Técnicas para la Actualización de Implementaciones de Protocolos y Mecanismos Criptográficos para TinyOS *

Edel Angel Rodríguez Sánchez
and Yanet Esmeralda Silva Pérez

Facultad de Ciencias Técnicas, Universidad de Granma. Cuba
{edel,ysilvap}@udg.co.cu

Resumen Actualmente existe un incremento del número de aplicaciones de las WSN y con ello surgen teorías e implementaciones de esquemas de seguridad para este tipo de redes. Este trabajo analiza dichas implementaciones y consecuentemente ofrece un conjunto de técnicas que permiten su actualización o adaptación descendente en las versiones del sistema operativo para sensores TinyOS.

Abstract At present there is an increase in the number of WSN applications, which leads to the making of new theories and implementations of security schemes for this kind of networks. This paper analyzes these implementations and accordingly offers techniques that allow its update or descending adaptation on the versions of the operating system for sensors TinyOS.

Palabras claves: redes inalámbricas de sensores, seguridad, implementaciones, actualización de implementaciones

Keywords: wireless sensor network, security, implementations, implementations update

1. Introducción

Después de iniciadas a finales de los años 90, las *Redes Inalámbricas de Sensores* (WSN) han evolucionado hasta convertirse en una solución tecnológica viable, implementada por diversas organizaciones. De modo general, este tipo de redes encuentra aplicación en escenarios con problemas de monitorización, rastreo y control (monitorización ambiental e industrial, rastreo de posición[1], control de la carga eléctrica en áreas residenciales, etc). Dentro de las WSN se han realizado varias investigaciones con el objetivo de proporcionales servicios de seguridad. A pesar del reto que esto significa debido a las características naturales de este tipo de redes (limitaciones de los recursos del hardware de los nodos y comunicación inalámbrica entre estos) estas investigaciones han dado lugar a varios protocolos de seguridad, entre los que podemos encontrar: Sistemas de Distribución de Claves, Criptografía de Clave Pública y Sistemas de Detección de Intrusos, entre otros. No obstante, pocos de estos protocolos han sido llevado a la práctica y algunas de esas implementaciones no funcionan sobre las versiones más recientes de los sistemas operativos para sensores. Además, no se dispone de una documentación que facilite el uso o actualización de estos protocolos.

El propósito de este trabajo es, en primer lugar, documentar las implementaciones de *Protocolos y Mecanismos de Seguridad* (PMS) existentes para el sistema operativo para sensores TinyOS [2]. Por tanto, se estudiará a cabalidad, desde las generalidades hasta los detalles de cada aplicación disponible que implemente un PMS para TinyOS. Dicha documentación será usada con el fin de crear un conjunto de *técnicas recomendadas* para a la hora diseñar o actualizar estos protocolos. El segundo propósito de este trabajo es validar la utilidad de estas *técnicas recomendadas* en la actualización de uno de una de las aplicaciones estudiadas. Por consiguiente, se muestran los detalles de las modificaciones realizadas al protocolo y los resultados en la puesta en práctica de su actualización. Finalmente se hace una comparación de los resultados obtenidos.

* Este trabajo fue apoyado por la Asociación Universitaria Iberoamericana de Postgrado, la Junta de Andalucía, Universidad de Málaga (España) y la Universidad de Ciencias Informáticas (Cuba)

2. Antecedentes

En la presente sección abordaremos algunos trabajos previos de los que veremos sus objetivos, resultados y relación con este trabajo. Estos y otros como [3], [4], [5] y [6] evidencian la importancia del desarrollo e implementación de mecanismos criptográficos sobre estas redes. Posteriormente en las **Conclusiones** (Sección 5 en la página 16) se compararán nuestros resultados con los obtenidos en trabajos anteriores.

Wireless Sensor Network Security: A Survey [7]
Este trabajo fue realizado en el año 2006 por John P. Walters, Zhengqiang Liang, Weisong Shi y Vipin Chaudhary de la Universidad Estatal de Wayne, en él se hace un análisis profundo de los cuatro asuntos de mayor importancia en el tema de la seguridad en redes de sensores. Primeramente se analizaron los obstáculos con los que nos encontramos cuando pretendemos implementar seguridad para estas redes, es decir, limitación de recursos y comunicación de manera insegura, entre otros. Luego se estudiaron los requisitos de seguridad que sería necesario satisfacer, se mostraron los ataques más comunes sobre este tipo de redes y finalmente se brindan medidas de defensa contra estos ataques.

El estudio de este trabajo nos permitió identificar los ataques y medidas de seguridad de mayor importancia en las WSN. Luego, partiendo de esto, enfocar las implementaciones que dan soporte a estas medidas de seguridad. Estas implementaciones serán descritas en la sección 4.2 en la página siguiente y en la sección 4.4 en la página 14 se estudiará una de las que tiene como objetivo la distribución de claves.

Analysis of Security Threats, Requirements, Technologies and Standards in Wireless Sensor Networks [8] En el año 2009 Javier López, Rodrigo Román y Cristina Alcaraz de la Universidad de Málaga en España, publicaron este artículo que tiene como objetivo primario evidenciar la relación entre el contexto de la aplicación para la WSN, sus requerimientos y mecanismos de seguridad empleados. Además, se analizan algunos estándares de seguridad que tienen en cuenta esta relación.

Este trabajo presenta inicialmente de forma clara y amena conceptos fundamentales acerca de los sensores, las redes de sensores, amenazas y tecnologías de seguridad. A pesar de no ser este su tema esencial, nos sirvió para fomentar una base sobre el tema de la seguridad en estas redes. Posteriormente en la parte de este trabajo dedicada a la actualización de una aplicación que implementa seguridad sobre una red, se tiene en cuenta el aporte de este trabajo.

Implementaciones de algoritmos de seguridad En la actualidad existen varias implementaciones libres de algoritmos de seguridad, estas han estado motivadas por las amenazas demostradas en trabajos como los mencionados anteriormente. Varios de estos han sido sido llevados a la práctica con resultados positivos [9] [10] y otros han servido de base al desarrollo de investigaciones posteriores. El cuadro 1 enumera y referencia cronológicamente las implementaciones de mecanismos de seguridad analizadas en este trabajo. Estas implementaciones son tratadas en detalles en la sección 4.2 en la página siguiente.

Cuadro 1. Implementaciones de mecanismos de seguridad

Año	Nombre	Objetivo	Versión de TinyOS	Referencia
2005	TinyKeyMan	Distribución simétrica de claves	1.x	[9]
2005 - 2007	TinyECC	Distribución pública de claves	1.1.11	[10]
2010	AES	Cifrado por bloques	2.x	[11]
2010	Triviun	Cifrado continuo	2.x	[11] [12]
2010	MMH	Función Hash (Autenticidad)	2.x	[11] [13]
2010	Poly	Función Hash (Autenticidad)	2.x	[11] [14]

3. Descripción del problema

Se pretende compilar las implementaciones existentes de los protocolos de seguridad para TinyOS, de tal forma que se disponga de una documentación que permita el uso de dichas implementaciones.

Además, se estudia y proporciona un conjunto de "técnicas recomendadas" que permitan actualizar aquellas implementaciones que no funcionen en la versión más reciente del sistema operativo para sensores TinyOS. Estas "técnicas recomendadas" se utilizarán en la actualización de alguna implementación, o en la creación de una nueva implementación de algún mecanismo especialmente importante que no haya sido implementado.

4. Detalles de la propuesta

En la presente sección se abordarán los detalles de nuestra propuesta; comenzando por la tecnología utilizada para la simulación y modificación de las implementaciones. Luego se describen brevemente las implementaciones compiladas, enfatizando en sus objetivos y funcionamiento. Posteriormente se expone un conjunto de técnicas recomendadas para actualizar una implementación y luego, estas técnicas son usadas en la actualización de uno de los PMS descritos en la sección 4.2. Finalmente se realiza un análisis comparativo de los resultados del trabajo con otros similares.

4.1. Tecnología Utilizada

Teniendo en cuenta los objetivos propuestos abordados en la introducción y la formulación del problema; se hicieron necesarias herramientas para trabajar sobre el código de las implementaciones de PMS desarrolladas para TinyOS.

TinyOS es probablemente el sistema operativo para sensores más usado en la actualidad. Este sigue un modelo de programación orientado a eventos y es escrito completamente en lenguaje nesC [15]. Dicho lenguaje es una extensión del lenguaje C [16] que incorpora su sintaxis y permite además la *Programación Orientada a Componentes* [17].

Debido a que el manejo del lenguaje nesC fue fundamental para desarrollar este trabajo, la tecnología seleccionada tuvo dos objetivos principales; en primer lugar la edición y depuración del código nesC de las implementaciones de PMS y en segundo lugar la simulación de estas implementaciones. Por consiguiente, se usaron editores de código nesC que permitieran además la depuración y se utilizó un simulador de TinyOS para Microsoft Windows. Cada una de estas herramientas es descrita a continuación.

4.1.1. Edición y depuración de código nesC NESCDT [18] es un plugin para la plataforma Eclipse SDK que permite la edición de código nesC. Realiza completación automática de código y resalto de la sintaxis. Además identifica gráficamente tipos de archivos (interfaces, componentes y módulos) de un proyecto en nesC.

Otro editor usado fue Yeti 2 [19]. Este cuenta con las mismas ventajas de NESCDT, además de otras como: validación en tiempo real, navegación por medio de vínculos a través de declaraciones, creación de gráfico de componentes, depuraración y compilación [1]. La figura 1 en la página siguiente muestra un ejemplo de la edición de un código en nesC sobre Yeti 2.

4.1.2. Simulador de TinyOS para Windows Normalmente la simulación de TinyOS se hace sobre el sistema operativo Linux. Los desarrolladores de este simulador distribuyen paquetes RPM que instalan las herramientas necesarias para la simulación. Además distribuyen paquetes para un simulador de Linux sobre Windows conocido como CygWin [21].

Para la simulación de las implementaciones de PMS se usaron los *Paquetes de TinyOS para CygWin* disponibles en [22]. Fueron instaladas conjuntamente las versiones 1.1.15 y 2.0.2 de TinyOS. La figura 2 en la página siguiente muestra la compilación paralela de dos aplicaciones para versiones distintas de TinyOS.

4.2. Implementaciones Existentes de Protocolos y Mecanismos de Seguridad

Asuntos como la *Autenticación* y la *Administración de Claves* son críticos dentro de la seguridad en una red de sensores. Debido a las restricciones de hardware de sus nodos no es possible el uso de

[1] Para la depuración y compilación del código es necesario C/C++ Development Tooling (CDT) [20]

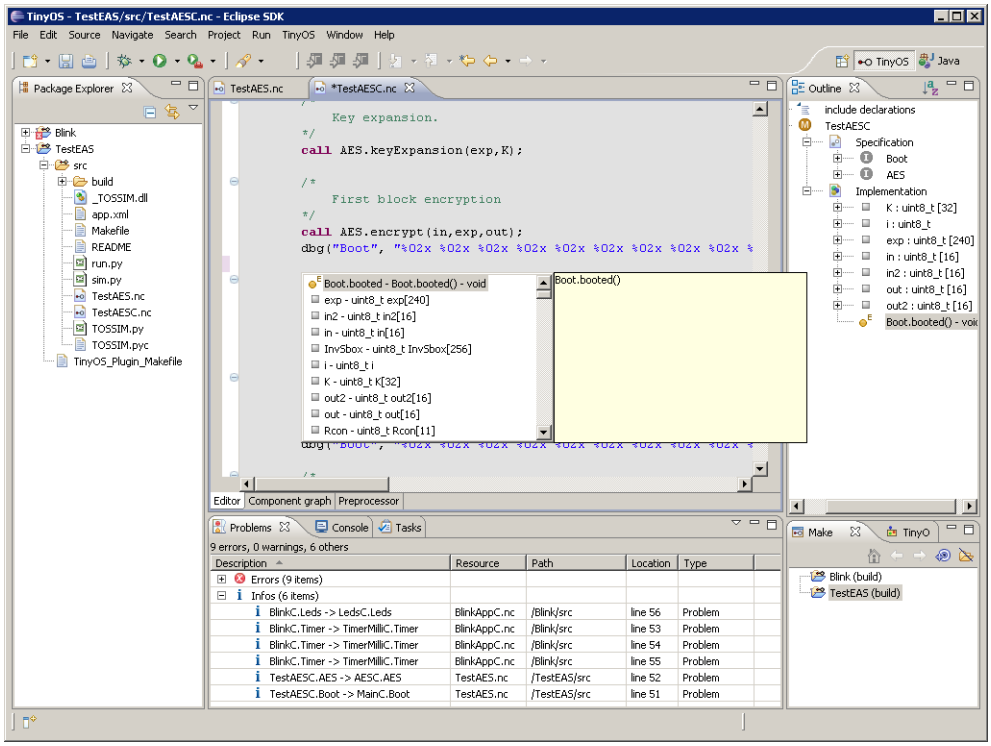

Figura 1. Edición de código nesC sobre Yeti 2

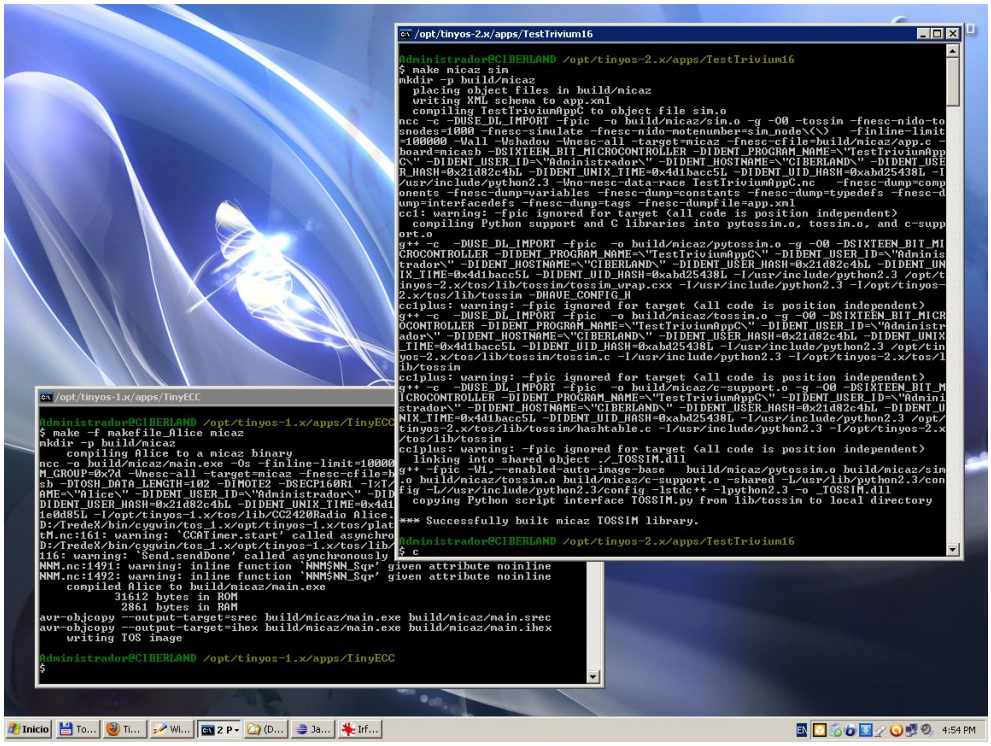

Figura 2. Versiones 1.1.15 y 2.0.2 de TinyOS instaladas conjuntamente

los esquemas tradicionales de seguridad como la *Criptografía de Clave Pública* (PKC) o un *Centro de Distribución de Claves* (KDC). Debido a esto han surgido varias propuestas que intentan proveer esquemas de seguridad para este tipo especial de redes, aunque no todas han sido llevadas a la práctica, se dispone actualmente de varias implementaciones basadas en estos esquemas.

A continuación analizaremos algunas de las implementaciones disponibles en la actualidad de PMS para redes de sensores. De cada una de estas veremos sus objetivos y versión de TinyOS para la que fue diseñada, describiremos su funcionamiento y en caso de ser necesario, se expondrá la estructura de la implementación con sus componentes e interfaces.

4.2.1. Algoritmos de cifrado y autenticidad para TinyOS

Sylvain Pelissier del *École Polytechnique Fédérale de Lausanne* ha desarrollado varias implementaciones de algoritmos criptográficos para TinyOS 2.x. Como resultado de esto ha obtenido las interfaces descritas a continuación y los componentes que las implementan, además para cada una de estas la autora proporciona aplicaciones de ejemplo [11].

AES Esta interfaz implementa una versión del *Algoritmo de Cifrado Estándar* [23] para la codificación por bloques, esta define los siguientes comandos: `encrypt` para cifrar un bloque de texto plano, `decrypt` descifra un bloque de texto (criptograma) y `keyExpansion` calcula la clave extendida por medio de un algoritmo de planificación; dicha clave extendida depende de la secreta conocida por el cliente y es utilizada para el cifrado y descifrado de los bloques.

Para el uso correcto de esta interfaz debe inicialmente contarse con una clave secreta. Esta pudiera obtenerse de los mecanismos descritos en las secciones 4.2.2 y 4.2.3 y partiendo de ella debe generarse la clave extendida mediante el comando `keyExpansion`, luego usar esta en la codificación y decodificación de los bloques con los comandos correspondientes. La figura 3 muestra el diagrama de componentes básico de una aplicación que usa la interfaz AES, y la figura 4 muestra el resultado de la codificación de 2 bloques de 16 bytes cada uno.

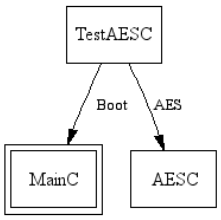

Figura 3. Diagrama de componentes de una aplicación que usa AES

```
First block encryption
Plain text: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Ciphertext: 0e dd 33 d3 c6 21 e5 46 45 5b d8 ba 14 18 be c8
Second block encryption
Plain text: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Ciphertext: e5 35 0d e3 b2 6a 17 ed 1d 23 5e a8 c2 04 a9 39
First block decryption
Ciphertext: 0e dd 33 d3 c6 21 e5 46 45 5b d8 ba 14 18 be c8
Plain text: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Second block decryption
Ciphertext: e5 35 0d e3 b2 6a 17 ed 1d 23 5e a8 c2 04 a9 39
Plain text: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Figura 4. Resultado de la codificación con AES

Trivium Esta interfaz proporciona comandos para el cifrado continuo basándose en el método Trivium [12]. Dicha interfaz cuenta con tres comandos: `gen_keystream` para generar bytes aleatorios que serán usados conjuntamente con el texto plano o el criptograma como entrada de una función XOR [24], el segundo comando se nombra `key_init` y es usado para la inicialización de la clave y del *Vector de Inicialización*(IV) [12], el tercer comando se nombra `process_bytes` y se usa para el cifrado y descifrado de los mensajes.

Junto a la implementación se proporcionan dos aplicaciones que ejemplifican el uso de esta interfaz: una de ellas para un procesador de 8 bits y otra para uno de 16 bits. En cada caso debe contarse con una clave secreta del cliente y un IV. Cada aplicación simula un trasmisor y un receptor, para esto define estados en los que realiza cada función. La clave secreta y el IV se usan para la inicialización de estos estados por medio del comando `key_init`. Este sería el primer paso, le sigue, la codificación y decodificación del mensaje con el comando que corresponde. En la figura 5 se muestra los resultados de la codificación/decodificación de un mensaje con Trivium.

```
Message: Hello world
Encryption (hexadecimal): ec 5d 00 01 19 04 ef 50 98 e1 da 73
Message decryted: Hello world
```

Figura 5. Resultado de la codificación con Trivium

MMH y Poly Estas interfaces implementan funciones hash universales para proporcionar autenticidad a los mensajes. MMH es una *Función Multimodular y Multilineal* [13], mientras que Poly se basa en el modelo descrito por Ted Krovetz y Phillip Rogaway en [14]. Estas interfaces deben usarse conjuntamente con un mecanismo de cifrado como los proporcionados por AES o Trivium, un ejemplo de esto se muestra en la figura 6.

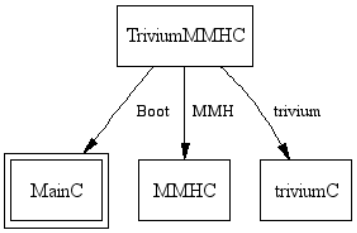

Figura 6. Diagrama de componentes de una aplicación que usa Trivium conjuntamente con MMH

Ambas interfaces solo disponen del comando `hash` por medio del cual se obtiene un número entero de 32 bits que representa el código de autenticidad del mensaje, este es calculado en función del mensaje y la clave del cliente. En la figura 7 se muestra el resultado de la aplicación de la función de autenticación de la interfaz MMH conjuntamente con Trivium.

```
Message: Hello world
Message hash: 92eb8243
Trivium key stream: 6d6c38a4
Authenticator: ff87bae7
```

```
Message: Hello world
Message hash: f82bfa1a
Trivium key stream: 6d6c38a4
Authenticator: 9547c2be
```

Figura 7. De izquierda a derecha: Resultado de la aplicación de MMH y Poly con Trivium

Puede notarse que no existen diferencias en cuanto a los resultados de cada función de autenticidad, pero si se realiza un análisis de complejidad a las correspondientes implementaciones se observa que Poly presenta mayor complejidad computacional que MMH lo cual debe ser considerado en las WSN.

4.2.2. Distribución de clave simétrica

En la *Criptografía de Clave Simétrica* el tema de mayor importancia es el *Establecimiento de Claves* [23] cuyo objetivo es hacer disponible una clave secreta compartida para dos o más partes interesadas en cifrar su comunicación. Seguidamente veremos una propuesta de los investigadores Donggang Liu y Peng Ning, ambos de la Universidad Estatal de Carolina del Norte en los Estados Unidos. Estos implementaron una aplicación para la versión 1.x de TinyOS nombrada TinyKeyMan.

TinyKeyMan es un mecanismo criptográfico para el establecimiento de claves simétricas en una WSN. Este mecanismo implementa el *Esquema Polinómico de Predistribución de Claves Basado en Depósitos*, un *Esquema Aleatorio de Asignación de Subconjuntos* y un *Esquema Matricial de Predistribución de Claves* [9]. Según los autores, estos esquemas garantizan una alta probabilidad de establecimiento de claves, tolerancia de captura de nodos y baja carga computacional y de comunicación.

El diagrama de la figura 8 corresponde a una aplicación sobre TinyKeyMan. En este se muestra la relación entre el componente principal de la aplicación (`Simulation`) y los componentes de TinyKeyMan. Desde este componente principal se utiliza el comando `SecretAssign` de la interfaz `SetupServer` para obtener una *Parte Polinómica* [25] en función del ID del nodo solicitante.

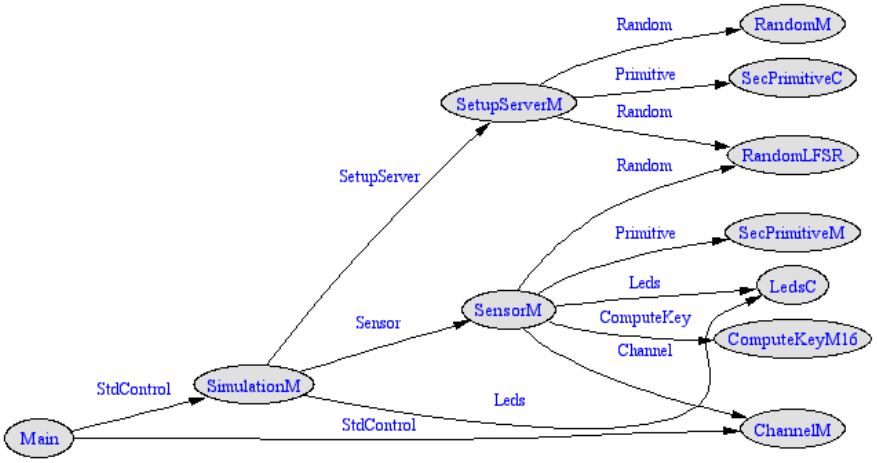

Figura 8. Diagramas de componentes de una aplicación sobre TinyKeyMan

Una vez que el nodo haya obtenido su parte polinómica puede establecer una clave compartida con otro nodo (conocido su ID) mediante el comando `establish_key` de la interfaz `Sensor`. Esta interfaz define además el evento `keyEstablished` para notificar que se ha acordado una clave con el nodo especificado y pasar dicha clave al nodo solicitante como parámetro del evento. Para establecer una clave compartida entre dos nodos, estos deben hacer llamadas semejantes a `establish_key`. En la figura 9 se muestra un ejemplo del establecimiento de una clave compartida entre el nodo 0 (`ID=0`) y el nodo 1 (`ID=1`), donde la constante `TOS_LOCAL_ADDRESS` contiene el ID del nodo.

Después que la aplicación recibe el evento `keyEstablished` notificando que ya se acordó la clave compartida con su vecino puede comenzar la transmisión de mensajes cifrados con esta clave. En la figura 10 en la página siguiente se muestra el resultado del uso de TinyKeyMan para establecer una clave entre dos nodos.

Una vez obtenida la clave compartida pudiera usarse uno de los algoritmos de cifrado tratados en la sección 4.2.1 en la página 5 para la codificación y decodificación de los mensajes.

Finalmente se tiene que TinyKeyMan implementa dos de los esquemas descritos anteriormente de forma separada: `Simulation` para el *Esquema Aleatorio de Asignación de Subconjuntos* y `SimulationG`

```
// El nodo 0 intenta establecer una calve compartida con el nodo 1
if (TOS_LOCAL_ADDRESS==0) {
    call Sensor.establish_key(1);
}
// El nodo 1 intenta establecer una calve compartida con el nodo 0
if (TOS_LOCAL_ADDRESS==1) {
    call Sensor.establish_key(0);
}
```

Figura 9. Código para establecer una clave entre dos sensores utilizando TinyKeyMan

```
0: 0 and 1 have set up key
0: key=B77B 8ADF 715A FBB2
1: 1 and 0 have set up key
1: key=B77B 8ADF 715A FBB2
```

Figura 10. Resultado de TinyKeyMan

para el *Esquema Matricial de Predistribución de Claves*. O sea, existen dos aplicaciones distintas en la distribución de TinyKeyMan. Gracias a esto podemos obtener versiones distintas de la aplicación y comparar los resultados de cada una [9]. Si se desea Compilar una o otra aplicación debe modificarse el `Makefile` del proyecto TinyKeyMan.

4.2.3. Distribución de clave pública

Anteriormente se ha descrito la dificultad del uso de algoritmos clásicos como RSA [26] o DSA [27] para la distribución de claves públicas dentro de una WSN. A continuación veremos una propuesta realizada por *An Liu* y *Peng Ning*, ambos del Departamento de Ciencias de la Computación de la Universidad Estatal de Carolina del Norte. Dicha propuesta se nombra TinyECC y pretende la distribución de claves públicas en una WSN.

TinyECC es una implementación del Esquema Criptográfico de Claves Públicas [23], esta se basa en una especificación para redes de sensores propuesta por *An Liu* y *Peng Ning* en [10] de la *Criptografía de Curva Elíptica* de Koblitz [28]. Se han desarrollado tres versiones de TinyECC para TinyOS 1.1.11, pero para el estudio en esta sección describiremos solo la versión 0.3, aunque en algún momento se hagan aclaraciones con respecto a las versiones anteriores.

Desde su primera versión, esta aplicación cuenta en su implementación con seis interfaces las cuales son comentadas a continuación.

NN Define operaciones algebraicas sobre números naturales grandes que normalmente no son soportadas por los tipos de datos primitivos de nesC.

ECC Define operaciones básicas sobre una curva elíptica, además de otras operaciones extendidas basadas en el método de la ventana deslizante [29].

ECDSA Define funciones para la generación y verificación de la firma de autenticidad adjunta al mensaje. Esta firma depende directamente del mensaje y de la clave privada.

SHA1 Define las funciones del Algoritmo de Hash Seguro SHA-1 [30].

CurveParam Define dos funciones: `get_param` para obtener los parámetros de la curva elíptica y `omega_mul` para la multiplicación optimizada con *omega*. En los módulos `secp128*.nc`, `secp160*.nc` y `secp192*.nc` se implementa esta interfaz. Dichos parámetros de la curva pueden ser modificados en la sección "*choose curve parameter*" del archivo `Makefile`.

Una aplicación que pretenda hacer uso de la distribución de claves públicas proporcionada por TinyECC debe de incluir los componentes listados en el cuadro 2 en la página siguiente y hacer uso de las interfaces de la segunda columna de este cuadro. La figura 11 en la página siguiente muestra el ejemplo de una aplicación que además de los componentes nativos de TinyOS, utiliza los componentes requeridos para el uso de TinyECC.

Puede notarse en la figura 11 en la página siguiente, que la aplicación que ella representa difiere de una aplicación clásica para recibo y envío de mensajes, solamente en la inclusión de `ECCC`, `ECDSAC` y `NNM`

Cuadro 2. Componentes e interfaces requeridas para el uso de TinyECC

Componente	Interfaz
ECCC	ECC
ECDSAC	ECDSA
NNM	NN

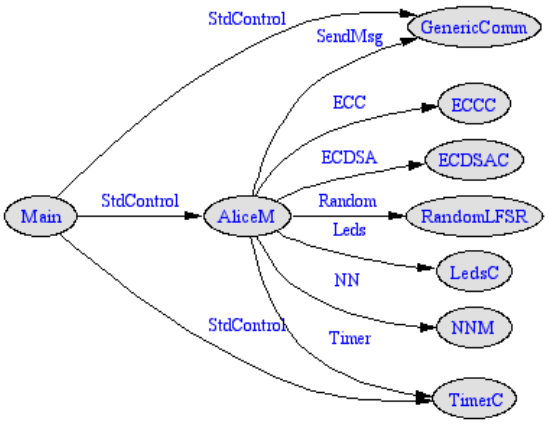

Figura 11. Diagrama de componente de una aplicación que usa TinyECC

proporcionados por TinyECC. Los dos primeros de estos son componentes, y la figura 12 muestra sus correspondientes diagramas, mientras que el tercero es un módulo donde se implementa la interfaz `NN`.

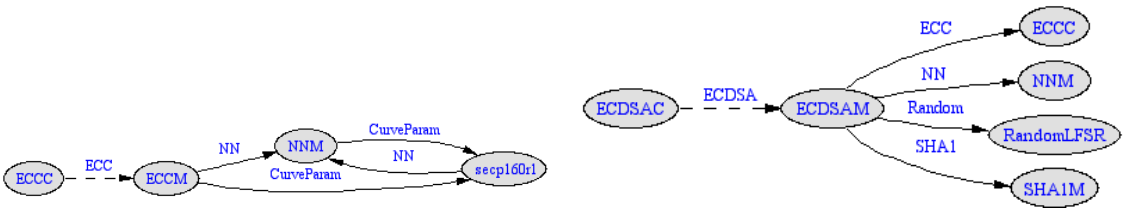

Figura 12. De Izquierda a derecha: Diagramas de componentes de ECCC y ECDSAC

Hasta ahora hemos descrito brevemente algunas de las implementaciones de PMS más populares para TinyOS. En el resto del documento veremos técnicas que nos servirán para actualizar estas aplicaciones y como son llevadas a la práctica.

4.3. Técnicas Recomendadas para la Actualización de Aplicaciones para TinyOS

Actualmente existen dos versiones (1.x y 2.x) del sistema operativo TinyOS. Desafortunadamente no hay compatibilidad descendente ni ascendente entre dichas versiones. Debido a esto en la presente sección se mostrarán en detalles algunas técnicas que facilitan el proceso de actualización (transición de la versión 1.x a la 2.x). Dichas técnicas surgieron de la actualización de TinyKeyMan (sección 4.2.2 en la página 7) y consisten mayormente en modificaciones al código fuente de la aplicación.

Aunque estás técnicas están enfocadas más bien a la actualización de aplicaciones, pueden ser usadas de forma inversa, transitando de la versión 2.x a la 1.x (sección 4.4 en la página 14).

4.3.1. Declarar una variable de entorno para Makerules Todo proyecto para TinyOS lleva junto a los archivos del código fuente un archivo de texto nombrado `Makefile`. Este contiene algunos de

los parámetros del proyecto y es interpretado por el programa `make` en el momento de la compilación. En TinyOS estos parámetros de configuración son mayormente comunes para todos los proyectos, por esta causa se ubican en un segundo archivo nombrado `Makerules` el cual es proporcionado por TinyOS. Gracias a esto, se ahorra espacio y se simplifica el `Makefile` de cada proyecto, pero cada archivo `Makefile` debe referenciar a este segundo archivo común. En el ejemplo de la figura 13 se muestran modos de referencia desde versiones distintas de TinyOS.

```
COMPONENT=MiAplicacion
include ../Makerules
```

```
COMPONENT=MiAplicacion
include $(MAKERULES)
```

Figura 13. Makefile de un proyecto para TinyOS. De izquierda a derecha: versión 1.x, versión 2.x

En la version 1.x de TinyOS el archivo `Makerules` se encuentra en el directorio de aplicaciones `apps`, por lo que cada aplicación ubicada en este directorio puede hacer referenciar al archivo `Makerules` como se muestra en la figura 13 (izquierda). Esto tiene la desventaja de que el programador se ve obligado a compilar su aplicación dentro del directorio `apps`, hacer referencia absoluta especificando la ruta completa del archivo `Makerules` o incluirlo dentro del directorio de su aplicación y de esta forma compilar desde cualquier ubicación.

La versión 2.x resuelve este problema definiendo una variable de entorno que contiene la ruta completa del archivo `Makerules` y así el `Makefile` puede referenciarlo independientemente de la ruta donde se encuentre la aplicación. La figura 13 (derecha) muestra un ejemplo de un archivo `Makefile` referenciando al archivo `Makerules` en TinyOS 2.x.

En el caso de que se desee desarrollar aplicaciones sobre la versión 1.x de TinyOS que posteriormente serán actualizadas a la versión 2.x, se recomienda declarar una variable de entorno que contenga la ruta completa del archivo `Makerules`. Si este archivo se encontrara en la raíz del directorio de aplicaciones de TinyOS, se declararía la variable de entorno de la forma siguiente:

```
$ export MAKERULES=$TOSROOT/apps/Makerules
```

Luego de esto el `Makefile` puede escribirse como el de la figura 13 (izquierda) y no necesitará cambios para la actualización. Si el caso es actualizar una aplicación, debe escribirse el `Makefile` como lo indica esta figura.

4.3.2. Incluir directivas en los archivos de cabecera Un programa en nesC puede incluir *archivos de definiciones de tipos* o *archivos de cabecera* como también se les conoce. La inclusión de estos se hace mediante la instrucción `includes` en la versión 1.x de TinyOS.

```
includes nombre_de_archivo;     // Sin la extensión del archivo
```

Nótese que en la versión 1.x se utiliza una instrucción y no una directiva como las de lenguaje C y además, no se especifica la extensión .h después del nombre del archivo. Esto cambia en la segunda versión de TinyOS; En esta se hace la inclusión de archivos de cabecera de forma muy similar [2] a como se hace en el lenguaje C. Consecuentemente, para incluir el texto contenido en un archivo de cabecera dentro del archivo fuente se escribe la directiva:

```
#include "nombre_de_archivo.h"
```

En este caso pueden ocasionarse *errores de redefinición de tipos* si se intenta incluir el mismo archivo de cabecera en varios archivos fuentes. Este es el motivo por el que se hace necesario en la versión 2.x que los archivos de cabecera contengan directivas que impidan la inclusión de sus definiciones más de

[2] En nesC no hay diferencia entre los archivos de cabeceras de la aplicación y los pertenecientes a la biblioteca del sistema a la hora de referenciarlos; por tanto todos los archivos se incluyen usando comillas dobles como en “nombre_de_archivo”.

una vez. Estas directivas no son necesarias en la version 1.x, ya que la instrucción `includes` se encarga de anexar el texto solo una vez, sin importar cuantas llamadas se hagan.

En la versión 2.x de TinyOS no existe la instrucción `include`, en su lugar se usan directivas que solo incluyen el texto del archivo de cabecera en el lugar de su llamada. Debido a esto es necesario el uso de otras directivas en los archivos de cabecera (al estilo C) de modo que si son incluidos por más de un archivo fuente no se produzcan errores de redefinición. Estas directivas en lenguaje nesC son iguales a las usadas en C. En la figura 14 se muestra un ejemplo de esto.

```
#ifndef ARCHIVO_H
#define ARCHIVO_H
    ...
    /* Cuerpo del archivo */
    ...
#endif
```

Figura 14. Directivas de un archivo de cabecera

Se recomienda que los archivos de cabecera sigan el modelo de la figura 14 independientemente de la versión de TinyOS para la que se creen. Después de esto, el proceso de actualización, solo requerirá del cambio (respetando su sintaxis) de la instrucción `includes` por la directiva `#include` en los archivos fuentes.

4.3.3. Remplazo del tipo `result_t` por `bool` La versión 1.x de TinyOS dispone de un tipo de dato identificado como `result_t`, este es un entero de 8 bits (`uint8_t`) que toma valores de una numeración de dos constantes (`SUCCESS` y `FAIL`). Este tipo es similar al booleano (`bool`), la diferencia sería que el booleano usa las constantes `TRUE` y `FALSE`. El tipo `result_t` es usado para los valores de retorno de los comandos, aunque también puede usarse cualquier tipo de dato.

En la versión 2.x este tipo de dato se ha anulado. En su lugar se suele utilizar el tipo booleano o `error_t` el cual es un nuevo tipo definido en la segunda versión [31]. Debido a esto se recomienda la utilización del tipo `bool` para los valores de retorno de los comandos de las aplicaciones que se desarrollen para TinyOS 1.x. Pero, en caso de que se trate de una actualización puede utilizarse la definición "`#define result_t bool`" al inicio de los archivos fuentes.

4.3.4. Actualizar la función de depuración NesC incluye la función `dbg` que permite mostrar texto en la consola durante la simulación. En la versión 1.x de TinyOS dicha función presenta la siguiente sintaxis:

```
dbg(<mode>, const char* format, ...);
```

Esta sintaxis cambió su primer parámetro en la versión 2.x, donde aparece como:

```
dbg(const char *chanel, const char *format, ...);
```

En la versión 1.x el primer parámetro (`mode`) es un entero de 8 bits y especifica el modo de depuración en el que se mostrará el texto del segundo parámetro. En la segunda versión de TinyOS dicho parámetro es una cadena con el nombre de la fuente donde se origina el mensaje; en este argumento suele ponerse el nombre de la aplicación.

Una técnica simple y que puede evitar modificaciones en varias líneas de depuración sería mediante el uso de la directiva `#define DBG_USR1 "Cadena"` al inicio del programa; donde `Cadena` sería el nombre de la aplicación y que a su vez identificaría el canal de salida para la simulación. Gracias a esto no será necesaria la modificación de cada llamada a la función `dbg` si se deseará actualizar la aplicación.

4.3.5. Cambio de la constante global que identifica al nodo En la versión 1.x de TinyOS se define una constante global `TOS_LOCAL_ADDRESS` que contiene un valor entero único para el nodo dentro de la red. Este valor suele conocerse como *identificador* del nodo o *dirección local*. Esta constante global ha cambiado en la segunda versión a `TOS_NODE_ID`. Otra vez una directiva "`#define TOS_LOCAL_ADDRESS TOS_NODE_ID`" puede emplearse en el proceso de actualización.

4.3.6. No usar excesivamente las primitivas de TOSSIM TinyOS 2.x proporciona primitivas (cuadro 3) para el modo de simulación TOSSIM que permiten conocer, entre otros datos, el tiempo trascurrido desde el inicio de la simulación y el identificador del nodo.

Cuadro 3. Algunas de las primitivas para TOSSIM

`sim_time_string`
`sim_print_time`
`sim_print_now`
`sim_node`
`sim_set_node`
`sim_random_seed`
`sim_random`

Estas primitivas son únicas de la versión 2.x y si se desarrolla una aplicación para TinyOS 2.x que pudiera luego necesitarse sobre la 1.x, no debe hacerse uso excesivo de estas primitivas.

4.3.7. Actualizar componente principal Para TinyOS 1.x; una aplicación es un componente que proporciona la interfaz `StdControl` para ser usada por el componente principal `Main`. Luego la aplicación debe implementar todos los comandos de dicha interfaz: `init`, `start` y `stop`.

En la práctica se ha demostrado que esta interfaz es insuficiente debido a que es completamente sincrónica y concentra en sí toda la actividad de arranque del sistema. TinyOS 2.x separa la funcionalidad de `StdControl` en tres interfaces distintas [32]:

Init Interfaz sincrónica para la inicialización de TinyOS. Ningún componente puede iniciarse hasta que esta parte es completada.
Scheduler Para la inicialización, ejecución y control de tareas [33].
Boot Proporciona el evento `booted` para notificar que el sistema ha iniciado correctamente.

Por otro lado, en la versión 2.x el nombre del componente principal cambió a `MainC` el cual ya no **usa** la interfaz `StdControl`. En su lagar `MainC` **proporciona** la interfaz `Boot`. En esta versión el único compromiso de la aplicación es implementar el evento de la interfaz `boot`.

Debido a estos cambios en la nueva versión de TinyOS se deben de realizar algunas actualizaciones al componente principal de nuestra aplicación. A continuación se lista una secuencia de pasos para realizar esta actualización.

Paso 1 Actualizar el componente principal.
1. Sustituir el componente `Main` por `MainC`
2. Sustituir el cableado de la interfaz del componente principal:
   "`Main.StdControl ->`*`NombreComponente`*`.StdControl;`" por
   "`MainC.Boot <-` *`NombreComponente`*`.Boot;`"
3. En caso de que exista cableado de otros componentes mediante la interfaz `StdControl` aplíquese para estos, el paso anterior.

Paso 2 Actualizar módulo de implementación del componente principal.
1. Eliminar la proporción de la interfaz `StdControl`:
   "`provides interface StdControl;`"
2. Adicionar el uso de la interfaz `Boot`:
   "`uses interface Boot;`"
3. Agregar la captura del evento `booted`
   "`event void Boot.booted() {...}`"
4. Mover el código contenido en la implementación de los comandos `init` y `start` de la interfaz `StdControl` a la captura del evento de paso anterior. El código debe de ser ubicado en este mismo orden. Luego de esto, la implementación de estos comandos puede eliminarse.

Con la realización de estos pasos conseguimos ajustar la estructura del componente principal de nuestra aplicación para la versión 2.x. Esto constituye un paso esencial en el proceso de actualización. No obstante, el mayor problema en la actualización esta formado por la incompatibilidad de las interfaces entre las versiones de TinyOS, de esto se comentará en la sección 4.3.9 en la página siguiente.

4.3.8. Actualización de la secuencia de arranque de la aplicación Lo más común es que nuestra aplicación requiera del envío y recibo de mensajes o del encendido y apagado de los Leds. Para esto se requiere que los componentes reales encargados de cada uno de estos servicios halla iniciado, lo cual se realiza en la secuencia de arranque. En la versión 2.x de TinyOS esta secuencia sigue los siguientes pasos [32]:

1. Inicialización del planificador
2. Inicialización de componentes
3. Señalización de que el proceso de arranque ha terminado (evento `Boot.booted`).
4. Ejecución del planificador

Para describir esta técnica nos basaremos en un ejemplo. Supongamos que en algún momento la aplicación necesita enviar y recibir mensajes sobre TinyOS 2.x. Para esto, debe ejecutar antes `start` de la interfaz `SplitControl`. Esta llamada debe hacerse una vez que el proceso de arranque halla terminado, es decir dentro del evento `Boot.booted`; pero, si inmediatamente después de esta llamada se intenta enviar un mensaje, el planificador tendrá en la cola este intento de envió antes de la ejecución del evento `startDone` de `SplitControl`, indicando que el componente está listo. En otras palabras, esto significa que se intenta enviar un mensaje antes de que el componente encargado esté listo.

Por lo antes descrito, surge la necesidad de usar el componente `TimerMilliC`, el cual se arranca con el comando `startPeriodic` de la interfaz `Timer` que este proporciona. En el evento `startDone` de `SplitControl` ejecutaríamos `startPeriodic`, en este punto el componente encargado de enviar y recibir los mensajes ya ha iniciado y se encuentra listo para ser usado. Luego, se hace el envío de mensajes en el evento `fired` de la interfaz `Timer` que es ejecutado cada cierto periodo de tiempo fijo. Este periodo es especificado en la llamada a `startPeriodic` y la primera ejecución del evento se hace después de trascurrido ese mismo tiempo desde el momento de la llamada.

Algunas aplicaciones para la versión 1.x y que usan el componente `GenericComm` para enviar y recibir mensajes no presentan esta dificultad, pero como se verá en la sección 4.3.9 este componente no está disponible para la versión 2.x. Por tanto la realización de las modificaciones correspondientes a esta técnica evitarán errores.

4.3.9. Buscar funciones similares entre las interfaces La versión 1.x de TinyOS incluye 88 interfaces primitivas, mientras que la versión 2.x solo incluye 62, de estas solo 11 son comunes entre versiones. Dichas interfases son listadas en el cuadro 4.

Cuadro 4. Interfaces comunes entre las versiones 1.x y 2.x de TinyOS

`I2CPacket.nc`
`Intercept.nc`
`Leds.nc`
`Random.nc`
`Receive.nc`
`Resource.nc`
`Scheduler.nc`
`Send.nc`
`SplitControl.nc`
`StdControl.nc`
`TaskBasic.nc`

A pesar de ser estas las únicas interfaces que han pasado a la versión 2.x, no han mantenido sus definiciones originales. Las que menos cambios incluyen, lo hacen en el tipo de retorno de los comandos y en los argumentos de estos. En otras palabras, en la práctica no se dispone de compatibilidad alguna entre las interfaces del sistema; tal vez esta sea una de las mayores dificultades en el proceso de actualización de una aplicación.

Para lograr avances en este proceso no basta con plantear técnicas, en su lugar debe estudiarse en detalles la función de cada comando primitivo del sistema usado o implementado por la aplicación. Una

vez conocidas estas funciones puede buscarse entre las interfaces de la versión siguiente, comandos o interfaces que realicen funciones iguales o similares. Es decir, se hace necesario en este punto, un estudio profundo sobre las interfaces, sus commandos y eventos en ambas versiones de TinyOS.

4.3.10. Adición parcial de componentes a la aplicación Hasta este punto se ha logrado actualizar la estructura general de la aplicación. Pero, es la incompatibilidad entre las interfaces lo que forma la gran parte del problema presente en la actualización. Cada interfaz usada o implementada por cada componente es fuente de conflicto en la que pocas veces se encontrará al menos el mismo nombre para dicha interfaz. Por esto se recomienda que los componentes que forman la aplicación se agreguen parcialmente, de forma que podamos analizar las interfaces de cada uno y buscar soluciones a los conflictos que puedan presentarse.

4.4. Aplicación de las Técnicas recomendadas

A continuación detallaremos la aplicación de las técnicas de la sección anterior en la actualización de TinyKeyMan; uno de los mecanismos de seguridad desarrollados para TinyOS 1.x y que fue descrito en la sección 4.2.2 en la página 7. Se verá la distribución de archivos que forma la implementación, se describirán las sustituciones de componentes, interfaces y eventos.

Inicialmente veamos brevemente la distribución de archivos que implementan a TinyKeyMan (cuadro 5). Esta cuenta con 25 archivos, de estos: 8 son interfaces, 12 son módulos, 3 son componentes y 2 son archivos de cabecera. De los 3 componentes, dos son aplicaciones (`Simulation.nc` y `SimulationG.nc`), ya que TinyKeyMan trae consigo dos implementaciones distintas (ver sección 4.2.2 en la página 7).

Cuadro 5. Distribución de archivos que implementan a TinyKeyMan

Interfaces	Módulos	Componentes	Cabeceras
BlockCipher.nc	CBCMAC.nc	SecPrimitiveC.nc	crypto.h
BlockCipherInfo.nc	ChannelM.nc	Simulation.nc	PolyOne.h
Channel.nc	ComputeKeyM16.nc	SimulationG.nc	
MAC.nc	RandomM.nc		
ComputeKey.nc	RC5M.nc		
Primitive.nc	SecPrimitiveM.nc		
Sensor.nc	SensorGM.nc		
SetupServer.nc	SensorM.nc		
	SetupServerGM.nc		
	SetupServerM.nc		
	SimulationGM.nc		
	SimulationM.nc		

Las técnicas 4.3.1 y 4.3.2 en la página 10 se aplicaron satisfactoriamente sobre los archivos correspondientes. La figura 6 en la página siguiente lista los archivos identificados en la implementación a los que se les aplicó la segunda técnica.

Lo descrito en las secciones 4.3.3, 4.3.4 y 4.3.5 puede combinarse con la inclusión, en los archivos del proyecto que lo requieran, de un archivo de cabecera como el que se muestra en la figura 15.

Pasemos ahora a la actualización del componente principal de nuestra aplicación como se indica en la sección 4.3.7 en la página 12. TinyKeyMan tiene dos de estos componentes; se describirá solamente la aplicación de la técnica sobre uno de estos (`Simulation.nc`) y su módulo de implementación (`SimulationM.nc`) ya que el proceso de actualización no cuenta con cambios relevantes para el caso de la segunda aplicación. En la figura 16 en la página siguiente se muestra (de izquierda a derecha) el componente antes y después de la modificación.

Hasta este punto se ha adaptado la estructura de los archivos que implementa la aplicación. Pasemos ahora a la adición de sus componentes como lo describe la sección 4.3.10.

La aplicación `Simulation.nc` hace referencia a 10 componentes, de los cuales 8 los contiene la aplicación y 2 son primitivos del sistema, comencemos con estos últimos (`LedsC` y `GenericComm`) en este

Cuadro 6. Archivos que utilizan la instrucción `includes`

BlockCipher.nc
BlockCipherInfo.nc
ChannelM.nc
ComputeKeyM16.nc
BlockCipher.nc
BlockCipherInfo.nc
MAC.nc
RC5M.nc
SensorGM.nc
SensorM.nc
SetupServerGM.nc
SetupServerM.nc
Simulation.nc
SimulationGM.nc
SimulationM.nc

```
#ifndef UPDATE_H
#define UPDATE_H

#define result_t                bool
#define DBG_USR1                "TinyKeyMan"
#define TOS_LOCAL_ADDRESS       TOS_NODE_ID
#define SUCCESS                 TRUE

#endif
```

Figura 15. Archivo de cabecera `udtate.h`

```
implementation {
components Main,...;
:
Main.StdControl ->ChannelM.StdControl;
Main.StdControl ->SimulationM;
:
}
```

```
implementation {
components MainC,...;
:
Main.Boot <- ChannelM.Boot;
Main.Boot <- SimulationM;
:
}
```

Figura 16. Actualización del componente `Simulation.nc`. De izquierda a derecha: Para TinyOS 1.x, para TinyOS 2.x

mismo orden. La adición de `LetsC` y su cableado no trae conflictos, ya que esta es una de las interfaces que no cambian de nombre en TinyOS. Aún así, es necesario tener en cuenta el cambio de nombre de los comandos, dado que ahora se identifica al led por un número y el color depende de la plataforma.

El componente encargado de las interfaces para enviar y recibir mensajes en la versión 1.x de TinyOS es `GenericComm`, este no existe en la versión 2.x, en su lugar existen otros dos componentes (`AMSenderC` y `AMReceiverC`) y otros complementarios como `ActiveMessageC`. Las interfaces `SendMsg` y `ReceiveMsg` se sustituyeron por `AMSend` y `Receive`, además se sustituyó `StdControl` por `SplitControl`. En la tabla 7 se muestran los componentes de la versión 1.x, los que los sustituyeron en la versión 2.x, las interfaces de estos y los eventos requeridos por estas interfaces.

Cuadro 7. Sustitución de componentes de la versión 1.x a la 2.x

Componentes en 1.x	Componentes en 2.x	Interfaces	Eventos
Main	MainC	Boot	booted
GenericComm	AMSenderC	AMSend Packet AMPacket	SendDone
	AMReceiverC	Receive	receive
	ActiveMessageC	SplitControl	startDone stopDone
	Timer	TimerMilliC	fired

Puede notarse en la tabla 7 que el componente `TimerMilliC`, en este caso no sustituye a ningún componente de la versión 1.x. Este fue añadido por la técnica descrita en la sección 4.3.8 en la página 13.

Como se había explicado anteriormente, las técnicas propuestas pueden ser generalizadas, además de la actualización de aplicaciones, también al paso entre versiones de TinyOS. Es decir; estas técnicas pueden emplearse en la adaptación descendente de una aplicación inicialmente desarrollada para la versión 2.x y que se quiera pasar a la versión 1.x de TinyOS. Esto pudo comprobarse en el desarrollo de dichas técnicas, para el cual se adaptaron los algoritmos de cifrado y autenticidad descritos en la sección 4.2.1 en la página 5. Para este caso fueron solo necesarias las técnicas 4.3.2, 4.3.4, 4.3.6 y 4.3.7. Los resultados de estas aplicaciones modificadas fueron iguales a los mostrados en la sección 4.2.1 en la página 5.

4.5. Análisis Comparativo de los Resultados

Como se describió en la sección 4.3 en la página 9, las incompatibilidades entre las versiones actuales de TinyOS traen como consecuencia que las implementaciones de PMS estén disponibles solamente para una de estas versiones (ver cuadro 1 en la página 2). Partiendo de la importancia que tiene la disponibilidad de estas aplicaciones para cualquier versión de TinyOS; Este trabajo nos aporta técnicas genéricas que consiguen la actualización de aplicaciones antiguas hacia la versión 2.x de TinyOS o la adaptación descendente de una aplicación hacia las primeras versiones de este sistema operativo. Se aporta además, las experiencias acumuladas de la aplicación de estas técnicas.

Después de la investigación sobre la actualidad del tema y el estudio de varios materiales, entre los que aparecen los analizados en los **Antecedentes** (sección 2) se expone claramente la importancia de nuestro aporte, al comprobarse que no se tienen precedentes similares.

5. Conclusiones

El presente trabajo ha tenido entre sus objetivos el análisis de las implementaciones actuales de seguridad para el sistema operativo TinyOS. En dicho análisis se describió el propósito, funcionamiento y modo de uso de cada implementación. Estas aplicaciones fueron colectadas, probadas y varias de ellas pasadas entre versiones con la aplicación de las técnicas aportadas.

Dichas técnicas permiten la actualización de implementaciones para la última versión de TinyOS, lo que se demostró mediante su aplicación en la actualización de TinyKeyMan; una implementación de un

esquema simétrico de distribución de claves que inicialmente fue desarrollado para TinyOS 1.x y se consiguió el paso de esta a la versión 2.x. En principio, estas técnicas pueden utilizarse para la actualización, pero además, para la adaptación descendente hacia la versión 1.x de TinyOS de las implementaciones de PMS.

Como trabajo futuro se propone un análisis profundo de los comandos y eventos de las interfaces primitivas de cada version de TinyOS, con el objetivo de obtener a una lista de diferencias y equivalencias entre estos, en función de su tarea dentro del sistema. Partiendo de esta, pudiera proponerse un modelo que tenga como objetivo el casamiento entre estas interfaces y los componentes que las implementan. La posterior implementación de dicho modelo daría lugar a un sistema que permita el paso automático de aplicaciones entre las versiones de TinyOS.

En dicha lista de diferencias y equivalencias deben aparecer los detalles de la secuencia de arranque de cada versión de TinyOS, de forma que queden al descubierto los cambios necesarios a una aplicación para que siga la filosofía de arranque adecuada para cada versión.

Referencias

1. Tseng, Y.C., Kuo, S.P., Lee, H.W., Huang, C.F.: Location tracking in a wireless sensor network by mobile agents and its data fusion strategies. In: Proceedings of the 2nd international conference on Information processing in sensor networks. IPSN'03, Berlin, Heidelberg, Springer-Verlag (2003) 625–641
2. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGPLAN Not. **35** (November 2000) 93–104
3. Çamtepe, S.A., Yener, B.: Combinatorial design of key distribution mechanisms for wireless sensor networks. IEEE/ACM Trans. Netw. **15** (April 2007) 346–358
4. Alcaraz, C., Roman, R., Lopez, J.: Análisis de primitivas criptográficas para redes de sensores. In: JITEL 2007: VI Jornadas de Ingeniería Telemática. (September 2007) 401–408
5. Castro, R.R.: Application-Driven Security in Wireless Sensor Networks. PhD thesis, Universidad de Málaga, BLV. Louis Pasteur, 35. 29071 Málaga (2008)
6. Cristina Alcaraz, Rodrigo Roman, J.L., Chen, H.H.: Selecting key management achemas for wsn applications. IEEE Wireless Communications (2009) Seleccionado para publicación.
7. Walters, J.P., Liang, Z., Shi, W., Chaudhary, V.: 17 in Security in Distributed, Grid, and Pervasive Computing. In: Wireless sensor network security: A survey. Auerbach Publications, CRC Press (2006)
8. Lopez, J., Roman, R., Alcaraz, C.: Analysis of security threats, requirements, technologies and standards in wireless sensor networks. In Aldini, A., Barthe, G., Gorrieri, R., eds.: Foundations of Security Analysis and Design V. Volume 5705 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 289–338 10.1007/978-3-642-03829-7_10.
9. Liu, D., Ning, P.: Establishing pairwise keys in distributed sensor networks. ACM Trans. Inf. Syst. Secur. **8** (February 2005) 41–77
10. Liu, A., Ning, P.: Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In: Proceedings of the 7th international conference on Information processing in sensor networks. IPSN '08, Washington, DC, USA, IEEE Computer Society (2008) 245–256
11. Pelissier, S.: Cryptography algorithms for tinyos. Web site www.tinyos.cvs.sourceforge.net/viewvc/tinyos/tinyos-2.x-contrib/crypto.
12. Canniere, C.D., Preneel, B.: Trivium specifications. eSTREAM, ECRYPT Stream Cipher Project **2006**
13. Carter, J.L., Wegman, M.N.: Universal classes of hash functions (extended abstract). In: Proceedings of the ninth annual ACM symposium on Theory of computing. STOC '77, New York, NY, USA, ACM (1977) 106–112
14. Krovetz, T., Rogaway, P.: Fast universal hashing with small keys and no preprocessing: The polyr construction. In Won, D., ed.: Information Security and Cryptology - ICISC 2000. Volume 2015 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2001) 73–89 10.1007/3-540-45247-8_7.
15. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesc language: A holistic approach to networked embedded systems. In: Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation. PLDI '03, New York, NY, USA, ACM (2003) 1–11
16. Kernighan, B.W., Ritchie, D.M.: The C programming language. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1978)
17. Szyperski, C.A., Bosch, J., Weck, W.: Component-oriented programming. In: Proceedings of the Workshop on Object-Oriented Technology, London, UK, Springer-Verlag (1999) 184–192
18. rup.inf(at)cbs.dk: Nescdt - an editor for nesc in eclipse (Noviembre 2009) docs.tinyos.net/index.php/NESCDT-_An_editor_for_nesC_in_Eclipse.

19. Burri, N., Flury, R., Nellen, S., Sigg, B., Sommer, P., Wattenhofer, R.: Yeti: an eclipse plug-in for tinyos 2.1. In: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. SenSys '09, New York, NY, USA, ACM (2009) 295–296
20. Leszek, P.: Integrated development environment: C/c++ development with the eclipse platform. IBM developerWorks (June 2006)
21. Lazenby, D.: Cygwin: For windows nt. Linux J. **2000** (July 2000)
22. CSU, C.S.U.: Tinyos home. Web site (2010) www.tinyos.net/download.html.
23. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. 1st edn. CRC Press, Inc., Boca Raton, FL, USA (1996)
24. de Renna E Souza, C.: R 68-44 mathematical logic. IEEE Trans. Comput. **17** (October 1968) 1003–
25. Blundo, C., De Santis, A., Vaccaro, U., Herzberg, A., Kutten, S., Yong, M.: Perfectly secure key distribution for dynamic conferences. Inf. Comput. **146** (October 1998) 1–23
26. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21** (February 1978) 120–126
27. Publication, F.I.P.S.: Digital signaure standard (dss). FIPS PUB 186 (July 2009)
28. Koblitz, N.: Constructing elliptic curve cryptosystems in characteristic 2. In: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '90, London, UK, Springer-Verlag (1991) 156–167
29. Hao, Y., Ma, S., Chen, G., Zhang, X., Chen, H., Zeng, W.: Optimization algorithm for scalar multiplication in the elliptic curve cryptography over prime field. In: Proceedings of the 4th international conference on Intelligent Computing: Advanced Intelligent Computing Theories and Applications - with Aspects of Theoretical and Methodological Issues. ICIC '08, Berlin, Heidelberg, Springer-Verlag (2008) 904–911
30. Eastlake 3rd, D., Jones, P.: US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational) (September 2001) Updated by RFC 4634.
31. Levis, P.: Tinyos 2.0 overview, overview.pdf (2006)
32. Levis, P.: Tinyos 2.x boot sequence. In: TinyOS Extension Proposal (TEP) 107, www.tinyos.net/tinyos-2.x/doc/pdf/tep117.pdf (2006)
33. Levis, P., Sharp, C.: Schedulers and tasks. In: TinyOS Extension Proposal (TEP) 106, www.tinyos.net/tinyos-2.x/doc/pdf/tep116.pdf (2006)

[3]

[3] *No hacen falta alas para hacer un sueño, basta con las manos, basta con el pecho, basta con las piernas y con el empeño.* ***Silvio Rodríguez***

...basta con personas dispuestas a ayudar.
Edel A. Rodríguez