2 haciendo uso de la llave y vector de inicialización proporcionados.

Signature

Contiene los métodos necesarios para hacer uso de la clase SHA1Managed de la plataforma .NET que permite calcular el valor hash para un conjunto de bytes, además de los métodos necesarios para hacer uso de la clase RSAPKCS1SignatureFormatter que permite calcular la firma digital para un conjunto de bytes. El acceso a estos métodos es concurrente en el lado del servidor y secuencial en el lado del cliente.

Métodos.

+GenerateHashValue (MessageBytes: Byte[]): Byte[] → Crea una nueva instancia de la clase SHA1Managed para calcular el valor hash del conjunto de bytes proporcionados.

+VerifyHashValue (MessageBytes: Byte[], SentHashValue: Byte[]): Boolean → Crea una nueva instancia de la clase SHA1Managed para calcular el valor hash del conjunto de bytes proporcionados y compararlo con en el valor hash proporcionado.

+GenerateSignature (RSAKeys: String, HashValue: Byte[]): Byte[] → Crea una nueva instancia de la clase RSACryptoServiceProvider con la pareja de llaves RSA proporcionada, además crear una instancia de la clase SAPKCS1SignatureFormatter haciendo uso de la

instancia del algortimo RSA la cual contiene la llave privada. Se utiliza el algoritmo SHA1 para calcular la firma digital para el valor hash proporcionado.

+VerifySignature (RSAPK: String, HashValue: Byte[], SignedHashValue: Byte[]): Boolean → Crea una nueva instancia de la clase RSACryptoServiceProvider con la llave pública proporcionada, además crea una instancia de la clase SAPKCS1SignatureFormatter haciendo uso de la instancia del algoritmo RSA la cual contiene la llave pública. Se utiliza el algoritmo SHA1 para verificar la firma digital.

Client

Implementa toda la funcionalidad del lado del cliente. En ésta clase se realiza la lectura de las mediciones de los sensores y se realizan los pasos necesarios para autenticar al servidor y transferir de forma segura esta información hacia el servidor. El acceso a estos métodos es secuencial.

Métodos.

-SetSrvPublicKey(): Boolean → Invoca de forma remota el método GetSrvPublicKey del servidor para obtener su llave pública, verifica la firma digital de ésta y guardar la llave pública del servidor en una variable local.

-SendPublicKey(): Boolean → Genera o recupera la llave pública del cliente, calcula la firma digital de su valor hash e invoca de forma remota el método SetCltPublicKey del servidor para registrar la llave pública del cliente en el servidor.

-IdentifySrv(): Boolean → Comprueba la identidad del servidor, para ello se cifra con la llave pública del servidor un reto que puede ser el identificador del cliente, invoca de forma remota el método Challenge del servidor para comprobar la identidad de éste. Descifra la respuesta obtenida del servidor con la llave privada del cliente y compara éste valor con la información enviada previamente (el identificador del cliente).

-SetSesionKey(): Boolean → Invoca de forma remota el método GetSesionKey del servidor para obtener la llave de sesión, descifra la respuesta del servidor (llave y vector de inicialización) con la llave privada del cliente y guarda los elementos de la llave simétrica (la llave y el vector de inicialización) en variables locales.

-SetAdjustTicks() → Invoca de forma remota el método GetSrvTicks del servidor para obtener los ticks del servidor y calcula el valor de ajuste entre los ticks del servidor y los ticks del cliente.

-SendFile(ptoOper: Byte, arch1: String, arch2: String, arch4: String, packages: ByRef Integer): Boolean → Lee las tres variables de medición de los sensores, las cifra con la llave de sesión e invoca de forma remota el método StoreFile del servidor para registrar las mediciones.

-Main() → Punto de entrada del programa cliente, creación del objeto cliente e invocación de los métodos de control.

-Run() → Crea y registra el canal del comunicación. Obtiene el tipo del objeto localizado en el servidor (interfaz) y crea el proxy.

-RunInitialize() → Control de la ejecución del programa (lógica de secuencia).

IServer

Interfaz de la clase Server.

Métodos.

+GetSrvPublicKey(): Boolean
+SetCltPublicKey(): Boolean
+Challenge(): Boolean
+GetSesionKey(): Boolean
+GetSrvTicks(ID:String): Long
+StoreFile(ID: String, ptoOper: Byte, ticks: Long, cipherAlt1: Byte[], cipherAlt2: Byte[], cipherAlt4: Byte[]): Boolean
+CloseSesion (ID:String): Boolean

Server

Implementa toda la funcionalidad del lado del servidor. En ésta clase se proporcionan los métodos necesarios para establecer el canal seguro de comunicación entre el cliente y el servidor y registrar la información de las lecturas de los sensores en la base de datos. El acceso a éstos métodos es concurrente.

Métodos.

+Server() → Constructora de la clase Server. Inicializa el objeto de la tabla hash (Hashtable), el manejador de errores (errHandler), manejador de eventos (eventHandler) y el administrador de bases de datos (DBManager).

-additem(ID: String, PK: String, Key: Byte[], IV: Byte[]) → Agrega una fila a la tabla hash compuesta por el identificador del cliente (ID), llave pública del cliente (PK), elementos de la llave de sesión llave y vector de inicialización (Key, IV).

-GetForeignerPK(ID: String, foreignerPK: ByRef String): Boolean → Recupera de la tabla hash la llave foránea (foreignerPK) para el identificador del cliente proporcionado (ID).

-RetrieveSesionKey(ID: String, rc2Key: ByRef Byte[], rc2IV: ByRef Byte[]): Boolean → Recupera de la tabla hash los elementos de la llave de sesión; llave (rc2Key) y vector de inicialización (rc2IV); para el identificador del cliente proporcionado (ID).

+GetSrvPublicKey(): Boolean → Recupera la llave pública del servidor del algoritmo asimétrico RSA del contenedor por defecto y la proporciona como parámetro de salida junto con la firma digital para su valor hash.

+SetCltPublicKey(): Boolean → Verifica la firma digital proporcionada para el valor hash de la llave pública del cliente y registra esta llave pública del cliente en la tabla hash del servidor.

+Challenge(): Boolean → Autentica al servidor por medio del reto proporcionado. El servidor descifra con su llave privada el mensaje proporcionado (message) y lo devuelve cifrado con la llave pública del cliente.

+GetSesionKey(): Boolean → Genera los componentes de la llave de sesión; llave (rc2Key) y vector de inicialización (rc2IV), los registra en la tabla hash del servidor y los cifra con la llave pública del cliente para devolverlos como parámetros de salida.

+GetSrvTicks(ID:String): Long → Invoca al método saveMasterRow de la clase DBManager para guardar la información de inicio de sesión en la tabla Maestro Sensores. Se retornan los ticks del servidor al momento que es invoca este método.

+StoreFile(ID: String, ptoOper: Byte, ticks: Long, cipherAlt1: Byte[], cipherAlt2: Byte[], cipherAlt4: Byte[]): Boolean → Descifra las mediciones de los sensores obtenidas (cipherAlt1, cipherAlt2, cipherAlt4) con la llave de sesión recuperada de la tabla hash del servidor por medio del identificador del cliente (ID) e invoca al método saveDetailRow de la clase DBManager para almacenar la información obtenida en la tabla Detalle de Sensores de la base de datos.

+CloseSesion(ID:String): Boolean → Guarda la información de finalización de sesión en la tabla Maestro Sensores y remueve la información del cliente de la tabla hash del servidor.

eventHandler

Esta clase proporciona los métodos necesarios para registrar en un archivo log los eventos que se producen en el lado del servidor. El acceso a estos métodos es secuencial.

Métodos.

+eventHandler (eventlogFile: String) → Constructora de la clase eventHandler. Instancia un objeto de manipulación de archivos con el nombre de archivo proporcionado (eventFile).

+eventWriter (sClientID: String, sClass: String, sTrace: String) → Registra el evento en el archivo log indicando la fecha y hora, identificador de cliente, módulo y pila de llamadas.

errHandler

Esta clase proporciona los métodos necesarios para registrar en un archivo log los errores que se producen al momento que se ejecutan los métodos del servidor. El acceso a estos métodos es concurrente.

Métodos.

+errHandler (errlogFile: String) → Constructora de la clase errHandler. Instancia un objeto de manipulación de archivos con el nombre del archivo proporcionado (errLogFile).

+errWriter (e: Object) → Registra el error en el archivo log indicando la fecha y hora, descripción del error, pila de llamadas.

ServerHost

Esta clase inicializa al objeto servidor, pública el puerto de comunicación.

Métodos.

-Main() → Crea y registra el puerto de comunicación e indica al servidor conectarse al punto de salida (endpoint).

DBManager

Implementa toda la funcionalidad de comunicación con la base de datos necesaria para registrar la información de las lecturas de los sensores en la base de datos. El acceso a éstos métodos es concurrente.

Métodos

+DBManager (connString: String) → Constructora de la clase DBManager. Establece la conexión con la base de datos e inicializa los objetos necesarios para la comunicación con las tablas Maestro Sensores y Detalle Sensores de la base de datos.

+saveMasterRow (ID: String, srvTicksIni: Long, guia: ByRef Long): Boolean → Almacena la información de inicio de sesión en la tabla Maestro Sensores.

+updateMasterRow (): Boolean → Actualiza la información de finalización de la sesión en la tabla Maestro Sensores.

+saveDetailRow (guia: Long, ptoOper: Byte, srvTicks: Long, cltTicks: Long, alt1: Decimal, alt2: Decimal, alt4: Decimal): Boolean → Almacena en la tabla Detalle Sensores la información de las lecturas de los sensores.

V.4. Diagramas de Secuencia.

Utilizamos el diagrama de secuencia UML para representar las interacciones entre objetos. Los diagramas muestran el conjunto de mensajes que intercambian los objetos como parte de su colaboración para alcanzar la operación o resultado deseado. A continuación se presentan y se describen cada uno de los diagramas de secuencia del caso de estudio Admire.

La primera etapa de ejecución del caso de estudio es la etapa de inicialización, en la que el servidor es dado de alta a través de la invocación de clase ServerHost, de esta forma el servidor queda listo para recibir solicitudes de los clientes por medio del canal de comunicación que se ha definido. El servidor es declarado del tipo Singleton lo que conlleva a que nunca haya más de una instancia de objeto servidor en un tiempo dado. La instancia del servidor es creada cuando el cliente realiza por primera vez una llamada de un método del objeto servidor. Si una instancia ya existe, todas las solicitudes de los clientes son atendidas por la misma instancia. Los objetos tipo Singleton tienen por defecto un tiempo de vida asociados a ellos, los clientes no siempre reciben una referencia a la misma instancia de la clase remota, aún cuando nunca haya más de una instancia disponible en un tiempo dado; es por esta razón que una instancia del objeto servidor no puede guardar información sobre los clientes porque cuando su tiempo de vida expira esta información se pierde.

Los clientes se conectan concurrentemente al objeto servidor por medio de la interfaz IServer cada vez que quieren transmitir un conjunto de información y para ello deben llevar acabo la etapa de inicialización y la etapa de transferencia de información.

La etapa de inicialización está compuesta de dos fases: Fase de intercambio de llaves públicas y Fase identificación de entidades y acuerdo de llave de sesión.

La etapa de transferencia de información está compuesta de fases: Fase de transferencia de información y la Fase de visualización de la información.

V.4.1. Fase intercambio de llaves públicas.

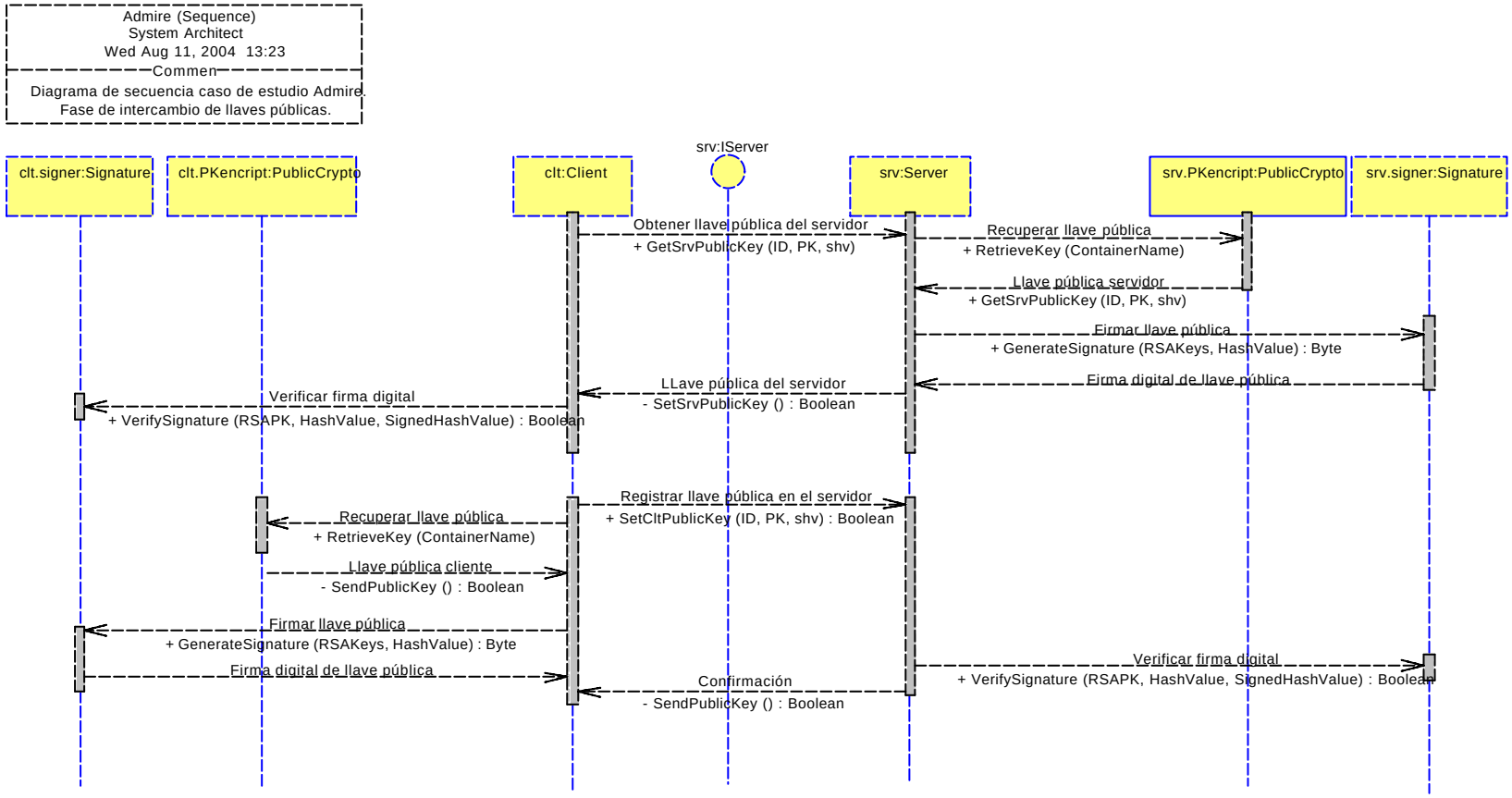

Fig. 15 Primer diagrama de secuencia del caso de estudio.

El primer diagrama de secuencia corresponde a la implementación de la "Fase de intercambio llaves públicas" del esquema de autenticación ver fig. 10.

La primera fase de la etapa de inicialización consiste en el intercambio de llaves públicas; generadas por el algoritmo asimétrico RSA; entre el cliente y el servidor.

El primer paso consiste en que en el cliente solicita la llave pública del servidor y debe realizarse debido a que los clientes no almacenan de forma local información acerca del objeto servidor; para ello se ejecuta el método *SetServPublicKey* en el cliente el cual invoca de forma remota el método *GetSrvPublicKey* del servidor que tiene como parámetros de entrada el identificador del cliente (ID) y como parámetros de salida la llave pública del servidor (PK) y la firma digital del valor hash calculado para la llave pública (PK).

Para dar respuesta a la solicitud del cliente en el método *GetSrvPublicKey* del servidor se invoca al método *RetrieveKey* de la clase PublicCrypto proporcionando el nombre del contenedor en el que se almacena la pareja de llaves asimétricas RSA del servidor; éste método devuelve la llave pública del servidor. El servidor realiza una llamada al método *GenerateSignature* de la clase Signature el cual calcula la firma digital correspondiente al valor hash de la llave pública del servidor.

El servidor finalmente devuelve al método *SetServPublicKey* del cliente por medio de los parámetros de salida su llave pública (PK) y la firma digital calculada para el valor hash de ésta y como valor de retorno un valor que indica si la operación tuvo éxito o no.

Cuando el método *SetServPublicKey* del cliente recibe la respuesta del servidor debe verificar que la firma digital del valor hash (shv) de la llave pública del servidor (PK) sea la correcta, para ello invoca al método *VerifySignature* de la clase Signature el cual comprueba si la firma digital obtenida (shv) corresponde a la llave pública del servidor (PK). De ésta forma el cliente comprueba que la llave pública obtenida proviene del servidor.

El segundo paso consiste en que el cliente registre su llave pública en el servidor esta fase es similar a la anterior solo que se invierten las llamadas. Del lado del cliente se ejecuta el método *SendPublicKey* el cual ínvoca al método *RetrieveKey* de la clase PublicCrypto proporcionando el nombre del contenedor en el que se almacena la pareja de llaves asimétricas RSA del cliente; éste método devuelve la llave pública del cliente. Posteriormente se invoca al método *GenerateSignature* de la clase Signature el cual calcula la firma digital correspondiente al valor hash de la llave pública del cliente. A continuación se invoca de forma remota el método *SetCltPublicKey* proporcionando como parámetros de entrada la identificación del cliente (ID), la llave pública del cliente (PK) y la firma digital para el valor hash de ésta (shv). Del lado del servidor se verifica la firma digital del valor hash de la llave pública del cliente, de frma que se compruebe su veracidad.

V.4.2. Fase identificación de entidades y acuerdo de llave de sesión.

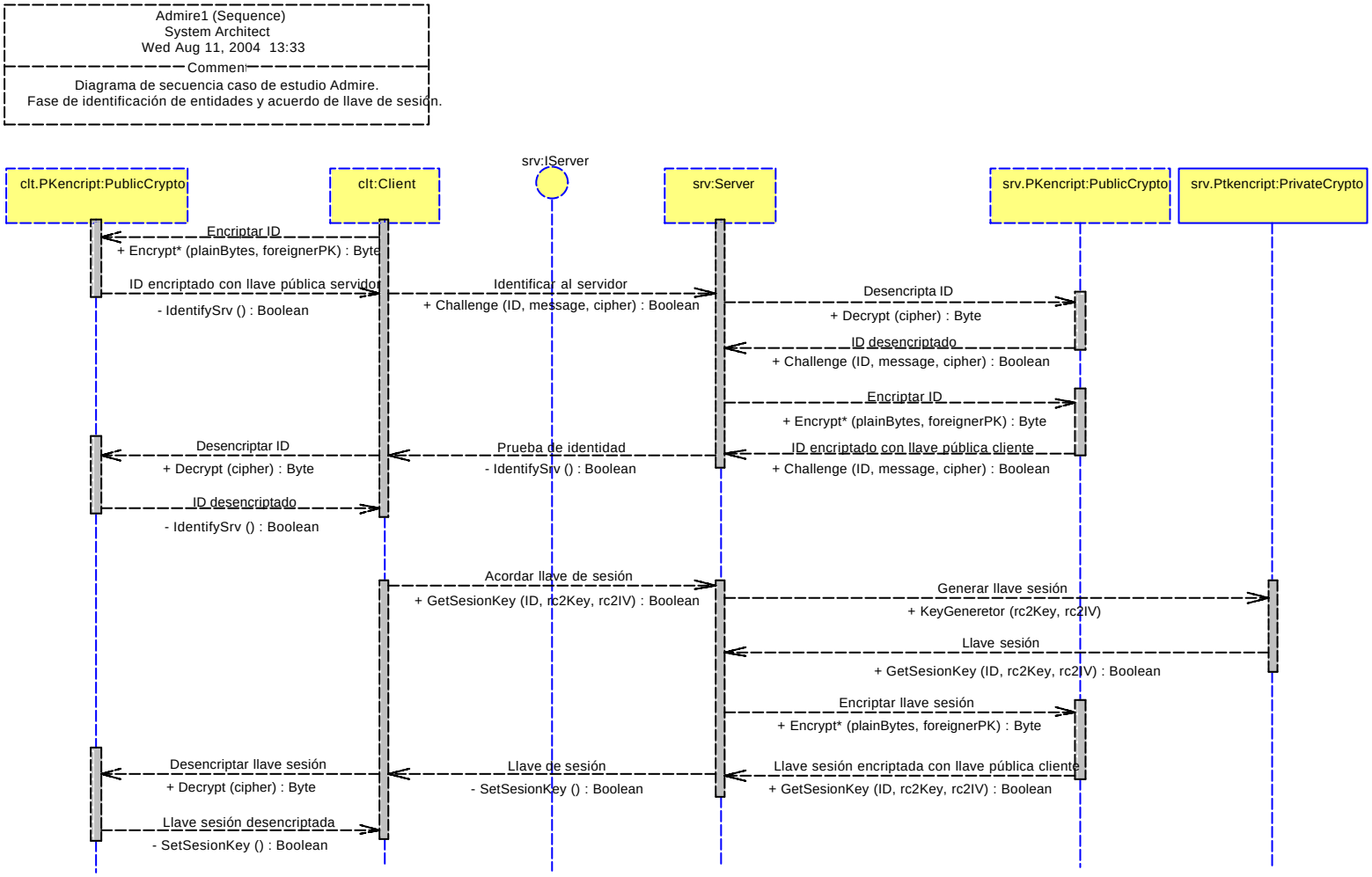

Fig. 16 Segundo diagrama de secuencia del caso de estudio.

El segundo diagrama de secuencia corresponde a la implementación de la "Fase identificación de entidades" y la "Fase acuerdo de llave de sesión" (primer llamado) del esquema de autenticación ver fig. 11 y fig.12 respectivamente.

La segunda fase de la etapa de inicialización consiste en la identificación de las entidades para asegurar que cada entidad es quien dice ser y realizar acuerdo de llave de sesión.

El primer paso consiste en que en el cliente se ejecuta el método *IdentifySrv* en el cual se invoca al método *Encrypt* de la clase PublicCrypto proporcionando el mensaje que se quiere cifrar (identificador) y la llave pública del servidor. El cliente invoca de forma remota el método *Challenge* del servidor que tiene como parámetros de entrada la identificación del cliente (ID), identificador cifrado con la llave pública del servidor (message). El servidor invoca el método *Decrypt* de la clase PublicCrypto para descifrar el identificador recibido haciendo uso de su llave privada, luego invoca el método *Encrypt* de la clase PublicCrypto para cifrar el identificador con la llave pública del cliente, éste valor es devuelto por el método Challenge por medio del parámetro de salida (cipher) y como valor de retorno un valor que indica si la operación tuvo éxito o no.

Del lado del cliente se invoca al método *Decrypt* de la clase PublicCrypto haciendo uso de su llave privada para descifrar el identificador (cipher) lo compara con el identificador que originalmente envió, si son iguales se comprueba la identidad del servidor.

El segundo paso consiste en que en el cliente se ejecuta el método *SetSesionKey* en el cual se invoca de forma remota el método *GetSesionKey* del servidor que tiene como parámetros de entrada el identificador del cliente (ID) y como parámetros de salida los elementos de la llave de sesión cifrados; la llave (rc2Key) y el vector de inicialización (rc2IV). Para dar respuesta a la solicitud del cliente el método *GetSesionKey* del servidor invoca al método *KeyGeneretor* de la clase PrivateCrypto con los parámetros de salida; la llave (iniKey) y el vector de inicialización (iniIV) que conforman la llave de sesión o privada del algoritmo RC2. Posteriormente, invoca al método *Encrypt* de la clase PublicCrypto para cifrar iniKey y iniIV haciendo uso de la llave pública del cliente y los devuelve como los parámetros de salida rc2Key y rc2IV respectivamente y como valor de retorno del método un valor que indica si la operación tuvo éxito o no.

Del lado del cliente se invoca al método *Decrypt* de la clase PublicCrypto para descifrar rc2Key y rc2IV con la llave privada del cliente. De esta forma el cliente obtiene los elementos de la llave de sesión que utilizará para cifrar y enviar la información de los sensores.

V.4.3. Fase de transferencia de información.

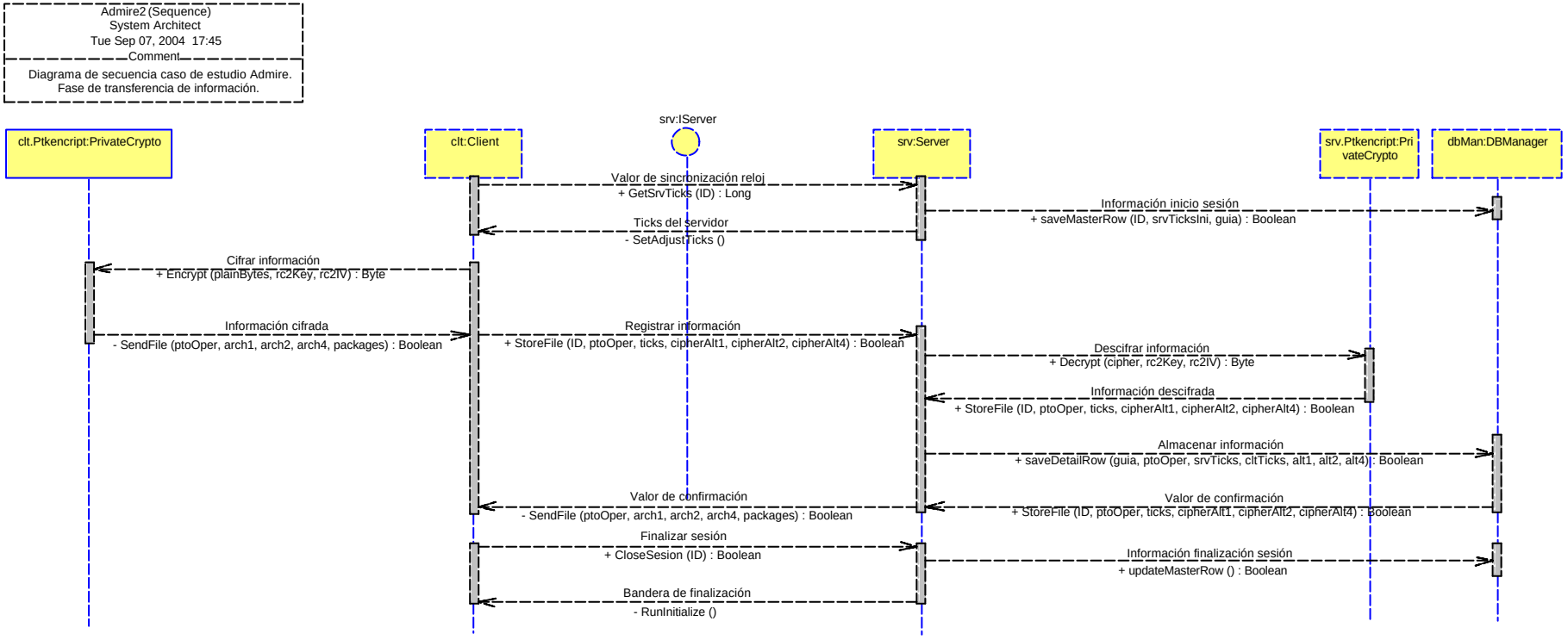

Fig. 17 Tercer diagrama de secuencia del caso de estudio.

El tercer diagrama de secuencia corresponde a la implementación de la "Fase acuerdo de llave de sesión" (segundo y subsecuentes llamados) del esquema de autenticación ver fig. 12.

La primera fase de la etapa de transferencia de información es la transferencia de información la información de los sensores en el cliente hacia el servidor.

El primer paso consiste en que en el cliente se ejecuta el método *SetAdjustTicks* en el cual se ejecuta de forma remota el método *GetSrvTicks* de servidor que guarda la información de inicio de sesión en la tabla Maestro Sensores al invocar el método saveMasterRow de la clase DBManger y además devuelve como valor de retorno un número correspondiente a los ticks que representa la fecha y hora en el servidor. Del lado del cliente se calcula la diferencia entre los ticks de éste y los del servidor, de forma que se pueda sincronizar el reloj del cliente con respecto al del servidor en cada lote de envío.

El segundo paso consiste en que en el cliente se ejecuta el método *SendFile* con los parámetros de entrada el punto de operación ó escenario (ptoOper), la ruta de ubicación del cada una de las fuentes de información (arch1, arch2, arch4) y el parámetro de salida que cuantifica el número de mensajes transferidos (packages). En el método *SendFile* se invoca al método *Encrypt* de la clase PrivateCrypto para cifrar cada una de las lecturas de los sensores y así poderlas transferir al servidor de forma segura. A continuación, se ejecuta de forma remota el método *StoreFile* del servidor que tiene como parámetros de entrada el identificador del cliente (ID), el punto de operación (ptoOper), los ticks del cliente (ticks) y las lecturas de los sensores cifradas (cipherAlt1, cipherAlt2, cipherAlt3). El servidor para dar respuesta a la solicitud del cliente realiza un llamado al método *Decrypt* de la clase PrivateCrypto para descifrar las lecturas de los sensores recibidas, posteriormente solicita que estas variables sean guardadas por medio de la llamada al método *saveRow* de la clase DBManager, éste método tiene como valor de retorno una confirmación, la cual a su vez devuelve el método *StoreFile* del servidor al método *SendFile* del cliente.

Una vez que se han transferido con éxito todos los lotes de información desde el cliente al servidor, en el cliente se ejecuta de forma remota el método *CloseSesion* del servidor que guarda la información de finalización de sesión en la tabla Maestro Sensores al invocar el método updateMasterRow, además éste método *CloseSesion* borra el registro que tiene el servidor sobre la sesión actual del cliente. Éste método tiene como parámetro de entrada el identificador del cliente (ID) y como valor de retorno una confirmación el éxito de la operación.

V.5. Pruebas realizadas.

Los datos de prueba de las mediciones de los sensores recolectados del simulador del modelo admire, se encuentran registrados en seis archivos de texto de formato simple.

Sensor	Escenario 0	Escenario 1	Escenario 2
Sensor de velocidad	alt1.dat	alt1_1.dat	alt1_2.dat
Sensor de ángulo de ataque	alt2.dat	alt2_1.dat	alt2_2.dat
Sensor de rotación del eje vertical	alt4.dat	alt4_1.dat	alt4_2.dat

Tabla 2 Archivos fuentes de la medición de sensores.

Un envío del cliente al servidor consiste de una terna compuesta con un dato para cada medición de los sensores (velocidad, ángulo de ataque y rotación del eje vertical), cada escenario consta de 5,001 mediciones para un total de 15,003 mediciones de la transmisión.

El instante de tiempo en el que el cliente envía los datos y el servidor recibe la información esta medido en ticks. La unidad ticks o pasos mide los valores de hora en unidades de 100 nano-segundos, y una fecha concreta es el número de ticks transcurridos desde las 12:00 de la noche del 1 de enero de 1 d.C. (era cristiana) en el calendario Gregoriano. [HREF 6]

Una sesión dura desde que el cliente envía la primera terna de datos hasta que finaliza el envío de ternas correspondiente a los tres escenarios o falla la el canal de comunicación entre el cliente y el servidor.

Justo después que se lleva acabo la fase de acuerdo de llave de sesión del esquema de autenticación y antes que se inicié la transferencia de los datos de las mediciones de los sensores el cliente invoca el método remoto SetAdjustTicks del servidor el cual le permite registrar el inicio de la sesión y además obtener el valor de ticks del servidor en el instante que se concreta el llamado; al recibir los ticks del servidor como parámetro de salida el cliente calcula un valor de ajuste con respecto a sus ticks y los del servidor. Este valor de ajuste permite que el cliente sincronice su reloj con respecto al del servidor.

Una vez que el cliente ha leído la terna de datos de los archivos y los ha cifrado con la llave de sesión invoca el método remoto StoreFile del servidor y establece en ese instante como parámetro de entrada el valor de sus ticks con el respectivo ajuste. De lado del servidor una vez que se ha descifrado la terna recibida se invoca al método SaveDetailRow de la clase DBManager y en el instante del llamado el servidor establece como parámetro de entrada el valor de sus ticks y el valor de ticks recibidos del cliente. De forma que cuando se guarde en la base de datos la terna, se guarda con ella los ticks del cliente en el momento que realizó el llamado a StoreFile y los ticks del servidor en el momento que realizó el llamado a SaveDetailRow.

Una vez que el cliente termina el envío de datos de los tres escenarios invoca el método remoto del servidor CloseSesion el cual le permite registrar el fin de la sesión. Ver fig. 17 diagrama de secuencia de la fase de transferencia de información.

Para probar el funcionamiento de la implementación del esquema de autenticación se realizaron dos experimentos que consisten en la transferencia de la información del escenario 0, 1 y 2 desde la máquina cliente a la máquina servidor.

En el experimento 1 la transferencia de los 15,003 registros se realizó con una tráfico de red que no afecta el desempeño en tiempo del sistema. Al realizar el experimento 2 se ejecutaron procesos que provocaron aumento del tráfico de red y mayor alternancia del tiempo del procesador, tanto del lado del servidor como del lado del cliente. Como resultado del aumento de carga de red impuesta, la transmisión de los 15,003 registros en el segundo experimento tomó un mayor tiempo.

La tabla siguiente se resume los resultados obtenidos para cada experimento.

Experimento	**Inicio Transmisión**		**Fin Transmisión**	
	Ticks	**Fecha/hora**	**Ticks**	**Fecha/hora**
1	632316318041718750	24/09/2004 02:10:04 p.m.	632316324791875000	24/09/2004 02:21:19 p.m.
2	632321732057656250	30/09/2004 08:33:26 p.m.	632321749247968750	30/09/2004 09:02:04 p.m.

Tabla 3 Detalles de transmisión experimento 1 y 2.

A continuación se presentan las gráficas de la tendencia de la transmisión de los 15,003 registros para los experimentos 1 y 2. En ambas se presenta la diferencia entre el cliente el servidor con respecto al envío y recepción de datos. Ambas mediciones están dadas en ticks.

La gráfica de la tendencia de la transmisión de los 15,003 registros con respecto al tiempo del experimento 1, (fig. 18) permite visualizar que la transmisión se realizó con una tendencia casi lineal de manera que se deduce que el tráfico de red no afecta de manera perceptible para la escala el desempeño del sistema durante el tiempo que duró la transmisión.

En cambio en la gráfica de la tendencia de la transmisión de los 15,003 registros con respecto al tiempo del experimento 2, (fig. 19) permite visualizar que la carga de la red tuvo efecto en la transmisión de los datos provocando un retrazo considerable. Se puede notar que la transmisión de los primero 5,000 registros la tendencia es lineal pero de los 5,000 a los 9,300 se forma una curva cóncava hacia arriba en el momento de mayor tráfico de red; la transmisión de los restantes registros recupera una tendencia lineal.

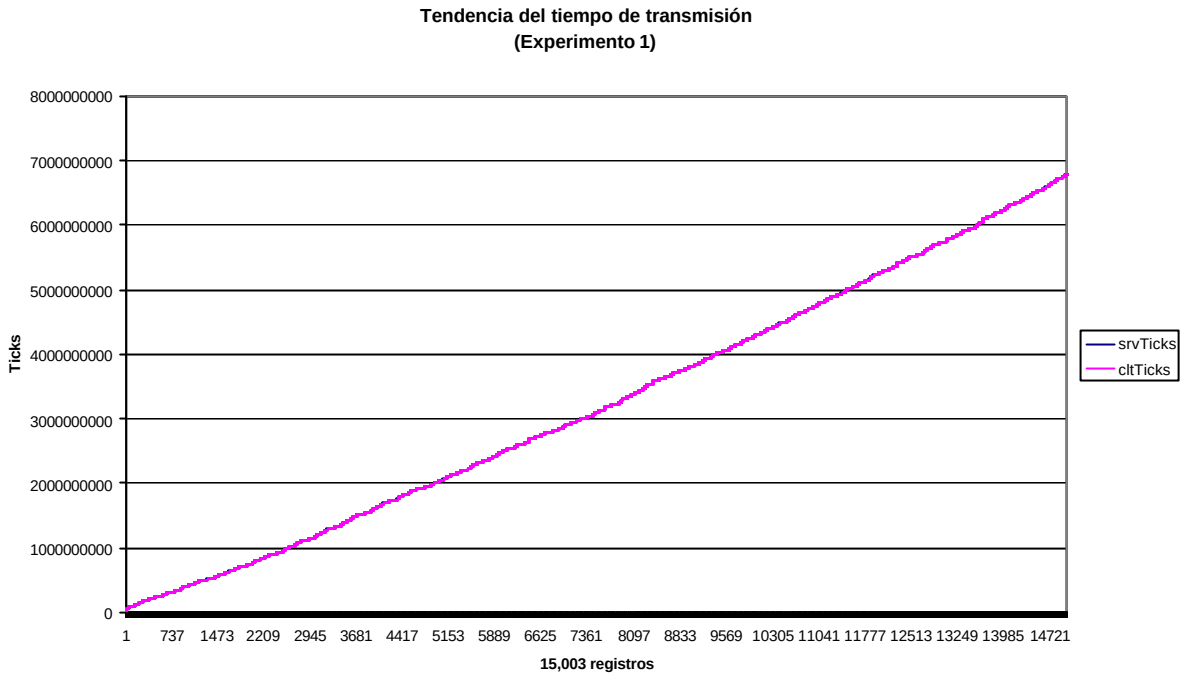

Fig. 18 Tendencia de transmisión experimento 1.

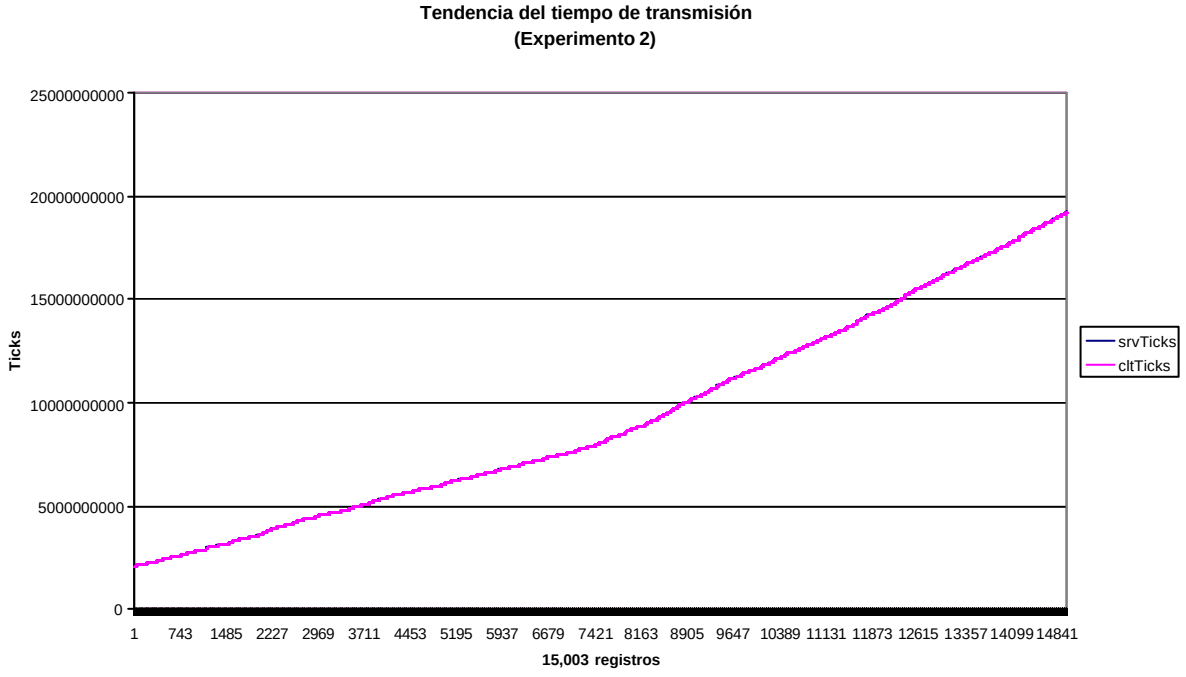

Fig. 19 Tendencia de transmisión experimento 2.

Debido a que el sistema operativo y la arquitectura de comunicación que se está utilizando no es de tiempo real no es posible analizar los elementos que conlleva a tal retrazo de la transmisión de la información, por lo tanto los experimentos tienen como único objetivo validar el funcionamiento del esquema de autenticación y determinar el impacto del tráfico de red en la transmisión de las mediciones de los sensores.

Por lo que la respuesta dinámica del sistema monitoreado tiende a aportar un desplazamiento en tiempo de manera no deseada y sobre todo no real como lo muestran las figuras 20 y 21. Así mismo, se puede observar que el retrazo de tiempo nos lleva a un comportamiento inamisible para cualquier sistema de monitoreo. Como se mencionó anteriormente, esto tiene su base en el tipo de arquitectura "middleware" usada dado que no garantiza una ejecución de procesos en tiempo real. De la misma forma, tanto la comunicación entre computadoras como el monitoreo de procesos llevado a cabo por el sistema operativo requieren de un protocolo en tiempo real que garantice el debido funcionamiento en tiempos acotados.

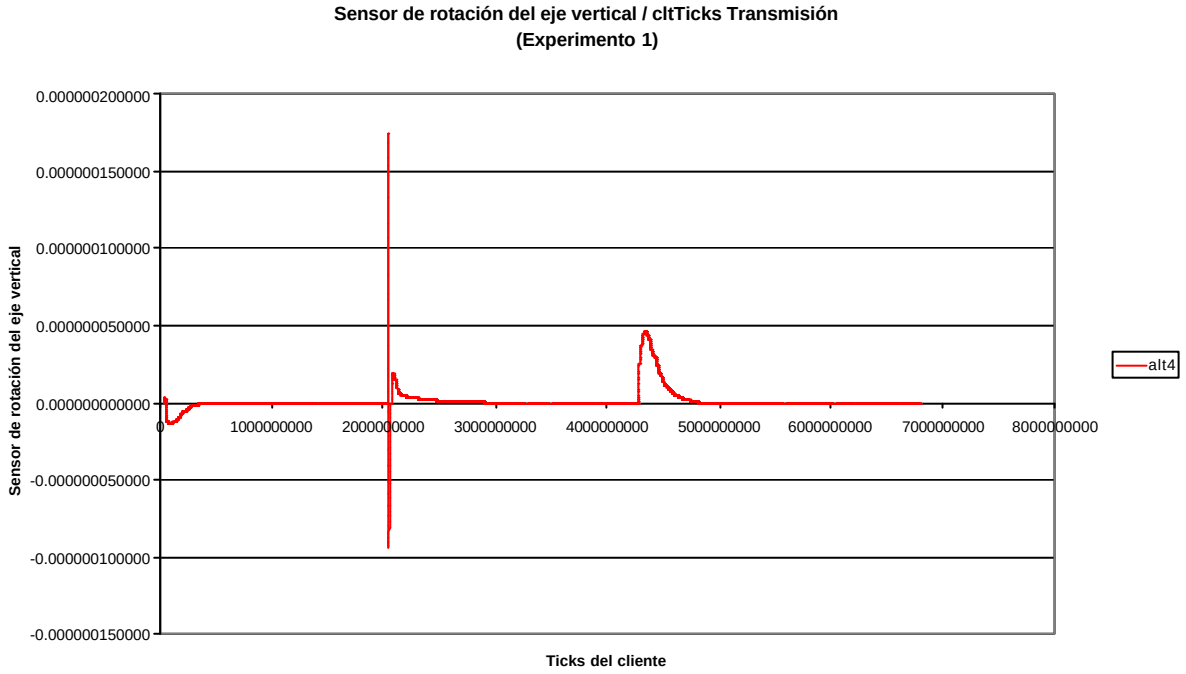

Fig. 20 Sensor de rotación del eje vertical / cltTicks experimento 1.

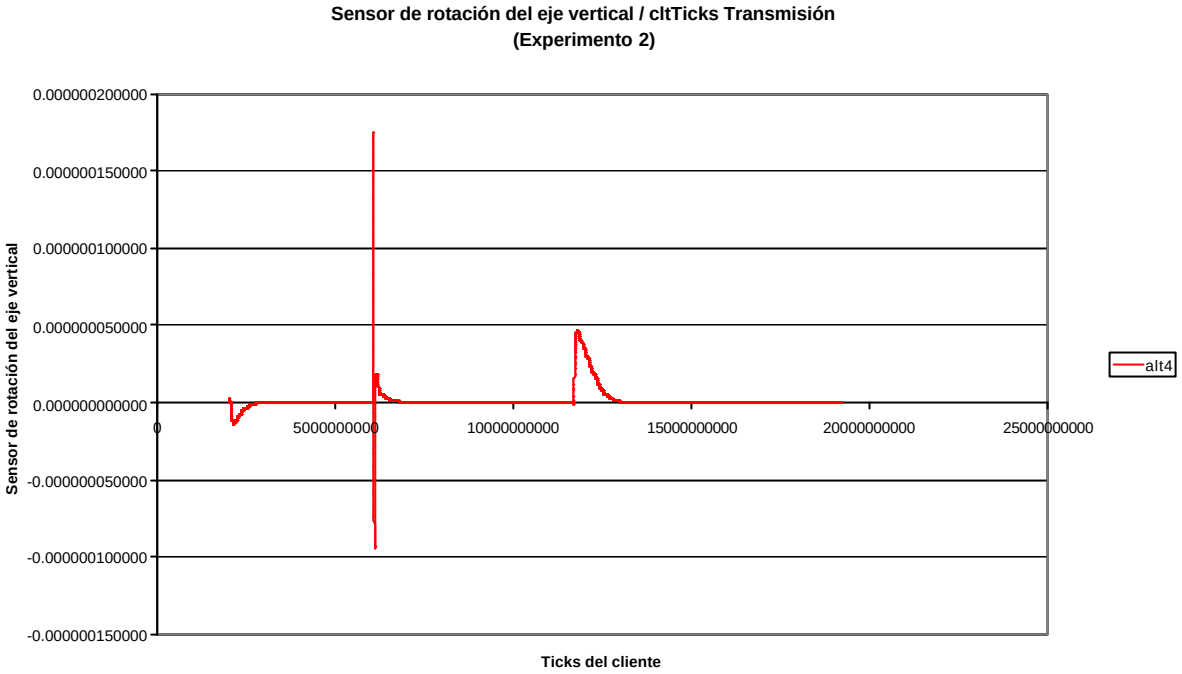

Fig. 21 Sensor de rotación del eje vertical / cltTicks experimento 2.

El principal logro de ésta tesis es establecer un método que nos permita comunicar componentes autónomos de manera segura con base a un "middleware" para sistemas distribuidos tal que, también permita el re-uso tanto del método de comunicación como del método de seguridad. Éste trabajo abre un campo de desarrollo dentro del área de tiempo real para sistemas tipo "middleware" debido a las necesidades mostradas en los resultados que se han comentado.

VI. CONCLUSIONES.

En los últimos años se ha incrementado considerablemente las aplicaciones distribuidas que se ejecutan en el ambiente Web, el cual es un ambiente abierto y hostil que requiere del establecimiento de mecanismos de seguridad. Además se ha identificado que debido a la falta de estándares y normas aún cuando se implementan los mecanismos de seguridad los sistemas presentan serios problemas de inseguridad. También se ha encontrado que los mecanismos de seguridad implementados de manera tradicional en el middleware conllevan a que haya una sobre carga en la comunicación entre las capas y para evitar esta sobrecarga es que se identifican funcionalidades que puedan ser realizadas a nivel de la aplicación.

En esta tesis se trabajó específicamente con la necesidad de proveer un mecanismo de autenticación entre componentes autónomos en base a una comunicación horizontal en el nivel tres (capa de la aplicación) de nuestra arquitectura Web de referencia ver fig. 2. Para lograrlo se ha delegado la ejecución del proceso de identificación de entidades y acuerdo del canal seguro de comunicación a los componentes autónomos involucrados en dicha comunicación, sin que se necesite intervención de un usuario.

Este trabajo pretende ser una guía genérica para facilitar la implementación de mecanismos de autenticación entre componentes autónomos que son parte de una arquitectura de aplicaciones Web distribuidas. Para ello proponemos un esquema de autenticación genérico para aplicaciones Web distribuidas y basadas en componentes.

Además como producto del trabajo desarrolló la librería Encrypt.dll que contiene los métodos necesarios para utilizar los algoritmos de encriptación RSA y RC2 y el algoritmo de firmas digitales SHA en los que se basa el esquema de autenticación propuesto. Esta librería puede ser reutilizada en aplicaciones desarrolladas en la plataforma .NET con sólo agregarla como referencia a un proyecto.

La implementación del prototipo y las pruebas realizadas nos permitieron realizar pruebas iniciales al comportamiento del esquema de autenticación que satisfacen los objetivos de ésta tesis y abren la posibilidad de trabajos futuros.

Con respecto a la plataforma .NET encontramos que proporciona facilidades para la desarrollo de mecanismos de seguridad los cuales pueden implementarse en tiempos razonables en un ambiente de desarrollo que permite la programación en diferentes lenguajes de programación con la posibilidad de ínteroperar con otros sistemas implementados en otras plataformas como por ejemplo J2EE.

Como trabajo futuro se propone establecer la estructura definida en esta tesis dentro de un ambiente en tiempo real, para tal efecto se tendrá que migrar a una plataforma middleware que ofrezca estas características.

Así mismo la implementación de la estrategia de autenticación deberá de ser revisada para definir el alcance de su seguridad así como su complejidad en el cálculo. Cabe señalar que este trabajo permite establecer los alcances del desarrollo de autenticación a nivel de componentes sin tener que modificar la comunicación entre elementos necesariamente seguros.

VII. REFERENCIAS.

[HREF 1] White, D., *Distributed Systems Security*, DBMS, Nov. 1997, http://www.dbmsmag.com/9711d13.html.

[HREF 2] Clikear.com – Portal para desarrolladores .NET Framework, http://www.clikear.com/manual_csharp/index.asp.

[HREF 3] The MSDN Library, http://msdn.microsoft.com/library/

[HREF 4] Microsoft ASP.net, Tutorials, http://www.asp.net/Tutorials/quickstart.aspx

[HREF 5] Garteour Project, 2001, http://www.foi.se/admire

[HREF 6] Msdn en español, http://www.microsoft.com/spanish/msdn/mexico/default.asp

[HREF 7] Rivest, R.L., *A Description of the RC2TM*, Encryption Algoritm File, draft-rivest-rc2des-00.txt, ftp://ftp.ietf.org/internet-drafts/.

---

[1] Monroe, R., Garlan D., *Style-Based Reuse for Software Architectures*, IEEE 4th International Conference on Software Reuse (ICSR '96), Apr. 23-26, 1996, Orlando, Florida, pág. 84-93.

[2] Booch, Rumbaugh, and Jacobson, *The UML Modeling Language User Guide*, Addison-Wesley, 1999.

[3] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Addison-Wesley Professional, 1998.

[4] Arch-int, S., Batanov, D., *Development of industrial information systems on the Web using business components*, Computers in industry 50, 2003, pág. 231-250.

[5] Kirova, V., Rossak, W., *Representing Architectural Designs: A Central Issue in the Development of Complex Systems*, IEEE 1st International Conference on Engineering of Complex Computer Systems, Nov. 06-10, 1995, Ft. Lauderdale, Florida, pág. 80-87.

[6] Stolle, R., Rossak, W., Kirova, V., *A Component-driven Architecture for Internet-Based, Directly Reactive Information Systems*, IEEE 7th International Conference and Workshop on the Engineering of Computer Based Systems, Apr. 03-07, 2000, Edinburgh, Scotland, pág. 129.

[7] Manuel P., AlGhamdi J., *A data-centric design for n-tier architecture*, Elsevier, Information Sciencies 150, 2003.

[8] Fielding Roy T., Taylor Richard N., *Principled Design of the Modern Web Architecture*, ACM Transactions on Internet Technology, Vol. 2, May 2002, Pages 115-150.

[9] Lee, S., Shirani, A., *A component based methodology for Web application development*, Elsevier, The Journal of Systems and Software, 2003.

[10] Probst, S., Essmayr, W., Weippl E., *Reusable Components for Developing Security-Aware Applications*, IEEE 18th Annual Computer Security Applications Conference (ACSAC'02), Dic. 09-13, 2002, San Diego California, pág. 239.

[11] Bidan, C., Issarny V., *Security Benefits from Software Architecture*, Institut National de Recherche en Informatique et en Automatique, No 3114, Feb. 1997, France.

[12] *Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication*, Microsoft Press, Redmond Washington, 2003.

[13] Popescu, B., Steen M., Tanenbaum, A., *A Security Architecture for Object-Based Distributed Systems*, IEEE 18th Annual Computer Security Applications Conference, Dic. 09-13, 2002, San Diego, California, pág. 161.

[14] Probst, S., Essmayr, W., Weippl E., *Reusable Components for Developing Security-Aware Applications*, IEEE 18th Annual Computer Security Applications Conference (ACSAC'02), Dic. 09-13, 2002, San Diego California, pág. 239.

[15] Bressoud, D., Wagon, S, *A Course in Computational Number Theory*, Key College Publishig. Innovator in Higher Education, in cooperation with Springer-Verlag, (1999).

[16] Delfst, H., Knebl, H., *Introduction to Cryptography, Principles and Applications*, Springer-Verlag, Berlin, Heidelberg, (2002).

[17] Hasse, H., *Number Theory*, Springer-Verlag, Berlin, Heidelberg New York, (1980).

[18] Ireland, K., Rosen, M., *A Classical Introduction to Modern Number Theory*, Springer-Verlag, Berlin, Heidelberg New York (1982).

[19] Niven, I., Zuckerman, H., Montgomery, H., *An Introduction to the Theory of Numbers*, John Willey and Sons Inc. (1991).

[20] Rivest, R.L., *A Description of the RC2TM*, Encryption Algoritm File, **draft-rivest-rc2des-00.txt**, available from ftp://ftp.ietf.org/internet-drafts/.

[21] Schneier, B., *Applied Cryptography. Protocols, Algorithms and Source Code in C*, John Willey and Sons Inc. (1996).

[22] Stephen W., *The MATHEMATICA Book*, Wolfram Media, Cambridge University Press (1999).

[23] Menezes, A. J., van Oorschot, P.C., Vanstone, S.A. *Handbook of Applied Cryptography*, CRC Press series on Discrete Mathematics and its applications. (1996)

[24] Rose H. E., *A Course in Number Theory*, Clarendon Press, Oxford, 1994.

APENDICE 1

Cuestiones básicas

Funciones

Sean A, B conjuntos no vacíos. Denotaremos por $f: A \to B$, la función f con dominio A y contradominio B.

El conjunto $f(A) := \{f(a) | a \in A\}$ denota la imagen del conjunto A bajo la función f. Dom(f), denota el dominio de f.

Sea $C \subseteq B$, el conjunto $f^{-1}(C) := \{a \in A | f(a) \in C\}$ se llama la imagen inversa del conjunto C bajo la función f.

Definición: Sea $f: A \to B$ función.

[a] La función f se dice inyectiva, si siempre que $x \neq x'$ se tiene que $f(x) \neq fx')$, para cada $x, x' \in A$.

[b] La función f se dice suprayectiva si para cada $b \in B$ existe algún $a \in A$ tal que $f(a) = b$.

[c] La función f se dice biyectiva, si f es inyectiva y suprayectiva.

[d] Sea $f : A \to B$ una función biyectiva. Se tiene la existencia de la función inversa de f, denotada por f^{-1} la cual satisface que $f^{-1} : B \to A$ y se cumple que $f(x) = y \Leftrightarrow x = f^{-1}(y)$.

[e] Sean X un conjunto no vacío y $g : X \to X$ función. La función g se dice permutación sobre X si g es función biyectiva.

Complejidad

Definición: El número de operaciones de bit necesarias para completar la realización de un algoritmo se llama complejidad computacional del algoritmo.

Definición: Sean $g, f : A \subseteq \mathbb{R} \to \mathbb{R}^+$ funciones. Si existe $c \in \mathbb{R}^+$ tal que $f(x) < cg(x)$ para todo $x \in A$ suficientemente grande, entonces escribimos $f(x) = O(g(x))$, o simplemente $f = O(g)$. "O" grande es el orden de magnitud de la complejidad de un algoritmo, es una cota superior sobre el número de operaciones de bit requeridas para la ejecución de un algoritmo en el peor escenario posible.

Definición: El tiempo de ejecución de un algoritmo $\mathcal{L}$ será de tiempo polinomial si la complejidad de $\mathcal{L}$ es $O(n^c)$ para alguna constante $c \in \mathbb{R}^+$ donde n es la longitud en bits del input para $\mathcal{L}$. Los algoritmos en tiempo polinomial son los algoritmos eficientes.

Definición: Un problema se dice computacionalmente difícil si ningun algoritmo en tiempo polinomial puede posiblemente resolverlo.

Definición: Un problema se dice computacionalmente fácil si puede ser resuelto usando algún algoritmo en tiempo polinomial.

Definición: Un problema se dice muy fácil, si el problema posee un algoritmo operando en tiempo polinomial bajo.

Definición: Sea $f : A \to B$ función. La función f se dice función en un sentido si es computacionalmente fácil calcular $f(a)$ para cada $a \in A$, pero dado $b \in \text{Im}(f) = \{f(x)|x \in A\}$ es computacionalmente difícil encontrar $a \in A$ tal que $f(a) = b$.

Enfatizemos en este punto que no se conocen funciones criptográficas de un sentido. Muchas funciones criptográficas f son conocidas tales que

[i] Es fácil calcular $f(x)$ a partir de x.

[ii] El cálculo de x a partir de $f(x)$ es computacionalmente difícil.

Sin embargo, para las funciones criptográficas $f(x)$ conocidas, no hay prueba conocida de lo afirmado en [ii].

Definición: Sea $f : A \to B$ función. La función f se dice función TOW, si es computacionalmente fácil calcular $f(a)$ para cada $a \in A$, y existe cierta información T que hace computacionalmente fácil para cada $b \in \text{Im}(f) = \{f(x)|x \in A\}$ encontrar $a \in A$ tal que $f(a) = b$.

Definición: Sean $a, b \in \mathbb{Z}$. Diremos que el entero b *divide* al entero a, denotado $b|a$ si existe un entero $c \in \mathbb{Z}$ tal que $a = bc$. Denotaremos por $gcd(a,b)$ el máximo común divisor de a y b.

Definición: Un *número primo* es un número natural p que tiene exactamente 4 divisores, a saber $\pm 1, \pm p$. Denotaremos el conjunto de números primos por $\mathbb{P}$.

Ejemplo: Usando Mathematica calculamos los primos entre 2 y 100, con un paso de 3 en el orden natural entre primos.

```
In[1]:= Table[Prime[i],{i,1,10^2,3}]}
Out[1]= { 2,7,17,29,41,53,67,79,97,107,127,139,157,173,191,199,227,239,257,271,
        283,311,331,349,367,383,401,421,439,457,467,491,509,541}
```

Proposición: **Algoritmo de la división**. Sean $a \in \mathbb{N}$ y $b \in \mathbb{Z}$ con $a > 0$. Entonces existen enteros únicos $q, r \in \mathbb{Z}$ tales que $0 \leq r < a$ y $b = aq + r$. Si $a \nmid b$ entonces r satisface la desigualdad más fuerte $0 < r < a$.

Demostración: [19], p. 5. □

El entero q se llama cociente y r se llama el residuo en la división de b por a.

$In[2] :=$ FullGCD$[215171279, 9182175,$ Labels $\to$ True$]$

$Out[2]//DisplayForm =$

Remainders	*Quotients*
$r_0 = a = 215171279$	
$r_1 = b = 9182175$	$q_1 = 23$
$r_2 = 3981254$	$q_2 = 2$
$r_3 = 1219667$	$q_3 = 3$
$r_4 = 322253$	$q_4 = 3$
$r_5 = 252908$	$q_5 = 1$
$r_6 = 69345$	$q_6 = 3$
$r_7 = 44873$	$q_7 = 1$
$r_8 = 24472$	$q_8 = 1$
$r_9 = 20401$	$q_9 = 1$
$r_{10} = 4071$	$q_{10} = 5$
$r_{11} = 46$	$q_{11} = 88$
$r_{12} = 23$	$q_{12} = 2$
$r_{13} = 0$	

Proposición: **Algoritmo euclideano en** $\mathbb{Z}$.

Sean $a, b \in \mathbb{Z}$ tales que $a \geq b > 0$. Definimos $r_{-1} := a$ y $r_0 := b$. Aplicando repetidamente el algoritmo de la división, construimos $r_{j-1} := r_j q_{j+1} + r_{j+1}$, con $0 < r_{j+1} < r_j$ para todo $0 \leq j < n$, donde n es el mínimo entero no negativo tal que $r_{n+1} = 0$ en cuyo caso $gcd(a,b) = r_n$.

Demostración: [19], p. 11. □

Hay un método conocido como algoritmo euclideano extendido el cual para calcular los enteros $x, y \in \mathbb{Z}$ tales que $gcd(a,b) = ax + by$. Notemos que si $gcd(a,b) = 1$ entonces el algoritmo euclideano extendido nos permite calcular el inverso de a módulo b, pues si para algunos $x, y \in \mathbb{Z}$ se tiene que $ax + by = 1$ entonces $ax \equiv 1 \pmod{b}$ y por tanto x será el inverso multiplicativo de a módulo b.

Proposición: **Algoritmo euclideano extendido en** $\mathbb{Z}$**.** Sean $a, b \in \mathbb{N}$ y para $i = 1, 2, \ldots, n+1$ sean q_i los cocientes obtenidos a partir de la aplicación del algoritmo euclidiano para encontrar $gcd(a,b)$ donde n es el mínimo entero no negativo tal que $r_{n+1} = 0$. Si $s_{-1} = 0$, $s_0 = 0$ y $s_i = s_{i-2} + q_{n-i+2} s_{i-1}$ para $i = 1, 2, \ldots, n+1$. Entonces $gcd(a,b) = a s_{n+1} + b s_n$.

Demostración: [19], p. 11. □

Ejemplo: Usando Mathematica, calculamos los enteros s_{n+1}, s_n mencionados en la definición anterior, dados los enteros $a = 11113412408741378431231274132749321074712387482$, $b = 12433243210973434131319891$.

```
In[3]:= ExtendedGCD[11113412408741378431231274132749321074712387482,\\
                                        12433243210973434131319891] }\\
Out[3]= {1,{350694042426038819473065O,-3134666841656736213912656192708642233 5\\
                76383089}}
```

Esto significa que

350694042426038819473065O(111134124087413784312312741327493210747123874 82)-

313466684165673621391265619270864223357638308 9(124332432109734341313198 91)=1

Teorema 1 (Teorema Fundamental de la Aritmética). *Sea n un número natural mayor que 1. Entonces n se puede expresar como un producto de potencias de primos* $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_s^{\alpha_s}$.

Demostración: [17], p. 3. □

Ejemplo: Usando Mathematica ejemplificamos la factorización de un enteros con al menos un factor primo pequeño.

```
In[4]:= FactorInteger[88828813969173994843998638822828985807281810752713600 41\\
                      167580525961371954774774179920589]
Out[4] =  79^{17}  8053^3  1299841^5  252097800623
```

Observación. El teorema 1 es un teorema de tipo existencial. Dice que dado $n \in N$ existe una descomposición de $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_s^{\alpha_s}$, sin embargo no dice como construir los factores primos p_i.

Función phi de Euler

Dado $n \in \mathbb{N}$, una útil función de conteo es la llamada función phi de Euler $\phi(n)$.

Definición: Para todo entero $n \geq 1$, $\phi(n)$ denota el número de enteros a tales que $1 \leq a \leq n$ y $gcd(a,n) = 1$. Esto es, $\phi(n) = \#(\{a \mid 1 \leq a \leq n,\ \ gcd(a,n) = 1\})$.

Ejemplo: Usando Mathematica, calculamos el valor de la función de Euler $\phi(n)$ para $n = 1, 2, \ldots, 49, 54$

```
In[5]:= FunctionTable[EulerPhi, 45, Columns -> 9]
Out[5]//DisplayForm=
```

n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$
1	1	6	2	11	10	16	8	21	12	26	12	31	30	36	12	41	40
2	1	7	6	12	4	17	16	22	10	27	18	32	16	37	36	42	12
3	2	8	4	13	12	18	6	23	22	28	12	33	20	38	18	43	42
4	2	9	6	14	6	19	18	24	8	29	28	34	16	39	24	44	20
5	4	10	4	15	8	20	8	25	20	30	8	35	24	40	16	45	24

Propiedades de la función phi de Euler.

Proposición 1.

[i] *Si p es número primo entonces* $\phi(p) = p - 1$.

[ii] *Sea p un número primo y* $n \in \mathbb{N}$. *Entonces* $\phi(p^n) = p^n - p^{n-1}$.

[iii] *Sean m y n son enteros primos relativos. Entonces* $\phi(mn) = \phi(m)\phi(n)$. *La función de Euler es una función multiplicativa.*

[iv] *Sea* $m = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ *la factorización en primos de* m. *Entonces* $\phi(m) = m\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\left(1-\frac{1}{p_k}\right)$.

[v] *Si n es un entero positivo mayor que 2 entonces* $\phi(n)$ *es par.*

[vi] *Sea n un entero positivo. Entonces* $\sum_{d|n} \phi(d) = n$.

Demostración: [19], p. 69. [17], p 30 □

Definición: Sean $a, b, m \in \mathbb{Z}$ con $m > 0$. Diremos que *a es congruente con b* módulo *m*, denotado por $a \equiv b \pmod{m}$, si *m* divide a $a-b$, esto es, si $a-b = km$ para algún $k \in \mathbb{Z}$. Pondremos $a \not\equiv b \pmod{m}$ si *a* no es congruente con *b* módulo *m*.

Ejemplo: Usando Mathematica mostramos que $131 \equiv 5 \pmod{7}$ y que la clase de equivalencia de 998638822828985807281810752713600411675805259613719547774 17 módulo 288139691739948439986 es 98176818738163667197

```
In[6]:= Mod[131, 7]
Out[6]= 5
In[7]:= Mod[998638822828985807281810752713600411675805259613719547774 17,
            288139691739948439986]
Out[7]= 98176818738163667197
```

Proposición 2.

Sean $a, b, m \in \mathbb{Z}$ con $m > 0$.

[a] [i] $a \equiv a \pmod{m}$ (Reflexividad).
[ii] Si $a \equiv b \pmod{m}$ entonces $b \equiv a \pmod{m}$ (Simetría).
[iii] Si $a \equiv b \pmod{m}$ y $b \equiv c \pmod{m}$ entonces $a \equiv c \pmod{m}$ (Transitividad).

[b] Dado $n \in \mathbb{Z}$, se tiene que en $\mathbb{Z}$ la relación de congruencia módulo *n* es una relación de equivalencia.

[c] Si $a \equiv b \pmod{m}$ y $c \equiv d \pmod{m}$ entonces $ax + cy \equiv bx + dy \pmod{m}$.

[d] Si $a \equiv b \pmod{m}$ y $c \equiv d \pmod{m}$ entonces $ac \equiv bd \pmod{m}$.

[e] Si $a \equiv b \pmod{m}$ entonces $a^k \equiv b^k \pmod{m}$, para todo $k \in \mathbb{N}$.

[f] Si $a \equiv b \pmod{m}$ y $d \mid m$, $d > 0$ entonces $a \equiv b \pmod{d}$.

Demostración: [19], p. 48, 49. □

Conjunto de clases de equivalencia $\mathbb{Z}_n := \{[0], [1], [2], \ldots, [n-1]\}$**:** Por la proposición 2, sobre el conjunto de los enteros la relación de congruencia módulo *n* es una relación de equivalencia. Por tanto el conjunto de los enteros se particiona en clases de equivalencia, denotemos por $[a]$ la clase de equivalencia que contiene al elemento $a \in \mathbb{Z}$.

Tenemos que dado $a, n \in \mathbb{Z}$ la clase de equivalencia de *a* módulo *n* es $[a] := \{nk + r \mid k \in \mathbb{Z}\}$ y donde *r* es el residuo resultante en la división de *a* por *n*, esto es, $a = nq + r$, con $0 \leq r \leq n-1$.

El conjunto de clases de equivalencia módulo n se denota por $\mathbb{Z}_n$ y se tiene que $\mathbb{Z}_n := \{[0],[1],[2],\ldots,[n-1]\}$.

Ejemplo: La clase de equivalencia de 15 módulo 8 es:

$$[15] = \{\ldots -65,-57,-49,-41,-33,-25,-17,-9,-1,7,15,23,31,39,47,55,63,71,79\ldots\}$$

Definición: Un *grupo* será una estructura algebraica $(G,\circledast)$, donde G es conjunto y $\circledast$ es una operación binaria sobre G, la cual satisface:

[a] Para todo $a,b \in G$ se tiene que $a \circledast b$ y $b \circledast a \in G$. (Cerradura).
[b] $a \circledast (b \circledast c) = (a \circledast b) \circledast c$ para todo $a,b,c \in G$. (Asociatividad)
[c] Existe un único $e \in G$ tal que $e \circledast a = a \circledast e = a$ para todo $a \in G$. (Existencia del elemento neutro e).
[d] Para cada $a \in G$ existe un único $a^{-1} \in G$ tal que $a \circledast a^{-1} = a^{-1} \circledast a = e$. (Existencia de inverso).

Si sucede que en el grupo $(G,\circledast)$, la operación $\circledast$ es conmutativa, esto es $g_1 \circledast g_2 = g_2 \circledast g_1$ para todo $g_1, g_2 \in G$, diremos que el grupo $(G,\circledast)$ es grupo abeliano.

Ejemplo:

[a] Sea $X \neq \emptyset$ un conjunto finito. Entonces el conjunto de permutaciones sobre X con la operación de composición $\circ$, es un grupo finito no abeliano. Este grupo se denota por S_X y es llamado el grupo simétrico sobre X.
[b] Sea $+$ la operación de suma usual en los enteros, entonces $(\mathbb{Z},+)$ es grupo abeliano infinito. Con la suma usual, tenemos que $(\mathbb{Q},+)$, $(\mathbb{R},+)$, $(\mathbb{C},+)$ son grupos abelianos infinitos.

Grupo $(\mathbb{Z}_n,+,[0])$**.** Al conjunto $\mathbb{Z}_n$ se le dota de estructura de grupo aditivo, donde la operación de suma se define por $[a]+[b]=[a+b]$ (Suma modular). Se tiene que $(\mathbb{Z}_n,+,[0])$ es grupo abeliano finito.

Anillo $(\mathbb{Z}_n,+,\cdot,[0],[1])$**.** Al conjunto $\mathbb{Z}_n$ se le dota de estructura de anillo conmutativo con unidad, denotado por $(\mathbb{Z}_n,+,\cdot,[0],[1])$, donde las operaciones de suma $(+)$ y producto $(\cdot)$, se definen por: $[a]+[b]=[a+b]$ (Suma modular), $[a]\cdot[b]=[ab]$ (Multiplicación modular).

Por abuso de notación, a la clase de equivalencia $[a]$ se le quitan los corchetes y se pone $\mathbb{Z}_n = \{0,1,2,\ldots,n-1\}$, esto es, si no hay peligro de confusión y estando en el contexto adecuado, se tendrá que i indica la clase de equivalencia $[i]$. Si no se indica lo contrario al tratar con $\mathbb{Z}_n$ entenderemos que estamos trabajando con su estructura de anillo conmutativo con unidad.

Aritmética modular: aritmética en el anillo $\mathbb{Z}_n$

La ventaja de la aritmética modular en $\mathbb{Z}_n$, es que no es necesario manipular números grandes. El tamaño de los números resultantes en las operaciones, depende de la base modular n.

[i] $[a] \pm [b] \pmod{n}$ (Cerradura aditiva modular)

[ii] $[a][b] = [ab] \pmod{n}$ (Cerradura multiplicativa modular)

[iii] $[a] + [b] = [b] + [a]$ (Conmutatividad de la suma modular)

[iv] $[a] + ([b] + [c]) = ([a] + [b]) + [c]$ (Asociatividad de la suma modular)

[v] $[0] + [a] = [a] + [0] = [a]$ (Neutro aditivo)

[vi] $[a] + [-a] = [-a] + [a] = [0]$ (Inverso aditivo)

[vii] $[a][b] = [b][a]$ (Conmutatividad de la multiplicación modular)

[viii] $([a][b])[c] = [a]([b][c])$ (Asociatividad de la multiplicación modular)

[ix] $[1][a] = [a][1] = [a]$ (Identidad multiplicativa modular)

[x] $[a]([b] + [c]) = [a][b] + [a][c]$ (Distributividad modular)

Definición: Sean $a, b, m \in \mathbb{Z}$ con $m > 0$. La operación $a^b \pmod{m}$ se llama *exponenciación modular*.

Definición: Sea g un elemento de un grupo finito $(G, *)$. El orden de g denotado por $o(g)$ será el mínimo natural $r > 0$ satisfaciendo que $g^r = 1$, donde 1 es el elemento neutro de G.

Ejemplo. Orden de un elemento en el anillo $\mathbb{Z}/n\mathbb{Z}$. Usando Mathematica, el orden de 98211313112311311311111111789473478897273121 en $\mathbb{Z}_{7112311123131443151211}$ es 142246222455904218416 y este valor se calcula por:

```
In[8]:= MultiplicativeOrder[98211313112311311311111111789473478897273121,
                            7112311123131443151211]
Out[8]= 142246222455904218416
```

Definición: Un elemento $a \in \mathbb{Z}_n$ se dice *unidad* en $\mathbb{Z}_n$, si existe $b \in \mathbb{Z}_n$ tal que $ab \equiv 1 \pmod{n}$. En este caso, se dice que b es el *inverso multiplicativo* de a módulo n. Se pone $b = a^{-1}$. El *grupo de unidades* de $\mathbb{Z}_n$ se denota por $\mathbb{Z}_n^*$.

Ejemplo: Usando Mathematica tenemos que el grupo de unidades $\mathbb{Z}_{84}^*$ tiene 24 elementos y este grupo es:

In[9]:= $\mathbb{Z}_{84}^*$

```
Out[9]=  {1, 5, 11, 13, 17, 19, 23, 25, 29, 31, 37, 41, 43, 47, 53, 55, 59,
           61, 65, 67, 71, 73, 79, 83}
In[10]:= EulerPhi[84]
Out[10]=24
```

Proposición 3. *Se tiene que $a \in \mathbb{Z}_n$ es unidad en $\mathbb{Z}_n$, si y sólo si $gcd(a,n) = 1$ y $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n | gcd(a,n) = 1\}$.*

Demostración: [18], p. 33 □

Proposición 4. *$\#(\mathbb{Z}_n^*) = \phi(n)$. Si p es número primo entonces $\#(\mathbb{Z}_p^*) = \phi(p) = p - 1$.*

Ejemplo: Usando Mathematica, tenemos que 13412408741378412741327493210747123874 82 es unidad en el anillo $\mathbb{Z}_{124332432109734341}$, pues el maximo común divisor de estos números es 1.

```
In[11]:= GCD[1341240874137841274132749321074712387482, 124332432109734341]
Out[11]= 1
```

Usando Mathematica, el inverso multiplicativo de 11179878972193278417498412412 1517 en $\mathbb{Z}_{179424673}$ es 80572507, lo cual es calculado por:

```
In[12]:=PowerMod[11179878972193278417498412412 1517, -1, 179424673]
Out[12]= 80572507
```

Rivest, Shamir, y Adleman diseñaron una fascinante algoritmo de encriptación, llamado RSA, con ayuda del Teorema chico de Fermat. La desencriptación es sólo posible para los pocos escogidos que tienen información extra.

El teorema chico de Fermat es una proposición acerca de potencias en aritmética modular en el caso especial donde la base modular es un número primo. Pohlig and Hellman estudiaron un esquema relacionado a RSA donde la exponenciación es hecha módulo un número primo.

Teorema 2 (Teorema chico de Fermat). *Sea p un número primo y $a \in \mathbb{Z}$ tal que $gcd(a,p) = 1$. Entonces $a^{p-1} \equiv 1 \pmod{p}$.*

Demostración: [19], p. 51. □

Teorema 3 (Teorema de Euler). *Sean $a,n \in \mathbb{Z}$ con $n \geq 2$ y $gcd(a,n) = 1$. Entonces $a^{\phi(n)} \equiv 1 \pmod{n}$.*

Demostración: [19], p. 51. □

APENDICE 2

Encriptando con RSA

Existen muchos algoritmos para encontrar un factor no trivial p del número compuesto N. Los algoritmos más útiles caen básicamente en dos categorias:

[A] El tiempo de corrido depende principalmente del tamaño de N y no tienen mucha dependencia del tamaño del factor p.

1. Algoritmo de Lehman, el cual en el peor escenario posible tiene tiempo de corrido de $O(N^{1/3})$.
2. El algoritmo de fracciones continuas y el algoritmo de la criba cuadrática multipolinomial (MPQS) los cuales bajo suposiciones plausibles tienen un tiempo esperado de corrido de $O(Exp(\sqrt{cLogNLogLogN}))$ donde c es una constante. Para MPQS, se tiene que $c \cong 1$
3. La Criba en Campos Numéricos (NFS), la cual bajo suposiciones plausibles tiene un tiempo esperado de corrido de $O(Exp(c(LogN)^{1/3}(LogLogN)^{2/3}))$ donde c es una constante que depende de la forma de N y características del algoritmo.

[B] El tiempo de corrido depende principalmente del tamaño del factor p. Se puede suponer que $p < \sqrt{N}$.

1. Algoritmo de división por ensayo y error. Tiene un tiempo de corrido $O(p(LogN)^2)$.
2. Algoritmo Rho de Pollard. El cual bajo suposiciones plausibles tiene un tiempo de corrido de $O(p^{1/2}(LogN)^2)$.
3. Algoritmo de Lenstra por Curvas Elipticas. El cual bajo suposiciones plausibles tiene un tiempo de corrido de $O(Exp(c\sqrt{cLogpLogLogp}))$, donde $c \cong 2$ es una constante.

Hasta el momento, no existe un algoritmo determínistico o probabílistico que en tiempo polinomial encuentre un factor p de un entero compuesto N. Para un algoritmo en tiempo polinomial, el tiempo de corrido debe ser polinomial en el tamaño de N, esto es, de $O((LogN)^c)$ para alguna constante c.

El problema de factorización de enteros, se supone que es un problema difícil. Inclusive, para los así llamados números RSA, los cuales son el producto de dos primos grandes, se cree que su factorización es un problema muy complicado.

Este hecho empírico es de gran interés, debido a que el más popular algoritmo para criptografía de llave pública, el algoritmo RSA, basa su seguridad en este hecho.

Ejemplo: Trivial multiplicar dos numeros primos grandes. Dados los primos

```
{PrimeQ[74556028256478842083373957362004549187833663426 57],
 PrimeQ[74712908749182741111111111232112987189172417 7] }
 Out[10] ={True, True}
p := 74556028256478842083373957362004549187833663426 57 ,
q := 74712908749182741111111111232112987189172417 7
```

Calcular $p * q$

```
In[13]:= p * q
Out[13]= 55702977358277937480679198751418000069846739185226421579773835618\\
          5986998462488340044231331828 9
```

Muy difícil el problema inverso....!!. Esto es, sabiendo que el número:

n =188198812920607963838697239461650439807163563379417382700763356422988859715234665485319060606504743045317388011303396716199692321205734031879550656996221305168759307650257059, es el producto de dos primos p, q. Encuentre los factores p y q.

```
 In[14]:= FactorInteger[n]
 In[14]:=   \$Aborted
```

Ejemplo de Encriptación con RSA

En este ejemplo, usando RSA el usuario *A* encripta el mensaje ” Autenticacion entre componentes en un prototipo de aplicación empresarial distribuida”, se trasmite al usuario *B* el cual recibe el mensaje encriptado:

```
835965691522841341344415525242676885980103518213217450554166140158394179449976
123225831040811716620739408935290442789273795383915436242760224935356398272841
206135776690861911009420652044144670966572465873756863698285466281178140566629
462096837740267825377670885413736604674457356322835607566312367184927522265773
111229451237625592940961672560161918411215328543648175731169960810731400902590
879088842
```

El usuario *B* usando su llave secreta d_B desencripta este mensaje y obtiene el mensaje ” *Autenticacion entre componentes en un prototipo de aplicacion empresarial distribuida”*.

A continuación, mostramos en detalle los pasos necesarios para encriptación y desencriptación del ejemplo anterior.

El usuario *B* genera aleatoriamente dos números primos p_B y q_B con 200 digitos cada uno y que sean aproximadamente del mismo tamaño. Se garantiza que los números p_B, q_B efectivamente son números primos, pues los primos se certifican.

```
In[15]:= Needs[CNT‘]
In[16]:= {CertifiedPrime[200], CertifiedPrime[200]}
Out[16]= {264420393935898476672247677327864331824373836632461778812390551405
902907415269148148489591097150162128035211274271803712659540191596804260492191
541954472593535828718356959745778243707235076800147685 9,
555173578550983900526598784812151614242142400397424748090112088887957144742272
272338765015189529008076060811585751400668357428382864996924166666230668986 2
851532825195511816197832599650939265217473943 1}
```

Los primos p_B y q_B se definen por:

```
In[17]:= p_B :=264420393935898476672247677327864331824373836632461778812390551
405390290741526914814848959109715016212803521127427180371265954019159680426049
219154195447259353582871835695974577824370723507680014768 59

In[18]:= q_B := 555173578550983900526598784812151614242142400397424748090112088
888795714474227227233876501518952900807606081158575140066835742838286499692416
666623066898628515328251955118161978325996509392652174739431
```

El módulo del esquema de encriptación para quién desee enviar un mensaje al usuario B será $n_B = p_B q_B$ el producto de los dos primos p_B, q_B. El usuario B obtiene:

```
In[21]:=n_B = p_B*q_B
Out[21]= n_B =146799216343253639968462427131428528754206301411175026610124819114164469981299022544320219172719664682344753218868972376373763303184383921370486626000089246277114553197695928578840621605599201572994718746191129747749125952358029355161487838944190750430585375232906670817051996359405920766248831364002454084553408342410497458638070617560774300825044401482530525051868652589971445148015563114150132722 9
```

Al azar el usuario B busca un número que cumpla con la condicion de ser primo relativo con $\phi(p_B q_B) = (p_B - 1)(q_B - 1)$. Propone $e_B = 23917$.

```
In[22]:= e_B := 23917
In[23]:= GCD[23917, (p_B - 1) (q_B - 1)]
Out[23]= 1
```

El usuario B hace del dominio público los enteros n_B y e_B. Note que B no hace públicos p_B, q_B.

El usuario B calcula su llave secreta d_B, para ello calcula $e_B^{-1} \pmod{\phi(n_B)}$:

Así, la llave secreta será $d_B = e_B^{-1}$:

```
In[26]:= d_B := PowerMod[e_B, -1, (p_B - 1) (q_B - 1)]
In[27]:= d_B
Out[27]=916935022348900835175339799689095953112467256253519974298053548834094 1
```

```
660955078386877951892884806082240867451338728010748202742761514869763572771274
889548229852466071080735364131371184371143840360279219426373447699401525553387
220135365154279912262097814361984090644027940598105333428824415147082334925568
116734312326165121869369479822682855412082173416483294866116346178023003790932
29389499890280233
```

El usuario *A* desea enviar al usuario *B* el mensaje: *Autenticacion entre componentes en un prototipo de aplicacion empresarial distribuida*. El esquema de encriptación RSA es un esquema de bloques.

Para encriptar este texto plano, el usuario *A* utilizará un solo bloque, por lo tanto convierte el texto plano a un solo entero. Para ello, usando código ASCII, obtiene:

```
In[19]:= m := StringToInteger[
                    Autenticacion entre componentes en un prototipo de
                    aplicacion empresarial distribuida
                   ]
In[20]:= m
Out[20]= 348685707985746866687480790170798583700168807881807970798570840170790
186790181838085808574818001697001668177746866874807901707881837084668374667701
6974848583746786746966
```

El usuario *A* encripta el mensaje.

Para ello calcula $m^{e_B} \pmod{n_B}$.

Así, el mensaje ya encriptado y que será enviado por *A*, quedará como:

```
In[24]:= c := PowerMod[m, e_B, n_B]
In[25]:= c
Out[25]=
835965691522841341344415525242676885980103518213217450554166140158394179449976
123225831040811716620739408935290442789273795383915436242760224935356398272841
206135776690861911009420652044144670966572465873756863698285466281178140566629
462096837740267825377670885413736604674457356322835607566312367184927522265773
111229451237625592940961672560161918411215328543648175731169960810731400902590
879088842
```

El usuario *B* recibe el mensaje *c* y procede a desencriptarlo: Para ello calcula $c^{d_B} \pmod{n_B}$, donde d_B es la llave secreta de *B*.

```
In[28]:= desencripte = PowerMod[c, d_B, n_B]
Out[28]=348685707985746866687480790170798583700168807881807970798570840170790
186790181838085808574818001697001668177746866874807901707881837084668374667701
6974848583746786746966
```

Finalmente, *B* lleva el bloque obtenido a código ASCII

```
In[29]:= IntegerToString[desencripte]
Out[29]=Autenticacion entre componentes en un prototipo de aplicacion
       empresarial distribuida
```

Si no tuviéramos la llave correcta d_B, obtendríamos un mensaje sin sentido. En el siguiente ejemplo, se desencripta el mensaje *c*, usando una llave que no es la correcta: la ”llave” que se utiliza es $d' = d_B - 1$

```
In[30]:=desencripte1 = PowerMod[c, d_B - 1, n_B]
Out[30]=396356306888759194006326123158839373486067103797092356320269533096234933
3310548287379200000984848299766694250860361822494114578722650196675290959401656
6258633801713075555164591693986594988097979620489462771056257980604637507396151
6956998667380931645472643779801527142782594324730149443456731685378002826678587
8797657503675317359715568492831384063950849071644524455905180562431695861346720
0451171902912293
```

Llevamos el bloque obtenido a código ASCII, obteniendo el siguiente mensaje, el cual no es aquel que fué trasmitido:

```
In[31]:= IntegerToString[desencripte1]
Out[31]=
"\.7fBˆ%wvZ2G%?\6.wFDAu%f"ne{Bˆ39˜@(]A|>$O;hn3\.1f\.1f\.81OO<\.80adIQu"\q7}*L
mg9Q2aj<(ZG/W8u@o0,&VV/LzdFuZPw(nn]#xM:f$]X\.81%#ˆj&F\RdW\.82ubE(>_Ug_Dno.:-:
qZJ7h\PKAWh/tDo!qamYnkXQCj>hZfVcP;>EGˆ˜’P&/KSKVyRoW7>dY\Ab3#R02!z5|
```

APENDICE 3

Tabla de sustitución y vector para RC2

El objetivo del algoritmo de expansión de llave es modificar el buffer de llave de tal forma que cada bit de la llave expandida dependa de alguna forma complicada sobre cada bit de la llave dada.

La expansión de llave y la encriptación harán uso de una tabla de sustitución que denotaremos por Π. Π es una tabla consistente de una permutación aleatoria de los números $0, 1, 2, \ldots, 127$ y fue derivada de la expansión de $\pi = 3.14159\ldots$

Detallamos Π en notación hexadecimal:

$*a$	$*b$															
	0	1	2	3	4	5	6	7	8	9	*a*	*b*	*c*	*d*	*e*	*f*
00 :	*d*9	78	*f*9	*c*4	19	*dd*	*b*5	*ed*	28	*e*9	*fd*	79	4*a*	*a*0	*d*8	9*d*
10 :	*c*6	7*e*	37	83	2*b*	76	53	8*e*	62	4*c*	64	88	44	8*b*	*fb*	*a*2
20 :	17	9*a*	59	*f*5	87	*b*3	4*f*	13	61	45	6*d*	8*d*	09	81	7*d*	32
30 :	*bd*	8*f*	40	*eb*	86	*b*7	7*b*	0*b*	*f*0	95	21	22	5*c*	6*b*	4*e*	82
40 :	54	*d*6	65	93	*ce*	60	*b*2	1*c*	73	56	*c*0	14	*a*7	8*c*	*f*1	*dc*
50 :	12	75	*ca*	1*f*	3*b*	*be*	*e*4	*d*1	42	3*d*	*d*4	30	*a*3	3*c*	*b*6	26
60 :	6*f*	*bf*	0*e*	*da*	46	69	07	57	27	*f*2	1*d*	9*b*	*bc*	94	43	03
70 :	*f*8	11	*c*7	*f*6	90	*ef*	3*e*	*e*7	06	*c*3	*d*5	2*f*	*c*8	66	1*e*	*d*7
80 :	08	*e*8	*ea*	*de*	80	52	*ee*	*f*7	84	*aa*	72	*ac*	35	4*d*	6*a*	2*a*
90 :	96	1*a*	*d*2	71	5*a*	15	49	74	4*b*	9*f*	*d*0	5*e*	04	18	*a*4	*ec*
*a*0 :	*c*2	*e*0	41	6*e*	0*f*	51	*cb*	*cc*	24	91	*af*	50	*a*1	*f*4	70	39
*b*0 :	99	7*c*	3*a*	85	23	*b*8	*b*4	7*a*	*fc*	02	36	5*b*	25	55	97	31
*c*0 :	2*d*	5*d*	*fa*	98	*e*3	8*a*	92	*ae*	05	*df*	29	10	67	6*c*	*ba*	*c*9
*d*0 :	*d*3	00	*e*6	*cf*	*e*1	9*e*	*a*8	2*c*	63	16	01	3*f*	58	*e*2	89	*a*9
*e*0 :	0*d*	38	34	1*b*	*ab*	33	*ff*	*b*0	*bb*	48	0*c*	5*f*	*b*9	*b*1	*cd*	2*e*
*f*0 :	*c*5	*f*3	*db*	47	*e*5	*a*5	9*c*	77	0*a*	*a*6	20	68	*fe*	7*f*	*c*1	*ad*

Tabla de sustitución Π especificada en notación hexadecimal para el byte de dato *ab*.

Vectores de prueba para la encriptación con RC2

Vectores de prueba para encriptación con RC2 se detallan a continuación:

Todas las cantidades estan dadas en notación hexadecimal.

Longitud de llave	Longitud efectiva de llave	llave	Texto simple	Texto cifrado
8	63	00000000 00000000	00000000 00000000	ebb773f9 93278eff
8	64	ffffffff ffffffff	ffffffff ffffffff	278b27e4 2e2f0d49
8	64	30000000 00000000	10000000 00000001	30649edf 9be7d2c2
1	64	88	00000000 00000000	61a8a244 adacccf0
7	64	88bca90e 90875a	00000000 00000000	6ccf4308 974c267f
16	64	88bca90e 90875a7f 0f79c384 627bafb2	00000000 00000000	1a807d27 2bbe5db1
16	128	88bca90e 90875a7f 0f79c384 627bafb2	00000000 00000000	2269552a b0f85ca6
33	129	88bca90e 90875a7f 0f79c384 627bafb2 16f80a6f 85920584 c42fceb0 be255daf 1e	00000000 00000000	5b78d3a4 3dfff1f1

Referencias

[15] Bressoud, D., Wagon, S, *A Course in Computational Number Theory*, Key College Publishig. Innovator in Higher Education, in cooperation with Springer-Verlag, (1999).

[16] Delfst, H., Knebl, H., *Introduction to Cryptography, Principles and Applications*, Springer-Verlag, Berlin, Heidelberg, (2002).

[17] Hasse, H., *Number Theory*, Springer-Verlag, Berlin, Heidelberg New York, (1980).

[18] Ireland, K., Rosen, M., *A Classical Introduction to Modern Number Theory*, Springer-Verlag, Berlin, Heidelberg New York (1982).

[19] Niven, I., Zuckerman, H., Montgomery, H., *An Introduction to the Theory of Numbers*, John Willey and Sons Inc. (1991).

[20] Rivest, R.L., *A Description of the RC2*^{TM}, Encryption Algoritm File, **draft-rivest-rc2des-00.txt**, available from ftp://ftp.ietf.org/internet-drafts/.

[21] Schneier, B., *Applied Cryptography. Protocols, Algorithms and Source Code in C*, John Willey and Sons Inc. (1996).

[22] Stephen W., *The MATHEMATICA Book*, Wolfram Media, Cambridge University Press (1999).

[23] Menezes, A. J., van Oorschot, P.C., Vanstone, S.A. *Handbook of Applied Cryptography*, CRC Press series on Discrete Mathematics and its applications. (1996)

[24] Rose H. E., *A Course in Number Theory*, Clarendon Press, Oxford, 1994.