Un caso práctico de aporte de seguridad en IoT

(Recibido: 07/05/2016; Aceptado: 07/07/2016)

Martinez-Caro, J.-M.; Cano, M.-D.
Departamento de Tecnologías de la Información y las Comunicaciones
Universidad Politécnica de Cartagena
Teléfono: 968325953
Email: martinezcaro92@gmail.com

Resumen. *Se espera que el nuevo ecosistema digital creado a partir de Internet de las Cosas sea el mayor en el ámbito de las Tecnologías de la Información y las Comunicaciones. En este contexto, la seguridad se presenta como uno de los mayores retos a abordar desde distintas vertientes: provisión de privacidad, relaciones de confianza, diseño nativo de aplicaciones seguras, etc. En este trabajo, presentamos un caso práctico de implementación software del algoritmo de cifrado AES en dispositivos inalámbricos Micaz usados en redes inalámbricas de sensores basadas en el estándar IEEE 802.15.4.*
Palabras clave. *IoT, Seguridad, IEEE 802.15.4, AES y TinyOS.*
Abstract. *It is expected that the new digital ecosystem created from the Internet of Things will be the greatest in the field of Information and Communication Technology. In this context, security is presented as one of the main challenges to be addressed from different areas: provision of privacy, trust relationships, native design of secure applications, etc. In this paper, we present an experimental example of software implementation of the AES encryption algorithm for Micaz motes used in IEEE 802.15.4-based wireless sensor networks.*
Keywords. *IoT, Security, IEEE 802.15.4, AES, and TinyOS.*

1. Introducción

La Internet de las Cosas (*Internet of Things*, IoT) es un nuevo paradigma en el que todas las "cosas", desde electrodomésticos hasta *wearables* pasando por vehículos o sensores estarán conectados a Internet. Se espera que el nuevo ecosistema digital creado a partir de IoT sea el mayor en el ámbito de las Tecnologías de la Información y las Comunicaciones (Aguzzi, 2015). Ejemplos clásicos de uso de IoT son: frigoríficos capaces de conocer los alimentos restantes en su interior y llevar a cabo la compra en función de los planes del usuario y el stock, los *weareables* que se conectan a Internet para registrar todos los datos que captan sobre su usuario, mejoras en el sistema de conducción de vehículos, la automatización del hogar o la automatización de las ciudades, haciéndolas más inteligentes (*SmartGrids*) y eficaces.

En este mundo IoT híper-conectado, la seguridad surge como uno de los principales retos a resolver (Keoh, 2014). Una forma lógica de analizar la seguridad en IoT es revisar las soluciones existentes para las ya establecidas redes de sensores inalámbricas (*Wireless Sensor Networks*, WSN). Las WSN están pensadas para que los diferentes nodos, denominados *motes*, se comuniquen entre ellos intercambiando información obtenida de sensores, tomen decisiones y/o lleven a cabo operaciones simples en base a la información obtenida, entre otras. Son varios los problemas y amenazas posibles en las WSN, por ejemplo (Dener, 2014): *Denial-of Service (DoS)*, *Attacks on Information in transit*, *Sybil Attack*, etc. Aunque algunos de estos problemas han sido solucionados, todavía quedan importantes temas a resolver como por ejemplo (Granjal, 2015) la autenticación de origen, el establecimiento de claves criptográficas, etc. En este trabajo, presentamos un estudio inicial de las prestaciones del algoritmo criptográfico AES (*Advanced Encryption Standard*) (FIPS, 2001), en términos de consumo energético, que pueda servir de ejemplo práctico de cómo introducir un alto nivel de confidencialidad y autenticación en WSN como base para un estudio posterior ampliado con IoT. Las pruebas se han llevado a cabo en un banco de pruebas experimental montado y configurado a tal efecto.

El resto del artículo se organiza de la siguiente forma. La sección 2 describe los elementos HW y SW del banco de pruebas e introduce el algoritmo criptográfico AES. La sección 3 incluye las pruebas realizadas y los resultados. El documento finaliza con las conclusiones en la sección 4.

2. Descripción del banco de pruebas

En esta sección se describen los componentes hardware (Micaz) y software (TinyOS y nesC) del banco de pruebas que se ha configurado para llevar a cabo el estudio. Asimismo, se resumen las características más importantes del algoritmo AES.

2.1 Hardware y software

TinyOS es un sistema operativo *(Operating System*, OS*)* diseñado para sistemas embebidos inalámbricos de baja potencia (TinyOS, 2016). Básicamente, se trata de un planificador de trabajo y una colección de *drivers* para microcontroladores y otros circuitos integrados comúnmente usados en plataformas embebidas inalámbricas. Algunos de los aspectos importantes para el diseño de TinyOS son los siguientes (Levis, 2005):

1. Recursos limitados. Los *motes* tienen recursos físicos limitados debido a que están planteados para localizarse en lugares

estratégicos, minimizando el coste y donde las fuentes de energía son limitadas.
2. Concurrencia reactiva. Un *mote* es el responsable de muestrear datos mediante el uso de sensores, procesar dichos datos y transmitirlos; también pueden participar en tareas de procesamiento distribuido.
3. Flexibilidad. Esta característica se atribuye a las diferentes aplicaciones y hardware que se pueden encontrar, donde el objetivo fundamental es reducir el espacio y el consumo de los *motes*.
4. Baja consumo. Para ahorrar en el consumo de energía sólo se activan las partes a utilizar del circuito que serán definidas en el código.

Para instalar TinyOS se puede optar por diferentes opciones, entre ellas:

1. Descarga de máquina virtual donde quedará todo listo para ser utilizado. Esta máquina virtual se importará en *VMWare* o *VirtualBox* para su utilización. Entre ellas, *XubuntOS* y *UbuntOS*.
2. Utilizando la herramienta clone de *Github* y definiendo ciertas variables dentro del fichero *~bashrc*.
3. Importando librerías y usando el *comando apt-get install*. Posteriormente se deberán definir ciertas variables dentro del fichero *~bashrc*.
4. Utilización de paquetes *RPM* para las versiones de *Unix* y *Windows*.

Por su parte, nesC es una extensión de *C* especialmente diseñada para correr sobre TinyOS y ser utilizada en WSN. Los conceptos básicos de este lenguaje de programación son (TinyOS, 2016; Levis, 2005; Gay, 2003):

1. Separación de la construcción y la composición: los programas son construidos a partir de componentes y son ensamblados para formar programas completos.
2. Las interfaces son bidireccionales y proporcionadas o usadas por componentes.
3. Los componentes se unen estáticamente entre sí a través de sus interfaces.
4. El modelo de concurrencia de nesC está basado en tareas de ejecución. Los controladores de interrupciones pueden interrumpir las tareas.

Finalmente, el nodo inalámbrico (*mote*) escogido para el banco de pruebas es el *Micaz* (Micaz, 2016) (véase Fig. 1). Este dispositivo utiliza la banda de radio 2.4GHz y el estándar IEEE 802.15.4 (IEEE802.15.4, 2006). Es capaz de medir parámetros como temperatura, presión, aceleración o sonido, entre otros, mediante el uso de la placa de sensores mts300. Además, utiliza la programadora mib520 (USB) o mib510 (puerto serie) (Micaz, 2016).

2.2. El algoritmo AES

Cuando se emplea el término seguridad se hace referencia a la consecución de tres objetivos: autenticación, integridad y confidencialidad. Mediante la autenticación se garantiza que un usuario/máquina es quien dice ser. Habrá integridad si durante el intercambio de información, ésta no ha sido modificada o eliminada por parte de usuarios malintencionados. Por último, la confidencialidad se produce cuando sólo los componentes autorizados del sistema pueden acceder al elemento protegido, por ejemplo cuando sólo transmisor y receptor pueden ver el contenido de un mensaje. El algoritmo AES es el estándar actual para el cifrado de comunicaciones (FIPS, 2001). Se trata de un algoritmo de cifrado en bloque simétrico que acepta bloques de entrada de 128, 192 o 256 bits y tamaños de clave de 128, 192 o 256 bits. Realiza todas las operaciones a nivel de byte. Es un algoritmo iterativo en el que el número de rondas depende del tamaño del bloque de texto plano y del tamaño de la clave. Hoy en día no se conoce ningún ataque por criptoanálisis a AES, siendo el ataque por fuerza fruta inviable.

El estándar 802.15.4 (IEEE802.15.4, 2006) define 8 *suits* de seguridad en la capa de control de acceso al medio que se pueden agrupar en: sólo autenticación

Fig. 1. *Mote* Micaz empleado en el estudio.

(AES-CBC-MAC), sólo cifrado (AES-CTR), cifrado y autenticación (AES-CCM), y no seguridad. Cada una de estas *suits* ofrece variantes en función del tamaño del código de autenticación del mensaje (*Message Authentication Code*, MAC) empleado para la autenticación pero en todas ellas se hace uso de AES para proporcionar confidencialidad (si procede). No todas las *suits* son obligatorias a la hora de crear implementaciones de este estándar. Es importante señalar que se han detectado fuertes vulnerabilidades a las *suits* propuestas, por ejemplo (Cao, 2016).

3. Caso práctico

En primer lugar, se trata de conocer el entorno en el que el investigador se va a manejar. Uso de una máquina que utiliza un sistema *Linux (XubuntOS)* sobre una máquina virtual *(VirtualBox)*. Una vez sea importado todo, se conocerán las herramientas o programas disponibles en abierto para su funcionamiento dependiendo de las necesidades del usuario. No sólo están disponibles herramientas o programas sino también código *nesC* y aplicaciones predefinidas, que ayudarán al usuario a comprender el funcionamiento de esta plataforma.
Se pretende desarrollar una solución de seguridad para la transferencia de datos por el medio. En este caso, se trata de la implementación software de un cifrado y descifrado AES para WSN implementado y testado en los dispositivos Micaz.

3.1. Primeros pasos

Una vez se conoce el entorno en el que el usuario va a trabajar, llega el momento donde hay que empezar a modificar y crear nuevo código para comprender y asimilar el lenguaje *nesC*. Con este fin, se han creado diferentes aplicaciones para modificar y/o ampliar las funcionalidades a los programas predefinidos (este trabajo no se ha incluido aquí dada la limitación de espacio).
Una vez que los cambios han sido definidos (o el nuevo código creado), es necesario compilar el código e introducirlo en los *motes* para comprobar que el código creado tiene la funcionalidad deseada. Es posible que sean necesarios varios *motes* con programas diferentes para testear los cambios realizados.

3.2. Desarrollo

Se trata de definir la estructura básica de las aplicaciones definidas en *TinyOS*. Consta de diferentes secciones como: definición de *#includes*, *module*{}, e *implementation*{}.
En primer lugar, se especifican una serie de *#includes* donde se importan las declaraciones definidas en otros ficheros, incluyendo los *#includes* anidados. Continúa con una sección *module*{} donde principalmente se declaran las interfaces que serán utilizadas en esta aplicación. Por último, se define la sección *implementation*{} donde las variables son definidas, inicializadas y se desarrollan operaciones con las mismas. En esta sección se define un *event void Boot.booted()*{} que tiene la misma funcionalidad que el clásico método *main()* definido en múltiples aplicaciones.
En el caso práctico que se desea abordar, son varios los tipos de datos a utilizar, desde datos tipo enteros hasta *arrays* unidimensionales y bidimensionales de datos enteros (de tipo *signed* y *unsigned*). Para optimizar el código se pretende llevar a cabo funciones o métodos de las operaciones más comunes como imprimir matriz o inicializar variables. Las etapas a desarrollar en nuestro caso para implementar AES son: *(1)SubBytes*, *(2)ShiftRows*, *(3)MixColums* y *(4)AddRoundKey*. De forma previa a cada etapa, se lleva a cabo una inicialización de variables, o *reset*, para evitar así que estas tomen valores erróneos de usos anteriores. Otro objetivo es el de minimizar el número de variables en la aplicación para que el código sea lo más eficiente posible, usando así una menor cantidad de memoria, debido a las limitadas capacidades disponibles en los dispositivos.
La etapa *(1)SubBytes* sustituye el dato de 8 bits introducido en función de la *S-box*. Los cuatro bits más significativos indican la fila y los cuatro bits menos significativos indican la columna del dato en la *S-Box*. Por ejemplo, véase la Fig.2, si el dato introducido es 0x31 buscamos en la *S-Box* fila 3, columna 1 y el nuevo dato será 0xC7.

	0	1	2
0	63	7C	77
1	CA	82	C9
2	B7	FD	93
3	04	C7	23

Fig.2. Sustitución de valores del proceso *SubBytes*.

En *(2)ShiftRows* se modificará la matriz rotando las columnas hacia la izquierda de la matriz a partir de la segunda fila (la primera fila se mantiene). El número de columnas rotadas se incrementará una unidad por cada fila (véase ejemplo en Fig. 3).
En el proceso *(3)MixColums*, se procede a multiplicar la matriz resultante con una matriz predefinida [4x4]. Como las multiplicaciones son computacionalmente costosas, se lleva a cabo un proceso de sustitución, similar al anterior, usando la *tabla L* y la *tabla E* de AES. Las operaciones llevadas a cabo se muestran en la Fig.4. Por último, en el proceso *(4)AddRoundKey* cada elemento de la matriz [i, j] es combinado con un elemento de la matriz de subclave [i, j] usando la operación XOR como indica la Fig. 5.

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & A & B \\ C & D & E & F \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 2 & 3 \\ 5 & 6 & 7 & 4 \\ A & B & 8 & 9 \\ F & C & D & E \end{bmatrix}$$

Fig. 3. Rotación de matrices del proceso *ShiftRows*.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} * \begin{bmatrix} D4 \\ BF \\ 5D \\ 30 \end{bmatrix} = \begin{bmatrix} 04 \\ 66 \\ 81 \\ E5 \end{bmatrix}$$

$$R_{0,0} = (D4 * 02) \oplus (BF * 03) \oplus (5D * 01) \oplus (30 * 01)$$
$$R_{0,0} = E(L(D4) + L(02)) \oplus E(L(BF) + E(03)) \oplus E(L(5D) + L(01)) \oplus E(L(30) + L(01))$$
$$R_{0,0} = E(19 + 41) \oplus E(01 + 9D) \oplus (0 + 88) \oplus (0 + 65)$$
$$R_{0,0} = E(5A) \oplus E(9E) \oplus E(88) \oplus E(65) = B3 \oplus DA \oplus 5D \oplus 30$$
$$R_{0,0} = 04$$

Fig. 4. Operaciones del proceso *MixColums*.

$$b_{2,2} = a_{2,2} \oplus k_{2,2}$$

Fig. 5. Operación *XOR* del proceso *AddRoundKey*.

Para una descripción completa del algoritmo por favor acudir a (FIPS, 2001).

3.3. Resultados

Los resultados se han obtenido una vez creado y compilado el código. El objetivo es conocer el tiempo necesario para llevar a cabo la operación, analizar el correcto funcionamiento del proceso de cifrado/descifrado y el consumo. Es importante señalar que para este estudio inicial sólo se ha tenido en cuenta una iteración de AES, por lo que los resultados serán del orden de diez veces menores que el final.

El tiempo de compilación e instalación de la aplicación en el sensor es de 9,6 s. La cantidad de memoria *ROM* necesaria para este caso es de 10950 bytes y 2292 bytes de memoria *RAM*. Los resultados obtenidos tras el cifrado se muestran en la Fig. 6.

Mediante la herramienta *Avrora* (Avrora, 2016) se pretende conocer el consumo energético necesario por el *mote* para llevar a cabo las operaciones programadas. Debido a que la ejecución tarda cinco segundos, la potencia eléctrica consumida por el dispositivo es 12,9 mW, tal y como se indica en (1) y en la Fig. 7.

```
Print Value to Cipher:
0x00000011 , 0x00000012 , 0x00000013 , 0x00000014
0x00000021 , 0x00000022 , 0x00000023 , 0x00000024
0x00000031 , 0x00000032 , 0x00000033 , 0x00000034
0x00000041 , 0x00000042 , 0x00000043 , 0x00000044
Print Key:
0x00000001 , 0x00000002 , 0x00000003 , 0x00000004
0x00000005 , 0x00000006 , 0x00000007 , 0x00000008
0x00000009 , 0x0000000a , 0x0000000b , 0x0000000c
0x0000000d , 0x0000000e , 0x0000000f , 0x00000010
Print RESULT CIFRADO:
0x0000001a , 0x000000a1 , 0x000000e0 , 0x0000004d
0x00000013 , 0x00000097 , 0x0000008b , 0x000000d1
0x00000054 , 0x00000075 , 0x00000068 , 0x0000002b
0x000000cb , 0x000000e1 , 0x000000e2 , 0x00000084
```

Fig. 6. Resultados de cifrado

```
CPU: 0.06448136112255859 Joule
   Active: 0.02567715432023112 Joule, 8339722 cycles
   Idle: 0.03880420680232747 Joule, 28524278 cycles
   ADC Noise Reduction: 0.0 Joule, 0 cycles
   Power Down: 0.0 Joule, 0 cycles
   Power Save: 0.0 Joule, 0 cycles
   RESERVED 1: 0.0 Joule, 0 cycles
   RESERVED 2: 0.0 Joule, 0 cycles
   Standby: 0.0 Joule, 0 cycles
   Extended Standby: 0.0 Joule, 0 cycles
```

Fig. 7. Resultados de consumo energético.

$$W = \frac{J}{s} = \frac{0.06448136}{5s} = 0.012896\frac{J}{s} = 0.012896W \quad (1)$$

4. Conclusión

En este trabajo, hemos mostrado de forma sucinta cómo implementar el algoritmo AES tal y como aparece en el estándar para el envío de datos cifrados en una WSN. Los nodos inalámbricos de las WSN son limitados en recursos, pero aun así, podrían ser suficientes para poder desarrollar medidas de seguridad. Como próximo trabajo, se finalizará la implementación completa de AES y se evaluarán soluciones de seguridad en la capa de control de acceso al medio de 802.15.4 como uno de los componentes de la futura IoT.

Referencias

[1] Aguzzi S. *et al.* (2015). *Definition of a Research and Innovation Policy Leveraging Cloud Computing and IoT Combination*. Available online: <https://goo.gl/TuLlpj>.

[2] Avrora. The AVR simulation and analysis framework, UCLA. Available online: <http://compilers.cs.ucla.edu/avrora/profiling.html>

[3] Cao X. *et al.* (2016), "Ghost-in-ZigBee: Energy Depletion Attack on ZigBee based Wireless Networks", IEEE Internet of Things Journal, DOI 10.1109/JIOT.2016.2516102.

[4] Dener M. (2014). "Security Analysis in Wireless Sensor Networks", Intl Journal of Distributed Sensor Networks, pp. 1-9.

[5] FIPS Pub 197 (2001). *Advanced Encryption Standard*, National Institute of Standards and Technology.

[6] Gay D. *et al.* (2003). "nesC 1.1 Language Reference Manual", manual.

[7] Granjal J. *et al.* (2015). "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues", IEEE Communication Surveys & Tutorials, 17 (3).

[8] IEEE Standard 802.15.4 (2006). "Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)".

[9] Keoh S. L. *et al.* (2014). "Securing the Internet of Things: A Standardization Perspective", IEEE Internet of Things Journal, 1 (3), 2014.

[10] Levis P. *et al.* (2005). "TinyOS: An Operating System for Sensor Networks", Ambient Intelligence, pp 115-148.

[11] Micaz datasheet, Memsic Inc (2016). Available online: <http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf>

[12] TinyOS Official Website (2016). Available online: <http://tinyos.stanford.edu/tinyos-wiki/index.php/>