Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Tests de Conformidad de IPsec en TTCN-3

Informe de Proyecto de Grado

Julio 2008

Santiago Romani

Supervisores:
Maria Eugenia Corti
Ariel Sabiguero

Resumen

El protocolo de Internet, IPv6 (Internet Protocol version 6) remplazará al actual protocolo (IPv4), por este motivo las implementaciones de IPv6 deben ser verificadas y validadas. IPsec (Internet Protocol security) es un mecanismo de seguridad incluido en la especificación de IPv6. La función de IPsec es asegurar las comunicaciones sobre el Protocolo de Internet (IP), actuando en la capa de red (capa 3 del modelo OSI) y proporcionando servicios de seguridad. La implantación de técnicas y procesos de testeo sobre aplicaciones o protocolos es importante ya que brinda una mayor credibilidad y confiabilidad del producto. Debido a la imposibilidad de que este testeo sea exhaustivo, es necesario seleccionar un conjunto de casos de prueba representativo que contemple su especificación. IPv6 Ready Logo Program es un programa de testeo desarrollado con la intención de incrementar la confianza del usuario demostrando que IPv6 está disponible y listo para ser usado. En particular IPv6 Ready Logo Program seleccionó un conjunto de casos de prueba con el propósito de validar el protocolo IPsec, mediante test de conformidad. Los test de conformidad son los que determinan si el sistema cumple la especificación, en este caso los RFC (Request For Comments) de IPsec. TTCN-3 (Testing and Test Control Notation version 3) es un nuevo lenguaje promovido por ETSI (European Telecommunications Standards Institute) para estandarización de pruebas formales, orientado a la especificación e implementación de casos de prueba.

Este proyecto tiene como objetivos el desarrollo de una metodología de implementación para la suite de tests de conformidad de IPsec (a partir de la especificación del IPv6 Ready Logo Program) y la implementación de esta suite en el lenguaje TTCN-3. Esta metodología tiene como objetivo la estandarización del proceso de desarrollo de los casos de prueba, lo cual busca mejorar la calidad en el contexto de IPv6 y seguridad sobre IP.

Palabras clave: IPsec, IPv6 Ready logo Program, TTCN-3

Índice general

Índice de figuras

Capítulo 1

Introducción

Internet nació en los EE. UU. como un proyecto militar llamado ARPANET que pretendía poner en contacto una importante cantidad de computadores pertenecientes al ejército norteamericano. A esta red se fueron añadiendo otras empresas, instituciones públicas como las universidades y también algunas personas desde sus casas. Así se logró que creciera por todo el territorio norteamericano. Fue entonces cuando se empezó a extender Internet por los demás países del mundo, abriendo un canal de comunicaciones entre varios países [30]. La comunicación entre computadores es realizada siguiendo el protocolo IP (Internet Protocol). La implementación de IP utilizada actualmente es IPv4 (Internet Protocol version 4). Ésta fue la primer versión que se implementó extensamente, utiliza direcciones de 32 bits, lo que permite 4.294.967.296 (2^32) direcciones IPv4 diferentes.

El protocolo de Internet versión 6 (IPv6, Internet Protocol version 6) [1] diseñado por Steve Deering (líder técnico de Cisco System) y Craig Mudge (director de la universidad de Macquarie), es el sucesor de IPv4, cuyo límite en el número de direcciones de red admisibles comienza a restringir el crecimiento de Internet. La adopción de IPv6 ha sido detenida por la traducción de direcciones de red (NAT, Network Address Translation) [33, 36], que soluciona parcialmente el problema de la falta de direcciones IP. NAT hace difícil el uso de algunas aplicaciones P2P (Peer to Peer, comunicación entre iguales), como lo son voz sobre IP (VoIP, Voice over Internet Protocol) [19] o juegos multiusuario. Además, NAT discrepa con la idea originaria de Internet donde todos pueden conectarse con todos. El nuevo protocolo mejora el servicio globalmente, proporcionando a futuras celdas telefónicas y dispositivos móviles, direcciones propias y permanentes, ya que permite asignar a cada dispositivo una dirección IPv6 diferente, dada la gran cantidad de direcciones disponibles. Actualmente, el gran catalizador de IPv6 es la capacidad de ofrecer nuevos servicios, como movilidad, Calidad de Servicio (QoS, Quality of Service),

privacidad, etc. El cambio más notorio de IPv4 a IPv6 es la longitud de las direcciones de red. Las direcciones IPv6, son de 128 bits, esto corresponde a 32 dígitos hexadecimales. El número de direcciones IPv6 posibles es de 2^128 $\approx$ 3.4 x 10^38. IPv6 tiene la funcionalidad de autoconfiguración y la capacidad de descubrir el camino de conexión a Internet (router discovery), es decir la posibilidad de asignar a una terminal, dirección IP, máscara de subred, DNS (Domain Name Server), etc en forma automática. Debido a la carencia de la funcionalidad mencionada, el protocolo anterior (IPv4) se vio forzado a desarrollar protocolos a nivel de aplicación que permitan esta funcionalidad, tales como DHCP o BootP.

IPv6 incluye mecanismos de seguridad en su especificación, como es el caso de IPsec (Internet Protocol security) [6]. Los protocolos de IPsec actúan en la capa de red, la capa 3 del modelo OSI (Open Systems Interconnection). Otros protocolos de seguridad para Internet son de uso extendido, como SSL (Secure Sockets Layer), TLS (Transport Layer Security) y SSH (Secure Shell), estos operan desde la capa de transporte (capas OSI 4 a 7). Ello hace que IPsec sea más flexible, ya que puede ser utilizado para proteger protocolos de la capa 4, incluyendo TCP (Transmissión Control Protocol) y UDP (User Datagram Protocol). IPsec tiene una ventaja sobre SSL y otros métodos que operan en capas superiores, ésta implica que para que una aplicación pueda usar IPsec no es necesaria la realización de ningún cambio, mientras que para usar SSL u otros protocolos de niveles superiores, las aplicaciones deben adaptar su código fuente. El protocolo está implementado por un conjunto de protocolos criptográficos para asegurar el flujo de paquetes, garantizar la autenticación mutua y establecer parámetros criptográficos. IPsec utiliza el concepto de asociación de seguridad (SA, Security Association) como base para construir funciones de seguridad en IP. Una asociación de seguridad se compone de un paquete de algoritmos y parámetros (tales como las claves que se están usando para cifrar y autenticar un flujo particular en una dirección). Por lo tanto, en el tráfico normal bidireccional, los flujos son asegurados por un par de asociaciones de seguridad.

La necesidad de mantener la seguridad, privacidad y desempeño confiable de las comunicaciones en redes, deriva en la construcción de programas de testeo que validen estos aspectos. IPv6 Ready Logo Program [8] es un programa de testeo desarrollado con el objetivo de incrementar la confianza del usuario demostrando que IPv6 está disponible y listo para ser usado. La administración de este programa se basa fundamentalmente en un foro creado por el IPv6 Logo Committee (v6LC) [7] conformado por vendedores de equipamiento, proveedores de servicios, organizaciones IPv6, etc. La misión de este comité es definir las especificaciones para tests IPv6 contemplando testeo de conformidad e interoperabilidad de forma de proveer acceso a herramientas automáticas de testeo y entrega de IPv6 Ready Logo a quienes cumplan con las especificaciones requeridas.

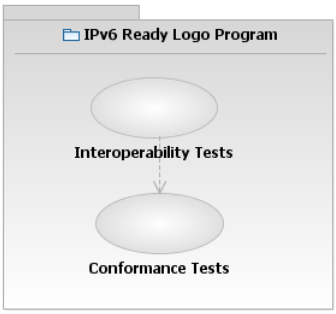

Figura 1.1: IPv6 Ready Logo Program

Los beneficios y objetivos claves de IPv6 Ready Logo Program son:

- Verificar la implementación de protocolos y validación de interoperabilidad de productos IPv6.
- Proveer acceso a herramientas para testeo automático.
- Proveer laboratorios de testeo IPv6 Ready Logo en todo el mundo.

La serie de tests definidos como parte de la suite de especificaciones de tests del IPv6 Ready Logo pueden ser divididos en dos tipos: conformidad e interoperabilidad, como se aprecia en la Figura 1.1.

El testeo de interoperabilidad es realizado en un entorno de laboratorio y el producto testeado es interconectado con otros productos (enrutadores, hosts, etc.) soportando configuraciones típicas. Los escenarios desarrollados se focalizan en verificar si el producto está habilitado para interactuar con productos de diferentes orígenes.

Por otro lado, los tests de conformidad tienen como objetivo la validación del producto a IETF RFC (Internet Engineering Task Force Request for Comments) [5, 13]. La validación del producto es obtenida a través de herramientas específicas que emulan un entorno de referencia para el producto testeado. El protocolo es analizado para cada una de las afirmaciones de especificaciones funcionales. Cabe destacar que los casos de prueba de conformidad para un protocolo pueden tratar con varios IETF RFC.

TTCN-3 (Test and Testing Control Notation) [40] es un lenguaje puramente orientado a la especificación e implementación de casos de prueba. El mismo ha sido desarrollado y mantenido constantemente por el ETSI (European Telecommunications Standards Institute) mediante un equipo de destacados expertos en pruebas

industriales, institutos y dentro del mundo académico. Es el único lenguaje de testeo estándar internacional reconocido por ETSI e ITU (International Telecommunication Union). Su propósito es la estandarización internacional de un lenguaje formal para la definición de escenarios de testeo y su implementación, garantizando la especificación unívoca de los casos de prueba.

Este lenguaje permite añadir cualquier otra especificación como diagramas UML o documentos XML, independiente del sistema a prueba. Las áreas típicas de aplicación son testeo de protocolos, testeo de componentes de bajo nivel (por ejemplo sistemas de comunicación, sistemas de automóviles, sistemas aeroespaciales, entre otros) y prueba de plataformas CORBA (Common Object Request Broker Architecture).

Una metodología se define como el conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal [1]. En este contexto, es fundamental contar con un proceso a seguir y por consiguiente desarrollar una metodología para la creación de nuevos casos de prueba, la cual aporta valor a trabajos futuros mediante la utilización de las herramientas mencionadas.

Este proyecto tiene como objetivo el desarrollo de la suite da casos de prueba correspondiente a la especificación que provee IPv6 Ready Logo Program dentro de los tests de conformidad para IPsec, y el desarrollo de una metodología de implementación de casos de prueba para trabajos futuros. Ésto permite desarrollar capacidades locales de testeo de protocolos y sistemas, aportando a la calidad en el ámbito IPv6 y a la seguridad sobre IP. Para la implementación de la suite de casos de prueba se utiliza el lenguaje TTCN-3. Es un requerimiento del presente proyecto el empleo de máquinas virtuales para representar los nodos de las topologías en las pruebas.

El alcance del presente documento abarca la temática de tests de conformidad correspondientes a la suite casos de prueba de IPv6 Ready Logo para IPsec. El resto del documento se estructura de la siguiente manera. El Capítulo 2 muestra el resultado de la investigación realizada, exponiendo el estado del arte de las tecnologías utilizadas, en el Capítulo 3 se describe la arquitectura utilizada en la implementación de los casos de prueba, mediante el lenguaje TTCN-3 y C++. El Capítulo 4 presenta la metodología de implementación desarrollada para la construcción de casos de prueba a partir de la especificación que proporciona IPv6 Ready Logo para IPsec. El Capítulo 5 describe la implementación de un caso de prueba en particular, mostrando cómo a partir de la metodología de implementación se logra una instancia concreta. En el Capítulo 6 se presenta cómo fue gestionado el proyecto y por último el Capítulo 7 presenta conclusiones y delinea posibles trabajos futuros de este proyecto.

[1] Definición de metodología: Real Academia Española.

Capítulo 2

Estado del Arte

En este capítulo se muestra el resultado de la investigación realizada sobre las herramientas empleadas para la implementación de la suite de casos de prueba y sobre el protocolo testeado. En la sección 2.1 se presenta el lenguaje utilizado para la implementación de los casos de prueba, TTCN-3 (Test and Testing Control Notation) [40], en la sección 2.2 se describe la herramienta T3DevKit desarrollada por INRIA (Institut National de Recherche en Informatique et en Automatique) [3] utilizada para el desarrollo de los casos de prueba, en la sección 2.3 se presenta una introducción a la tecnología de máquina virtual, VMware [16], mientras que en la sección 2.4 se describe la biblioteca utilizada para el cifrado de mensajes, GNU Cryptographic Library [2]. Por último se presenta IPsec [6] en la sección 2.5, explicando aspectos de sus funcionalidades.

2.1. TTCN-3

TTCN-3 es un lenguaje fuertemente tipado, flexible y potente, aplicable a la especificación de todos los tipos de tests, normalmente usado para testeo de conformidad en sistemas de comunicaciones y servicios web [40]. Las áreas típicas de aplicación son testeo de protocolos (incluyendo protocolos móviles y de Internet), testeo de servicios, testeo de módulos, testeo de aplicaciones basadas en CORBA (Common Object Request Broker Architecture), testeo de APIs (Application Programming Interface), etc. TTCN-3 no está restringido a testeo de conformidad y puede ser usado para muchos otros tipos de testeo incluyendo interoperabilidad, robustez, regresión, testeo de sistema y de integración. Es el único lenguaje de testeo estándar internacional reconocido por ETSI e ITU(International Telecommunication Union and European Telecommunications Standards Institute) [4]. Desde un

punto de vista sintáctico, TTCN-3 es muy diferente de versiones anteriores del lenguaje como se define en ISO/IEC 9646-3. Sin embargo, muchas de las funcionalidades básicas bien probadas de TTCN han sido mantenidas, y en algunos casos aumentadas. TTCN-3 [40] incluye las siguientes características esenciales:

- La habilidad de especificar dinámicamente configuraciones concurrentes de testeo
- Operaciones para comunicación sincrónica y asincrónica
- Especificar información de codificación y otros atributos (incluyendo extensibilidad de usuarios)
- Parametrización de tipos y valores
- Asignación y manejo de veredictos de test
- Parametrización de suite de tests y mecanismos de selección de casos de test
- Uso combinado de TTCN-3 con ASn.1 (Abstract Syntax Notation number One) y uso potencial con otros lenguajes como IDL (Interface definition language)
- Sintaxis bien definida, intercambio de formatos y semántica estática
- Diferentes formatos de presentación (por ejemplo formatos de presentación tabular y gráfico)
- Un algoritmo preciso de ejecución (semánticas operacionales)

Un sistema de pruebas TTCN-3 puede ser visto conceptualmente como un conjunto de entidades que interactúan entre sí, donde cada una implementa una funcionalidad de testeo específica. En la Figura 2.1 se observan los diferentes componentes que participan en un modelo de pruebas básico. El TE (TTCN-3 Executable) interpreta y ejecuta el test compilado. El componente SA (System Adaptor), adapta las comunicaciones entre el TTCN-3 y el sistema a testear (SUT, System Under Test). El componente PA (Platform Adaptor) permite la ejecución de funciones externas escritas en C++. El componente TM (Test Management) es responsable de la gestión del sistema de testeo. Finalmente, el componente CD (Coding - Decoding) es responsable de codificar y decodificar los valores TTCN-3 en *bitstrings* adecuados para la comunicación con el sistema a testear. Por más detalles sobre la descripción de esta arquitectura ver [28].

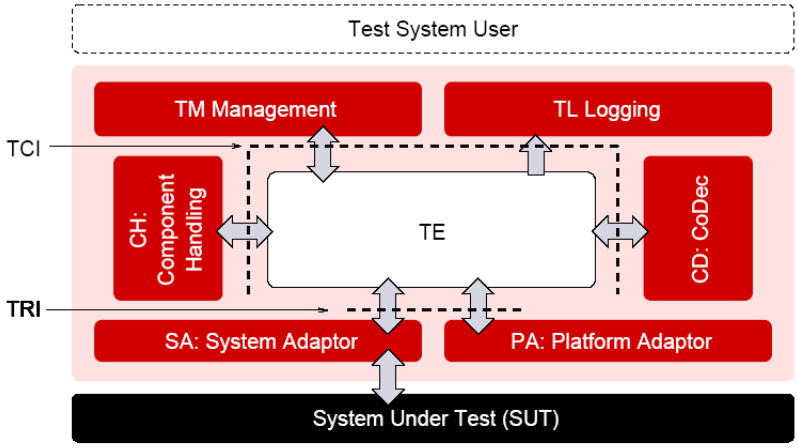

Figura 2.1: TTCN-3 Adaptador de Test

2.2. T3DevKit

Es un conjunto de herramientas libres desarrolladas con el fin de facilitar la creación de casos de prueba utilizando TTCN-3. Se rige por la licencia CeCILL-C en el derecho Francés y el cumplimiento de las normas de distribución de software libre. Este framework es utilizado para continuar con la línea de trabajo, la herramienta se compone de dos partes:

- T3DevLib: Es una biblioteca que cumple con la estandarización ETSI TRI (TTCN-3 Runtime Interface) y TCI-CD (TTCN-3 Control Interface - Coding and Decoding) [27]. Proporciona un marco de fácil manipulación mediante diferentes niveles de abstracción y mecanismos por defecto de codificación y decodificación. También provee un conjunto de clases C++ que facilitan el manejo de puertos, temporizadores y funciones externas en TTCN-3.

- T3CDGen. Genera a partir de los tipos definidos en código TTCN-3 el código C++ necesario para la aplicación de CoDec, estos son utilizados tanto en la transmisión como en el envío de mensajes.

Los codets son piezas del código de la plataforma, que al ser combinados con el código generado producen el CoDec (Code-Decode). Cuando el generador de código está corriendo, automáticamente chequea la presencia de codets, los incluye

en la definición de la clase y realiza las llamadas correspondientes en *Encode()* y *Decode()*.

Mediante este conjunto de herramientas es posible desarrollar casos de prueba en TTCN-3 con mayor facilidad [10].

2.2.1. Integración con framework T3DevKit

Se debe desarrollar un sistema de Codificador-Decoficador (CoDec) para establecer la conexión entre los componentes abstractos y los instanciados, por ejemplo en la creación de puertos a partir de las clases que provee T3DevLib. En la Figura 2.2 se muestra la integración del código realizado por el desarrollador y el framework.

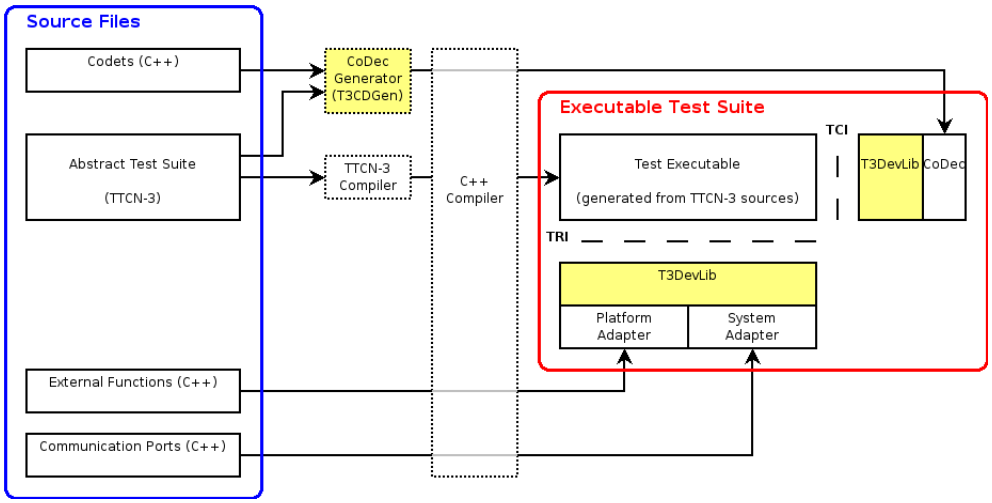

Figura 2.2: Integración con Framework

Es importante señalar que el desarrollador no necesita manipular el código generado, evitando en consecuencia la ocurrencia de errores en la codificación manual. Por otro lado el framework brinda automatización en la ejecución de tareas costosas y rutinarias, como es el caso del desarrollo y mantenimiento de los CoDec. Las funciones que permite desarrollar la herramienta son las siguientes[10]:

- Sincronización de definiciones de tipos: La definiciones de tipos son escritas en TTCN-3. Sin embargo el interior del CoDec (escrito en C++), el desarrollador necesita saber con precisión cómo los mensajes son representados en TTCN-3 a fin de corregir la estructuras a través de las llamadas a TCI-CD. Básicamente este trabajo debe ser realizado dos veces, y el mismo es

una tarea tediosa y propensa a introducir errores, especialmente cuando la definición de tipos es actualizada. Con el generador de CoDec que provee el framework la mayoría del código C++ se genera automáticamente, y en consecuencia la cantidad de código a ser escrito de forma manual se reduce al mínimo, por lo que es mucho más rápido para escribir y mucho más fácil de mantener.

- Manipulación de datos TTCN-3: En TTCN-3 los valores son manejados de manera abstracta en la vía TCI-CD, no siendo de fácil manejo en C++. Es necesaria la utilización de bibliotecas para la representación de tipos básicos; éstas se encuentran dentro del framework y son de gran ayuda al desarrollador.

- Portabilidad: A pesar de que TTCN-3 es un lenguaje estandarizado existen incompatibilidades entre los compiladores de diferentes proveedores. Las incompatibilidades pueden ocurrir debido a bugs, imprecisión en los estándares o simplemente porque algunos errores de instalación no son cubiertos por la especificación. Con este framework el usuario no debe preocuparse por las interfaces, las mismas están implementadas en módulos separados. Algunos compiladores ya están soportados (por ejemplo Java y C++) y portar la biblioteca hacia otros compiladores resulta simple ya que todos los cambios a realizar están localizados en el mismo lugar.

Existen algunas limitaciones en T3DevKit, como lo es el soporte a tipos primitivos, la biblioteca no soporta los siguientes tipos:

- Float, tipo que representa el punto flotante
- Charstring, cadena de caracteres
- Address, direcciones

Los subtipos primitivos de TTCN-3 (integer, octectstring, etc) no son soportados por el generador de CoDec, ellos necesitan ser declarados manualmente en un archivo «.h» provisto por el usuario. Los atributos type y field (excepto la palabra reservada optional) son ignorados por el generador de CoDec. A pesar de que los nombres de módulos están listos para ser manejados internamente no están completamente soportados. Los conflictos de nombres entre identificadores de diferentes módulos deben ser evitados durante la escritura de los casos de prueba. No se soportan definiciones de tipos anidados [10].

2.2.2. Generador de CoDec

Al momento de crear los CoDec, el generador recibe como entrada los archivos que contienen código TTCN-3 con las definiciones de tipos, los archivos C++ (código fuente y/o encabezados) conteniendo los codets y opcionalmente se puede incluir un archivo de encabezados C++ que permite la definición de tipos primitivos los cuales no son soportados por el generador.

El proceso de generación comprende la creación de dos archivos C++, un archivo de encabezado y otro con código fuente. Luego para el desarrollo de los codets se debe incluir el archivo de encabezado generado.

En la Figura 2.3 se muestra las dependencias entre código fuente C++, las clases gen_classes.h y gen_classes.cpp que son producidas por el generador de CoDec.

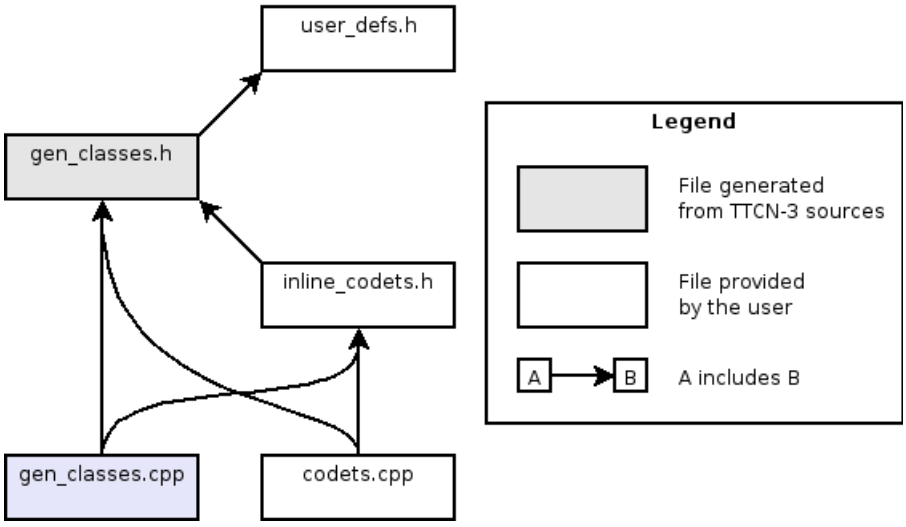

Figura 2.3: Dependencias entre código fuente C++

2.3. Máquinas Virtuales

Una máquina virtual es un software que crea un entorno virtual entre la plataforma de una computadora y el usuario final, permitiendo que éste ejecute un software determinado.

El concepto de máquina virtual surge con el sistema VM/370 de IBM [12] en 1972. La idea principal es la de permitir ejecutar varios sistemas operativos simultáneamente sobre el mismo hardware. Para ello, separa las dos funciones básicas

que realiza un sistema de tiempo compartido: multiprogramación y abstracción del hardware.

En la actualidad existen varias implementaciones que permiten la creación y gestión de máquinas virtuales, como lo son: Virtual PC [14], QEMU [12], VMware [16] y VirtualBox [14]. De forma que un usuario puede estar ejecutando varios sistemas operativos como Windows NT [20] y Linux [9] de forma simultánea. Dentro de las implementaciones existentes de máquinas virtuales se optó por VMware.

2.3.1. VMware

WMware proporciona la mayor parte del software de virtualización disponible para ordenadores compatibles X86 [22]. Dentro este software se incluyen VMware Workstation, VMware Server [18] y VMware Player [17]. El software de VMware puede funcionar en Windows, Linux, y en la plataforma Mac OS X que corre en procesadores INTEL, bajo el nombre de VMware Fusion. El nombre coorporativo de la compañía es un juego de palabras usando la interpretación tradicional de las siglas "VM" en los ambientes de computación, como Máquinas Virtuales (Virtual Machines).

Esta tecnología provee un sistema de virtualización por software, que simula un sistema físico con determinadas características de hardware. Cuando se ejecuta el programa (simulador), proporciona un ambiente de ejecución similar al de un ordenador físico (excepto en el puro acceso físico al hardware), con CPU (puede ser más de una), BIOS, tarjeta gráfica, memoria RAM, tarjeta de red, sistema de sonido, conexión USB, disco duro (pueden ser más de uno), etc. Soporta tres alternativas dentro la configuración de red:

- Bridge
- NAT
- Host-Only

En este proyecto se emplea VMware Server dentro de las posibilidades de software que provee VMware, la cual es ampliada en la sección 2.3.1.1.

2.3.1.1. VMware Server

En un principio fue creada como una versión no libre, recientemente ha sido liberada y puede ser descargada y utilizada de forma gratuita. Esta versión, a diferencia de la anterior, tiene un mejor manejo y administración de recursos. También corre dentro de un sistema operativo (host), permite la creación de máquinas

virtuales a diferencia de VMplayer que sólo permite la ejecución de máquinas virtuales.

2.4. Criptografía

La criptografía es la ciencia de cifrar y descifrar información utilizando técnicas que hagan posible el intercambio de mensajes de manera segura de forma que sólo puedan ser leídos por las personas a quienes van dirigidos. Cuando se habla de esta área de conocimiento como ciencia se debería hablar de que engloba tanto las técnicas de cifrado, como sus técnicas complementarias: el criptoanálisis, que estudia los métodos que se utilizan para romper textos cifrados con objeto de recuperar la información original en ausencia de las claves.

La finalidad de la criptografía es, en primer lugar, garantizar el secreto en la comunicación entre dos entidades (personas, organizaciones, etc.) y, en segundo lugar, asegurar que la información que se envía es auténtica en un doble sentido: que el remitente sea realmente quien dice ser y que el contenido del mensaje enviado, habitualmente denominado criptograma, no haya sido modificado en su tránsito.

Otro método utilizado para ocultar el contenido de un mensaje es ocultar el propio mensaje en un canal de información, esta técnica no se considera criptografía, sino esteganografía. Por ejemplo, mediante la esteganografía se puede ocultar un mensaje en un canal de sonido, una imagen o incluso en reparto de los espacios en blanco usados para justificar un texto. La esteganografía no tiene por qué ser un método alternativo a la criptografía, siendo común que ambos métodos se utilicen de forma simultánea para dificultar aún más la labor del criptoanálisis.

Una de las ramas de la criptografía que más ha revolucionado el panorama actual de las tecnologías informáticas es el de la firma digital: tecnología que busca asociar al emisor de un mensaje con su contenido de forma que aquel no pueda posteriormente repudiarlo. Es de interés destacar a GNU Cryptographic Library [2] dentro de las implementaciones criptográficas existentes, ya que es utilizada en el desarrollo de los casos de prueba, la misma es explicada en la sección 2.4.1.

2.4.1. GNU Cryptographic Library

Existen varias implementaciones criptográficas, GNU Cryptographic Library reúne una combinación de ventajas sobre otras bibliotecas que realizan un trabajo similar. Es software libre, cualquiera puede usarlo, modificarlo y redistribuirlo bajo los términos de la licencia GNU Lesser General Public License. Encapsula criptografía de bajo nivel, esta provee una interfaz de alto nivel para la construcción de

bloques criptográficos usando una API extensible y flexible, resuelve los siguientes algoritmos [26]:

- Criptografía Simétrica: AES, DES, Blowfish, CAST5, Twofish, Arcfour
- Algoritmos de Hash : MD4, MD5, RIPE-MD160, SHA-1, TIGER-192
- Criptografía Asimétrica: RSA, ElGamal, DSA

La biblioteca GNU Cryptographic Library posee la característica de thread-safe. Existen algunas funciones criptográficas no thread-safe que modifican cierto contexto almacenado en indicadores. Si el usuario intenta utilizar funciones de difentes hilos en el mismo indicador debe tener cuidado sobre la serialización de algunas funciones por sí mismo, si no se describe de otra manera la opción por defecto de cada función es thread-safe.

2.5. IPsec

IPsec se compone de un conjunto de protocolos cuya función es asegurar las comunicaciones sobre el Protocolo de Internet (IP) autenticando y/o cifrando cada paquete IP en un flujo de datos. IPsec también incluye protocolos para el establecimiento de claves de cifrado. Los protocolos de IPsec actúan en la capa de red, la capa tres del modelo OSI. Esto hace que IPsec sea más flexible, ya que puede ser utilizado para proteger protocolos de la capa cuatro, incluyendo TCP y UDP [24, 37].

IPsec está implementado por un conjunto de protocolos criptográficos para asegurar el flujo de paquetes, garantizar la autenticación mutua y establecer parámetros criptográficos.

La arquitectura de seguridad de IP utiliza el concepto de asociación de seguridad (SA, Security Association) como base para construir funciones de seguridad en IP. Una asociación de seguridad es el paquete de algoritmos y parámetros que se está usando para cifrar y autenticar un flujo particular en una dirección.

Para decidir qué protección se va a proporcionar a un paquete saliente, IPsec utiliza el índice de parámetro de seguridad (SPI), un índice a la base de datos de asociaciones de seguridad (SAD), junto con la dirección de destino de la cabecera del paquete, que identifican de manera únivoca una asociación de seguridad para dicho paquete. Para un paquete entrante se realiza un procedimiento similar, este procesamiento es ampliado en la sección 2.5.1.

IPsec es parte obligatoria de la especificación de IPv6, mientras que su uso es opcional sobre IPv4. Los protocolos de IPsec se definieron originalmente en las

RFCs 1825 y 1829 [35, 29], éstos fueron sustituidos por los RFCs 2401 y 2412[35, 29], no compatibles con los anteriores pero conceptualmente similares. Luego se produjo la tercera generación de documentos, los RFCs 4301 y 4309 [34, 31]. Esta tercera generación de documentos estandarizó la abreviatura de IPsec como «IP» en mayúsculas y «sec» en minúsculas además de describir la arquitectura de la seguridad para ambas versiones IP (IPv4 e IPv6).

IPsec opera en dos modalidades [6]:

- Modo transporte: Sólo la carga útil del paquete IP es cifrada y/o autenticada. El enrutamiento permanece intacto, ya que no se modifica ni se cifra la cabecera IP.
- Modo túnel: En el modo túnel, todo el paquete IP (datos más cabeceras del paquete) es cifrado y/o autenticado.

En la Figura 2.4 se muestra la diferencia entre los distintos modos en términos de contenido de mensaje.

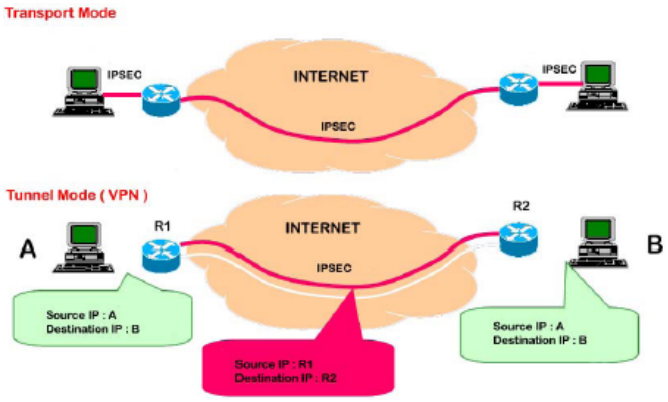

Figura 2.4: Modo túnel y modo transporte

Estos modos pueden ser combinados con distintos protocolos que han sido desarrollados para proporcionar seguridad a nivel de paquete, estos son:

- Authentication Header (AH): proporciona integridad en los datos, autenticación y no repudio por parte del emisor.
- Encapsulating Security Payload (ESP): proporciona confidencialidad, además de opcionalmente autenticación y protección de integridad.

Los algoritmos criptográficos definidos para usar con IPsec incluyen HMAC-SHA1 para protección de integridad, y Triple DES-CBC y AES-CBC para confidencialidad, estos son detallados en la RFC 4305[23].

La cabecera de autenticación (AH) está dirigida a garantizar integridad sin conexión y autenticación de los datos de origen de los datagramas IP. Además, puede opcionalmente proteger contra ataques de repetición utilizando la técnica de ventana deslizante y descartando paquetes viejos. Protege la carga útil IP y todos los campos de la cabecera de un datagrama IP excepto los campos «mutantes», es decir, aquellos que pueden ser alterados en el tránsito.

En la Figura 2.5 se muestra el contenido de los mensajes correspondientes al protocolo AH.

0 - 7 bit	8 - 15 bit	16 - 23 bit	24 - 31 bit
Next header	Payload length	RESERVED	
Security parameters index (SPI)			
Sequence number			
Authentication data (variable)			

Figura 2.5: Protocolo AH

Los campos del mensaje AH son:

- Next header: Identifica el protocolo de los datos transferidos
- Payload length: Tamaño del paquete AH.
- RESERVED: Reservado para uso futuro es rellenado con ceros por el momento.
- Security parameters index (SPI): Indica los parámetros de seguridad que en combinación con la dirección IP, identifican la asociación de seguridad implementada con este paquete.
- Sequence number: Un número siempre creciente, utilizado para evitar ataques de repetición.
- Authentication data: Contiene el valor de verificación de integridad (ICV) necesario para autenticar el paquete; puede contener relleno.

En la Figura 2.6 se muestra el contenido de los mensajes correspondientes al protocolo ESP.

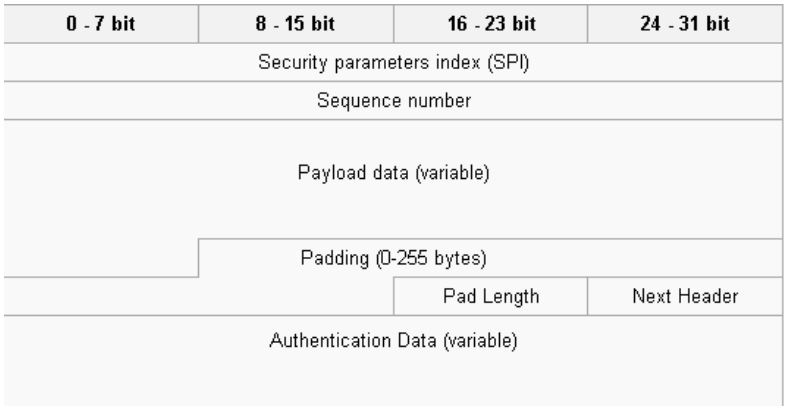

Figura 2.6: Protocolo ESP

Los campos del mensaje ESP son:

- Security Parameters Index (SPI): Identifica los parámetros de seguridad en combinación con la dirección IP.
- Sequence number: Un número siempre creciente, utilizado para evitar ataques de repetición.
- Payload data: Los datos a transferir.
- Padding: Usado por algunos algoritmos criptográficos para rellenar por completo los bloques.
- Pad length: Tamaño del relleno (Padding) en bytes.
- Next header: Identifica el protocolo de los datos transferidos.
- Authentication data: Contiene los datos utilizados para autenticar el paquete.

2.5.1. Procesamiento de paquetes

Para efectuar el procesamiento se utilizan dos bases de datos, una conteniendo las asociaciones de seguridad (SAD, Security Association Database) y otra que contiene las políticas de seguridad (SPD, Security Policy Database). La SPD consta de los siguientes campos:

- Referencia a las SAs activas

- Selector de campos

Luego, SAD es un conjunto de SAs, las cuales contienen los siguientes campos:

- Destination IP address: Dirección IP de destino
- Ipsec proto (AH or ESP): Protocolo de IPsec utilizado, el mismo puede ser ESP o AH
- SPI (cookie): Security Parameters Index (SPI) análogo a el utilizado en ESP y AH
- Sequnce counter: Análogo a el utilizado en ESP y AH (Sequence number)
- Seq O/F flag: Bandera de sequencia O/F
- Anti-replay window info: Información de window anti-replay
- AH type and info: Información acerca del tipo AH
- ESP type and info: Información acerca del tipo ESP
- Lifetime info: Información acerca del tiempo de vida
- Tunnel/transport mode flags: Bandera que indica la modalidad (túnel o transporte)
- PATH MTU info: Informacion de MTU

El procesamiento de paquetes es realizado tanto para paquetes entrantes como salientes. En la sección 2.5.1.1 se desarrolla el procesamiento de paquetes salientes, y en la sección 2.5.1.2 el procesamiento entrante.

2.5.1.1. Procesamiento del paquete saliente

En caso de que sea requerido el procesamiento IPsec, el paquete se procesa en primer lugar según las reglas de la SPD. Si el paquete hace un match con una regla que especifica IPSec, se busca la SAD para ver si ya existe una SA establecida. Si hay una SA, se aplica al paquete saliente. Si no hay una SA, se negocia una y se almacena en la SAD después de negociada [31].

2.5.1.2. Procesamiento del paquete entrante

Para los paquetes entrantes se debe encontrar la SA en la SAD, que haga un match con los siguientes valores:

- Destination IP address: Dirección IP de destino
- Ipsec proto (AH or ESP): Protocolo IPSec
- Security Parameters Index (SPI): Identifica los parámetros de seguridad en combinación con la dirección IP.

La dirección IP de destino y el protocolo IPSec se obtienen del paquete IP. El SPI se obtiene del header AH o ESP. Si se encuentra una SA, se aplican al paquete entrante los servicios de seguridad especificados en la SA. Luego el paquete pasa a ser procesado por las reglas de la SPD[31].

Capítulo 3

Arquitectura de Implementación

En este capítulo se presenta la arquitecura utilizada para la creación de los casos de prueba. La arquitectura de implementación define de manera abstracta los componentes que llevan a cabo las diferentes tareas y la comunicación entre ellos. La ejecución de los casos de prueba se logra a partir de la interacción de estos componentes.

La arquitectura utilizada se ilustra en la Figura 3.1 mediante un diagrama de componentes, el cual muestra como interaccionan entre sí. Se presenta su estructura para facilitar la compresión del desarrollo de la metodología de implementación de casos de prueba correspondiente al capítulo 4.

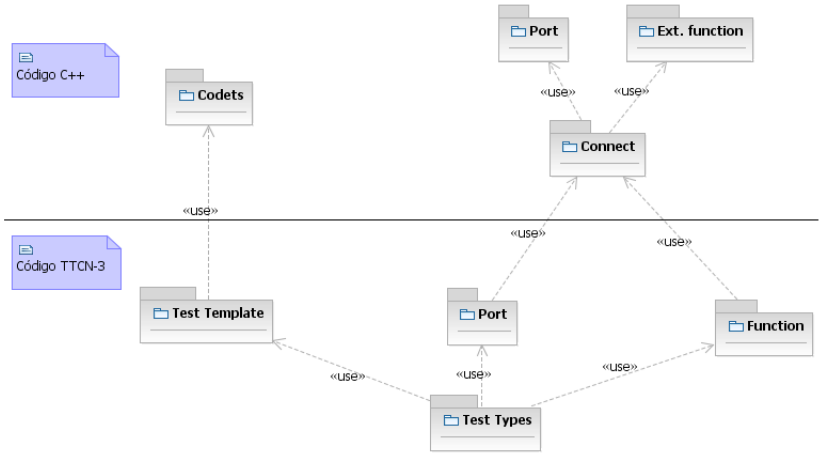

Figura 3.1: Arquitectura

En la Figura 3.1 se observa que intervienen dos lenguajes de programación,

uno donde los componentes son desarrollados en C++ y otro donde el lenguaje es TTCN-3, por ende requiere un mecanismo de comunicación entre ellos. En la sección 3.1 se explica el mecanismo utilizado. En la sección 3.2 se presentan los componentes especificados en TTCN-3 y por último en la sección 3.3 se describen los componentes implementados en C++.

3.1. Comunicación entre componentes

La comunicación entre los componentes que son implementados en C++ y los que son desarrollados en TTCN-3 es llevada a cabo siguiendo la arquitectura observada en la Figura 2.2, correspondiente a la integración con el framework T3DevKit.

El componente codets implementado en el lenguaje C++ es utilizado como entrada en el generador de CoDec (T3CDGen, TTCN-3 CoDec Generator), así como también la especificación de tipos y templates utilizados que se encuentran en el componente TestTypes (lenguaje TTCN-3), de esta manera se implementa la interfaz TCI (TTCN-3 Control Interface) que proporciona T3DevLib.

Los componentes Port y External Function desarrollados en C++ son compilados y utilizados como plataforma de adaptación y sistema de adaptación respectivamente, de esta manera implementan las funcionalidades que la biblioteca T3DevLib ofrece correspondiente a la interfaz TRI (TTCN-3 Runtime Interface) [27].

3.2. Código TTCN-3

En esta sección se explican los roles de los componentes especificados en el lenguaje TTCN-3 utilizados para la resolución de los casos de prueba. Son descritos de la siguiente forma, en la sección 3.2.1 el componente Types template, en la sección 3.2.2 el componente Port, en la sección 3.2.3 el componente Functions y por último en la sección 3.2.4 el componente Test case.

3.2.1. Tests template

Este componente define las templates (plantillas) de los tipos que serán usados para la implementación de los casos de prueba, tanto para la entrada de mensajes como para la salida. Los templates de tipos permiten la manipulación de los campos del mensaje y proporcionar una mayor adaptabilidad. La definición del template se desprende de la especificación del protocolo. Los campos de los mensajes pueden

tener longitud variable, en este caso se debe especificar la longitud de los mismos, por otro lado existen protocolos con campos opcionales a los cuales se les permite asignar presentismo. Estas asignaciones se realizan en el componente *Codets* explicado en la sección 3.3.1.

3.2.2. Port

Aquí se definen los puertos que se van a utilizar en el envío y recepción de mensajes, se crea un tipo de puerto diferente para cada protocolo. Es una definición, la implementación concreta se realiza en C++. En la sección 3.3 se explica con mayor detalle cómo se obtiene la implementación a partir de la definición.

3.2.3. Functions

El componente Functions define las funciones utilizadas que no son implementadas en el lenguaje TTCN-3. De esta manera quedan disponibles para ser utilizadas desde el código TTCN-3, independientemente de su implementación. De manera similar a la conexión de la definición de `t3devlib::Port` con su implementación, se conectan las definiciones de funciones con su correspondiente implementación, utilizando la biblioteca T3DevLib.

3.2.4. Test case

Las implementaciones de casos de prueba se agrupan en este componente, de forma que para cada caso de prueba, se encuentra una implementación asociada. En este componente se encuentra el programa principal que ejecuta los casos de prueba en orden secuencial.

3.3. Código C++

En esta sección se explican los roles de los componentes implementados en el lenguaje C++ utilizados para la resolución de los casos de prueba. Son descritos de la siguiente manera, en la sección 3.3.1 el componente Codets, en la sección 3.3.2 el componente Port y por último en la sección 3.3.3 el componente External Functions.

3.3.1. Codets

En este componente se realizan las tareas de codificación y decodificación de los tests. Es posible implementar funciones para cada tipo definido en el componente Tests Types especificado en TTCN-3, las mismas son invocan en diferentes instancias: previo a la decodificación del paquete, durante la decodificación (en cada campo del paquete) y luego de la decodificación del mismo. Cuando un campo del paquete tiene valor indefinido u opcional es necesario tomar la decisión de emplearlo o no. Esta tarea es llevada a cabo utilizando la biblioteca *T3DevLib*, en particular la clase `t3devlib::GeneratedRecord`.

3.3.2. Port

Este componente especifica el o los puertos utilizados en la prueba y las clases de este componente son realizadas por intermedio al framework *T3DevKit*. Aquí se define el envío y recepción de paquetes, implementando las funciones abstractas definidas en el framework. La clase utilizada es `t3devlib::Port`, y es importante destacar la necesidad de la redefinición de la función *Map* de esta clase.

3.3.3. External Functions

El conjunto de funciones externas se implementan dentro de este componente, la comunicación de parámetros se realiza utilizando la clase `t3devlib:: ParameterList` para emplear los parámetros de entrada y la clase `t3devlib::Bitstring` para los parámetros de salida.

Capítulo 4

Metodología de implementación

En este capítulo se describe el eje central del presente proyecto, el cual consiste en desarrollar una metodología de implementación para la suite de casos de prueba de IPsec sobre IPv6, especificados por el comité IPv6 Ready Logo Program [8].

La relevancia de establecer una metodología radica en brindar un conjunto de métodos para la resolución de los casos de prueba y en consecuencia estructurar el proceso facilitando el desarrollo de éstos. La bondad de esta metodología es contar con el suficiente nivel de abstracción para contemplar los diferentes casos de prueba. Estos casos de prueba son definidos mediante un documento técnico (ATS, Abstract Test Suite) que establece su especificación.

Debido a que IPsec funciona en dos modalidades (transporte y túnel) existen diversos escenarios que deben ser testeados. En la especificación de casos de prueba se incluyen cuatro topologías de red que abarcan todos los escenarios posibles, y se indica qué topología aplica en cada caso de prueba.

Los test de conformidad generalmente, se ejecutan generando paquetes dentro del nodo testeador (TN, Test Node), que luego se envían al nodo bajo prueba (NUT, Node Under Test). Estos paquetes simulan el resultado de haber atravesado toda la topología hasta llegar al NUT. Una alternativa a este método es la virtualización, donde se utilizan máquinas virtuales para representar las topologías mencionadas previamente. Una ventaja de este escenario es que el TN ya no debe generar completamente el mensaje, pues el mismo se completa en diferentes etapas al atravesar las distintas máquinas virtuales. Para que esto sea posible es necesario configurar las diferentes topologías para cada caso de prueba. Implementar los tests de conformidad utilizando el enfoque de virtualización es un requerimiento del presente proyecto.

La metodología planteada es dividida en dos partes, por un lado la metodología de ejecución de casos de prueba, la cual describe las actividades de configuración

de las topologías de red y ejecución de estos casos, y por otro la metodología de especificación y codificación, la cual describe las actividades relacionadas con el desarrollo de los casos de prueba.

Previo al detalle sobre la metodología, se proporciona la descripción del documento técnico (ATS) mencionado anteriormente en la sección 4.1. Luego en la sección 4.2 se desarrolla la metodología correspondiente a la ejecución de los casos y por último en la sección 4.3 la metodología de especificación y codificación en el lenguaje TTCN-3.

4.1. Descripción de ATS

La suite de tests abstractos (ATS) se compone de casos de prueba abstractos los cuales se definen como una especificación completa e independiente de la acción requerida para alcanzar un propósito de testeo especifíco [32].

A fin de verificar las actuales implementaciones de IPsec en redes IPv6, el comité del IPv6 Ready Logo Program definió una ATS correspondiente a los tests de conformidad, a partir de los cuales se pretende realizar la cobertura de las especificaciones IPsec, basada en los IETF RFC [5].

Para obtener el **Logo** de IPv6 Ready para IPsec, el NUT debe satisfacer los siguientes requerimientos:

- Tipo de Equipo: se definen dos posibilidades para tipos de equipo como se muestra a continuación
  - End-Node: es el nodo que puede usar IPsec solamente para sí mismo. Tanto un Host como un Router pueden ser un End-Node.
  - Security GateWay (SGW): es el nodo que provee el modo túnel de IPsec para nodos detrás de él. Sólo un Router puede ser SGW.
- Protocolo de Seguridad: Se requiere que el NUT pase todos los tests ESP (Encapsulating Security Payload) sin importar el tipo de equipo. El IPv6 Ready Logo Program no se focaliza en AH (Authentication Header).
- Modo: este requerimiento depende del tipo de NUT
  - End-Node: si el NUT es un End-Node debe pasar todos los tests de modo transporte. Si el NUT soporta el modo túnel, además debe pasar todos los tests de modo túnel (por ejemplo, el modo túnel es una funcionalidad avanzada para el End-Node).

  - SGW: si el nodo es un SGW debe pasar todos los tests de modo túnel.

- Algoritmo de Encriptación: El IPv6 Logo Comitee ha definido dos categorías de algoritmos de encriptación, algoritmos base (BASE ALGORITHM) y algoritmos avanzados (ADVANCED ALGORITHM). Todos los NUTs deben pasar los test base para obtener un IPsec Logo. Aquellos NUTs que soporten algoritmos calificados como avanzados deben pasar todos los tests avanzados (ADVANCED). El requerimiento de algoritmo es independiente del tipo NUT.

  - Algoritmo Base: 3DES-CBC
  - Algoritmos Avanzados: AES-CBC, AES-CTR, NULL

- Algoritmo de Autenticación: El IPv6 Logo Committee ha definido algoritmos de autenticación base y avanzados. Todos los NUTs deben pasar todos los tests de algoritmos base para obtener el IPsec Logo. Los NUTs que soportan los algoritmos que son listados como avanzados, deben pasar todos los tests correspondientes. El requerimiento algoritmo de autenticación es independiente del tipo de NUT.

  - Algoritmo Base: HMAC-SHA1
  - Algoritmos Avanzados: AES-XCBC-MAC-96, NULL

Para cada caso de prueba se cuenta con la siguiente información [38]:

- **Propósito:** breve descripción de lo que se intenta lograr a partir de la prueba.
- **Categoría:** cada caso de prueba se puede clasificar y por ende categorizar, existen dos categorías BASIC y ADVANCED; cada NUT (Node Under Test) debe soportar todos los casos de la categoría BASIC, mientras que los casos ADVANCED son requeridos únicamente en los NUTs que soportan encriptación avanzada y/o algoritmos de autenticación avanzados.
- **Inicialización:** describe cómo inicializar y configurar el NUT antes de comenzar cada prueba; si un valor no está previsto, el protocolo define el valor por defecto utilizado.Se indica a qué topología aplica el caso de prueba en esta sección.
- **Paquetes:** describe los paquetes enviados y recibidos que participan en la prueba.

- **Procedimiento:** indica paso a paso las instrucciones para llevar a cabo la prueba.
- **Fallo:** describe el resultado esperado, el NUT pasa la prueba si se observa lo mismo que el resultado.

4.2. Metodología de ejecución

En esta sección se explican las actividades que se deben realizar para el correcto desempeño de los casos de prueba. Para lograr total independencia entre sucesivas ejecuciones de los casos de prueba, es necesario que el sistema retorne al estado inicial luego de cada ejecución. Además es fundamental llevar registro de los paquetes enviados y recibidos como método de validación.

La metodología de ejecución consiste en una serie de etapas, las cuales son empleadas en todos los casos de prueba y se basa en la ISO-9646 [25]. Estas etapas son:

- **Preámbulo**: en esta etapa se logra la configuración necesaria para la ejecución del caso de prueba.
- **Comienzo de registro**: se inicia el proceso de registro de los paquetes enviados y recibidos.
- **Ejecución del caso de prueba**: en esta etapa se ejecuta el caso de prueba.
- **Fin de registro**: se finaliza el proceso de registro de paquetes, generando un archivo correspondiente al caso de prueba.
- **Postámbulo**: en esta instancia se restaura la configuración del sistema, deshaciendo los cambios realizados.

En la Figura 4.1 se presentan gráficamente las etapas de la metodología descritas anteriormente.

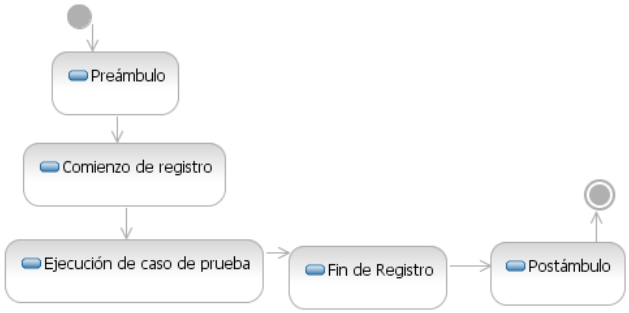

Figura 4.1: Etapas de la metodología de ejecución

4.2.1. Preámbulo

En esta sección se cubren los aspectos de configuración de los casos de prueba. Los parámetros necesarios para esta configuración se obtienen de la sección *Inicialización* del documento técnico de especificación. Previo a la ejecución de un caso de prueba se deben llevar a cabo dos actividades que conforman la etapa de *Preámbulo*:

- Configuración de topología de red
- Configuración de IPsec

Estas actividades pueden ser ejecutadas en cualquier orden ya que son independientes. Sin embargo, la metodología de implementación propone que se realice en primera instancia la actividad *Configuración de topología de red* y a continuación la *Configuración de IPsec*, ya que luego de la primera es posible probar la conectividad entre los extremos sin que interfiera la configuración de IPsec. Luego, al configurar IPsec se puede probar la conectividad segura, sabiendo que existe conectividad a nivel de IPv6. Para probar conectividad luego de configurar IPsec, se utilizó el programa *ping6* [11] (Packet Internet Groper versión 6), éste envía paquetes ICMPv6, los cuales son cifrados a nivel de capa 3 por la asociación de seguridad (SA, Security Association) de IPsec. Para confirmar el correcto funcionamiento de IPsec, es posible la visualización del tráfico de la red mediante programas de *packet sniffer*, por ejemplo *wireshark* [21].

4.2.1.1. Configuración de topología de red

La configuración de topología de red hace referencia a conceptos tales como direcciones IPv6, máscaras de subred, tablas de enrutamiento, asignación de MTU

```
Initialization:

Use common topology described as Fig.1

Set NUT's SAD and SPD as following:

    HOST1_Link1 ---------------------- NUT
                --------------------> SA1-I
                <-------------------- SA1-O
                --------------------> SA2-I
                <-------------------- SA2-O
```

Figura 4.2: Inicialización - Especificación de casos de Prueba

(Maximun Transfer Unit), etc. Por otra parte se desarrollan tareas complementarias a la configuración de la topología, tales como la automatización de configuraciones intermedias y creación de VLANs.

Dentro del punto *Inicialización*, correspondiente al documento técnico de especificación de casos de prueba, son definidas las tareas de configuración de topología de red que deben llevarse a cabo para cada test. En la Figura 4.2 se muestra un ejemplo del punto *Inicialización* extraído de la especificación de casos de prueba correspondiente al test 5.1.1 Select SPD. Se indica aquí qué topología es usada para llevar a cabo el caso de prueba, así como también la configuración del NUT en la prueba. Cabe destacar que el NUT puede ser configurado en más de una ocasión por prueba.

La topología es definida mediante diagramas que se encuentran en las primeras páginas del documento técnico de especificación de casos de prueba. En la Figura 4.3 se presenta la topología 1, donde se aprecian los participantes de la topología, sus direcciones IPv6, máscaras de red y la distribución de los routers.

Automatización de la configuración

La automatización de los casos de prueba permite su ejecución de manera continua, sin intervención humana. Su importancia radica en que las tareas de testeo pueden consumir demasiado tiempo y demandar interacción con el usuario, la cual es fuente de posibles errores y por tanto puede disminuir la calidad del testeo. Además es deseable que las pruebas sean reproducibles, de modo de poder ejecutar una serie de casos de prueba predeterminada, en distintos momentos. Por lo tanto, es de gran utilidad la configuración automática de la topología utilizada en la ejecución. Para lograrla, se creó una subred administrativa encargada de llevar

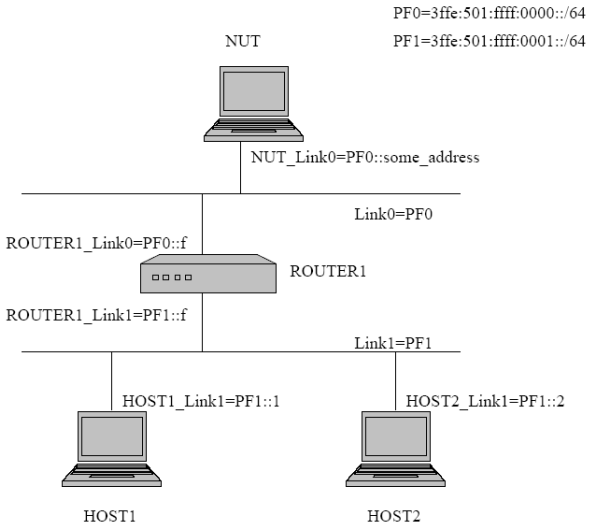

Figura 4.3: Topología 1

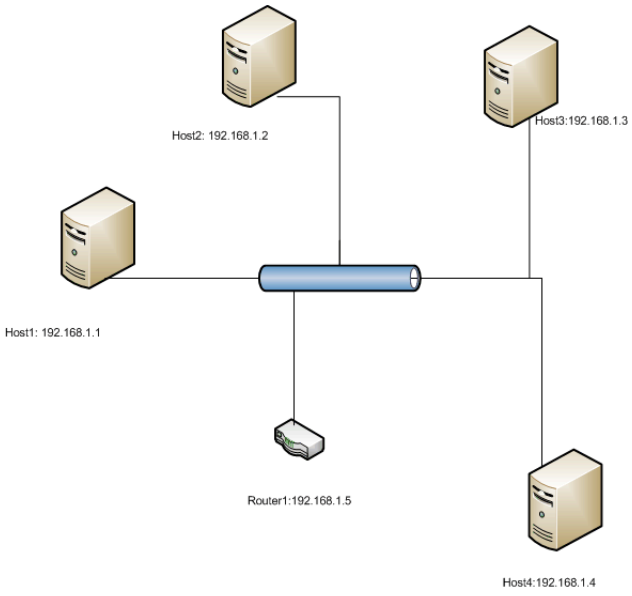

Figura 4.4: Subred Administrativa

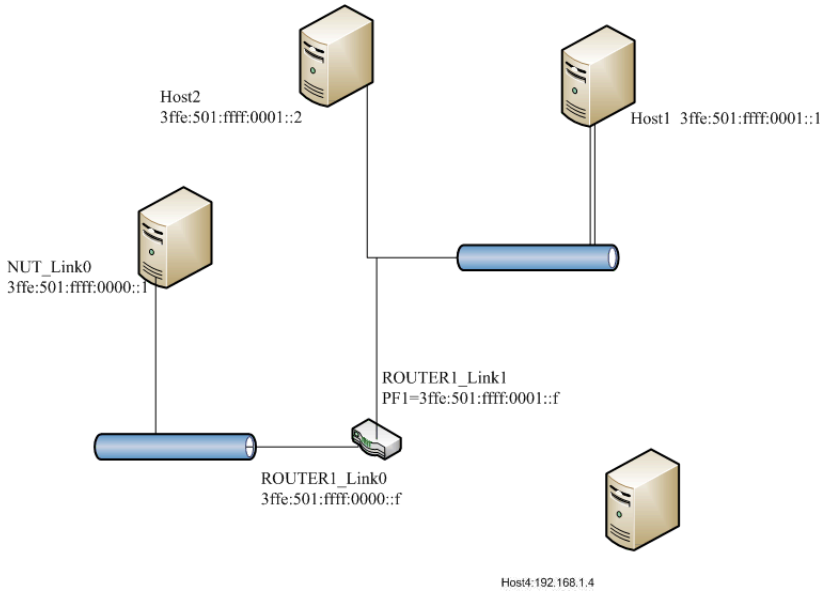

Figura 4.5: Subred generada sobre IPv6

a cabo las configuraciones intermedias. Esta subred tiene acceso a todos los nodos y a partir de ellos puede generar remotamente las diferentes topologías.

A continuación en las figuras 4.4 y 4.5 se presenta un ejemplo de cómo es creada la topología de manera automática, mostrando la misma subred en los diferentes estados. En la Figura 4.4 se muestra la subred administrativa sobre IPv4. A partir de ésta, se crea la topología sobre IPv6 necesaria al momento de la ejecución de un caso de prueba como se muestra en la Figura 4.5. Mediante la ejecución de comandos de manera remota, es llevada a cabo la actividad de *Configuración de topología de red* en forma automática. La automatización se consigue mediante una llamada al sistema, la cual ejecuta el programa *ssh* con claves RSA (pública y privada).

Debido a la utilización de máquinas virtuales existe una única interfaz de red física y por lo tanto todas las interfaces virtuales comparten este recurso. En la herramienta de virtualización utilizada (VMware) se cuenta con diferentes opciones de configuración de red: bridge, nat o host-only. Dado que es necesario que al NUT no le lleguen paquetes no deseados (que no forman parte de la prueba) se deben aislar las topologías de red. La alternativa seleccionada para esta separación es la utilización de interfaces bridgeadas y VLAN (Virtual LAN). Otra solución podría ser el uso de interfaces host-only, pero tiene como inconveniente la necesidad de recompilar VMware para cada cambio de topología.

Creación de VLAN

Las redes de área local virtuales (VLAN) [15] brindan la posibilidad de crear redes lógicamente independientes dentro de una misma red física, favoreciendo la confidencialidad de los paquetes enviados por los diferentes nodos. Las VLAN son etiquetadas para saber a que red virtual pertenecen, el protocolo de etiquetado es el IEEE 802.1Q [17].

Se utiliza VLAN para aislar las topologías de red utilizadas en los casos de prueba, cabe destacar que es necesaria la creación de una VLAN por cada topología de la especificación.

A continuación se describe la secuencia de pasos a seguir para la creación de VLAN utilizando máquinas virtuales. Notar que VMware no proporciona soporte para VLAN, por lo que debe aplicarse una modificación al archivo *bridge.c*, la misma se especifica en [39], y fue proporcionada para este proyecto. Por más información ver Apéndice B, el cual contiene un manual de instalación y configuración de los casos de prueba realizados.

Secuencia de pasos para la creación de VLAN:

1. Declaración de interfaces puenteadas (bridged) en VMware: mediante la ejecución de «vmconfig.pl» se compila VMware; durante la compilación se declaran las interfaces puenteadas (las interfaces vmnetX son puenteadas a ethY.X) que son usadas en las máquinas virtuales.

2. Modificación de interfaces en máquinas virtuales: en cada máquina virtual utilizada se debe configurar las interfaces de red «Custom», seleccionando para cada caso la «vmnetX», de esta forma a cada máquina virtual le corresponde una interfaz de red no conectada con las demás.

3. Creación de puente (bridge): se debe crear un puente al cual son agregadas las VLAN necesarias.

4. Creación de VLAN: para cada ethY.X declarada en el paso *1* se debe crear una VLAN.

5. Agregado de VLANs al puente: las VLANs creadas en el paso *4* deben ser agregadas al puente, es aquí donde se definen los participantes del puente.

Las actividades realizadas hasta el momento corresponen a la configuración de la topología de red, también es necesario la configuración IPsec, la misma es explicada en la sección 4.2.1.2.

Security Association Database (SAD) for SA1-I

source address	HOST1_Link1
destination address	NUT_Link0
SPI	0x1000
mode	transport
protocol	ESP
ESP algorithm	3DES-CBC
ESP algorithm key	ipv6readylogo3descbcin01
ESP authentication	HMAC-SHA1
ESP authentication key	ipv6readylogsha1in01

Figura 4.6: Inicialización correspondiente a IPsec [32]

4.2.1.2. Configuración de IPsec

Las actividades realizadas en esta etapa corresponden a la configuración de IPsec sobre los nodos de la topología. En esta sección veremos cómo, a partir del punto *Inicialización* de la especificación de los casos de prueba, se obtienen los valores necesarios para la configuración. En la Figura 4.6 se muestra un ejemplo de la sección de *Inicialización* extraída de la especificación de un caso de prueba. La forma de indicar qué Base de Datos de Asociación de Seguridad (Security Asociation Database, SAD) es utilizada es mediante un identificador, el mismo se puede apreciar en la Figura 4.2. Para la especificación completa de una SAD es necesaria la siguiente información:

- Dirección IP origen
- Dirección IP destino
- Tipo de Protocolo IPsec
- SPI (Índice parámetro de seguridad)
- SN (Número de secuencia)
- Algoritmo ESP de encriptación y clave de encriptación
- Algoritmo ESP de autenticación y clave de autenticación

En la Figura 4.6 se muestra un ejemplo de cómo se especifican todos los valores iniciales, necesarios para configurar la asociación de seguridad.

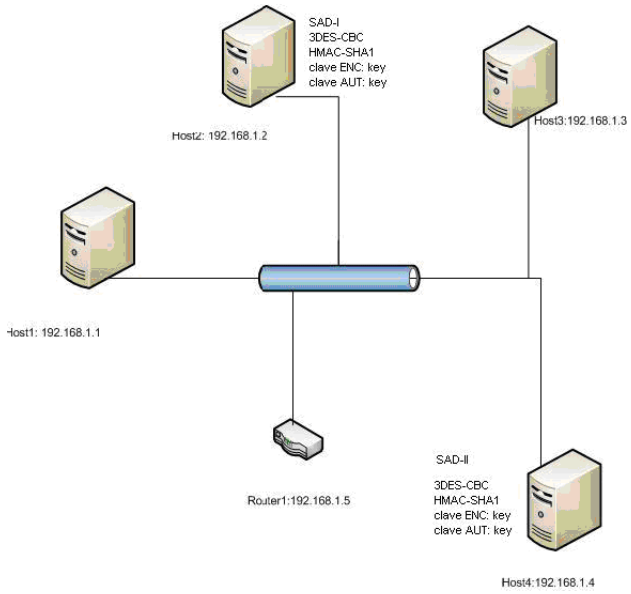

Figura 4.7: Configuración de IPsec

En esta etapa se deben asignar las asociaciones de seguridad (SA), para lo cual en los sistemas Linux se utiliza el programa *setkey*. Cabe destacar que las asociaciones de seguridad para el NUT se configuran automáticamente para facilitar el testeo. Finalmente se efectúan pruebas de conectividad utilizando IPsec, por ejemplo, mediante el comando *ping6* [11].

En esta etapa son creados los scripts de configuración de IPsec a partir de la información obtenida de la especificación, los cuales son invocados antes de la ejecución del caso de prueba de forma autómatica con el objetivo de que sea transparente para el usuario.

Automatización

De manera análoga a la automatización correspondiente a la *Configuración de topología de red*, se efectúa la *Configuración de IPsec* empleando la misma subred administrativa para configurar los nodos correspondientes. Los scripts generados anteriormente son ejecutados de manera remota. A partir de su ejecución se logra la automatización de la configuración. En la Figura 4.4 se aprecia la subred administrativa que corresponde al estado inicial de la red y en la Figura 4.7 se observa la misma subred luego de efectuada la ejecución de los scripts de configuración de IPsec.

En la Figura 4.7 se observa la subred configurada sobre IPv4, donde se configura manualmente IPsec asignando tanto asociaciones de seguridad como claves y SPI.

4.2.2. Comienzo de registro

Durante la ejecución resulta de utilidad la posibilidad de capturar los paquetes enviados y recibidos para comprobar su correcto funcionamiento, es decir, observar si los paquetes enviados poseen el formato e información esperados. También es importante contar con el registro de los mismos ya que es utilizado como comprobante que el test fue ejecutado correctamente.

En la actividad *Comienzo de registro* se comienza a registrar tanto los paquetes enviados como los recibidos durante la ejecución del caso de prueba. Esto se realiza por ejemplo con el programa *wireshark [21]*. Adicionalmente, se guarda el identificador del proceso (Pid) que es utilizado en la actividad de *Finalización de monitoreo de mensajes*.

4.2.3. Ejecución del caso de prueba

En la actividad *Ejecución de la prueba* se ejecuta la prueba, almacenando la información de los resultados obtenidos en ella (la implementación del caso de prueba es descrita en la sección 4.2). La metodología planteada propone dos modos de ejecución: modo normal y modo debug. El modo debug permite la detención del sistema testeador antes y después de realizar las actividades de preámbulo, luego de la recepción de un mensaje y antes de ejecutar el postámbulo. Mientras que el modo normal ejecuta los casos de prueba sin detenerse.

4.2.4. Fin de registro

Al culminar la ejecución se lleva a cabo la actividad de *Fin de registro*, la misma consiste en terminar el proceso de monitoreo de paquetes y en la creación de un archivo con la información de los paquetes enviados y recibidos durante el caso de prueba. Este archivo permite observar los mensajes enviados por el NUT y confirmar los resultados de los tests. De esta forma se procede a la finalización del caso de prueba correspondiente a la etapa de *Postámbulo*.

4.2.5. Postámbulo

En esta etapa se restaura el sistema luego de la ejecución de un caso de prueba.

Al igual que la etapa de *Preámbulo* consiste en las siguientes actividades:

- Configuración de topología de red
- Configuración de IPsec

La restauración del sistema consiste en realizar las actividades de la etapa de *Preámbulo* de manera inversa, deshaciendo los cambios descritos en la sección 4.1. Cada comando ejecutado tiene la opción para quitar las configuraciones asignadas previamente.

4.3. Metodología de Especificación y Codificación

Una vez descrita la *Metodología de ejecución* nos concentraremos en el segundo componente de la *Metodología de Implementación* el cual consta de la especificación y codificación en TTCN-3, utilizando C++ como lenguaje complementario.

Tanto la especificación como la codificación son independientes de la configuración realizada en la sección 4.2, sin embargo la *Inicialización* es tenida en cuenta al momento de la construcción de los casos de prueba. Los valores iniciales son usados en el envío y la recepción de paquetes, estos valores son utilizados para los cálculos criptográficos realizados en la generación de paquetes enviados y en el procesamiento de los paquetes recibidos, así como también al momento de efectuar el envío a una dirección IPv6 determinada.

Esta sección se compone de la subsección 4.3.1 *Consideraciones Previas*, donde se presentan particularidades del lenguaje TTCN-3; la subsección 4.3.2 *Paquetes*, donde se describen las actividades realizadas en la metodología de implementación que se corresponden al punto *Paquetes* de la especificación; la subsección 4.3.3 *Procedimiento*, donde se explican los pasos a seguir para llevar a cabo el caso de prueba; en la sección 4.3.4 *Fallo*, donde se describe la obtención del veredicto final, por último se presenta la sección 4.3.5 *Validación,* aquí se especifica cómo se comprueba que la implementación se apega a la especificación.

4.3.1. Consideraciones Previas

En esta sección se muestran algunas consideraciones particulares del lenguaje TTCN-3 que serán de ayuda para comprender mejor la metodología de especificación y codificación en TTCN-3. En la sección 4.3.1.1 se explica la creación de plantillas (templates) de tipos en TTCN-3, esta actividad es llevada a cabo dentro del componente *Test Template*, desarrollado en TTCN-3 y es empleada para

```
type record DnsMsg // simplified message structure!
{
  DnsMsgKind  kind,
  charstring  question,
  charstring  answer optional
}
type enumerated DnsMsgKind {e_query, e_response}

template DnsMsg m_dnsQuestion( charstring p_question )
{
  kind := e_query,
  question := p_question,
  answer := omit  // no answer
}


template DnsMsg mw_dnsAnswer( charstring p_answer )
{
  kind := e_response,
  question := ?, // any question ok
  answer := p_answer
}
```

Figura 4.8: Mensaje de DNS

el manejo de datos de entrada y salida. Luego en la sección 4.3.1.2 se presenta la creación de funciones externas al lenguaje TTCN-3.

4.3.1.1. Creación de Templates en TTCN-3

El lenguaje TTCN-3 proporciona un mecanismo que permite el manejo de entrada y salida de manera transparente al programador. Éste debe especificar el tipo de datos que desea utilizar y las plantillas (templates) que necesita. En la Figura 4.8 se observa un ejemplo de uso obtenido de *Introduction to TTCN-3* [40] en el cual se crea un tipo de datos *DnsMsg* y luego se generan dos templates (*m_ dnsQuestion* y *mw_ dnsAnswer*) que devuelven un valor de tipo *DnsMsg*.

Un tipo de datos puede tener asociado más de un template. Los templates son definidos en el componente *Test Templates* visto en el Capítulo 3 correspondiente a la arquitectura de implementación, y son los responsables de manejar la información referente a la creación de un tipo de datos a partir de los datos de entrada, así como la creación de los mensajes a enviar. Estos tipos de datos junto a sus valores asignados son enviados y recibidos por intermedio de los enlaces definidos en el componente *Port* desarrollado en TTCN-3. La definición de un template también es utilizada para controlar el valor de algunos campos, por ejemplo, puede ser útil definir un template con el valor esperado luego de una respuesta de forma de poder compararlo con el valor real obtenido en la ejecución; en caso de observar diferencias el programador puede tomar las medidas al respecto.

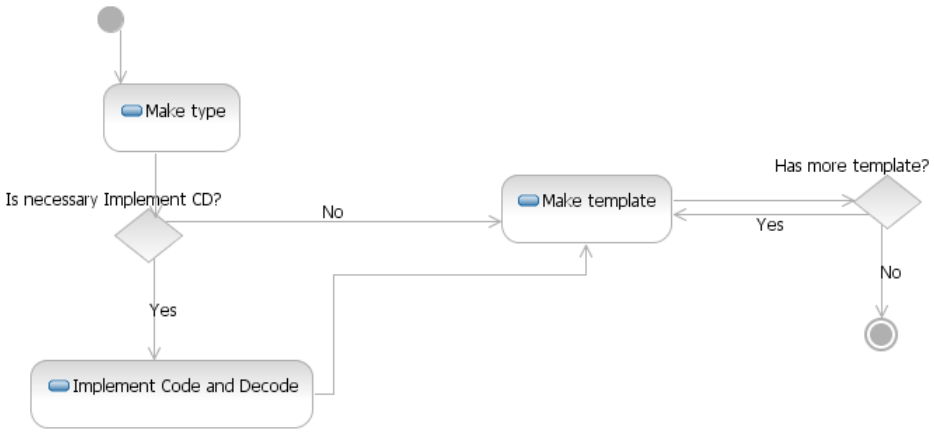

Figura 4.9: Creación de tipo y template

Debido a que los campos de los protocolos en general son de largo variable y omitibles, no se logra abstraer completamente al programador del manejo de enlaces, por este motivo se debe poder configurar la presencia, ausencia y largo de los campos que pertenecen a un paquete determinado. Para resolver este problema se utiliza la biblioteca *T3DevLib* que proporciona la implementación de funciones que asisten el proceso de codificación y decodificación (CD, Code and Decode) para cada tipo de datos definido en el código TTCN-3. Esta implementación es efectuada en el componente *Codets* desarrollado en C++, visto en el Capítulo 3.

En la Figura 4.9 vemos un diagrama de flujo de actividades que nos indica cómo se realiza la creación de un tipo con su correspondiente conjunto de templates.

Como se observa en la Figura 4.9 en primera instancia se debe crear el tipo, esto se logra de manera análoga a la definición de tipo en lenguajes estructurados como se aprecia en el ejemplo con DNS en la Figura 4.8. Luego en caso de ser necesario se agregan las funciones que permiten la codificación y decodificación de los mensajes dentro del componente *Codets* en la sección de código C++. Luego se procede a la creación de los templates necesarios para llevar a cabo la prueba.

4.3.1.2. Funciones Externas

Las funciones externas son de gran utilidad, ya que permiten reutilizar bibliotecas implementadas con anterioridad en otros lenguajes (por ejemplo C++) reduciendo el tiempo de desarrollo en contraposición con una implementación que prescinda de éstas. Por otro lado mejora la calidad del testeo, ya que estas bibliotecas son ampliamente utilizadas, como es el caso de la biblioteca GNU Cryptographic

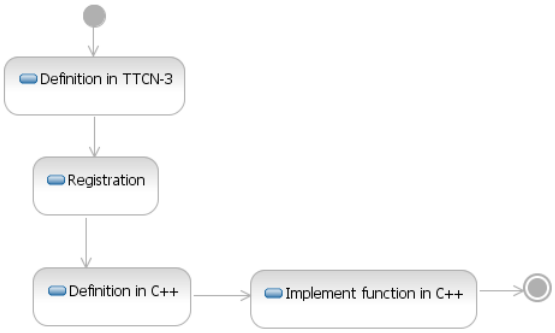

Figura 4.10: Creación de función externa

Library [2].

IPsec basa su seguridad en técnicas de encriptación y desencriptación, para ejecutar el testeo de este protocolo es necesaria la inclusión de algoritmos que desarrollen estas técnicas. Para ello se utilizan funciones externas debido a la complejidad de los mismos. La biblioteca GNU Cryptographic Library es la utilizada al momento de encriptar y desencriptar los mensajes, así como también para autenticar los mismos, ya que provee las operaciones que la especificación requiere.

Las funciones externas se deben declarar en el componente *Function* desarrollado en TTCN-3. Luego de declarar las funciones deben ser registradas esto se logra utilizando la biblioteca *T3DevLib* [10], y se efectúa dentro del componente *Connect* desarrollado en C++. Luego debe implementarse una función con el nombre que fue registrada, esta será ejecutada cada vez que se invoque a la función externa.

El pasaje de parámetros de un lenguaje a otro se realiza por intermedio de una lista, por consiguiente, al momento de crear una función en C++ debemos obtener los parámetros según su posición en la lista. Para que el código sea más legible es recomendable la creación de una función auxiliar que se encargue de la obtención de parámetros. Para ello se crea un tipo estructurado con los parámetros, una función se encarga a partir de la lista de parámetros de entrada de asignar los campos del la estructura recientemente mencionada.

La Figura 4.10 muestra las actividades realizadas para la creación de una función externa, éstas son presentadas mediante un diagrama de flujo.

La creación de una función externa se divide en cuatro pasos, primero se declara en el lenguaje TTCN-3 la función que es implementada en C++, se registra la función utilizando el framework *T3DevKit [10]*, luego se registra la función en el componente Function, y se declara en un archivo de cabecera, por último se implementa en C++. Su implementación se realiza utilizando las clases `t3devlib::ParameterList` y `t3devlib::Bitstring` para el pasaje de paráme-

Packets:

ICMP Echo Request with SA1-I's ESP

IP Header	Source Address	HOST1_Link1
	Destination Address	NUT_Link0
ESP	SPI	0x1000
	Algorithm	3DES-CBC
	Key	ipv6readylogo3descbcin01
	Authentication Algorithm	HMAC-SHA1
	Authentication Key	ipv6readylogsha1in01
ICMP	Type	128 (Echo Request)

Figura 4.11: Sección paquetes

tros de entrada y salida respectivamente.

4.3.2. Paquetes

En esta sección se explica cómo a partir del punto *Paquetes* de la especificación de los casos de prueba se obtiene la información necesaria para comenzar la implementación del caso de prueba.

A partir de la lista de paquetes que intervienen en el caso de prueba que proporciona la especificación, se generan los templates como fueron vistos en la sección 4.2.1.1, estos templates permiten el envío y recepción de paquetes. Como se observa en la Figura 4.11 cada paquete muestra los distintos campos y sus valores. La información obtenida de la lista de paquetes es usada en la creación de tipos de datos, se crea un tipo de dato para cada conjunto de campos diferente. Es recomendable la creación de subtipos, para de esta manera reducir la complejidad y facilitar la comprensión. Cada tipo declarado en código TTCN-3 se relaciona a una clase en C++ con el mismo nombre, este es el trabajo del generador de *CoDec* correspondiente al framework *T3DevKit*. Dentro del componente *Codets* son implementadas las clases que permiten codificar y decodificar los paquetes recibidos y enviados, antes y después del procesamiento.

El protocolo IPsec basa su seguridad en dos protocolos AH (Autentication Header) y ESP (Encapsulating Security Payload), sin embargo en la suite de casos de prueba no se testea el protocolo AH, ya que ESP cumple con sus funciones y además proporciona confidencialidad en los datos. Por este motivo es importante la declaración de un tipo que represente los campos del protocolo ESP. La

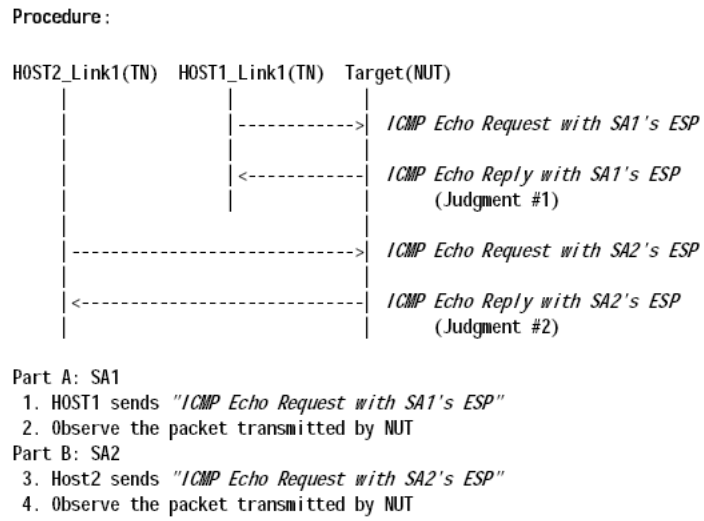

Figura 4.12: Sección procedimiento

declaración de un tipo de datos es similar a los lenguajes estructurados. IPsec, al igual que otros protocolos, contiene campos de longitud variable, por lo tanto únicamente declarando la estructura básica del tipo de datos no es suficiente. Durante el procesamiento del paquete se asignan longitudes a los campos y en caso de no encontrarse algún campo se indica la ausencia del mismo. Este procesamiento se lleva a cabo mediante la utilización de la biblioteca *T3DevLib*. Como fue explicado en la sección 4.1.1.1 un tipo de datos puede tener asociado más de un template, los templates son definidos en el componente *Test Templates*.

Al concluir esta actividad se deben crear los tipos y templates correspondientes a la especificación de paquetes que intervienen en la prueba.

4.3.3. Procedimiento

En esta sección se muestra cómo a partir del punto *Procedimiento* de la especificación de casos de prueba se obtiene la información necesaria para la implementación, especificando la estructura general y el diálogo entre los nodos participantes. Esta información comprende la interacción entre los nodos de la topología, y los mensajes observados. En la Figura 4.12 se muestra un ejemplo de especificación de *Procedimiento.*

Mediante el diagrama de la Figura 4.12 se describe la comunicación entre los nodos de la topología empleada para llevar a cabo la prueba. De esta forma son in-

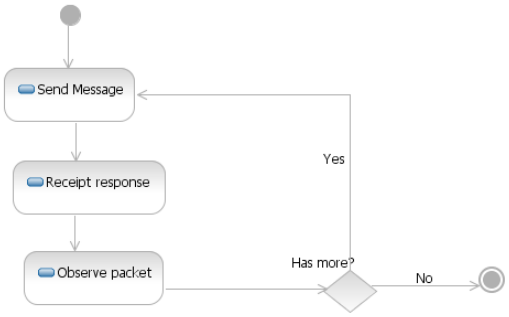

Figura 4.13: Diagrama de creación de procedimiento

dicados los paquetes enviados y recibidos por cada nodo. Es importante mencionar que los paquetes que intervienen en el procedimiento son los mismos que los de la sección 4.2.2 *Paquetes.*

Para establecer la comunicación entre los nodos participantes se debe definir un nuevo enlace para cada par de nodos, esta definición se realiza en el componente *Port* incluido en la sección de código TTCN-3. La implementación concreta del enlace se realiza especializando el tipo `t3devlib::Port` que proporciona el framework T3DevKit, una única vez. Cabe destacar que puede existir más de un nodo encargado de llevar adelante la prueba (más de un TN).

En la Figura 4.11 se especifica qué asociaciones de seguridad utiliza cada host (conjunto de datos necesarios para configurar IPsec). Para llevar a cabo el test es necesario encriptar y desencriptar el mensaje, así como autenticar y comprobar la autenticación del mismo. Esto es realizado utilizando funciones externas a TTCN-3, implementadas mediante la biblioteca Libgcrypt[2], que contiene los algoritmos necesarios para IPsec, como fue mencionado en el Capítulo 3. El manejo de las funciones externas se explicó en la sección 4.3.1.2. Se debe crear una función externa encargada de la encriptación, que será utilizada al momento del envío del mensaje. Así mismo es necesario crear una función externa encargada de la desencriptación de mensajes, que será utilizada en la recepción de los mismos. Análogamente a la encriptación de mensajes se realiza la autenticación. Las implementaciones de criptografía son realizadas en el compone *External Function* correspondiente a la sección de código C++.

Las actividades que se deben realizar para la implementación del procedimiento se detalla mediante un diagrama flujo en la Figura 4.13.

Una vez enviado un mensaje se espera a recibir una respuesta, cuando ésta llega se examina su contenido, como se ve en la sección 4.3.4. En caso que resten mensajes por enviar se retorna al paso inicial. Notar que para cada envío de mensaje

se utilizan los enlaces definidos en el componente *Port* correspondiente a la sección de código.

Al finalizar esta actividad es codificado lo referente a envío y recepción de mensajes para lograr la prueba.

4.3.4. Fallo

En esta sección se muestra cómo a partir del punto *Fallo* de la especificación de casos de prueba se establecen las actividades realizadas para la especificación en TTCN-3. En la Figura 4.14 se presenta un ejemplo del punto *Fallo* extraído del documento técnico de especificación de casos de prueba.

```
Judgment:

Part A: Judgment #1
 Step-2: NUT transmits "ICMP Echo Reply with SA1's ESP".
Part B: Judgment #2
 Step-4: NUT transmits "ICMP Echo Reply with SA2's ESP".
```

Figura 4.14: Sección procedimiento del archivo de especificación [32]

Como se observa en la Figura 4.14, es necesario recibir del NUT dos paquetes ICMPv6, uno con Asociación de seguridad 1 (SA2) y otro con la Asociación de seguridad 2 (SA2).

De manera que es necesaria la desencriptación y comprobación de integridad de los mensajes recibidos, esto se realiza invocando funciones externas a TTCN-3 que devuelven el contenido del mensaje. Esta etapa está fuertemente relacionada con el punto *Procedimiento*, dado que los mensajes observados en el mencionado punto son evaluados durante el *Fallo*. Aquí se les compara (luego de realizar la desencriptación) con el valor esperado y de esta forma se concluye la prueba. Cabe destacar que el caso de prueba es aprobado únicamente si todas las partes que lo componen son exitosas. En caso contrario el fallo es no aprobado.

4.3.5. Validación

Luego de realizar la implementación es necesario validar que el caso de prueba fue implementado correctamente. A continuación se especifican 2 escenarios de validación (configuración correcta e incorrecta de IPsec). No se realizan pruebas con configuración de topología de red incorrecta ya que cuando se realice la prueba sobre el NUT se supone que estará configurada correctamente (es parte del caso de

prueba). La configuración de IPsec es realizada en el NUT, la validación se lleva a cabo de la siguiente manera:

1. **Configuración IPsec correcta:** Son configurados todos los nodos de forma adecuada para llevar a cabo el test. Mediante un programa de *packet sniffer* se visualiza el tráfico de la red, el mismo debe coincidir con la especificación. Se compara el tráfico visualizado en el programa de *packet sniffer* con la especificación (ATS) y luego se verifica que la prueba es aprobada.

2. **Configuración IPsec incorrecta:** Se configura la topología de red sobre IPv6 de manera correcta, pero no la configuración de IPsec. En este caso se observa de manera análoga al punto 1 el tráfico de la red. Son tomadas en cuenta las siguientes alternativas de configuración incorrecta:

   a) Configuración incorrecta de claves del protocolo ESP

   b) Configuración correcta en autenticación e incorrecta en encriptación

   c) Sin configuración de IPsec

A partir de las pruebas mencionadas validamos el caso de prueba desarrollado. Esta comprobación es a nivel de caso de prueba y es realizada en forma manual.

Síntesis

En este capítulo se estableció una metodología de implementación para el desarrollo de la suite de tests especificada en el documento técnico que proporciona el comité de IPv6 Ready Logo Program para IPsec.

La metodología de implementación presentada es dividida en dos partes, por un lado la metodología de ejecución de casos de prueba, en donde se describe las actividades de configuración de topologías de red, configuración de IPsec y automatización de estas configuraciones, es especificada también la creación de VLAN para las diferentes topologías, el registro de los paquetes enviados y recibidos, y la ejecución de los casos de prueba. Por otro lado la metodología de especificación y codificación describe las actividades relacionadas con el desarrollo de los casos de prueba en TTCN-3 a partir de la especificación de tests.

La metodología de ejecución se compone de las siguientes etapas: preámbulo, comienzo de registro, ejecución del caso de prueba, fin de registro y postámbulo. En la etapa de preámbulo se realiza la configuración necesaria para llevar a cabo

la prueba, la misma se realiza de manera automática para de esta forma ahorrar tiempo en las tareas de testeo y mejorar la calidad del mismo ya que elimina posibles errores manuales. Luego se comienza el registro de paquetes enviados y recibidos, para ejecutar el caso de prueba. Al culminar la ejecución se procede a la finalización del registro de paquetes y luego se restaura el sistema.

En la metodología de especificación y codificación se obtiene el código fuente de cada caso de prueba en TTCN-3. La metodología de implementación es basada en el documento técnico de especificación, utiliza los puntos: Paquetes, Procedimiento y Fallo. Para cada uno de estos puntos se especifican las actividades que deben llevarse a cabo. Los paquetes especificados en el documento técnico son utilizados para la creación de tipos y templates en TTCN-3. Mientras que el procedimiento especificado es utilizado para la construcción del cuerpo del caso de prueba. Se especifica en TTCN-3 los paquetes enviados y los paquetes que deben observarse, para el envío de paquetes se utilizan los templates diseñados en el punto Paquetes y funciones externas al lenguaje para la encriptación de estos. En el fallo los paquetes observados son evaluados según la especificación, y de esta forma obtenemos el veredicto final de la prueba.

Capítulo 5

Implementación de un caso de prueba

En este capítulo vemos cómo se implementa un caso de prueba a partir de la metodología de implementación descrita en el Capítulo 4. Las secciones que componen este capítulo están directamente relacionadas con las secciones del Capítulo 4. En la sección 5.1 se presenta la implementación correspondiente al punto *Metodología de Ejecución* y en la sección 5.2 se describe la solución correspondiente al punto *Metodología de Especificación y Codificación*.

El caso de prueba seleccionado para su implementación es el *5.2.1 Transport Mode ESP=3DES-CBC HMAC-SHA1*, en esta prueba se verifica que el nodo bajo prueba (NUT, Node Under Test) implemente correctamente los algoritmos 3DES-CBC (cifrado) y HMAC-SHA1 (autenticación). En el *Apéndice A* podemos ver los casos de prueba seleccionados, su propósito y sobre qué IETF RFCs se basan.

5.1. Ejecución

En la presente sección se describe cómo se obtiene una implemetación concreta para el caso de prueba seleccionado, tomando como referencia el punto *Metodología de Ejecución* de la metodología vista en el Capítulo 4.

En la sección 5.1.1 se explica la etapa de *Preámbulo*, en la sección 5.1.2 se presentan las etapas de *Ejecución del caso de prueba, Comienzo de registro y Fin de registro*. La etapa de *Postámbulo* se desarrolla de manera análoga a la de *Preámbulo*, y por este motivo no será descrita, ya que no aporta nuevos conocimientos.

5.1.1. Preámbulo

Como se describe en el Capítulo 4, la etapa de *Preámbulo* obtiene la información necesaria para su implementación del punto *Inicialización* perteneciente a la especificación de casos de prueba [32]. En él se especifica la «topología 1» como la topología de red a utilizar, la misma se observa en la Figura 4.3. En ella se especifican los nodos que participan, lo cual, primera instancia da la idea de la cantidad de máquinas virtuales necesarias. Se deben crear tres máquinas virtuales y un router (realizado con otra máquina virtual) para la realización de esta topología.

En la sección 5.1.1.1 se describen las actividades necesarias para la configuración de la topología de red e IPsec. En la sección 5.1.1.2 se detalla la etapa de *Creación de VLAN* y por último en la sección 5.1.1.3 es desarrollada la etapa de *Automatización*.

5.1.1.1. Configuración de topología de red e IPsec

La configuración de la topología de red consta de dos partes, por un lado configuración de direcciones IPv6 y por otro la configuración de tablas de ruteo. Luego se describen las actividades realizadas durante la configuración de IPsec.

Configuración de direcciones IPv6

En la presente sección se describe la configuración de direcciones IPv6 necesarias para implementar la topología 1 de la especficación de casos de prueba.

En la Figura 5.1 se especifican las direcciones IPv6 necesarias correspondiente al nodo testeador (TN).

```
#Interface utilizada
ETH="eth1"
#Ruta del comando ifconfig
IFCONFIG="ifconfig"
#Ruta del comando ip
IP="/sbin/ip"
#Dirección IPv6 a setear
IPv6_ADDRESS="3ffe:501:ffff:1::1"
#Mascara de subred
IPv6_NETMASK="64"

$IFCONFIG $ETH up
$IP -6 addr add $IPv6_ADDRESS/$IPv6NETMASK dev $ETH1
```

Figura 5.1: Script configuración TN

Las configuraciones en términos de IPv6 y máscara de red correspondientes al router y al NUT son análogas a las realizadas para configurar el TN. Se debe cambiar IPv6_ADDRESS por los valores indicados, en caso del router se debe modificar la variable ETH y la variable IPv6_ADDRESS.

Configuración de tablas de enrutamiento

En la presente sección se describe la configuración de tablas de enrutamiento necesarias para implementar la topología 1. En la Figura 5.2 se especifica la configuración de tablas de enrutamiento correspondiente al nodo que representa el router.

```
#Interfaces utilizada
ETH_1="eth1"
ETH_2="eth2"
#Ruta de archivo para habilitar forwarding
FORWARDING="/proc/sys/net/ipv6/conf/all/forwarding"
#Ruta del comando ifconfig
IFCONFIG="ifconfig"
#Ruta del comando ip
IP="/sbin/ip"
#Dirección IPv6 a setear
IPv6_ADDRESS1="3ffe:501:ffff::"
IPv6_ADDRESS2="3ffe:501:ffff:1::"
LOCAL ="f"

#Mascara de subred
IPv6_NETMASK="64"

#Habilito reenvío de paquetes
echo "1" > $FORWARDING

#Agrego ruta en tabla de ruteo
$IP -6 route add to $IPv6_ADDRESS1/$IPv6NETMASK via  $IPv6_ADDRESS$LOCAL  dev  $ETH_2

#Agrego ruta en tabla de ruteo
$IP -6 route add to $IPv6_ADDRESS2/$IPv6NETMASK via  $IPv6_ADDRESS$LOCAL  dev  $ETH_1
```

Figura 5.2: configuración de tablas de ruteo en Router

Las configuraciones de TN y NUT son similares se debe agregar una sola ruta y no se debe configurar el reenvío de paquetes.

Configuración de IPsec

La configuración de IPsec son efectudas en la NUT y en TN, en la Figura 5.3 se especifica la configuración de IPsec correspondiente al nodo que representa NUT.

```
add 3ffe:501:ffff:1::1 3ffe:501:ffff::1 esp 0x00000100 -m transport -E 3des-cbc
"ipv6readylogo3descbcin01" -A hmac-sha1 "ipv6readylogsha1in01";
add 3ffe:501:ffff::1 3ffe:501:ffff:1::1 esp 0x00000200 -m transport -E 3des-cbc
"ipv6readylogo3descbcout1" -A hmac-sha1 "ipv6readylogsha1out1";
spdadd 3ffe:501:ffff::1 3ffe:501:ffff:1::1 icmp6 -P out ipsec
esp/transport//require;
spdadd 3ffe:501:ffff:1::1 3ffe:501:ffff::1 icmp6 -P in ipsec
esp/transport//require;
```

Figura 5.3: configuración de IPsec en NUT

La configuración correspondiente al TN es análoga a la del NUT, intercambiando in por out. La configuración es lograda ejecutando el programa *setkey* y como

parámetro de entrada el archivo que contiene la especificación de las SPDs y las SADs, en la configuración correspondiente al NUT. Se observa en la Figura 5.3 el archivo pasado como parámetro al programa *setkey*.

5.1.1.2. Creación de VLAN

La creación de VLAN se incluye para separar virtualmente las redes. Se debe crear una VLAN para cada topología correspondiente a la especificación y una VLAN administrativa la cual es la encargada de la automatización de los casos de prueba.

En primera instancia debemos modificar el fuente de VMware, ya que no soporta la implementación de VLAN. Luego debemos compilar VMware, esto lo logramos ejecutando «VMware-config.pl». Posteriormente se debe agregar las interfaces que serán usadas en la implementación de las VLAN. En la Figura 5.4 se aprecia un ejemplo de cómo se agregan interfaces brigeadas a VMware, por más información ver *Apéndice B*.

```
Would you like to skip networking setup and keep your old settings as they are?
(yes/no) [no]

Do you want networking for your virtual machines? (yes/no/help) [yes]

Would you prefer to modify your existing networking configuration using the
wizard or the editor? (wizard/editor/help) [editor]


Which virtual network do you wish to configure? (0-99) <numero de vlan>

(bridged,hostonly,nat,none) [bridged]

Your computer has multiple ethernet network interfaces available:
eth1. Which one do you want to bridge to vmnet18? [eth1.<numero de vlan>]
```

Figura 5.4: Agregar redes bridgeadas sobre VMware

Se agregan tantas interfaces virtuales como nodos de la topología, también es necesario agregar una interfaz más para la subred administrativa que se encuentra en una VLAN diferente.

Al finalizar la compilación se debe configurar cada interfaz de las máquinas virtuales como «Custom», haciendo referencia a una de las interfaces agregadas en el paso anterior.

En la Figura 5.5 se ve mediante comandos Linux la implementación de la VLAN de trabajo, correspondiente a la topologia 1.

```
HOSTS_VLANS="13 14 15 16 17"
ETH="eth1"
VCONFIG="vconfig"
IFCONFIG="ifconfig"
BRCTL="brctl"
TEST_VLAN="IPsec"

$BRCTL addbr $TEST_VLAN
# levanto interface brige
$IFCONFIG $TEST_VLAN up

# creo vlans
for i in $HOSTS_VLANS; do
        $VCONFIG add $ETH $i
        sleep 1
        $IFCONFIG $ETH.$i 0.0.0.0
        $IFCONFIG $ETH.$i up
done
# agrego intefaces al brigde
for i in $HOSTS_VLANS; do
        $BRCTL addif $TEST_VLAN $ETH.$i
done
$BRCTL show
```

Figura 5.5: Creación de VLAN en Linux

5.1.1.3. Automatización

Se debe crear una subred sobre las máquinas virtuales creadas, donde el nodo testeador (TN) acceda a todos los nodos de la topología y pueda configurar de forma remota los nodos de la misma. Para lograr la ejecución de comandos de forma remota se utiliza *ssh* con autenticación de claves RSA (claves pública y privada). Para su uso, es necesario generar la clave, esto se logra mediante la ejecución del comando *ssh-keygen*. Luego se debe agregar la clave pública en todos los nodos de la topología. La ejecución de los scripts que «construyen» la topología se realiza de la siguiente manera: `ssh <ip>-l <user><script_constructor>`. Por más información ver *Apéndice B* el cual sirve de guía en la instalación y configuración de la implementación realizada.

Durante la ejecución de un caso de prueba se ejecuta el script encargado de la configuración de la topología y otro encargado de la configuración de IPsec, los mismos fueron presentados en la sección 5.1.1.1.

5.1.2. Registro de mensajes

El registro de mensajes es realizado mediante el comando *tcpdump* de la siguiente manera: `tcpdump -i ethX -w "testID.dump" & echo $! >PID_TEST`

Además de registrar el tráfico de la red, se guarda el identificador de proceso, que será utilizado en la etapa de finalización de registro.

5.1.3. Ejecución

La implementación de los modos de ejecución es realizada de manera simple por intermedio de una variable (flag) global, la cual indica el modo. En caso de ser modo debug espera recibir un carácter para continuar con el proceso, en caso contrarío continua la ejecución.

5.1.3.1. Desplegar resultados

Luego de obtener el veredicto del test, el mismo es almacenado y desplegado en un archivo HTML, para de esta forma presentar una interfaz amigable al usuario final.

5.1.4. Fin de registro

La finalización del registro de mensajes se realiza en Linux mediante la ejecución del siguiente comando: `kill 'cat PID_TEST' rm PID_TEST`. Esta tarea es llevada a cabo luego de ser almacenado el identificador del proceso en la etapa de comienzo de registro. De esta forma se termina con el registro de los mensajes enviados durante la ejecución de la prueba, creando un archivo `"testID.dump"` que contiene el registro de los mismos.

5.2. Especificación y Codificación

Se describe aquí la implementación en TTCN-3, y en menor medida la implementación en C++. La misma se divide en secciones para de esta manera continuar con el formato de la *Metodología de Implementación* descrita en el Capítulo 4. La sección 5.2.1 contiene la implementación correspondiente al punto *Paquetes*, la sección 5.2.2 describe la implementación del punto *Procedimiento*; por último la sección 5.2.3 presenta la implementación concerniente al punto *Fallo*.

5.2.1. Paquetes

En esta subsección se presenta una implementación posible siguiendo con la metodología descrita en el capítulo 4. Los paquetes enviados en este caso de prueba son del tipo ICMPv6 con ESP.

Se obtiene la información necesaría para la implementación del tipo y template a partir de la especificación de la prueba en el punto *Paquetes*, ésta se observa en

la Figura 2.6. Para su implementación se especifica el tipo ESPMessage, el mismo representa un mensaje del protocolo ESP (Encapsulating Security Payload). Los campos necesarios para la implementación del caso de prueba son, el índice de parámetro de seguridad (SPI, Security Parameter Index), Número de secuencia (SeqNum), datos a transferir (Payload) y por último los datos de autenticación (ICV, Integrity Check Value). En la Figura 5.6 se presenta la definición del tipo ESPMessage así como también la creación de dos templates que resuelven el envío y recepción de mensajes ESP. Estos templates son utilizados en la creación del template de ICMPv6, el mismo se desarrolla de manera análoga al de ESP.

```
/**
Definición de tipo ESPMessage, utilizado para el envío y recepción.
*/
  type record ESPMessage {
   octetstringSPI length(4),
   UInt32 SeqNum,
   octetstringPayload,
   octetstringICV optional // Authentication Data
  }

/**
Definición de template del tipo ESPMessage, utilizado para el envío, se crea una estructura del tipo ESPMessage
asignando los valores de entrada a los campos correspondiente al tipo.
*/
template ESPMessage ICMPv6ESPMessage (Ipv6AddressType src, Ipv6AddressType dst, octetstring m_spi, octetstring m_data,
UInt16 checksum, octetstring encpay) := {
  SPI := m_spi,
  SeqNum := 1,
  Payload := encpay,
  ICV := GetICV(m_spi,1,Icmpv6EchoRequestType,encpay,checksum,src, dst)
}
/**
Definición de template del tipo ESPMessage, utilizado para la recepción, se crea una estructura del tipo
ESPMessage asignando los valores decepcionados y confirmando la presencia del campo SPI y su valor, los demás
campos serán analizados luego.
*/
template ESPMessage ICMPv6ESPMessage_Answer (octetstring m_spi) := {

  SPI:= m_spi,
  SeqNum := ?, // valores que por el momento no son conocidos.
  Payload := ?,
  ICV := ?
}
```

Figura 5.6: Definición de template

En la Figura 5.6 se muestra cómo es creado el tipo ESPMessage y dos templates, uno para enviar y otro para recibir mensajes de este tipo.

```
bool ESPPort::Map(const PortId& port_id) {

socket_ = socket (PF_INET6, SOCK_RAW, IPPROTO_RAW);
struct sockaddr_in6 my;
my.sin6_family = PF_INET6;
inet_pton(AF_INET6, FROM_IP_ADDR, &(my.sin6_addr));
my.sin6_port=0;
socket_ = socket (PF_INET6, SOCK_RAW, IPPROTO_ESP); // se crea el socke seteando IPPROTO_ESP
if ((bind(socket_, (struct sockaddr*) &my, sizeof(my)) == -1)
||(socket_ == -1))
{
   return false;
}
listen_s = socket (PF_INET6, SOCK_RAW, IPPROTO_ESP);

if (listen_s == -1) {
   return false;
}

if (pthread_create (&thread_, NULL, &icmp_thread, this)) {
   close (listen_s);
   listen_s = -1;
   return false;
}

// Recuerdo el id de port
connectedPort_ = &port_id;
}
```

Figura 5.7: Implementación de ESPPort

```
testcase tc_5_2_1 () runs on TestComponentTest5_2_1 {
timer replyTimer;

      var ESPMessageAnswer Myvar;
// Genero checksum que será incluido en el mensaje a enviar utilizando la función externa
// GetCheckSum(PF1_1, PF0_1, Icmpv6EchoRequestType, DATA, NextHeaderIcmpV6);

var UInt16 checksum := GetCheckSum(PF1_1, PF0_1, Icmpv6EchoRequestType, DATA, NextHeaderIcmpV6);
// Genero el Payload encriptado que será incluido en el mensaje a enviar
var octetstring encpay := EncriptPayload(PF1_1, PF0_1, Icmpv6EchoRequestType, DATA,checksum);
// envío el mensaje, este mensaje es generado a partir del template que genera un tipo ICMPv6_EchoRecuest
Link1.send(ICMPv6WithESP_EchoRequest_AuthSHA1(PF1_1, PF0_1, SPI_SA1, DATA, checksum,encpay));
// Inicializo timer en 30 segundos
      replyTimer.start(30.0);
// Se espera a recibir un mensaje
      alt {
      //Recibo el mensaje correcto
[] Link1.receive(ICMPv6ESPMessage_Answer(PF0_1, PF1_1, SPI_SA2,
DATA, checksum)) -> value Myvar {
// Actuar según fallo
}
//Se recibio un mensaje incorrecto, no cumple con la especificación del template
[] Link1.receive {
// Actuar según fallo

}
//Receive no answer
[] replyTimer.timeout {
// Actuar según fallo
}
}
  stop;
  }

}
```

Figura 5.8: Implementación del procedimiento TTCN-3

5.2.2. Procedimiento

En esta sección se presenta la implementación del caso prueba tomando en cuenta el punto *Procedimiento* de la especificación de casos de prueba. El procedimiento correspondiente a este caso, se efectúa al enviar y recibir un mensaje ICMPv6 con ESP. Se utiliza un enlace del tipo ESPPort, el cual se utiliza para enviar y recibir los mensajes ESP. Los mensajes son enviados utilizando templates, en este caso debemos crear un template para ICMPv6, que contenga el template ESPMessage. Estos templates fueron generados en la etapa de *Paquetes.*

El método *ESPPort::Map* se destaca dentro de la implementación en el lenguaje C++, ya que es el encargado de el envío y recepción de los paquetes. En la Figura 5.7 se presenta la implementación de ese método.

Luego se debe especificar el esqueleto del caso de prueba, es decir la estructura general del mismo, son indicados qué mensajes son emitidos, recibidos y en que orden se ejecuta cada acción. En la Figura 5.8 se observa la implementación en TTCN-3 siguiendo la metodología del Capítulo 4. Son utilizadas para su implementación varias funciones externas antes de enviar y luego de recibir los mensajes. La función *GetChecksum* es utilizada para generar la suma de verificación que será enviada en el mensaje. La función *EncPayload* se utiliza para encriptar el mensaje. El valor de ICV se asigna durante la ejecución del template de envío, como se observa en la Figura 5.6.

```
//Recibo el mensaje correcto
var bitstring encpayload := Myvar.Payload;//obtengo Payload del mensaje recibido
var UInt32 secNum := Myvar.SeqNum; //obtengo Numero de Secuencia del mensaje recibido
var octetstring icv := Myvar.ICV; //obtengo ICV del mensaje recibido
// comparo ICV recibido con el correcto.
if (match(icv, GetICVDec(SPI_SA2, secNum,encpayload)))
{
var UInt8 payloadLength := lengthof(encpayload)/8;
// genero el payload correcto, desencriptando el mensaje
var EncPayload payload := DecriptPayload(encpayload,
payloadLength);
// comparo payload obtenido con el payload correcto
if (match(payload, ICMPv6EncPayload_Answer(PF0_1, PF1_1, DATA)))
{
//seteo que el test pasa con éxito la prueba
setverdict(pass);
}
else
{//seteo que el test NO pasa con éxito la prueba
setverdict(fail);
}
}
else {
//seteo que el test NO pasa con éxito la prueba
setverdict(fail);
}
//detengo el timer
replyTimer.stop;
```

Figura 5.9: Implementación correspondiente a Fallo

5.2.3. Fallo

En esta sección se presenta la implementación del caso prueba 5.2.1 tomando en cuenta el punto *Fallo* de la especificación de IPv6 Ready Logo Program. Este es el punto final, donde se decide si el cumple según lo especificado. Para ello se requiere calcular los valores adecuados y luego comparar con los recibidos. En base a funciones externas son calculados los valores esperados. Las funciones utilizadas para comprobar integridad (ICV) y comprobar encriptación. Para la resolución de este caso de prueba se debe comprobar qué el valor de ICV sea correcto, luego comprobar si fue cifrado correctamente el mensaje. Teniendo en cuenta que fue recibido un paquete que cumple con las características del protocolo ESP y además el valor obtenido en SPI es el esperado según la especificación (SPI_SA2) (asegurado por la utilización del template ESPMessage realizado en la sección 5.2.1). El código de la Figura 5.9 es introducido luego de recibir el mensaje con formato correcto.

En los casos donde la recepción no fue la de un mensaje ESP correspondiente con el template, o no se recibió ningún mensaje (luego de finalizar el timer) se considera como no aprobada la prueba.

Capítulo 6

Gestión del Proyecto

En este capítulo se describe cómo fue administrado el presente proyecto. En la sección 6.1 se presenta la planificación elaborada para llevarlo a cabo, y en la sección 6.2 la ejecución de ésta.

6.1. Planificación

En esta sección se especifican las actividades desarrolladas, el período estimado y los objetivos de las mismas. La planificación es dividida en las siguientes actividades:

1. **Aprendizaje de herramientas y tecnologías a utilizar:** Período estimado: Abril 2007 / Mayo 2007. Se realiza la investigación de las tecnologías utilizadas para la resolución de los objetivos de este proyecto, éstas son:

   a) **SVN:** Es utilizado para el manejo de versiones de código fuente y de documentación

   b) **IPsec:** Es el protocolo a prueba, por ende es importante el conocimiento del mismo

   c) **TTCN-3:** Lenguaje utilizado para el desarrollo de los casos de prueba, éste es requerimiento del presente proyecto

   d) **T3DevKit:** Esta biblioteca es necesaria para la implementación de los casos de prueba. Para continuar la línea de trabajo[39] es utilizado T3DevKit

   e) **VMware:** El software de virtualización utilizado, el cual continúa la línea de trabajo mencionada en el punto (d)

Al comienzo del proyecto, de las tecnologías mencionadas, únicamente contaba con experiencia en uso de CVS (similar conceptualemente a SVN) y VMware Server.

2. **Creación de documento mapeo entre casos de prueba y RFC:** Período estimado: Abril 2007. Se realiza un documento que víncula cada caso de prueba especidificado en el documento técnico con los RFC que éste trata

3. **Búsqueda de patrones comunes en los casos de prueba:** Período estimado: Junio 2007. Se buscan patrones comunes en los casos de prueba. En esta actividad se trata de factorizar las abstracciones encontradas en la especificación de los casos de prueba. De forma de encontrar patrones que serán utilizados en el *punto 4* , para la generación de herramientas que asistan el desarrollo

4. **Generación de herramientas para asistir la construcción de los casos de pruebas:** Período estimado: Junio 2007 / Julio 2007. En esta etapa se crean herramientas que ayudan a la construcción de los casos de prueba. Debido a que TTCN-3 es un lenguaje de especificación

5. **Implementación de casos prueba:** Período estimado: Junio 2007 / Setiembre 2007. En esta actividad se especifica y codifican los casos de prueba, así como también las actividades de configuración de las máquinas virtuales.

6. **Validación de casos de prueba:** Período estimado: Julio 2007 / Setiembre 2007. En esta actividad se valida la implementación realizada en el *punto 5*

7. **Visualización web de resultados:** Período estimado: Octubre 2007. Son agregados en esta actividad las modalidades de ejecución y la visualización de los resultados a medida que estos trancurren, en formato web

8. **Desarrollo de metodología de implementación:** Período estimado: Agosto 2007 / Noviembre 2007. Se lleva a cabo la documentación correspondiente a la metodología de implementación

9. **Documentación final:** Período estimado: Noviembre 2007. En esta actividad lleva a cabo la documentación final del proyecto

6.2. Ejecución

En esta sección se describe la ejecución de la planificación implantada en la sección 6.1. Continúa con la estructura de la sección anterior y se describe

1. **Aprendizaje de herramientas y tecnologías a utilizar:** Período estimado: Abril 2007 / Mayo 2007. Esta actividad se llevó a cabo sin mayores complicaciones, se cumplieron los tiempos previstos

2. **Creación de documento mapeo entre casos de prueba y RFC:** Período estimado: Abril 2007. Esta actividad se llevó a cabo sin mayores complicaciones, se cumplieron los tiempos previstos. Este documento corresponde al Apéndice B

3. **Búsqueda de patrones comunes en los casos de prueba:** Período estimado: Junio 2007. La búsqueda de patrones fue una tarea complicada debido al gran nivel de abstracción que TTCN-3 provee. Los patrones comunes a los casos de prueba encontrados, son muy similares a los que el lenguaje contiene. Esta actividad tomo mayor tiempo del esperado y al no tener resultados relevantes se decidió finalizar la actividad. Fecha de finalización Julio 2007

4. **Generación de herramientas para asistir la construcción de los casos de pruebas:** Período estimado: Junio 2007 / Julio 2007. Esta etapa no fue realizada ya que depende directamente con la actividad 3

5. **Implementación de casos prueba:** Período estimado: Junio 2007 / Setiembre 2007. Esta actividad no se llevó a cabo en tiempo debido a falta de dedicación por temas de trabajo y de salud. Fueron agregados a la implementación de los casos de prueba, los requerimientos de automatización de la ejecución de casos de prueba y utilización de VLANS para separar las topologías. Fecha de finalización Enero 2008

6. **Validación de casos de prueba:** Período estimado: Julio 2007 / Setiembre 2007. Esta actividad depende directamente de la actividad anterior, por este motivo vió afectado de igual forma su duración. Fecha de finalización Febrero 2008

7. **Visualización web de resultados:** Período estimado: Octubre 2007. Esta actividad se desarrollo para brindar una interfaz mas amigable de los resultados de los casos de prueb. La misma finalizó con la implementación de los casos de prueba, Enero 2008

8. **Desarrollo de metodología de implementación:** Período estimado: Agosto 2007 / Noviembre 2007. El desarrollo de la metodolgía resultó ser una tarea más difícil de lo esperada, debido a las abstracciones que ésta maneja. Fecha de finalización Julio 2008

9. **Documentación final:** Período estimado: Noviembre 2007. Esta actividad se ve directamente afectada por la demora en las actividades anteriores, ya que engloba las mismas. Fecha de finalización Julio 2008

6.3. Comentarios generales

Mi principal interés en el proyecto fue el de conocer a fondo el protocolo de seguridad IPsec, así como también el desarrollo de la suite de casos en un lenguaje nuevo y orientado a la especificación, como lo es TTCN-3.

La planificación no se llevó a cabo de la manera esperada debido a varios factores, principalmente a períodos de baja dedicación provocados por problemas de salud, poca disposibilidad horaria a causa de actividades laborales y a la estimación optimista en la asignación de tiempo a tareas por errores en el análisis inicial de complejidad. Las dificultades en el desarrollo se debieron a su complejidad y a la falta de información, ya que no existe mucho material disponible sobre TTCN-3 dado que es un lenguaje nuevo.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se presentan las conclusiones obtenidas al finalizar el trabajo, y se delinean posibles trabajos futuros a partir de este proyecto.En la sección 7.1 un resumen de las actividades realizadas. En la sección 7.2 se expresan las conclusiones y por último en la sección 7.3 se describen posibles trabajos futuros.

7.1. Resumen

Fue realizada parte de la especificación de casos de prueba proporcionada por IPv6 Ready Logo Program, la mayor parte de los tests realizados fueron utilizando al NUT como END-Node, y dentro de este subconjunto la mayoría son los casos de prueba correspondientes a la sección Arquitectura de IPsec de la especificación de casos de prueba.

Se desarrolló la metodología de implementación de casos de prueba basada en la especificación mencionada, utilizando el lenguaje TTCN-3 y C++ como lenguaje complementario. Se presentó la implementación de un caso de prueba en particular, para de esta forma ejemplificar la metodología y obtener una mayor comprensión de la misma. Fueron adquiridos durante el proyecto conocimiento de protocolos, en particular IPsec e IPv6. Así como conocimiento a cerca el lenguaje TTCN-3, el cual es de gran utilidad para la verificación formal de protocolos.

7.2. Conclusiones

Si bien TTCN-3 es muy bueno para la especificación abstracta de tests, requiere

de mucho trabajo sobre el lenguaje y también sobre el lenguaje complementario (C++) para el desarrollo de los casos de prueba. El framework (T3DevKit) desarrollado por INRIA (Institut National de Recherche en Informatique et en Automatique) [3] facilita las labores de codificación y decodificación necesarias para la implementación de los casos de prueba. Esto mejora la calidad del testeo ya que elimina la introducción de errores manuales, además de facilitar la tarea del desarrollador de estos casos.

La metodología de implementación desarrollada facilita la resolución de la suite de casos de prueba, ya que estructura la solución. Organiza la realización de los tests en etapas bien diferenciadas y fáciles de seguir, lo cual reduce la complejidad en cada caso y permite la reutilización de los componentes realizados en otros casos de prueba. Esta metodología consta con el suficiente nivel de abstracción para contemplar los diferentes casos de prueba.

El caso de prueba seleccionado para su implementación en el Capítulo 5 ejemplifica la metodología de implementación desarrollada, llevando las abstracciones vistas en la metodología de implementación a la práctica en un caso de prueba especifico.

La utilización de máquinas virtuales simplificó la tarea de generación de paquetes, sin embargo trajo inconvenientes tales como las configuraciones de las topologías sobre IPv6, las configuraciónes de IPsec y la creación de VLAN de forma automática.

La selección de los casos de prueba implementados comprenden un subconjunto representativo de la suite de tests, pues abarcan las diferentes modalidades del protocolo testeado, así como también las distintas topologías de red. La implementación de los tests restantes es análoga a los tests desarrollados durante este proyecto.

7.3. Trabajos Futuros

Ha quedado pendiente el desarrollo de la suite completa de casos de prueba de esta forma comprobar si un dispositivo cumple las especificaciones del protocolo IPsec. Un posible trabajo futuro es el de realizar esta suite, basándose en la metodología realizada y los casos de prueba implementados.

La ejecución de los casos de prueba especificados en TTCN-3, es realizada en mediante línea de comandos, ya que TTCN-3 no cuenta con entorno gráfico ni un

entorno web, un posible trabajo futuro podría ser construir un entorno amigable. Un item importante es que este entorno permita observar el tráfico de paquetes enviados y recibidos en todo momento, así como también el resultado en cada prueba.

Se utilizó VMware para representar los diferentes hosts de las topologías de red, un posible trabajo futuro podría ser la investigación de software de virtualización alternativo y un posible reemplazó del mismo.

Bibliografía

[1] Internet Protocol versión 6. http://www.ipv6.org/.

[2] Criptografía GNU. http://www.gnu.org/software/gnu-crypto/, Último acceso en Mayo de 2008.

[3] Institut National de Recherche en Informatique et en Automatique. http://www.inria.fr, Último acceso en Mayo de 2008.

[4] International Telecommunication Union and European Telecommunications Standards Institute. http://www.itu.int/ITU-R/study-groups/docs/rsg6-etsi/index.html, Último acceso en Mayo de 2008.

[5] Internet Engineering Task Force Request for Comments. http://www.ietf.org/rfc.html, Último acceso en Mayo de 2008.

[6] IPsec, Internet Protocol security. http://www.ipsec.org, Último acceso en Mayo de 2008.

[7] IPv6 Logo Comitee. http://www.ipv6ready.org/, Último acceso en Mayo de 2008.

[8] IPv6 Ready Logo. http://www.ipv6ready.org/, Último acceso en Mayo de 2008.

[9] Linux. http://www.linux.org, Último acceso en Mayo de 2008.

[10] Manual de usuario T3DevKit. http://www.itu.int/ITU-R/study-groups/docs/rsg6-etsi/index.html, Último acceso en Mayo de 2008.

[11] Ping versión 6. http://www.freebsd.org/cgi/man.cgi?query=ping6, Último acceso en Mayo de 2008.

[12] QEMU. http://www.qemu.org/, Último acceso en Mayo de 2008.

[13] The Internet Engineering Task Force. http://www.ietf.org, Último acceso en Mayo de 2008.

[14] Virtual Box. http://www.virtualbox.org/, Último acceso en Mayo de 2008.

[15] Virtual LAN. http://www.vlan.org, Último acceso en Mayo de 2008.

[16] VMware. http://www.vmware.com, Último acceso en Mayo de 2008.

[17] VMware Player. http://www.vmware.com/products/player/, Último acceso en Mayo de 2008.

[18] VMware Server. http://www.vmware.com/products/server/, Último acceso en Mayo de 2008.

[19] Voz sobre IP. http://www.voiceip.org, Último acceso en Mayo de 2008.

[20] Windows NT. http://www.microsoft.com/, Último acceso en Mayo de 2008.

[21] Wireshark. http://www.wireshark.org/, Último acceso en Mayo de 2008.

[22] X86. http://www.x86.org/, Último acceso en Mayo de 2008.

[23] D. Eastlake 3rd. RFC 4305 - Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). http://www.rfc-editor.org Diciembre de 2005.

[24] D. Eastlake 3rd. RFC 793 - Transmission Control Protocol. http://www.rfc-editor.org Setiembre 1981.

[25] ISO 9646. Information technology - Open Systems Interconnection - Conformance testing methodology and framework . http://www.eurescom.de/ public-seminars/2000/IP/05Grupp/sld008.htm 1995.

[26] A. Fúster Sabater, D. de la Guía Martínez, L. Hernández Encinas, F. Montoya Vitini, J. Muñoz Masqué. Introducción a la criptografía.

[27] Scott Moseley Sebastian Mueller Anthony Wiles, Theofanis Vassiliou-Gioles. Experiences of Using TTCN-3 for Testing SIP and OSP. http://portal.etsi.org/ptcc/downloads/TTCN3SIPOSP.pdf 2007.

[28] César Viho Ariel Sabiguero, María Eugenia Corti. The new Internet Protocol security IPsec testing with TTCN-3. http://www.irisa.fr/tipi/publi/t3uc2007paper.pdf 2007.

[29] D. Eastlake 3rd. RFC 2401 - Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH). http://www.rfc-editor.org - Noviembre de 1998.

[30] Duncan Clark Eric Harwit. Shaping the Internet in China: Evolution of Political Control Over Network Infrastructure and Content. http://www.TTCN-3.org/TTCN3UC2007/Presentations/Wed/Session1/SabCorVihT3UC2007.pdf Junio de 2001.

[31] R. Housley. RFC 4309 - Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP). http://www.rfc-editor.org - Diciembre de 2005.

[32] IPv6 Logo Committee IPv6 Forum. IPv6 Ready Logo Phase II Test Specification IPsec. 2007.

[33] P. Francis K. Egevang. RFC - The IP Network Address Translator (NAT). http://www.rfc-editor.org Mayo de 1994.

[34] S. Kent. RFC 4301 - Security Architecture for the Internet Protocol. http://www.rfc-editor.org - Diciembre de 2005.

[35] H. Orman. RFC 2412 - The OAKLEY Key Determination Protocol. http://www.rfc-editor.org - Noviembre de 1998.

[36] K. Egevang P. Srisuresh. Traditional IP Network Address Translator (Traditional NAT) RFC 3022. http://www.rfc-editor.org Enero de 2001.

[37] J. Postel. RFC 768 - User Datagram Protocol. www.rfc-editor.org, Agosto de 1980.

[38] IPv6 Ready Logo Program. IPsec Phase 2 technical document. http://www.ipv6ready.org/Wed/Session1/ipsecFase218.pdf 2007.

[39] Ariel Sabiguero. From Abstract Test Suites (ATS) to Executable Test Suites (ETS): A Contribution to Conformance and Interoperability Testing - Des suites de tests abstraits aux suites de tests exécutables: Une contribution au test de conformité et d'interoperabilité PhD Thesis Work. http://www.fing.edu.uy/ asabigue/publi/these20071114-BU.pdf, Université de Rennes I / Universidad de la República - 2007.

[40] C. Willcock. Introduction to TTCN-3. http://www.ttcn-3.org/TTCN3UC2005/program/TTCN-3T3UC05.pdf - Noviembre de 2002.

Glosario

Pid: Identificador del proceso

T3devkit: Framework para desarrollar casos de prueba sobre TTCN-3

ETSI: European Telecommunications Standards Institute

ITU: International Telegraph Union

TRI: TTCN-3 Runtime Interface

TCI-CD: TTCN-3 Control Interface - Coding and Decoding

IPv4: Internet Protocol version 4

IPv6: Internet Protocol version 6

IPsec: Internet Protocol security

CD: abreviación de CoDec

CoDec: Codifica y decodifica

IRISA: Institute de Recherche en Informatique et Systèmes Aléatoires

ISO : International Organization for Standardization

thread-safe: Programas multi-hilos

TE: Ejecutable de TTCN-3 de los casos de prueba

TN: Nodo de la topología que comienza el test, contiene el TE y lo ejecuta

NAT: Network Address Translation

NUT: Nodo bajo prueba

SUT: Sistema bajo prueba

ESP: Encapsulating Security Payload, protocolo que permite encriptación y autenticación sobre IPsec

AH: Authentication header, protocolo que permite autenticación sobre IPsec

ICV: Valor de integridad utilizado por los protocolos AH y ESP para autenticar

SPI: Security Parameter Index, indice de parámetro de seguridad utilizado en las asociaciones de seguridad de IPsec

ICMP: Internet Control Message Protocol

Ping: Packet Internet Grouper, comando util para probar conectividad entre nodos de una red

DHCP: Dynamic Host Configuration Protocol, protolo de asignación autómatica de direcciones de red

BOOTP: Bootstrap Protocol, protolo de asignación autómatica de direcciones de red

TTCN-3: Lenguaje de programación orientado a la especificación de casos de prueba

ISAKMP: Internet Security Association and Key Management Protocol

IKE: Internet Key Exchange