Departamento de Informática

PROYECTO FIN DE CARRERA

Puzzles Criptográficos: Implementación y Evaluación

Autor: Alejandro Monasterio Rubio

Tutora: Esther Palomar González

Leganés, diciembre de 2012

Título: Puzzles Criptográficos: Implementación y Evaluación
Autor: Alejandro Monasterio Rubio
Directora: Esther Palomar González

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 18 de diceimbre de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

Quisiera agradecer su apoyo en la realización de este trabajo a mis padres, los cuales me han echado una mano siempre que han podido en todos los aspectos de mi vida, y en este proyecto me han transmitido todos sus ánimos y han demostrando tener una paciencia admirable.

No quisiera olvidar tampoco a mis hermanas y mi tía que siempre me han apoyado, así como a mi novia que me ha dado el empujón definitivo. A todos ellos ¡muchas gracias!

Resumen

En este Proyecto Fin de Carrera se implementan y evalúan diferentes aplicaciones de los protocolos de pruebas de esfuerzo, también conocidos como puzzles criptográficos, que se han propuesto hasta la fecha como solución a ataques como el spam, en redes peer-to-peer, redes vehiculares, e incluso en tecnología RFID *(Radio Frecuency Identification*).
Para ello, se va a comprobar la viabilidad de algoritmos criptográficos tales como el TEA (*Tiny Encryption Algorithm*) y el AES (*Advanced Encryption Standard*) como base de las pruebas de esfuerzo computacional.

En el escenario de los RFID, por ejemplo, los puzzles criptográficos implementados pretenden aliviar algunos de los problemas relacionados con la privacidad de la información almacenada en las etiquetas RFID.

Los puzzles desarrollados otorgan mayor seguridad ya que en ningún momento la etiqueta transmitirá su identificador en claro. En su lugar, la etiqueta cifrará la suma XOR del identificador (*ID*) y el número aleatorio (*n*). A continuación se la enviará al lector junto con la función resumen del identificador, una parte de la clave de cifrado (k_s), y el número aleatorio (*n*). Formando todo ello el puzzle criptográfico

$$Puzzle = enc_k(ID\ (+)\ n),\ h(ID),\ k_s,\ n$$

El lector deberá resolver este puzzle invirtiendo de esta manera tiempo y recursos computacionales para poder obtener el identificador de la etiqueta y acceder así a su información almacenada en base de datos.

Abstract

In this project, different applications of proof of work protocols are implemented, also known as cryptographic puzzles, which have been proposed to combat attacks like spam in peer-to-peer networks, vehicular networks, and even in RFID technology (Radio Frequency Identification).
In this regard, we focus on evaluating the feasibility of cryptographic algorithms such as TEA (Tiny Encryption Algorithm) and AES (Advanced Encryption Standard) as the basis for testing computational proofs of work protocol in resource-constraint scenarios.

For example, assuming a RFID scenario, our cryptographic puzzles aim at alleviating some of the problems related to the privacy of the information stored in RFID tags.

Moreover, our developed puzzle-based protocol provides security because of the tag will never transmit its identifier in clear. Instead, tags will encrypt the identifier (*ID*) XOR a nonce (*n*) using an encryption key (k_s). Thus, it will be sent to the reader together with the hash of the identifier, a part of the encryption key (l-bits of k_s), and the nonce (*n*). All these parts make the cryptographic puzzle:

$$Puzzle = enc_k(ID\ (+)\ n),\ h(ID),\ k_s,\ n$$

Reader should solve this puzzle so investing time and computational resources in order to get tags' identifiers and thereby gaining access to the information stored in the database.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

Mediante el siguiente documento se pretende dar al lector una idea global del proyecto llevado a cabo. Por ello, este capítulo resumirá el contenido completo del proyecto, que será expuesto en los sucesivos capítulos del documento de manera más amplia, así como se expondrán los objetivos perseguidos para la realización del mismo. Además de dar un resumen global del proyecto también serán resumidos los diferentes apartados mediante la explicación de la estructura del mismo.

1.1. Introducción

El uso de pruebas de esfuerzo (*Proof of work - PoW*) es una idea que se lleva utilizando desde hace tiempo en informática.

El concepto es el siguiente. Hay dos entidades que se ven involucradas en el proceso. Una entidad *A* que es la interesada en obtener un servicio, y una entidad *B* que es la poseedora del servicio que *A* solicita. *A* realiza una petición (request) a *B*. *B* antes de proporcionarle la información, solicita a *A* que resuelva un reto (puzzle). De esta forma *A* deber resolver dicho puzzle, invirtiendo tiempo y recursos para obtener su objetivo.

El esquema podría describirse como sigue:

1. *A* → *B*: *request*
2. *B* → *A*: *puzzle*
3. *A* → *B*: *solución* del puzzle
4. *B* → *A*: *response*

Las pruebas de esfuerzo tienen distintos usos, los cuales cito a continuación pero serán explicados con detalle más adelante.

- Para limitar accesos

  En este caso, el uso de pruebas de esfuerzo se emplea principalmente para evitar *spam* en el correo electrónico. La idea básica es: “Si no te conozco y deseas enviarme un correo, entonces debes demostrarme que tu correo es digno de recibir, gastando tiempo en resolver un reto en tu ordenador".

- Contar accesos:

  Las pruebas de esfuerzo también se han utilizado para medir accesos a sitios webs. Para ello se emplea una agencia auditora que genera un desafío, es decir, una tarea computacional, y la envía al servidor a través de un canal seguro. A cada cliente se le pedirá que realice una pequeña parte de esta tarea, cuya resolución final demostrará la visita de *k* clientes usando sus respuestas durante el periodo de tiempo.

- Combatir DoS (Denial of Service)

  Hay varios trabajos que usan puzzles criptográficos como una solución para combatir ataques de denegación de servicio (*DoS*). La idea de este protocolo es exigir a todos los clientes que se conectan a un servidor que resuelvan correctamente un puzzle criptográfico antes de establecer la conexión.

Considerando el primero de los usos, las pruebas de esfuerzo han sido recientemente propuestas para limitar los problemas de privacidad en tecnología RFID (*Radio Frequency Identification*) [POP+12]. RFID es una tecnología punta, muy utilizada actualmente, con la que se realiza la identificación de objetos de cualquier tipo y permite una rápida captación de datos de manera automática mediante radiofrecuencia. Se emplea principalmente, en aquellas áreas a los que ya no llegan las prestaciones de otras tecnologías de identificación, como los códigos de barras, (por ejemplo: en logística, gestión de materiales, automatización industrial, identificación de pacientes de hospitales). Esta tecnología permite que la recogida de datos se haga en tiempo real, que aumente su fiabilidad y que se reduzca el costo, permitiendo a la empresa tener una gestión más ágil y eficaz potenciando su competitividad.

Los elementos principales de un sistema RFID son los siguientes: las etiquetas electrónicas (*transponders* o *tags*) que almacenan la información, los lectores (*readers*) que se utilizan para acceder a la información; y los sistemas de información (*servidores* y *bases de datos*) que utilizan la información de las etiquetas para localizar toda la información asociada a la misma. Las etiquetas de estos sistemas pueden ser pasivas, semipasivas o activas. Este proyecto trabajará bajo etiquetas pasivas, que son las que no requieren de fuente de alimentación interna, ya que se alimentan del campo eléctrico generado por el lector.

El modelo de comunicación en RFID con etiquetas pasivas sigue los mismos pasos descritos en la página anterior. El lector inicia la comunicación lanzando una petición de identificación (*request*), a la que la etiqueta responde (*response*) con la información que tiene guardada. Esta información la utiliza el lector como índice de búsqueda en una base de datos para obtener toda la información asociada a la etiqueta.
Entre el lector y los sistemas de información no consideramos que haya amenazas de seguridad ya que estos equipos pueden ejecutar mecanismos de cifrado, además utilizan canales de comunicación seguros como interfaces cableadas de red local.
Lectores y etiquetas, sin embargo, utilizan un canal de radio para la comunicación que se supone inseguro. Por tanto, cualquier lector compatible puede tener acceso al identificador de las etiquetas en el rango del canal. Por si fuera poco, el diseño de las etiquetas está diseñado para reducir su coste, por lo que su capacidad es muy reducida y carece de mecanismos de seguridad y autenticación fiables.

Es por ello que este trabajo tratará de poner remedio, mediante el uso de puzzles criptográficos, a las posibles amenazas que pueden ocurrir en la comunicación entre ambos.

1.2. Objetivos

Con la realización de este proyecto fin de carrera se pretende resolver uno de los problemas de seguridad que amenazan a los sistemas con tecnología RFID, limitar el acceso a la información almacenada en las etiquetas a lectores deshonestos o atacantes.

Para ello se ha elaborado un protocolo de seguridad que emplea puzzles criptográficos como pruebas de esfuerzo, para garantizar que los lectores de etiquetas RFID son realmente los permitidos en el sistema y no lectores deshonestos.

Para su desarrollo se han tenido en cuenta las limitaciones que tienen las etiquetas RFID en cuanto a tamaño de memoria. Se han utilizado AES y TEA como algoritmos de cifrado para implementar las pruebas de esfuerzo debido a que requieren poca capacidad de almacenamiento.

1.3. Fases del desarrollo

Para el desarrollo de este proyecto, en un primer lugar se han estudiado protocolos de seguridad RFID implementados con anterioridad, así como se ha investigado acerca del uso de pruebas de esfuerzo. A continuación se ha desarrollado en JAVA un algoritmo de generación de puzzles criptográficos empleando los algoritmos de cifrado TEA y AES-128. Tras la implementación del algoritmo de generación, se desarrolló un algoritmo que fuera capaz de resolver el puzzle empleando fuerza bruta. Una vez desarrollado este algoritmo de resolución se procedió a realizar las pruebas experimentales para medir los tiempos de resolución del puzzle.
Finalmente se procedió a desarrollar este documento en el que se incluyen datos acerca de protocolos de autenticación en RFID, se explica de modo general en qué consisten las pruebas de esfuerzo, se expone el protocolo propuesto y finalmente se muestran y se analizan los resultados.

1.4. Medios empleados

Para la realización de este proyecto se han empleado dos ordenadores, el primero de ellos se ha empleado para desarrollar el algoritmo y las pruebas experimentales que se desvelarán en capítulos posteriores. Este PC tiene las siguientes características:

- Procesador AMD Athlon 64 Processor 3400, 2,19 GHz
- 2,00 GB de memoria RAM
- Sistema operativo: Windows XP Service Pack 3

El otro ordenador se utilizó para la redacción de este documento y tiene las siguientes características:

- Procesador Intel Core i5-2400 3,10 GHz
- 8,00 GB de memoria RAM
- Sistema operativo: Windows 7 64 bits

1.5. Estructura de la memoria

A lo largo de este documento se pasará a explicar el algoritmo propuesto con más detenimiento así como los resultados obtenidos tras su ejecución. El documento queda por tanto dividido en los siguientes capítulos

1.5.1. Capítulo 2.- Preliminares

En este capítulo se explicará en qué consisten los puzzles criptográficos, así como para que se han utilizado a lo largo del tiempo. También se dará una visión general de la seguridad en los sistemas RFID y de las amenazas a las que se enfrentan.

1.5.2. Capítulo 3.- Puzzles criptográficos en tecnología RFID

En este capítulo se procederá a explicar y desarrollar el protocolo propuesto, explicando sus características y analizando su seguridad.

1.5.3. Capítulo 4.- Resultados experimentales

En este capítulo se mostrarán los resultados obtenidos al realizar experimentos utilizando dos algoritmos de cifrado distintos para la generación del puzzle criptográfico que se empleará en el protocolo propuesto. También se realizará un análisis del hardware necesario para su implementación y una comparativa entre los resultados de los dos algoritmos de cifrado.

1.5.4. Capítulo 5.- Gestión del proyecto

En este capítulo se describen las fases en las que se ha dividido el proyecto, se explican los medios técnicos empleados y finalmente se detalla la gestión económica del mismo.

Capítulo 2

Preliminares

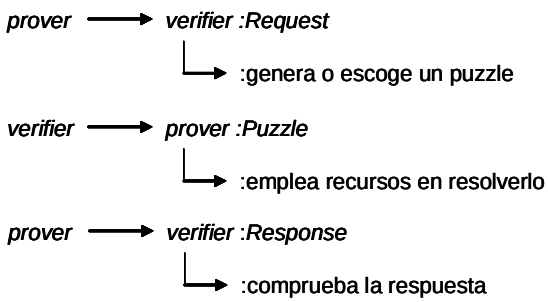
En este capítulo, en el primer apartado se explicará en qué consisten los puzzles criptográficos, así como para que se han utilizado a lo largo del tiempo. En el segundo apartado se dará una visión general de las amenazas a las que se enfrenta un sistema RFID.

2. 1 ¿Qué es un puzzle?

Cuando nos referimos a un puzzle, criptográficamente hablando, nos referimos a un reto que crea un elemento interviniente en una comunicación -en nuestro caso la etiqueta RFID- para que otro elemento que hace una petición sobre el primero -el lector- lo resuelva invirtiendo así tiempo y recursos computacionales. Con ello se pretende evitar la participación de participantes deshonestos en la comunicación.

2.1.1.Algunas definiciones y propiedades

La idea de demostrar un coste computacional realizado en un intervalo de tiempo, es decir, el sistema de pruebas de esfuerzo, es todavía la base de muchos protocolos de seguridad recientes. Básicamente, dos entidades están involucradas en dicho proceso. Una parte, (*verifier*) pide a la otra (*prover*) que complete un sencillo test antes de garantizarle acceso a un cierto servicio. *Provers* no podrán obtener el material requerido sin gastar una cantidad minima de recursos computacionales, y demostrar la respuesta esperada. A continuación ilustro un esquema interactivo básico:



En el enfoque no interactivo de POW, una gran cantidad de puzzles son calculados y a continuación son almacenados juntos de forma centralizada. *Provers* seleccionarán sus

propios retos o, en otros casos, un valor aleatorio de inicio, como se explicará a continuación. Este hecho significa que hay sólamente una ronda de comunicación desde el *prover*.

Por otro lado, quizás la propiedad más interesante de las POW recientes es la dificultad del puzzle, es decir, la complejidad de la operación criptográfica computacional. Varias propuestas dirigen el reto de establecer este coste de forma dinámica y de acuerdo a diferentes parámetros como la calidad del servicio demostrada en interacciones pasadas.

Trapdoors

En lo que concierne a la definición de coste computacional de resolver un puzzle, un nuevo concepto aparece a mediado de los 70 [Syv98] , llamado *trapdoor*. Se dice que F es una función trapdoor si existe alguna información secreta k, tal que dado F(x) y k, es fácil de calcular x y de otra forma no. Un valor tiene el compromiso de que no puede ser descubierto hasta que el que lo compromete revele o bien el valor (o algún otro secreto) o realice un cálculo privado. Varios tipos de funciones de compromiso han sido propuestas de forma que mantienen un secreto durante un tiempo o hasta que una cantidad moderada y predecible de cálculos se han producido.

El bit de compromiso es un medio de exigir a una entidad que mantenga un valor oculto hasta revelarlo en un momento posterior.

Este puede ser un ejemplo para introducir este concepto: Alice genera dos cadenas de bits aleatorios $\{R_1,R_2\}$, le añade un mensaje M mediante el cálculo $h(R_1||R_2||M)$ y envía $\{R_1, h(R_1||R_2||M)\}$ a Bob. Cuando ella quiere revelar M a Bob, elle le envía $\{R_2||M\}$. Por las propiedades de las funciones resumen: 1) Bob no puede obtener M del primer mensaje que Alicia le envía; 2) Alicia no puede encontrar otro par diferente $\{R_2' \,||\, M'\}$ tal que $h(R_1 \,||\, R_2 \,||\, M) = h(R_1 \,||\, R_2 \,||\, M')$.

Las funciones Weakly Secret Bit Commitment (WSBC) trabajan bajo el mismo principio, pero con la diferencia notable de que el secreto del bit de compromiso se puede desvelar después de un límite aceptable predefinido en términos de tiempo y/o computo. La resistencia a la segunda preimagen (*2nd preimage resistance*) y la resistencia a la preimagen débil (*weak-preimage resistance*) son las propiedades que una función WSBC $\omega()$ debe tener.

El primer sistema basado en puzzles

El sistema de puzzles fue concebido por primera vez en 1974 por Ralph Merkle [Mer78], para asegurar que dos partes se pueden comunicar de forma segura sobre un canal inseguro. Las dos entidades deben ponerse de acuerdo acerca de un secreto compartido a través de un intercambio de mensajes. El *puzzle de Merkle* consiste en una gran cantidad de puzzles en forma de mensaje encriptado con una clave desconocida. Los puzzles emplean funciones de cifrado de sentido único y la clave debe ser lo suficientemente corta para permitir ataque por fuerza bruta. Dados A y B, se ponen de acuerdo en una función de cifrado, G, también conocida por posibles espías. G es una función común de cifrado con dos argumentos: (1) el mensaje que debe cifrar y (2) la clave de cifrado. La siguiente figura muestra los mensajes del protocolo y las interacciones entre ambas partes.

A (verifier):
Primero escoge de forma secreta una función de cifrado de sentido único, F, y cra un conjunto de N puzzles:
$i^{th} \rightarrow G(k, (N * F(K,i) + i)), \forall i = 1, \ldots, N$
Donde K es la clave usada con la función F, y k es la clave compartido por G

Puzzles
→
B elige uno de los puzzles y aleatoriamente, dice j^{th} $(j \in [1,\ldots, N])$, *y resuelve* $G(k,(N * x + j))$
Básicamente, B encontrará x de manera simple probando todas las posibilidades, entonce se lo transmitirá en claro a A

x:
←
Aplica la función de descifrado, F^{-1}, *a x, y determina j, la cual se puede usar como clave por ambas A y B en futuras comunicaciones*

Al contrario que los enfoques descritos en la siguiente sección, el primer puzzle criptográfico mencionado en la literatura no fue diseñado como prueba de esfuerzo (POW), sino como una construcción primitiva de un cifrado de clave publica. En su lugar, las primeras aproximaciones de POW se concentraron en encontrar mecanismos para regular el comportamiento egoísta de los nodos mediante funciones de coste vinculadas a la CPU (*CPU-bound cost-functions*), las cuales parametrizan la cantidad de trabajo necesario par obtener un recurso. Además, diferentes primitivas se han aplicado como defensa contra spam y agotamiento de la conexión, entre otros.

2.1.2.Primeras aproximaciones a pruebas de esfuerzo

Limitación de acceso

La primera vez que se empleó un protocolo de pruebas de esfuerzo fue en 1992 [DN92] . Cynthia Dwork and Moni Naor emplearon el concepto de pruebas de esfuerzo para combatir el correo basura (*spam*). Su contribución consistió en solicitar al remitente de los correos resolver una función moderadamente complicada denominada *pricing function*, antes de poder enviar el email. En otras palabras, la idea básica era: "Si no te conozco y deseas enviarme un correo, entonces debes demostrarme que tu correo es digno de recibir, gastando tiempo en resolver una función en tu ordenador". Los autores sugieren tres funciones candidatas a *pricing function*: extraer raíces cuadradas, el esquema de identificación Fiat-Shamir, y el esquema de firma de Ong-Schnorr-Shamir. Esta función debe tener la propiedad de ser difícil de resolver para el remitente, y, en comparación, fácil de comprobar para el receptor. Por un lado, el objetivo global es limitar las capacidades (recursos y tiempo) de los adversarios, ya que los *spammers*, incluso usando *botnets*, no pueden tener cantidades ilimitadas de tiempo de procesamiento a su disposición. Por otro lado, la *pricing function* debería ser fácil de evaluar dado algún tipo de trampilla secreta (*trapdoor*) o de acceso directo. Así, para obtener la función *trapdoor* se confía en la familia de funciones criptográficas utilizada.

En 1996, Rivest, Shamir y Wagner [RSA96] resolvieron el problema de un atacante capaz de resolver puzzles en paralelo, por medio de puzzles con tiempo de bloqueo (*time-lock puzzles*). Un puzzle con tiempo de bloqueo presenta el problema computacional de que no puede ser resuelto sin una ejecución continua durante una cantidad precisa de tiempo. Por lo tanto estos puzzles se pueden usar para implementar retardos, siendo la cantidad de retardo controlada. Los puzzles con tiempo de bloqueo emplean funciones de coste fijo (*fixed-cost functions*), basadas en supercifrado en RSA y *trapadoors*.

Además el sistema de *Hascash* de Back (*Back's Hascash system*) [Bac02] , anunciado por primera vez en 1997, trata principalmente el problema del *spam* y también el ataque de denegacion de servicio (*Denial of service - DoS*), mediante la aplicación de un sistema de pruebas de esfuerzo no interactivo libre de *trapdoor*. Una función de coste, basada en encontrar colisiones parciales en la función resumen SHA-1, ayuda en el filtrado de

clientes de correo electrónico. Las implementaciones modifican la cabecera del mensaje añadiendo un token *hascash*, el cual consiste en datos relacionados con el receptor tales como una dirección, un *timestamp* y un *nonce* aleatorio. La principal diferencia entre la función de coste *Hashcash* y la función *pricing* mencionada con anterioridad es que la primera no tiene *trapdoors* para evitar que los remitentes puedan emitir tokens de forma sencilla hacia otros. Sin embargo, los remitentes también tendrán que invertir un tiempo de procesamiento requerido para crear una cabecera válida.

Contar accesos

Las pruebas de esfuerzo también se han utilizado para medir accesos.
Franklin y Malkhi propusieron un sistema seguro para contar accesos a webs que fue publicado en 1997 [FM97] y patentado en el 2000, para controlar los accesos a sitios webs mediante la participación de usuarios en la resolución de una función de tiempo (*timing function*) para una determinada entrada. La función de tiempo está basada en una función resumen de sentido único y una semilla única generada por cada visita. Más tarde y debido a la importancia de la publicidad web, en 1998 Naor y Pinkas [NP98] presentaron un eficiente sistema de medición que se basa en el sistema de compartición de secretos polinomial de Shamir. En esencia, una agencia auditora genera un desafío, es decir, una tarea computacional, y la envía al servidor a través de un canal seguro. A cada cliente se le pedirá que realice una pequeña parte de esta tarea, cuya resolución final demostrará la visita de *k* clientes usando sus respuestas durante el periodo de tiempo. Hay que tener en cuenta que hay otro rol intermedio, esto es, la entidad auditora, que modifica el protocolo interactivo de reto-respuesta y por lo tanto, debe prestar especial atención a la eficiencia de la comunicación y la colaboración de los clientes.

Combatir DoS (Denial of Service)

Hay varios trabajos que usan puzzles criptográficos como una solución elegante para combatir ataques de denegación de servicio (DoS). El protocolo Cliente-Puzzle introducido por Juels y Brainard en 1999 [JB99] utiliza puzzles de cliente para impedir que un protocolo de comunicación como TCP y SSL se le agote la conexión por la limitación de velocidad de las conexiones TCP. La idea de este protocolo es exigir a todos los clientes que se conectan a un servidor que resuelvan correctamente un puzzle

criptográfico antes de establecer la conexión. Tras resolver el puzzle, el cliente deberá devolver la solución al servidor, la cual comprobará rápidamente y si no es correcta rechazará y cancelará la conexión. El puzzle es simple y de fácil resolución pero requiere al menos una mínima cantidad de cálculos en el lado del cliente. Usuarios legítimos experimentarán un coste computacional despreciable, pero se disuadirá el uso abusivo. Aquellos clientes que quieran establecer una gran cantidad de conexiones simultáneas serán incapaces de hacerlo debido al coste computacional que implica. Para crear el puzzle el servidor genera un nonce aleatorio n_s y el nivel de dificultad k del puzzle. Esta información se le envía a los clientes. Para resolver el puzzle el cliente ID_c genera un *nonce* aleatorio n_c y resuelve X (la solución esperada) de la siguiente ecuación utilizando para ello fuerza bruta:

$$h(IDc,n_s,n_c,X)=000...000Y$$

dónde Y es el resto de bits de la función resumen y representa cualquier patrón de bits. El cliente deberá intentar 2^k soluciones antes de encontrar la correcta. Hay que tener en cuanta que n_s se deberá cambiar periódicamente para evitar que los atacantes empleen soluciones precalculadas. En este sentido los protocolos de pudin de pan de Jakobbson y Juel (*Jakobsson and Juels's Bread Pudding Protocols*) presentados en 1999 [JJ99], ampliaron el protocolo Cliente-Puzzle para otro propósito.

2.1.3.Enfoques recientes

Ahora, la idea de emplear pruebas de esfuerzo criptográficas para incrementar el coste de enviar un email y hacer el envío de spam poco rentable se está extendiendo a otras áreas de investigación recientes. Es el turno de las nuevas plataformas con recursos limitados como, por ejemplo, *Wireless Sensor Networks* (WSNs), sistemas RFID y también sistemas peer-to-peer (P2P) y redes vehiculares [PAL12]. Del mismo modo, hay varias publicaciones que tratan de mitigar los ataques DoS y *free-riding* además de proporcionar soluciones para el enrutamiento y autenticación en tales dominios. Para hacer frente al ataque de *free-riding*, la idea subyacente es que se puede animar a compartir estableciendo un coste a las descargas, pero asegurando que aquellos que compartan más no se verán afectados por este coste. En el esquema de micropagos de Mankins et al. [MKB+01], los usuarios reciben un incentivo para trabajar juntos hacia un fin común a

través de la introducción de estos puzzles. Los autores presentan y valoran varios tipos de micropagos.

Mientras tanto, la contribución de Abadi et al. en [ABM+03], presenta un enfoque alternativo a las pruebas de esfuerzo computacionales basada en latencia de memoria, ya que la latencia de memoria varía en mucha mayor medida que la velocidad de la CPU. Los autores llamaron a esta idea*: Memory-Bound Functions* (MBFs). Antes de MBFs, los participantes más poderosos eran capaces de resolver puzzles más rápido que otros.

Ahora, los puzzles son procesos o cálculos vinculados a la memoria, y se pueden usar para asegurar que cada nodo gastará la misma cantidad de recursos críticos. El análisis y evaluación de diversas MBFs presentado por Dwork et al. en [DGN03] da como resultado un rendimiento constante en diferentes máquinas. Como consecuencia de tales validaciones, muchos trabajos basados en MBFs fueron presentados en distintos campos de investigación. Por ejemplo, Serjantov y Lewis [SL03] consideraron usar puzzles de cliente para proporcionar incentivos en sistemas P2P. Sin embargo, hay controversia sobre la conveniencia de imponer dicho esfuerzo a cada nodo en el sistema, sin importar cuál es el comportamiento de cada nodo. Este hecho se discute en varios artículos, como la publicación de Laurie y Clayton en [LC04], que cuenta la carga adicional, injusta y contraproducente a los participantes honestos.

Por otro lado, la propuesta de Borisov, presentada en [Bor06] , impone un coste computacional a la ocupación de una posición dentro de una red con el fin de asegurar su participación en el sistema P2P, evitando así el ataque Sybil donde los atacantes con gran cantidad de recursos computacionales pueden conseguir un gran rango de identidades en la red.

Por otra parte el mecanismo de autenticación débil de Ning et al. propuesto en [NLD08] emplea cadenas de claves de un sólo sentido para asegurar la autenticación de los paquetes difundidos en WSNs. Así mismo, las pruebas de esfuerzo se están adaptando a sistemas RFID mediante la aplicación de funciones ligeras. Por ejemplo, en el protocolo reto-respuesta de Burmester et al. [BMM08], la etiqueta RFID puede ocultar su identificador de forma que sólo el servidor de back-end puede descubrirlo usando una trampilla (trapdoor) que únicamente el posee. Los autores emplean funciones de un sentido de clave pública.

Finalmente es aún un reto el establecer claramente cual debe ser la dificultad del puzzle que se va a enviar. De hecho, Narasinmhan et al. en 2010 [NVR10] aplicaron la Teoría de Juegos para analizar formalmente un protocolo de cliente-puzzle, y afirmaron que la dificultad del puzzle no debería ser determinada sin un número mínimo de cálculos.

2.2. Protocolo de seguridad RFID

En un esquema de identificación básico, completamente inseguro, en primer lugar el lector envía una petición (request) a la etiqueta, al cual responde con un identificador estático (ID).

reader ⟶ *tag: Request*

tag ⟶ *reader: ID*

En este esquema se pueden destacar las siguientes vulnerabilidades que pueden ser explotadas por parte de un adversario:

- El canal etiqueta-lector es un canal inseguro.
- Cualquier lector compatible puede acceder a la información de las etiquetas en el rango del canal lector-etiqueta.
- El diseño de las etiquetas está optimizado para reducir su coste, por lo que su capacidad es muy reducida y carece de mecanismos de seguridad y autenticación fiables.

Las cuatro principales amenazas de un sistema RFID son las siguientes: [MGH]

1. Escuchas fraudulentas:

   Éstas se definen como la presencia de lectores no autorizados con acceso a la comunicación del canal lector-etiqueta.

   El canal de comunicación entre lectores y etiquetas es fácilmente accesible dada la inseguridad del canal inalámbrico, con lo que la confidencialidad de los datos transmitidos es fácilmente vulnerable.

2. Suplantación de identidades:

   Puesto que el protocolo RFID no dispone de mecanismos de autenticación, un adversario no encontraría ninguna dificultad para conseguir la misma información que podría obtener un usuario autorizado dentro del sistema. Un lector no autorizado podría suplantar a uno autorizado, obteniendo la identificación de etiquetas legítimas. Esta información se podría reproducir en etiquetas ilegítimas, lo que significaría un caso de clonación de etiquetas. Un lector autorizado no podría discernir entre una etiqueta legítima y otra clonada al no haber mecanismos de autenticación para la identificación de etiquetas.

3. Divulgación de información:

   El riesgo derivado de la suplantación de identidad va más allá de la posible falsificación o clonación de etiquetas electrónicas. Dado que la información almacenada en las etiquetas puede ser muy valiosa, un adversario podría obtener beneficio económico de la venta de esa información.

4. Denegación de servicio:

   La denegación de servicio (*DoS*) es una amenaza que tiene por objeto limitar o anular la funcionalidad de un sistema de información. En nuestro caso, la denegación de servicio significaría dejar inoperativo el canal de comunicación haciendo inviable el intercambio de información. Un lector no autorizado en el rango de lectura del canal lector-etiqueta emitiendo constantemente peticiones de identificación, reduciría considerablemente la eficiencia de lectura de los lectores autorizados, retrasando los procesos de inventariado del sistema atacado.

Se han publicado gran variedad de estudios que han intentado resolver los problemas de seguridad relacionados con el uso de etiquetas RFID y han propuesto protocolos para lograr la autenticación segura y evitar el acceso no autorizado a la información contenida en dichas etiquetas.

En alguno de estos trabajos se han empleado puzzles criptográficos en la autenticación RFID. En éstos, para evitar que la etiqueta devuelva su identificador directamente al lector, se transmite un puzzle del identificador.

reader ⟶ *tag: Request*

tag ⟶ *reader: Puzzle(ID)*

Capítulo 3

Puzzles criptográficos en tecnología RFID

A lo largo de este capítulo se procederá a explicar y desarrollar el protocolo propuesto.

3.1. Nuestra aplicación de puzzles en tecnología RFID

En esta sección se va explicar un protocolo basado en puzzles criptográficos. Este protocolo debe ofrecer protección a la privacidad de la información confidencial almacenada en las etiquetas. Esto es, su identificador estático (*ID*). Este identificador permite la identificación unívoca de la etiqueta y se puede usar como índice para localizar toda la información adicional vinculada a la etiqueta que se encuentra almacenada en una base de datos off-line. Los mensajes deben ser anónimos para evitar los ataques de replicación y rastreo. El protocolo ofrece una protección moderada en lo que concierne a la privacidad cuando se considera una sola etiqueta. Decimos que el nivel de seguridad es "moderado" porque un adversario después de un elevado consumo computacional podría descubrir la información secreta almacenada en la etiqueta, poniendo en riesgo su privacidad. No obstante, el mecanismo resulta muy efectivo cuando se considera una gran cantidad de etiquetas y en tal caso esta tarea se hallaría fuera del alcance del adversario. Es por ello que el protocolo resulta muy útil para encarar problemas reales como revelación de inventario o clonación de etiquetas.

3.1.1. Notación

R y *T* denotan las dos partes involucradas en el protocolo, lector y etiqueta respectivamente. Asumimos que lectores y etiquetas emplean un canal de comunicación no seguro. También asumimos que el *ID* es la información que las dos entidades les gustaría intercambiar de forma segura, donde *ID* simboliza el número de identificación único de cada etiqueta. Además, $enc_k(x)$ es un algoritmo de cifrado de clave simétrica (por ejemplo AES o TEA que cifran el mensaje *x* bajo la clave *k*). Mientras que $enc_{ks}^{-1}(x)$ será el algoritmo de descifrado.

Así mismo $\omega^{\pi}_{j}(k)$ representa una función WSBC, esto es una *trapdoor*, que consiste en una selección de *l* bits de la clave *k*. Nos vemos forzados a elegir una función tan sencilla

debido a las limitaciones que ofrecen las etiquetas RFID de bajo/moderado coste. Finalmente *h(x)* representa una función resumen (SHA-2) cuya entrada es *x*.

3.1.2.Protocolo propuesto

Preparativos iniciales

Cada etiqueta tiene asignado un identificador único (*ID*) y una clave secreta (*k*), los cuales son asignados en el proceso de inicialización. El identificador y la información vinculada a la etiqueta se encuentran almacenados en una base de datos off-line. Los lectores legítimos únicamente tendrán acceso a la información que guarda la base de datos una vez que se hayan autenticado de forma satisfactoria.

Protocolo de identificación

En esta sección introduciremos un protocolo de identificación seguro entre una etiqueta y un lector. En este proceso no se ve involucrada la base de datos en ningún momento. Por lo tanto el lector puede ejecutar el protocolo varias veces para identificar un gran número de etiquetas. Entonces, una vez que el identificador estático de esas etiquetas haya sido revelado, el lector puede establecer una conexión segura y comprobar todos esos valores en un sólo paso, evitando así una conexión permanente a la base de datos.

Los pasos que sigue el protocolo para la identificación de una etiqueta son los siguientes:

1. R → T: request
   El lector R envía una petición (*request*) a la etiqueta T
2. T → R: puzzle(ID) = $enc_k(x)$, $h(ID)$, $\omega^{\pi}_{j}(k)$, n
   La etiqueta T envía un puzzle con el identificador al lector R
   El puzzle se genera de la siguiente forma
   1. Se genera un nonce *n*
   2. Se hace una suma XOR del *ID* con *n*: $x=ID\ (+)\ n$
   3. Se cifra *x* bajo la clave *k*: $enc_k(x)$
   4. Se genera una función resumen del *ID*: $h(ID)$
   5. Se genera la trapdoor $\omega^{\pi}_{j}(k)$
3. R → T: *ID*
   El lector R debe resolver el puzzle, devolviéndole como solución el *ID*

Denotaremos $o_w=enc_k(x)$

Para resolver el puzzle el lector R deberá probar todos los valores posibles de k^{-1}

```
while !enc
      dv=enc_ks^-1(ow)
      IDs= dv (+) n
      h(IDs)
      Si h(IDs)=h(ID)
            enc=true;
      s++
```

Deberá iterar valores de la clave de descifrado, hasta que encuentre la solución correcta.

En cada iteración, tiene una clave de descifrado k_s^{-1} con la cual descifra o_w y obtendrá d_v

Tras ello hará una suma XOR de d_v con en nonce n y obtendrá un identificador candidato ID_s

Hará una función resumen de ID_s : $h(ID_s)$

Finalmente si $h(ID_s)$ coincide con el $h(ID)$ recibido entonces sabrá que ID_s es el identificador que busca. De no ser así, probaría con un nuevo valor de k_s^{-1}

Análisis de la seguridad

El protocolo propuesto trata de dar solución a los problemas relacionados con la privacidad y la trazabilidad de las etiquetas RFID.

Se mantiene la privacidad ya que el identificador de la etiqueta no es enviado nunca en claro a través del canal inseguro. Concretamente, una versión cifrada del identificador $enc_k(ID(+)\ n)$, la cual requiere de la clave secreta k para su resolución, es utilizada en la generación del puzzle. Del mismo modo se envía una función resumen del ID para que el lector pueda verificar el puzzle sin comprometer ninguna información confidencial. Además la etiqueta proporciona parte de la clave secreta $\omega^{\pi}_{j}\ (k)$ al lector. La elección del tamaño de esta parte de la clave se analizará en este trabajo más adelante.

La información confidencial será entregada a los lectores una vez que el puzzle criptográfico sea resuelto. Después de ello, el lector podrá obtener la información privada vinculada a la etiqueta. Es cierto que si un lector no autorizado quisiera obtener la información de una única etiqueta podría hacerlo tras consumir una cantidad significativa de tiempo, sin embargo el objetivo principal del protocolo propuesto es evitar la revelación de los contenidos de un gran número de etiquetas, en cuyo caso la tarea resultaría inviable ya que el tiempo invertido sería excesivo. Del mismo modo, si la información privada no se ve comprometida, el rastrear un grupo de etiquetas sería en vano ya que el lector deshonesto no podría distinguir entre las respuestas emitidas por diferentes etiquetas.

En lo que respecta al rendimiento, se puede cuestionar si el protocolo propuesto es lo suficientemente eficiente. Si lo comparamos con un simple y no seguro esquema de identificación en el cual las etiquetas revelan su identificador estático indiscriminadamente, no lo es. No obstante, si comparamos el tiempo invertido en resolver un puzzle -identificar una etiqueta en nuestro esquema- con el lento ratio de lectura de la tecnología de código de barras, nuestro protocolo resultaría efectivo, eficiente y fiable. Como ejemplo decir, usando este protocolo podríamos comprobar el stock de un almacén en unas pocas horas, mientras que esta tarea llevaría varios días empleando códigos de barras.

Una de las ventajas más importantes de esta propuesta es la ausencia de una base de datos on-line. El lector puede conectar con la base de datos cada cierto tiempo y comprobar un conjunto de etiquetas identificadas, todo al mismo tiempo. Además, se podría hacer que cada vez que una etiqueta sea leída sea actualizada su clave. Se podría hacer está actualización en dos momentos. O bien, una vez que la etiqueta envía el puzzle, o una vez que el lector haya resuelto correctamente el puzzle. La segunda opción sería mejor cuando se necesite una conexión permanente a la base de datos. Sin embargo, para nuestro protocolo sería mejor la primera opción ya que impide en gran medida la posibilidad de un ataque satisfactorio de un oponente escuchando en el canal. En virtud de esta actualización de la clave, nuestra propuesta permitiría seguridad hacia atrás, es decir, las comunicaciones pasadas están protegidas incluso cuando el contenido de la etiqueta es revelado.

Otro importante aspecto en lo referente al uso de las etiquetas RFID es la resistencia a los ataques de clonación. Nuestro protocolo se puede ver como una contramedida ante ellos. Un atacante puede clonar una etiqueta en particular después de resolver el puzzle criptográfico enviado por ella. Sin embargo, el ratio de éxito de este ataque es nulo cuando el número de etiquetas se incrementa por el excesivo tiempo que consume en resolver todos los puzzles criptográficos asociados a las etiquetas. El problema de este protocolo es que los lectores honestos sufrirían el mismo problema. Es por ello que sería conveniente hallar una forma de cómo podrían las etiquetas descubrir si el lector que pretende descubrir su identificador es honesto o deshonesto. En otros protocolos propuestos [POP+12], se utiliza la distancia que hay entre el lector y la etiqueta para enviarle un puzzle de mayor o menor complejidad, ya que entiende que los lectores honestos estarán más cerca de las etiquetas y recibirán por tanto un puzzle de menor complejidad, mientras que los lectores no autorizados se encontrarías a mayor distancia y por ello recibirán un puzzle de mayor complejidad.

Capítulo 4

Resultados experimentales

En este capítulo, en un primer apartado se mostrarán los resultados obtenidos al realizar experimentos utilizando dos algoritmos de cifrado distintos para la generación del puzzle criptográfico que se empleará en el protocolo propuesto. A continuación se realizará un análisis del hardware necesario para la implementación de dichos algoritmos en etiquetas RFID. Finalmente se hará una comparativa entre los dos algoritmos de cifrado.

4.1. Algoritmos implementados

Se han realizado pruebas empleando dos algoritmos de cifrado distintos para la creación de nuestro puzzle criptográfico. Estos son el TEA y el AES. Para ambos emplearemos clave de cifrado de 128 bits.

Para la implementación del algoritmo TEA hemos adaptado a Java la versión en C que desarrollaron David Wheeler y Roger Needham [WN94] .

El algoritmo AES lo hemos implementado utilizando las librerías del paquete javax.crypto de Java.

Elegimos estos algoritmos por su ligereza, y por tanto por su mejor adaptación a las limitaciones de las etiquetas RFID. Más adelante se hará un análisis de todo esto. En primer lugar mostramos el diagrama UML de las clases implementadas (Figura 1) para construir nuestros puzzles y evaluar su coste. Además, por claridad presentamos el código java de las clases principales.

4.1.1.Diagrama UML de clases Java

A continuación se muestra el diagrama UML de las clases más importantes implementadas para la generación y resolución de puzzles.

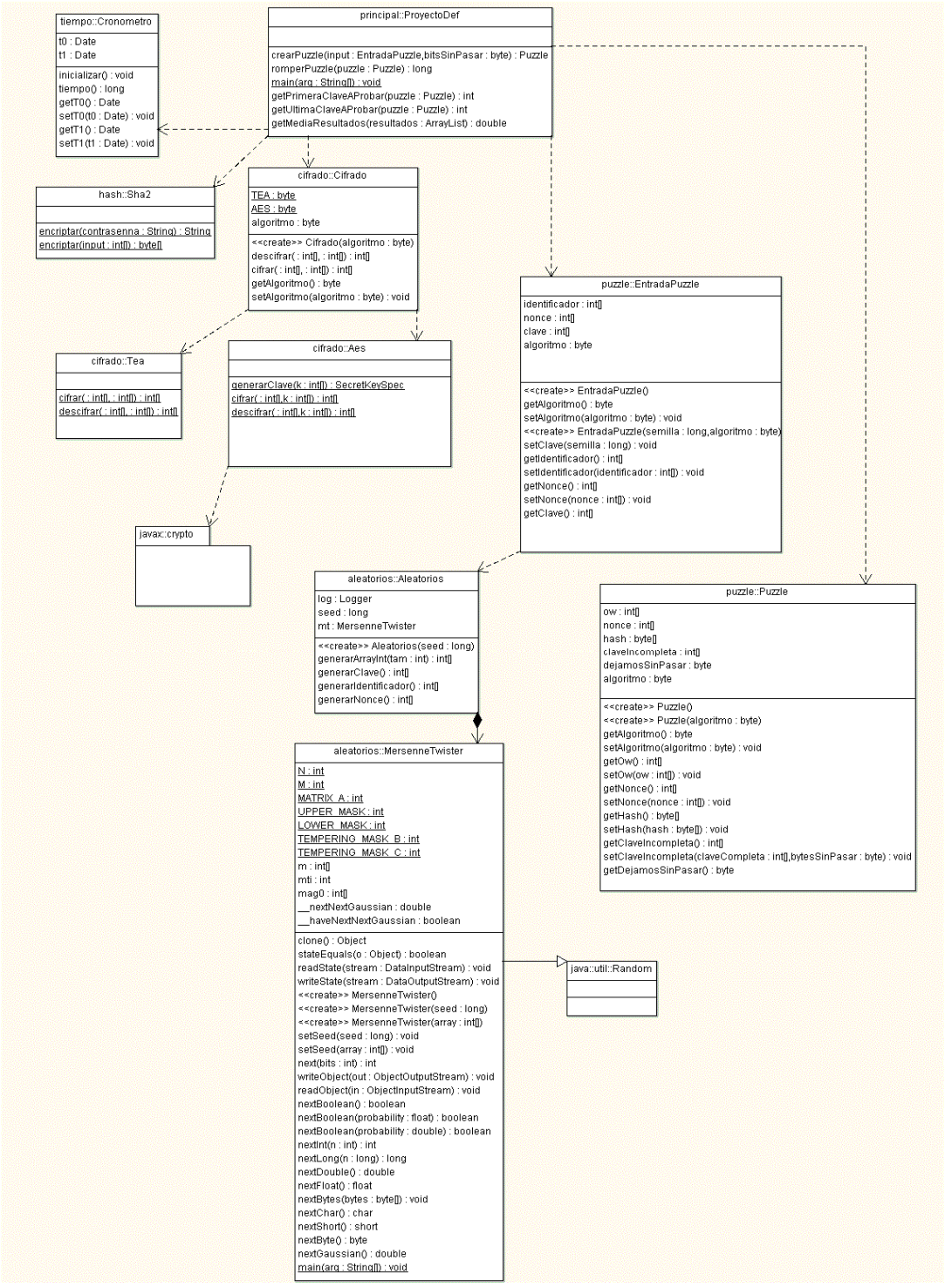


Figura 1. Diagrama UML de clases JAVA

4.1.2.Código Java

Clase principal

```
//Método principal
public static void main(String args[]){
	byte algoritmo=Cifrado.AES; //byte algoritmo=Cifrado.TEA;
	ProyectoDef proy= new ProyectoDef();
	int numPruebas=100;//Número de pruebas
	long semilla;
	EntradaPuzzle input=null;
	Puzzle puzzle=null;
	byte bitsSinPasar=22; //Número de bits de la clave que dejamos sin pasar

	ArrayList<Long> resultados= new ArrayList<Long>();
	Long t=null;
	for(byte i=0;i<numPruebas;i++){
		semilla=new Date().getTime();
		//Generamos los datos de entrada del puzzle
		input=new EntradaPuzzle(semilla, algoritmo);
		//Creamos el puzzle
		puzzle=proy.crearPuzzle(input, bitsSinPasar);
		//Rompemos el puzzle
		t=proy.romperPuzzle(puzzle);
		resultados.add(t);
	}
	//Calculamos la media de los resultados
	double media=proy.getMediaResultados(resultados);

    }

/**
* Genera un puzzle a partir de los datos necesarios para ello
* @param input <EntradaPuzzle> Datos de entrada del puzzle
* @param bitsSinPasar <byte> Número de bits de la clave que dejaremos de
	pasar (Entre 1 y 32)
* @return <Puzzle> Devuelve el puzzle generado
*/
public Puzzle crearPuzzle(EntradaPuzzle input, byte bitsSinPasar){
	Puzzle puzzle=new Puzzle(input.getAlgoritmo());

	//establecemos el algoritmo
	puzzle.setAlgoritmo(input.getAlgoritmo());

	//establecemos el nonce
	puzzle.setNonce(input.getNonce());

	//hacemos la sum xor del id y el nonce
	int[]idXORNonce=SumaXOR.sumaXOR(input.getIdentificador(),
	input.getNonce());

	//hacemos la funcion resumen del identificador
	byte[]hash=Sha2.encriptar(input.getIdentificador());
	puzzle.setHash(hash);

	//ciframos con el algoritmo deseadpo: idXORNonce con la clave
	Cifrado cifrado=new Cifrado(input.getAlgoritmo());
	int[] ow=cifrado.cifrar(idXORNonce,input.getClave());
```

```
		puzzle.setOw(ow);

		//pasamos la clave incompleta
		puzzle.setClaveIncompleta(input.getClave(), bitsSinPasar);

		return puzzle;
	}


	/**
	* Rompe el puzzle y devuelve el tiempo (en milisegundos) que se ha empleado
		para ello.
	* @param puzzle<Puzzle> Puzzle a romper
	* @return <long> Tiempo (en ms) que se ha tardado en romper el puzzle
	*/
	public long romperPuzzle(Puzzle puzzle){
		boolean enc=false; //solución encontrada
		int[] dv= new int[2]; //resultado de descifrar (ID XOR nonce)
		int[] candidato= new int[2]; //ID candidato = dv XOR nonce
		byte[] hashCandidato= null;  //hash del ID candidato
		int[] clavePrueba= new int[4]; //Clave de cifrado que probamos en cada
	iteración
		//Inicializamos el cronometro
		Cronometro c= new Cronometro();
		c.inicializar();
		long tiempo=-1;
		int x=0;//iterardor

		int primeraClave; //Primera clave a probar
		int ultimaClave;  //Última clave a Probar
		primeraClave=getPrimeraClaveAProbar(puzzle);
		ultimaClave=getUltimaClaveAProbar(puzzle);

		//Clave con la que probamos para romper el puzzle
		clavePrueba=puzzle.getClaveIncompleta();


		//Desciframos con el algoritmo que se generó el puzzle: idXORNonce con
		la clave
		Cifrado cifrado=new Cifrado(puzzle.getAlgoritmo());
		while(x<=ultimaClave&&!enc){
			clavePrueba[3]=primeraClave+x;
			dv=cifrado.descifrar(puzzle.getOw(), clavePrueba);
			candidato=SumaXOR.sumaXOR(dv, puzzle.getNonce()); //ID candidato
			hashCandidato=Sha2.encriptar(candidato); //Hash(candidato)
			if(Utilidades.sonIgualesArray(puzzle.getHash(), hashCandidato)){
			//Si son iguales el hash que recibimos en el puzzle y el hash de
			//nuestro candidato, hemos encontrado la solución
				tiempo=c.tiempo();
				enc=true;
			}
			x++;
		}

		return tiempo;
	}
```

Cifrado TEA

```
/**
* Cifrar
* @param v int[2] Entrada. Es una array de 2 posiciones de 32 bits cada una.
* @param k int[4] Clave. Es una array de 4 posiciones de 32 bits cada una.
* @return int[2]. Es una array de 2 posiciones de 32 bits cada una.
*/
public static  int[] cifrar(int v[], int k[]){
	int []c= new int[2] ;
	c[0]=v[0];
	c[1]=v[1];
	int y=c[0];
	int z=c[1];
	int delta=0x9E3779B9; //Constante de cifrado
	int n=32; //Número de rondas
	int sum=0;
	while(n-->0){ //Ciclo de cifrado
		sum+=delta;
		y+=(z<<4)+k[0]^z+sum^(z>>5)+k[1];
		z+=(y<<4)+k[2]^y+sum^(y>>5)+k[3];
	}
	c[0]=y;
	c[1]=z;
	return c;
}

/**
* Descifrar
* @param w int[2] Entrada. Es una array de 2 posiciones de 32 bits cada una.
* @param k int[4] Clave. Es una array de 4 posiciones de 32 bits cada una.
* @return int[2]. Es una array de 2 posiciones de 32 bits cada una.
*/
public static int[] descifrar(int w[], int k[]){
	int []d= new int[2] ;
	d[0]=w[0];
	d[1]=w[1];
	int y=d[0];
	int z=d[1];
	int delta=0x9E3779B9; //Constante de cifrado
	int n=32; //Número de rondas
	int sum=delta<<5;
	while(n-->0){ //Ciclo de descifrado

		z -= (y << 4)+k[2] ^ y+sum ^ (y >> 5)+k[3];
		y -= (z << 4)+k[0] ^ z+sum ^ (z >> 5)+k[1];
		sum -= delta;
	}
	d[0]=y;
	d[1]=z;
	return d;
}
```

Cifrado AES

```
/**
* Cifrar
* @param v int[2] Entrada. Es una array de 2 posiciones de 32 bits cada una.
* @param k int[4] Clave. Es una array de 4 posiciones de 32 bits cada una.
* @return int[2]. Es una array de 2 posiciones de 32 bits cada una.
*/
public static  int[] cifrar(int v[], int[] k ){
	byte[] bCifrado=null;
	int[] cifrado=null;
	try {
		Cipher cipher = Cipher.getInstance("AES");
		SecretKeySpec skeySpec=generarClave(k); //Generamos la clave de
			cifrado
		cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
		byte[] input=Utilidades.intArrayToByteArray(v, true);
		//Ciframos
		bCifrado=cipher.doFinal(input);
		// Transformamos en array de int
		cifrado=Utilidades.byteArrayToIntArray(bCifrado, true);
	} catch (NoSuchAlgorithmException e) {
		e.printStackTrace();
	} catch (NoSuchPaddingException e) {
		e.printStackTrace();
	} catch (InvalidKeyException e) {
		e.printStackTrace();
	} catch (IllegalBlockSizeException e) {
		e.printStackTrace();
	} catch (BadPaddingException e) {
		e.printStackTrace();
	}
	return cifrado;
}


/**
* Descifrar
* @param w int[2] Entrada. Es una array de 2 posiciones de 32 bits cada una.
* @param k int[4] Clave. Es una array de 4 posiciones de 32 bits cada una.
* @return int[2]. Es una array de 2 posiciones de 32 bits cada una.
*/
public static int[] descifrar(int w[], int[] k){
	byte[] bDescifrado=null;
	int[] descifrado=null;
	try {
		Cipher cipher = Cipher.getInstance("AES");
		SecretKeySpec skeySpec=generarClave(k); //Genramos clave de
	descifrado
		cipher.init(Cipher.DECRYPT_MODE, skeySpec);
		byte[] input=Utilidades.intArrayToByteArray(w, true);
		//Desciframos
		bDescifrado=cipher.doFinal(input);
	// Transformamos en array de int
		descifrado=Utilidades.byteArrayToIntArray(bDescifrado, true);
	} catch (NoSuchAlgorithmException e) {
		e.printStackTrace();
	} catch (NoSuchPaddingException e) {
		e.printStackTrace();
```

```
		} catch (InvalidKeyException e) {
			e.printStackTrace();
		} catch (IllegalBlockSizeException e) {
			e.printStackTrace();
		} catch (BadPaddingException e) {
			//Salta cuando la clave no es correcta para descifrar
			descifrado=new int[4];
		}
		return descifrado;
}


/**
* Genera la clave de cifrado a partir de la clave pasada como parámetro
* @param k Clave.Es una array de 4 posiciones de 32 bits cada una.
* @return <SecretKeySpec> Clave de cifrado
*/
private static SecretKeySpec generarClave(int[] k){
		// Get the KeyGenerator
		SecretKeySpec skeySpec =null;
		// Transformamos en array de bytes
		byte[] raw=Utilidades.intArrayToByteArray(k, true);
		skeySpec = new SecretKeySpec(raw, "AES");
		return skeySpec;
}
```

4.2. Pruebas en PC

Pata probar la viabilidad de nuestro protocolo, se han realizado pruebas experimentales en un PC con las siguientes características:

- Procesador AMD Athlon 64 Processor 3400, 2,19 GHz
- 2,00 GB de memoria RAM
- Sistema operativo: Windows XP Service Pack 3

El código se ha realizado en Java y se ha compilado bajo la versión 1.6

Aparte de la elección del algoritmo de cifrado, también hay que tener en cuenta el tamaño de trapdoor que elegiremos, es decir, la cantidad de bits de la clave que se deben de averiguar a la hora de resolver el puzzle. Las pruebas las hemos realizado con tamaños de: 20, 24, 28 y 32 bits.

Hemos llevado a cabo 100 experimentos para diferentes valores de (n-l) bits, variando aleatoriamente los retos y claves usadas. Hemos considerado que más de 32 bits

escondidos serían inviables. Los resultados obtenidos son los que mostramos en las tablas del Anexo I. Para cada caso, si el valor de l disminuye, el número de claves candidatas aumenta.

Las siguientes figuras proporcionan una caracterización de nuestros resultados empleando el algoritmo TEA con clave de 128 bits y 20, 24, 28 bits de trapdoor (Figura 2), así como el algoritmo AES con el mismo tamaño de clave y mismos bits de trapdoor (Figura 3). El propósito de estas figuras es mostrar los valores más extremos en nuestro conjunto de pruebas (máximos y mínimos valores que debemos considerar como situaciones extremas), el primer y tercer cuartil y la mediana. Los resultados se muestran con un diagrama de caja (*boxplot*) dónde la caja representa el rango intercuartílico y los bigotes se extienden hasta los valores mínimos y máximos. Si se consideran los valores de la variable comprendidos entre las dos barreras interiores, el valor mínimo de la variable y el valor máximo son los extremos de los bigotes. Si existen valores de la variable comprendidos entre las barreras interiores y exteriores se consideran valores atípicos. Si existieren valores fuera de las barreras exteriores se consideran valores todavía más atípicos. Por otra parte, este tipo de gráfico nos proporciona información con respecto a la simetría o asimetría de la distribución. Se utilizan los siguientes criterios: si la mediana está en el centro de la caja o cerca de él, constituye un indicio de simetría de los datos, si la mediana está considerablemente más cerca del primer cuartil indica que los datos son positivamente asimétricos y si está más cerca del tercer cuartil, señala que los datos son negativamente asimétricos. Asimismo, la longitud relativa de los bigotes se puede emplear como un indicio de su asimetría.

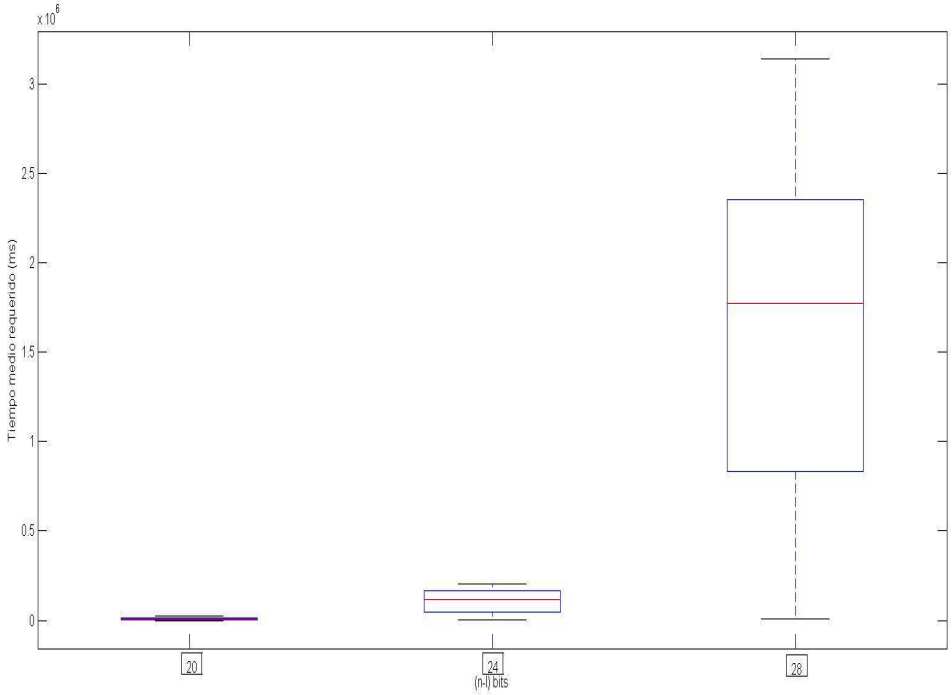


Figura 2. Tiempo medio requerido (ms) en pruebas con TEA para diferentes valores de (n-l) bits, y variando aleatoriamente los retos y claves empleadas.

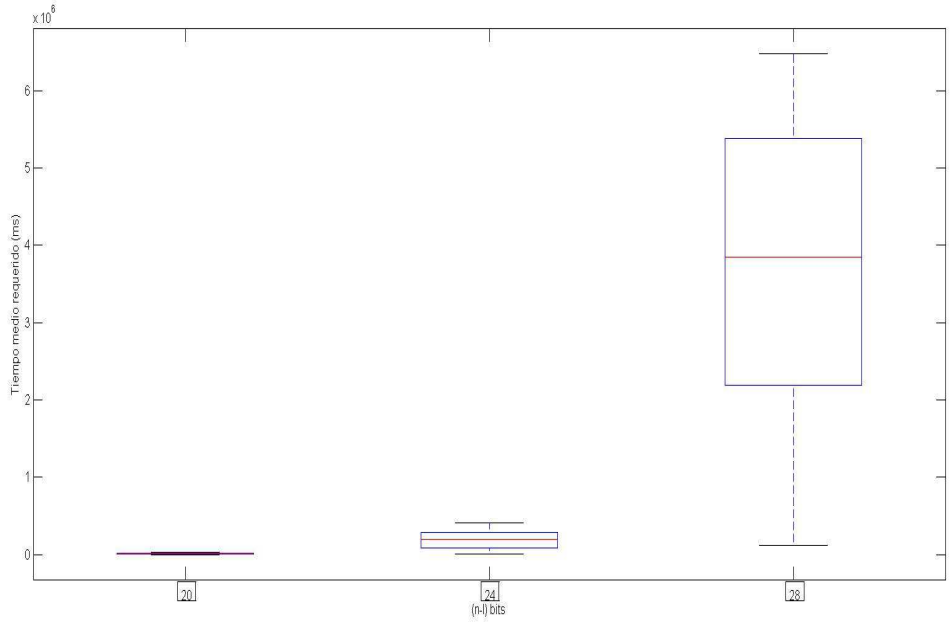


Figura 3. Tiempo medio requerido (ms) en pruebas con AES-128 para diferentes valores de (n-l) bits, y variando aleatoriamente los retos y claves empleadas.

.

4.3. Análisis de hardware

En esta sección se va a realizar un análisis de los algoritmos propuestos (TEA y AES-128) para la generación del puzzle, para ver así si serían factibles de implementar en una etiqueta RFID teniendo en cuenta las limitaciones que ofrece ésta.
Vamos a considerar una etiqueta RFID pasiva estándar (recibe la alimentación a través del lector por inducción) con las siguientes características:

64 bits TID
240bits EPC
512bits user

El puzzle iría en la memoria *user* que es la que se utiliza para guardar información para la optimización de procesos.

Algoritmo TEA:

Memoria que ocupa:

Identificador(v)-> 64 bits. Es un array de 2 posiciones de int (32 bits cada uno)
Clave(k)-> 128 bits. Es un array de 4 posiciones de int (32 bits cada uno)

Variables auxiliares:
y=v[0] int->32 bits
z=v[1] int->32 bits
sum int->32 bits
Contador(n) int->32 bits

Constante (delta)-> int->32 bits

Total=352 bits.

Operaciones que realiza:

Para el cifrado:

```
int y=v[0];
int z=v[1];
int delta=0x9E3779B9;
int n=32;
int sum=0;
        5 asignaciones

Durante 32 veces:
        n-->0
```

1 resta
1 comparación
sum+=delta;
1 suma
1 asignación
y+=(z<<4)+k[0]^z+sum^(z>>5)+k[1];
1 desplazamiento de los bits de z 4 posiciones a la izquierda (z<<4)
1 XOR (k[0] XOR z)
1 desplazamiento de los bits de z 5 posiciones a la derecha (z>>5)
1 XOR (sum XOR(z>>5))
4 sumas
1 asignación
z+=(y<<4)+k[2]^y+sum^(y>>5)+k[3];
1 desplazamiento de los bits de y 4 posiciones a la izquierda (y<<4)
1 XOR (k[2] XOR y)
1 desplazamiento de los bits de y 5 posiciones a la derecha (y>>5)
1 XOR (sum XOR(y>>5))
4 sumas
1 asignación
v[0]=y;
v[1]=z;
2 asignaciones

Para el descifrado:

int y=w[0];
int z=w[1];
int delta=0x9E3779B9;
int n=32;
4 asignaciones
int sum=delta<<5;
1 desplazamiento de los bits de delta 5 posiciones a la izquierda (delta<<5)
1 asignación
Durante 32 veces:
n-->0
1 resta
1 comparación
z -= (y << 4)+k[2] ^ y+sum ^ (y >> 5)+k[3];
1 desplazamiento de los bits de y 4 posiciones a la izquierda (y<<4)
1 XOR (k[2] XOR y)
1 desplazamiento de los bits de y 5 posiciones a la derecha (y>>5)
1 XOR (sum XOR(y>>5))
3 sumas
1 resta
1 asignación
y -= (z << 4)+k[0] ^ z+sum ^ (z >> 5)+k[1];
1 desplazamiento de los bits de z 4 posiciones a la izquierda (z<<4)
1 XOR (k[0] XOR z)
1 desplazamiento de los bits de z 5 posiciones a la derecha (z>>5)
1 XOR (sum XOR(z>>5))

```
                3 sumas
                1 resta
                1 asignación
        sum -= delta;
                1 resta
                1 asignación
w[0]=y;
w[1]=z;
        2 asignaciones
```

Algoritmo AES:

En lo que respecta al algoritmo AES no se ha podido realizar el análisis de hardware debido a que para su implementación se han empleado las clases del paquete javax.crypto de Java.

4.4. Comparativa entre AES y TEA

En esta sección vamos a analizar las diferencias que hay entre emplear el algoritmo AES-128 y el algoritmo TEA en la generación del puzzle.

En primer lugar, tras realizar las pruebas experimentales en un PC, se observa que hay una diferencia notable en el tiempo consumido a la hora de resolver el puzzle entre ambos algoritmos. Estos datos pueden verse en la siguiente tabla.

Tabla 1. Esfuerzo medio computacional hecho (en 100 experimentos) por cada lector para varias cantidades de bits conocidos.

		Tiempo medio requerido (segundos)	
n-l bits	l bits	AES-128	TEA
32	96	53913	23134
28	100	3677	1613
24	104	194	107
20	108	14	10

A continuación, se van a mostrar las gráficas obtenidas al comparar los tiempos obtenidos empleando TEA y AES-128, para diferentes valores de n-l.

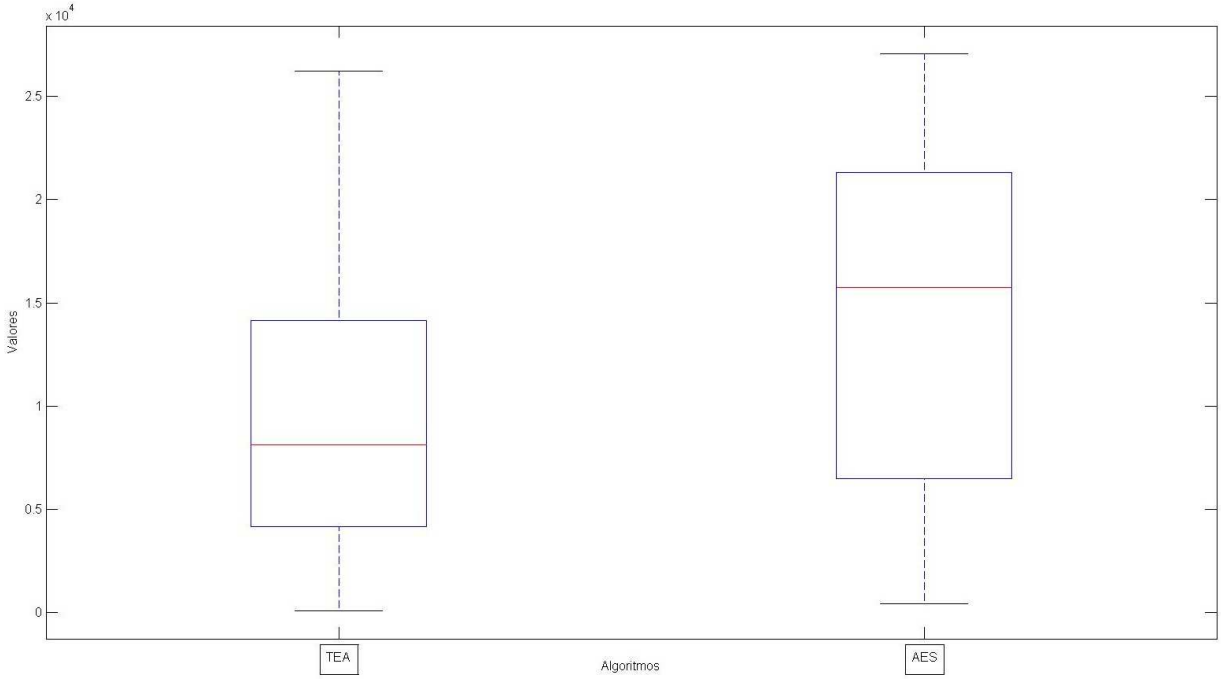


Figura 4. Tiempo medio requerido (ms) en pruebas con TEA y AES-128 para n-l=20 bits.

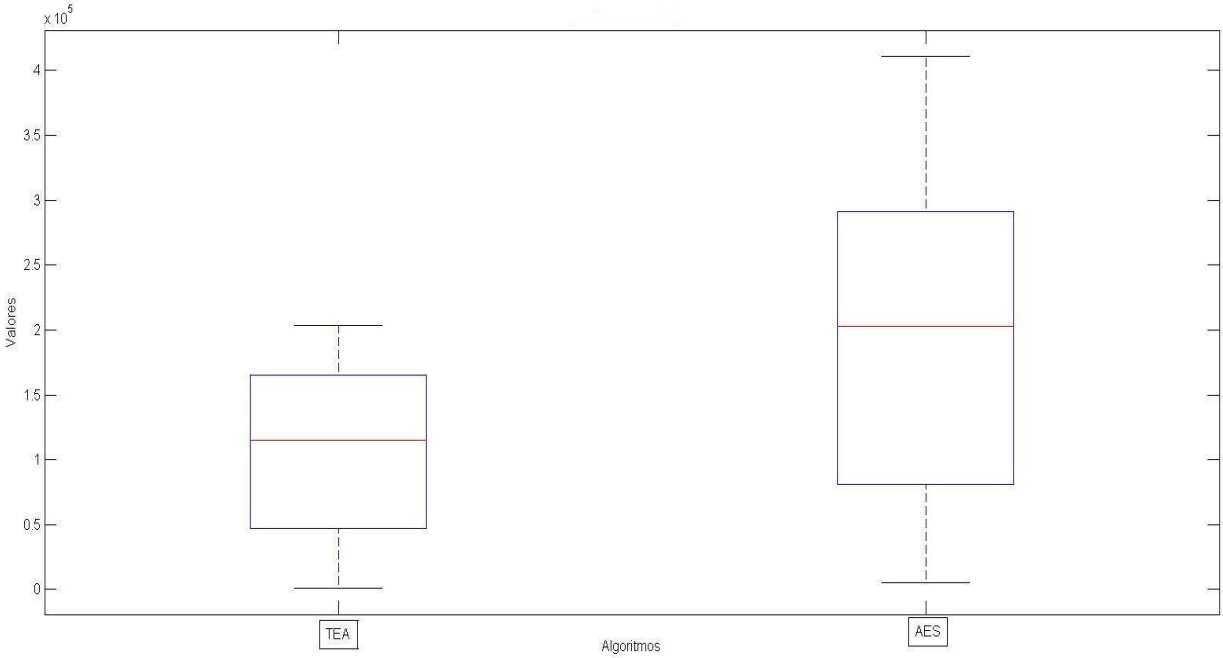


Figura 5. Tiempo medio requerido (ms) en pruebas con TEA y AES-128 para n-l=24 bits.

En la Figura 4 se puede ver como para el caso del algoritmo TEA la distribución es asimétrica positiva (la mediana se encuentra más cerca del límite inferior). Sin embargo, para el caso del AES ocurre al contrario, se trata de una distribución asimétrica negativa, puesto que la mediana se sitúa cerca del extremo superior.

En la gráfica mostrada en la Figura 5 se puede apreciar, que tanto para el caso del algoritmo TEA como para el AES, la distribución es asimétrica negativa, ya que la mediana se sitúa cerca del extremo superior.

En el caso de la siguiente gráfica (Figura 6), se muestra para el caso del TEA una distribución asimétrica negativa, ya que la mediana se sitúa cerca del extremo superior. Sin embargo par el AES la distribución es simétrica, la mediana se sitúa en el centro de la caja.

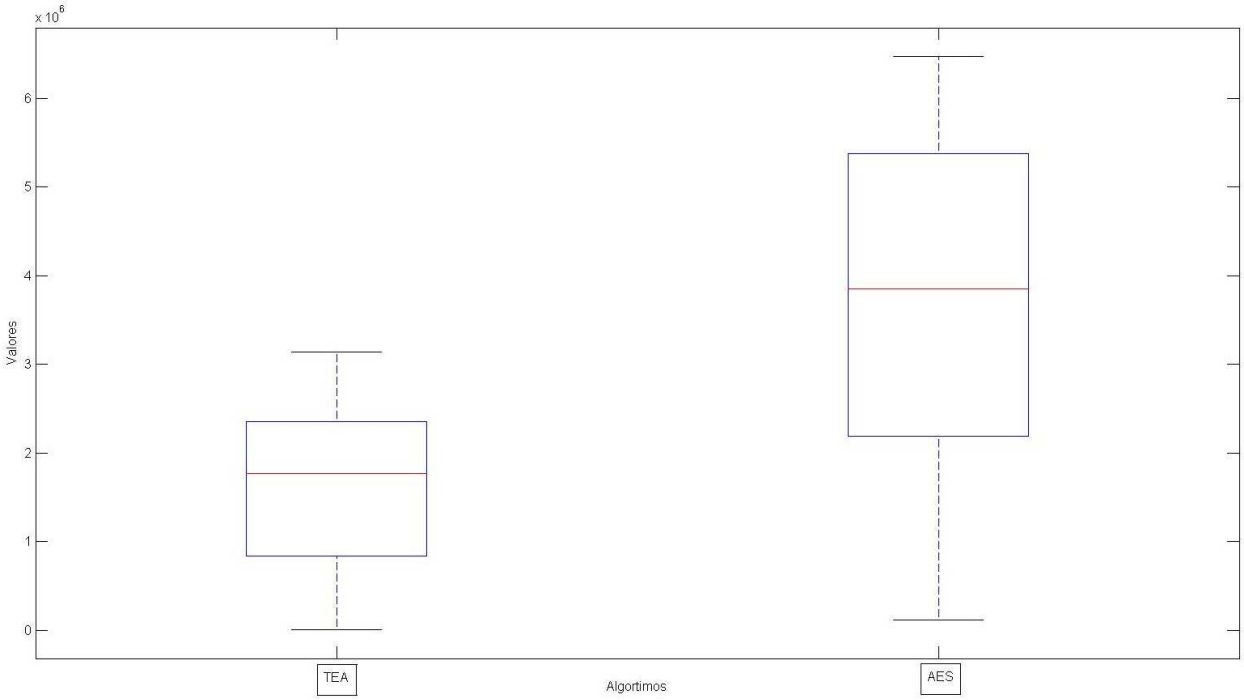


Figura 6. Tiempo medio requerido (ms) en pruebas con TEA y AES-128 para n-l=28 bits.

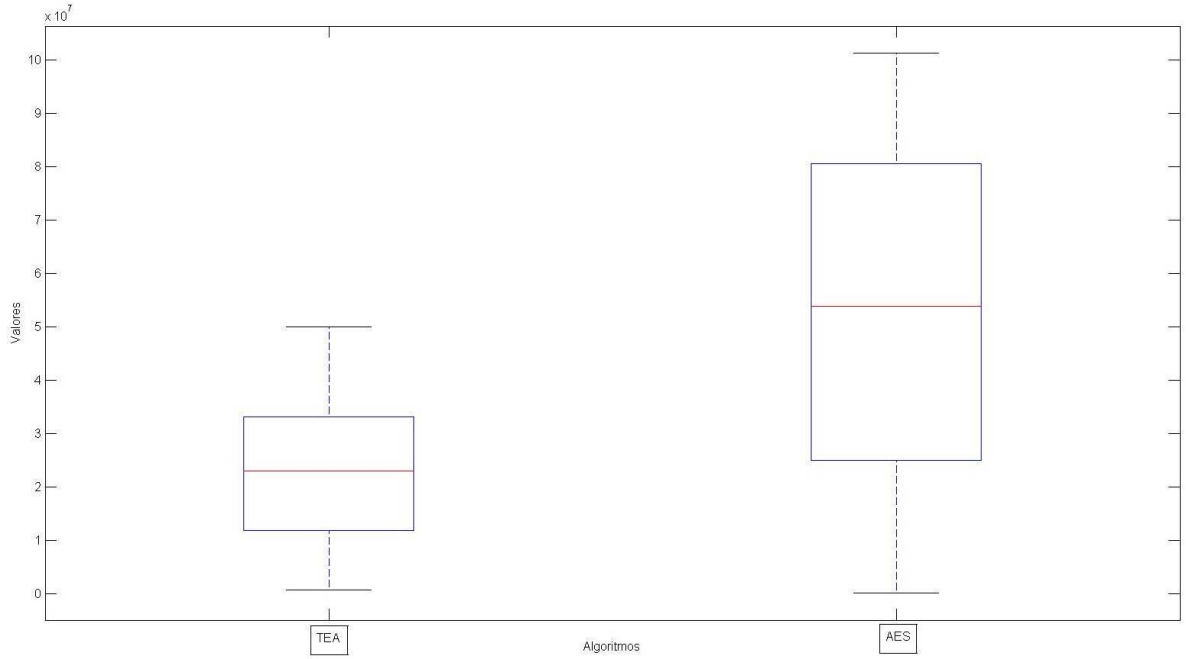


Figura 7. Tiempo medio requerido (ms) en pruebas con TEA y AES-128 para n-l=32 bits.

En este último caso (Figura 7) las dos distribuciones, la representada por el algoritmo TEA y la representada por el AES, son simétricas ya que en ambas la mediana se sitúa aproximadamente en el centro de la caja.

Si observamos en conjunto las figuras anteriores (Figuras 3, 4, 5 y 6), todas ellas tienen una característica común. En todas se puede apreciar como los datos para el algoritmo AES presentan mayor variabilidad que para el algoritmo TEA, puesto que la caja es más ancha.

Tras observar y analizar los tiempos de resolución del puzzle para cada uno de los dos algoritmos propuestos, así como analizar cada gráfica expuesta, la conclusión que obtenemos es la siguiente: el tiempo de resolución del puzzle que emplea AES-128 es sensiblemente superior al que emplea TEA además los resultados muestran una mayor variabilidad. Si se pusiera en práctica este protocolo en un sistema RFID en el que los lectores trabajaran constantemente leyendo un número elevado de etiquetas, dado que no podríamos distinguir entre lectores honestos y deshonestos, sería más funcional la implementación del puzzle con el algoritmo de cifrado TEA debido a que requiere menos tiempo para resolver el puzzle criptográfico.

Capítulo 5

Gestión del proyecto

En este capítulo se muestra la planificación que se ha realizado al principio del proyecto así como los resultados reales obtenidos.

5.1. Planificación del proyecto

Las tareas que se han propuesto para este proyecto son las siguientes:

Hito de Inicio del Proyecto

- **Fase de Análisis**
  - Estudio de la tecnología RFID: estudio en general de la tecnología RFID, analizando las amenazas a las que se ve sometida.
  - Estudio de puzzles criptográficos: Estudio del estado del arte de puzzles criptográficos y pruebas de esfuerzo.
  - Estudio de protocolos de autenticación RFID: Estudio de algunos protocolos de autenticación RFID y sus amenazas.
  - Estudio de algoritmos de cifrado: Estudio de los algoritmos de cifrado AES y TEA que posteriormente emplearemos en el protocolo.
  - Análisis de Requisitos: recopilación del catálogo de requisitos que tiene que cumplir el protocolo propuesto.
  - Reunión de Seguimiento Fase Análisis: Reunión con la tutora del Proyecto Fin de Carrera para verificar los puntos de la fase de análisis.

  Hito Fin Fase de Análisis

- **Fase de Diseño**
  - Diseño de Diagrama de Clases: Representación de la estructura del programa para su posterior implementación.
  - Reunión Seguimiento Fase Diseño: Reunión con la tutora del Proyecto Fin de Carrera para verificar los puntos de la fase de diseño.

  Hito Fin Fase de Diseño
- **Fase de Implementación**
  - Algoritmo de Cifrado TEA: Implementación de la clase que permite el cifrado y el descifrado con el algoritmo TEA.

- Algoritmo de Cifrado AES: Implementación de la clase que permite el cifrado y el descifrado con el algoritmo AES.
- Función resumen SHA-2: Implementación de la clase que permite la encriptación con la función resumen SHA-2 de 512 bits.
- Implementación clase Puzzle: Implementación de la clase que va a representar el puzzle criptográfico.
- Implementación clase EntradaPuzzle: Implementación de la clase con los datos necesarios para generar el puzzle criptográfico.
- Desarrollo de clases auxiliares: Implementación de clases auxiliares necesarias para la aplicación como: la generación de números aleatorios, cronómetro, tratamiento de arrays, suma or-exclusiva.
- Reunión Seguimiento Fase Implementación: Reunión con la tutora del Proyecto Fin de Carrera para verificar los puntos de la fase de implementación.

Hito Fin Fase de Implementación

- **Fase de Pruebas**
  - Pruebas experimentales: Pruebas experimentales del algoritmo propuesto con el fin de analizar los resultados.
  - Reunión Seguimiento Fase Pruebas: Reunión con la tutora del Proyecto Fin de Carrera para analizar los resultados obtenidos en la fase de pruebas.

Hito Fin Fase de Pruebas

- **Documentación**
  - Confección Memoria: Tarea que se realiza al final de cada día de trabajo, y se completa tras la fase de pruebas en la que se recogen los aspectos más importantes de los estudios realizados durante cada jornada.
  - Redacción completa Memoria: Tras finalizar las pruebas experimentales se incluyen en la memoria los resultados y análisis de las mismas. Se completa la memoria con los puntos que faltan.

Hito Fin Fase de Documentación

- **Presentación del Proyecto**: Preparación de la Presentación del Proyecto Fin de Carrera ante el Tribunal de profesores. Esta etapa incluye la preparación de

diapositivas y el estudio de toda la documentación obrante en la Memoria del Proyecto.

Hito Fin del Proyecto

5.1.1.Planificación estimada

Se han estimado 185 días para la realización del proyecto. La fecha de comienzo fue el lunes 26 de septiembre de 2011.

Realizando 5 jornadas laborales de 2'5 horas cada una de ellas, de 18:30 a 21:00, a lo largo de la semana, se ha empleado un esfuerzo continuo del 90% en las tareas correspondientes al ciclo de vida (análisis, diseño, implementación y pruebas), y un 10% de esfuerzo en cada jornada, para la fase de documentación.

La fecha estimada de finalización del proyecto ha sido la del viernes 8 de junio de 2012.

En la siguiente figura se puede ver el diagrama de Gantt de la planificación del proyecto.

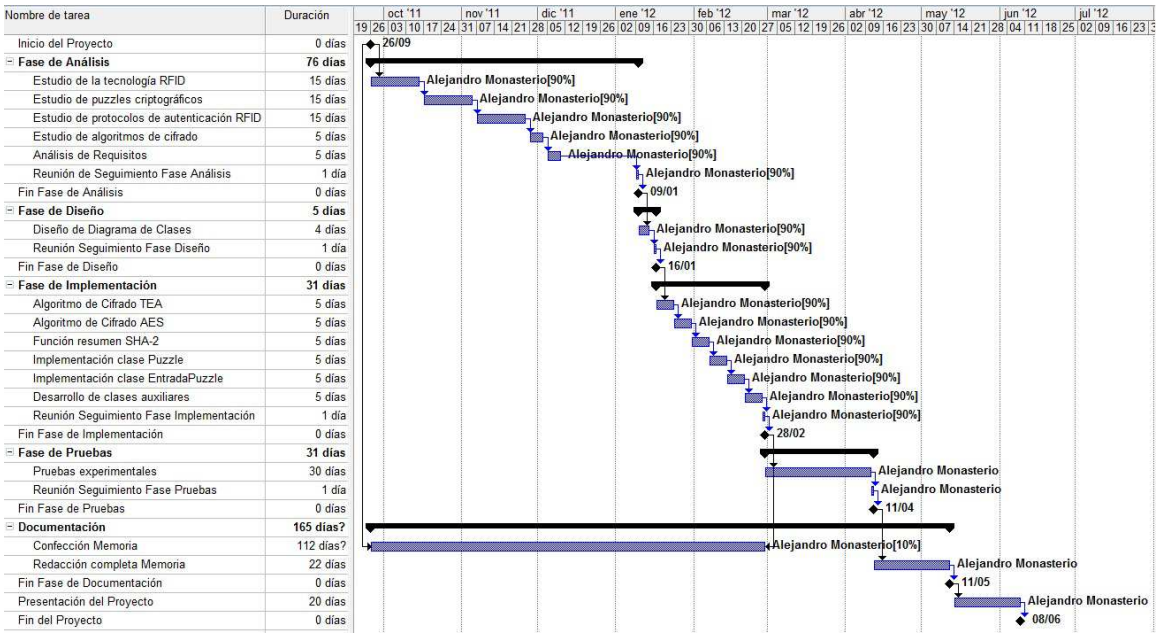


Figura 8. Microsoft Project: Diagrama de Gantt planificación estimada del proyecto.

5.1.2.Planificación real

Tras la realización del proyecto, y llevar un seguimiento exhaustivo del mismo, la duración del proyecto se ha alargado hasta los 308 días, con una desviación de 123 días al final del proyecto.

La fecha real para la finalización del proyecto, es decir, par la presentación del mismo ha sido el martes 27 de noviembre de 2012.

En la siguiente figura se puede ver el diagrama de Gantt de la planificación real del proyecto.

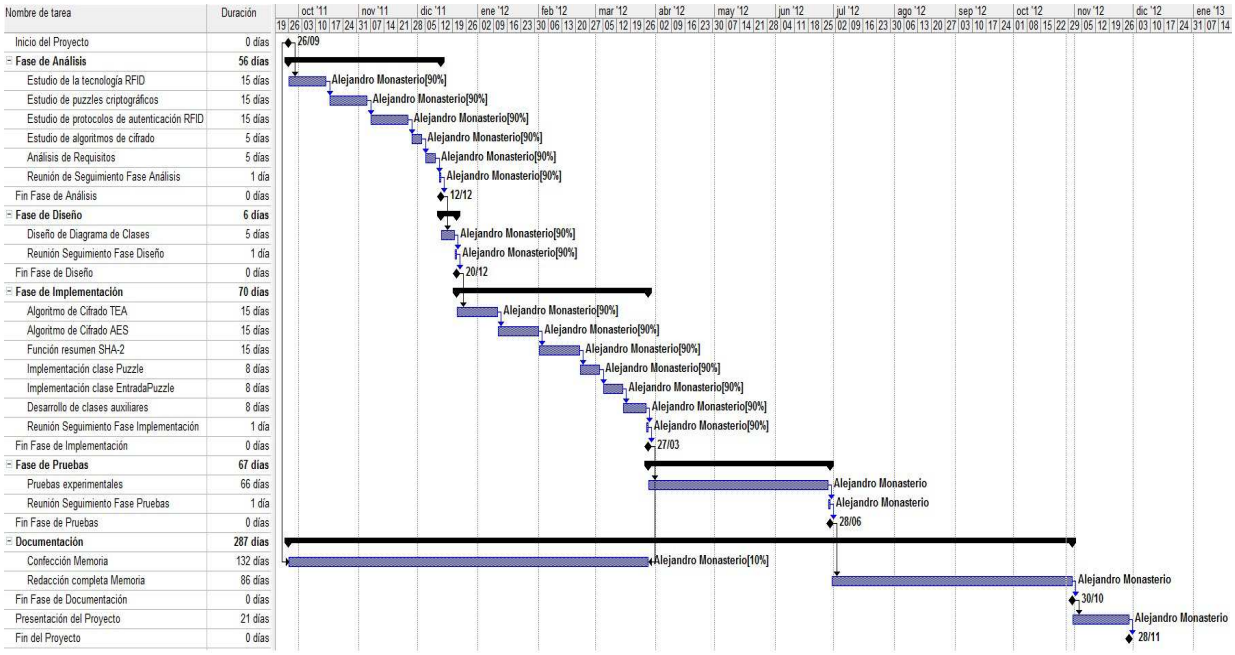


Figura 9. Microsoft Project: Diagrama de Gantt planificación real del proyecto.

5.1.3. Análisis de planificación del proyecto.

El principal motivo de la desviación entre la planificación inicial y la planificación real es que la fase de pruebas duró más de lo previsto (36 días más). También habría que añadir que se tuvo que compaginar el desarrollo del proyecto fin de carrera con el trabajo, habiendo días en los que resultó imposible cumplir con el horario establecido. Otro hecho a destacar es que la redacción completa de la memoria duró más de lo esperado. Al prolongarse la duración del proyecto, hubo que contemplar las vacaciones de verano, lo que sumaron tres semanas más de retraso.

Por estos motivos la desviación del proyecto es de 123 días, que impidieron que se pudiera presentar el proyecto fin de carrera a principios de junio y se tuviera que retrasar a finales de noviembre.

5.2. Medios técnicos empleados

En este aparatado se muestras los distintos medios y herramientas, tanto de hardware como de software que se han utilizado durante todo el ciclo de vida de este Proyecto Fin de Carrera.

5.2.1. Medios hardware

A continuación se muestra una tabla con los distintos dispositivos hardware que se han necesitado para realizar el proyecto.

Tabla 2. Medios hardware.

Tipo	Nombre	Distribuidor
Ordenador	Ordenador HP con Windows XP	Hewlett-Packard
Ordenador	Ordenador Sony con Windows 7	Sony
Impresora Multifunción	Impresora HP Deskjet 3050	Hewlett-Packard

5.2.2.Medios software

A continuación se muestra una tabla con los distintos dispositivos software que se han necesitado para realizar el proyecto.

Tabla 3. Medios software.

Tipo	Nombre	Distribuidor
Sistema Operativo	Windows XP Home Edition Service Pack 3	Microsoft
Sistema Operativo	Windows 7 Home Premium	Microsoft
Navegador	Firefox	Mozilla
Entorno de Programación	Eclipse Java EE IDE for Web Developers	Eclipse
Lenguaje de Programación	Java 1.6	Oracle Java
Bloc de Notas	Notepad ++	Notepad Team (GNU)
Procesador de Texto	Office - Word 2003	Microsoft
Planificador	Office - Project 2003	Microsoft
Tabla de cálculo	Office - Excel 2003	Microsoft
Diagramas UML	ArgoUML	ArgoUML

5.2.3.Lenguajes de programación

Para la creación de la aplicación se ha utilizado el lenguaje de programación Java (1.6) dado que contiene un gran número de librerías y código de libre acceso, lo cual ha facilitado la implementación

Un factor importante para la elección de Java fue el conocimiento del mismo, así como la gran comunidad de desarrolladores que dispone, lo cual es un punto de vista positivo para poder obtener ayuda, ya sea a través de otros desarrolladores o a través de guías oficiales.

5.3. Gestión económica

Para realizar la gestión económica del proyecto, se ha seguido la plantilla guía que ofrece la universidad.

Dentro de la gestión económica del proyecto se distinguen tres tipos de gastos:

- Recursos humanos: Personas físicas que participan en la elaboración de este proyecto fin de carrera.
- Equipos: Recursos materiales utilizados para la confección del proyecto.
- Otros gastos: Listado de otros gastos vinculados al proyecto.

Para determinar el coste total del proyecto, se determinará el sumatorio de todos los tipos de gastos.

5.3.1. Costes estimados

En este apartado se representa el presupuesto inicial del proyecto.

Los costes de los recursos humanos durante todo el proyecto han sido los siguientes:

Tabla 4. Planificación de coste de recursos humanos.

Puesto	Coste/hora	Total horas	Coste total
Alejandro Monasterio (Ingeniero Técnico)	10,00 € / hora	462,5	4625 €

Los costes de los equipos utilizados durante todo el proyecto han sido los siguientes:

Tabla 5. Planificación de coste de los equipos utilizados en el proyecto.

Descripción	Coste (€)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable (€)
Ordenador HP con Windows XP	510,95	100	8,5	24	180,96 €
Ordenador HP con Windows 7	750	100	7	24	218,75 €
Paquete Microsoft Office 2003	320,00	100	8,5	24	113,33 €
Impresora HP	64,95	100	8,5	24	23 €
				Total	536,04 €

Otros gastos del proyecto han sido estos:

Tabla 6. Planificación de coste de otros gastos del proyecto.

Descripción	Coste imputable
Material de Oficina	140 €
Consumibles	50 €
Total	190 €

El resumen del presupuesto es el siguiente:

Tabla 7. Planificación del coste total del proyecto.

Costes totales	
Personal	4.625 €
Amortización	536,04 €
Costes de funcionamiento	190 €
Costes indirectos	1.070,21 €
Total	**6.421,25 €**

5.3.2.Costes reales

En este apartado se reflejan los costes finales del proyecto.

Los costes de los recursos humanos durante todo el proyecto han sido los siguientes:

Tabla 8. Coste final de recursos humanos.

Puesto	Coste/hora	Total horas	Coste total
Alejandro Monasterio (Ingeniero Técnico)	10,00 € / hora	770	7.700 €

Los costes de los equipos utilizados durante todo el proyecto han sido los siguientes:

Tabla 9. Coste final de los equipos utilizados en el proyecto.

Descripción	Coste (€)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable (€)
Ordenador HP con Windows XP	510,95	100	14	24	298,05 €
Ordenador HP con Windows 7	750	100	11	24	343,75 €
Paquete Microsoft Office 2003	320,00	100	14	24	186,67 €
Impresora HP	64,95	100	14	24	37,89 €
				Total	866,36 €

Otros gastos del proyecto han sido estos:

Tabla 10. Coste final de otros gastos del proyecto.

Descripción	Coste imputable
Material de Oficina	140 €
Consumibles	50 €
Total	190 €

El resumen del coste total es el siguiente:

Tabla 11. Coste total final del proyecto

Costes totales	
Personal	7.700 €
Amortización	866,36 €
Costes de funcionamiento	190 €
Costes indirectos	1.751,27 €
Total	**10.507,63 €**

5.3.3.Análisis de costes del proyecto

Según se puede apreciar en los costes presupuestados el total daba como resultado 6.421,25 euros mientras que en el final el valor del coste del proyecto es de 10.507,63 euros.

Esto supone que la desviación del presupuesto, en estos 123 días extra ha sido de (10.507,63€ - 6.421.25€) =4086,38 euros

La desviación económica del proyecto en datos porcentuales ha sido de un 63.64% más del presupuesto inicial por los motivos mencionados en el apartado de *Análisis de la planificación del proyecto.*

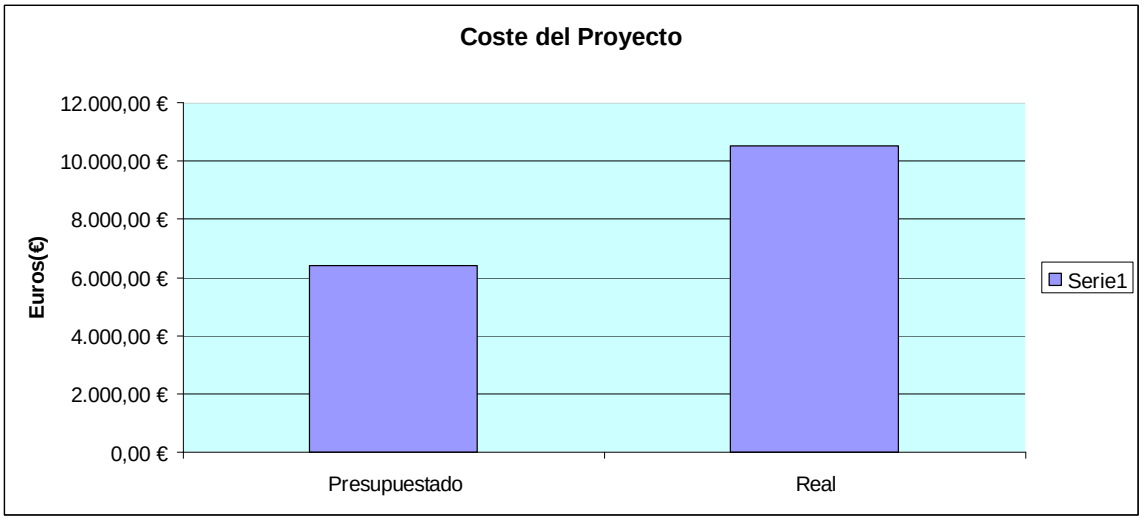


Figura 10. Grafica comparativa entre el coste presupuestado y el coste real del proyecto.

Capítulo 6

Conclusiones y Líneas futuras

6.1. Conclusiones

Tal y como se indicó en el capítulo introductorio de este trabajo, el objetivo del presente proyecto es tratar de resolver uno de los problemas de seguridad más importantes que amenazan a los sistemas con tecnología RFID, limitar a lectores deshonestos el acceso a la información confidencial almacenada en las etiquetas RFID. Este objetivo se ve satisfecho con el protocolo propuesto en el capítulo 3.

Con el uso de puzzles criptográficos en el proceso de autenticación se consigue mantener la privacidad, ya que el identificador de la etiqueta no se envía nunca en claro al lector, y este último no lo descubrirá hasta que resuelva el puzzle correctamente. Es cierto que este protocolo es vulnerable para el caso de una ínfima cantidad de etiquetas, puesto que un lector deshonesto podría descubrir dicha información tras consumir una cantidad significativa de tiempo. Sin embargo esta no sería nuestra meta, ya que el objetivo principal del protocolo propuesto es evitar la revelación de la información privada de un gran número de etiquetas, en cuyo caso la tarea resultaría inviable ya que el tiempo invertido sería excesivo. Esto último queda demostrado en le capítulo 4 del presente documento. Tras realizar pruebas experimentales con los dos algoritmos de cifrado propuestos (AES y TEA) comprobamos que los tiempos empleados para resolver los puzzles generados resultarían muy elevados para lectores ajenos al sistema que pretenden obtener información confidencial de una gran cantidad de etiquetas. Así mismo, tras analizar estos tiempos, concluimos que el tiempo de resolución del puzzle que emplea AES-128 es sensiblemente superior al que emplea TEA, además muestran una mayor variabilidad. Si se implementara este protocolo en un sistema RFID real, en el que participaran un número elevado de etiquetas, puesto que no se podría distinguir entre lectores autorizados y no autorizados, sería más funcional la generación de los puzzles empleando el algoritmo de cifrado TEA debido a que se requiere menos tiempo para resolver el puzzle criptográfico.

Si comparamos los resultados obtenidos en este proyecto con los obtenidos en el trabajo [POP+12], vemos que los tiempos obtenidos en el nuestro son mayores. Esto no depende sólo de las características del PC empleado al realizar las pruebas, sino que también

influye que en nuestro trabajo hemos empleado Java y en el trabajo mencionado emplean C, que tiene mejores tiempos de ejecución.

6.2. Líneas futuras

Como posibles trabajos futuros, se puede considerar trabajar en la mejora del rendimiento y la seguridad del protocolo.

En lo que respecta a la seguridad se podría desarrollar un algoritmo que permitiera actualizar la clave cada vez que una etiqueta sea leída, de este modo se conseguirían proteger las comunicaciones pasadas. Esta actualización se podría hacer en dos momentos. O bien, una vez que la etiqueta envía el puzzle, o una vez que el lector haya resuelto correctamente el puzzle. La segunda opción sería mejor cuando se necesite una conexión permanente a la base de datos. Sin embargo, para nuestro protocolo sería más eficaz la primera opción ya que impide en gran medida la posibilidad de un ataque satisfactorio de un oponente escuchando en el canal.

Para mejorar el rendimiento se podría investigar en alguna forma sencilla de poder distinguir a los lectores pertenecientes al sistema (autorizados) de los ajenos al mismo (atacantes), de esta forma se podría enviar un mayor número de bits de la clave a los lectores autorizados y así éstos podrían resolver el reto invirtiendo menos recursos.

También sería recomendable el realizar las pruebas en un entorno RFID real, puesto que éstos tienes mayores restricciones de hardware que el ordenador dónde se han realizado los experimentos.

Otro desarrollo interesante sería, en lugar de utilizar el paquete javax.crypto de Java, el implementar el algoritmo AES ayudado por librerías de código abierto que nos proporcionan el código fuente del algoritmo. De esta forma se podría analizar y razonar también la complejidad del AES en sus operaciones.

Glosario

Acrónimo	Definición
RFID	*Radio Frecuency Identification*
TEA	*Tiny Encryption Algorithm*
AES	*Advanced Encryption Standard*
SRAC	*Semi-Randomized Access Control*
DoS	*Denial of Service*
POW	*Proof of Work*
WSBC	*Weakly Secret Bit Commitment*
SHA	*Secure Hash Algorithm*
UML	*Unified Modeling Language*

Referencias

[POP+12] Pedro Peris-Lopez, Agustin Orfila, Esther Palomar y Julio C. Hernandez-Castro. *A secure distance-based RFID identification protocol with an off-line back-end database.* Personal and Ubiquitous Computing Volume 16, Issue 3 , pp 351-365 Marzo, 2012.

[Syv98] Syverson P. *Weakly secret bit commitment: applications to lotteries and fair exchange.* In: Proceedings of the 11th IEEE computer security foundations workshop, pp 2–13, 1998.

[Mer78] Merkle RC. *Secure communications over insecure channels.* Commun ACM 21(4):294–299, 1978.

[DN92] Dwork C, Naor M. *Pricing via processing or combatting junk mail.* In: Proceedings of the 12th annual international cryptology conference on advances in cryptology, Springer, Berlin, pp 139–147, 1992.

[RSA96] Rivest RL, Shamir A, Wagner DA. Technical report mit/lcs/tr-684. time-lock puzzles and timed-release crypto. Technical report, 1996.

[Bac02] Back A. *Hashcash. A denial of service counter-measure.* Technical report. August 2002. Disponible [Internet]: <http://www.hashcash.org/hashcash.pdf>, [19 de diciembre de 2012]

[FM97] Franklin M, Malkhi D. *Auditable metering with lightweight security.* In: Proceedings of financial cryptography, vol 1318, Springer, Berlin, pp 151–160, 1997.

[NP98] Naor M, Pinkas B. *Secure and efficient metering.* In: Proceedings of advances in cryptology (EUROCRYPT'98), vol 1403, Springer, Berlin, pp 576–590, 1998.

[JB99] Juels A, Brainard J. *Client puzzles: a cryptographic defense against connection depletion attacks.* In: Proceedings of the networks and distributed security systems, California, pp 151–165, 1999.

[JJ99] Jakobsson M, Juels A. *Proofs of work and bread pudding protocols.* In: Proceedings of the IFIP TC6/TC11 joint workingconference on secure information networks, Kluwer, Dordrecht, pp 258–272, 1999.

[PAL12] Esther Palomar, José M. de Fuentes, Ana I. González-Tablas, Almudena Alcaide, *Hindering false event dissemination in VANETs with proof-of-work mechanisms*, Transportation Research Part C: Emerging Technologies, Volume 23, Pages 85-97, August 2012.

[MKB+01] Mankins D, Krishnan R, Boyd C, Zao J, Frentz M. *Mitigating distributed denial of service attacks with dynamic resource pricing.* In: Proceeidngs of the 17th annual conference on computer security applications, 2001.

[ABM+03] Abadi M, Burrows M, Manasse M, Wobber T (2003) *Moderately hard, memory-bound functions.* In: Proceedings of the 10th annual network and distributed system security symposium,pp 25–39, 2003.

[DGN03] Dwork C, Goldberg A, Naor M. *On memory-bound functions for fighting spam.* In: Proceedings of advances in cryptology, Springer, Berlin, pp 426–444, 2003.

[SL03] Serjantov A, Lewis S. *Puzzles in p2p systems*. In: Proceedings of the 8th CaberNet Radicals Workshop, 2003.

[LC04] Laurie B, Clayton R. *Proof-of-work proves not to work.* In: Proceedings of the 3rd workshop on the economics of information security, 2004.

[Bor06] Borisov N. *Computational puzzles as sybil defenses.* In: Proceedings of the 6th international conference on Peer-to-Peer computing, IEEE, pp 171–176, 2006.

[NLD08] Ning P, Liu A, Du W. *Mitigating dos attacks against broadcast authentication in wireless sensor networks.* ACM Trans Sensor Networks 4(1):1–35, 2008.

[BMM08] Burmester M, de Medeiros B, Motta R . *Robust, anonymous rfid authentication with constant key-lookup.* In: Proceedings of the 2008 ACM symposium on Information, computer and communications security, ACM, pp 283–291, 2008.

[NVR10] Narasimhan H, Varadarajan V, Rangan C. *Game theoretic resistance to denial of service attacks using hidden difficulty puzzles.* In: Proceedings of information security, practice and experience, vol 6047, Springer, Berlin, Lecture Notes in ComputerScience, pp 359–376, 2010.

[MGH] Joan Melià-Seguí, Joaquin Garcia-Alfaro y Jordi Herrera-Joancomartí. *Clasificación de las amenazas a la seguridad en sistemas RFID - EPC Gen2*. Disponible [Internet]: <http://joan.melia.cat/pubs/Melia-RECSI2010.pdf>[19 de diciembre de 2012]

[WN94] Wheeler, David J.; Needham, Roger M. *TEA, a Tiny Encryption Algorithm*. In: Lecture Notes in Computer Science (Leuven, Belgium: Fast Software Encryption: Second International Workshop) vol. 1008: pp. 363–366, 1994.

Anexo I

Las tablas a continuación recogen los tiempos en milisegundos obtenidos en la ejecución de 100 pruebas de esfuerzo empleando los algoritmos TEA y AES para diferentes tamaños de trapdoor.

Tabla 12. Datos obtenidos en milisegundos empleando el algoritmo TEA.

n-l=20	l=108	n-l=24	l=104	n-l=28	l=100	n-l=32	l=96
	4281		199610		677594		39498672
	4219		162000		919265		18435359
	4234		181766		942079		18859984
	2219		1344		721734		3347188
	4094		80484		2884578		42375922
	13093		161094		2675734		11601437
	9688		8703		365047		14884422
	5750		201687		1859844		4821703
	1375		63688		1011281		16936407
	6359		151844		3127235		12643312
	3938		173281		2456562		25238625
	5000		4156		1065672		29213328
	8093		198359		2421171		25382922
	7454		66094		302438		10543609
	1140		1688		794953		9870407
	1797		145578		2289532		50068797
	5250		53390		2956453		13146687
	11265		36375		3136172		6903109
	4907		50235		2331437		21375578
	796		30734		1849234		26243531
	6859		119735		2200360		16129563
	8516		82297		1616156		18831109
	9031		13656		234313		29803250
	703		76907		2765531		15674063
	11422		173796		509390		13518078
	23922		21235		2827735		37618890
	4000		170484		2156219		15512938
	7078		58500		1849719		20430218
	18657		87610		2551672		14849594
	14562		170359		2928094		2613328
	24078		4875		922109		9148797

9063	146781	1342969	33963172
24609	186500	569906	11900234
24641	156313	1919578	5344907
22797	88359	700922	29509406
219	32953	2807500	23694094
23437	88016	137218	32214329
19657	92453	1781203	10523844
4203	150750	1373547	24849828
14734	68156	2233313	11222188
8156	192563	9094	27981531
13391	183422	2442906	39141391
1047	192640	1378969	41084172
13734	32203	2112671	15921984
3360	117344	3092047	5957219
9172	142578	19485	30523516
3171	163735	588343	45705234
63	139484	2550922	703938
11562	29547	622157	3549468
2657	83829	493250	27589391
78	149468	1658437	46891891
6390	146953	1845672	22922890
12250	142359	1624656	39682547
15079	177516	770485	33322484
5765	170266	30406	1578266
15781	130406	7109	49754766
2829	195468	1981750	3545453
2343	182860	2155219	43096797
11125	120531	2589172	32958422
10828	31531	3129297	27844859
2063	147031	2404250	41931484
5984	21797	1048328	19260625
1891	172578	449109	3191703
4453	171141	1882235	40107860
4734	7609	2368937	791062
7563	70688	102484	32003516
12828	12547	708360	23039265
500	85625	2026235	23242641
11797	161250	1104390	20912266
8546	32109	3052922	49676172
5766	166860	1801953	14343843
1250	38718	1828391	30786953
10625	37391	1065750	30239375
10391	61781	1763125	11301906
11218	203328	1088968	14489141
26235	173828	192344	24487875
19219	6063	2807250	11520375
6625	61781	1378703	49787875
6391	3813	2114875	35617046
10515	112812	604531	12276891
3641	94188	2581719	11947484
8359	103640	2115625	28561407
23157	41219	2796219	39230875
6375	119682	1491281	41090422

19032	172171	876485	2780015
24797	68969	91468	3441625
15156	30797	2883657	31992438
12859	181547	190234	44162797
20765	139859	2143219	16673078
4281	155547	2238328	34599469
14859	181813	1143828	42595015
15125	89500	2151812	24501500
19313	136156	2323250	30405344
19485	43422	1031390	9416906
3250	151625	2437235	12673329
16781	140219	1676593	19469000
5406	56250	1695500	25622500
13688	190343	2062438	7090672
2281	38172	1426547	34225516
22890	94406	874718	45412843

Tabla 13. Datos obtenidos en milisegundos empleando el algoritmo AES-128.

n-l=20	l=108	n-l=24	l=104	n-l=28	l=100	n-l=32	l=96
	796		63843		3347250		24553360
	17281		7469		1655296		82990719
	2438		261219		6473282		68248906
	797		38281		3428859		46536531
	17187		375156		2204781		93581891
	16859		337750		603532		91128032
	3219		274500		3342265		99528922
	12625		352079		615703		67872547
	24750		69140		4365375		5681156
	6125		200110		5581563		88295531
	14375		361093		5596031		100774859
	3391		58547		5790516		79524796
	18984		387110		1158375		92613984
	16672		173562		5250297		45225969
	22578		235094		4330125		97621078
	19859		107765		680453		94742016
	7032		117235		4368125		13238375
	26187		81031		5065703		49793140
	6016		387094		4519844		45932641
	10047		255844		5442125		58642469
	20062		56093		5710781		21519438
	19250		256750		6159063		60929718
	25281		54000		397875		66294860
	6594		7078		6100281		74886312
	26578		407328		1714250		69777907
	21063		148719		5018938		93702828
	10515		393546		5240687		92912203
	26407		64797		3059656		73572313
	578		45094		4418735		36855687
	18843		207859		6470812		45015297

6719	396516	1737453	43501891
7125	291547	5197282	1277609
7063	236375	5246656	14615938
17718	293281	2227516	657281
4219	297203	949734	44427266
10844	180641	593266	28341047
20641	90562	5559375	85668046
27078	286391	2924937	75921829
4375	286578	2224906	19753437
7984	294703	4806860	60790828
17953	130641	2706062	50127000
875	30953	1781890	24803344
7016	298094	6116938	50616703
21625	55000	2214547	81038469
1687	64859	574750	18833500
22250	316391	1341031	18613250
3172	120172	3284047	995141
20235	51250	2665656	67977156
9546	336859	1201906	31776563
10516	197922	6180407	22680828
16781	57156	5437359	80258406
24594	281922	4168109	99109344
22500	206016	3115704	90151453
23703	230781	5319968	49579313
4625	9687	3592594	20317000
10047	367391	117969	75336532
20547	185063	2464125	36514593
23062	283187	3909875	77700235
12938	256750	2167875	5029500
5062	62859	3566047	15839375
5641	396985	1486968	90831344
24016	21062	233375	77245110
1484	85828	3808750	51069687
24578	270438	6386781	8499453
20063	291094	2246016	21835094
24359	80937	1941344	28677140
4125	230531	5181047	59408281
24422	138250	5881718	86585485
3234	175657	4277188	69935422
2782	54797	5103500	25319078
23703	317984	5560656	93991406
14562	34250	4902828	36635781
17766	16297	1666813	13108922
9078	118140	1246656	51050329
14844	217063	6257266	25666406
17062	410578	6448062	90260500
6500	138797	3892735	100630703
11250	109891	4488047	90060531
22281	37703	422093	62860422
21891	315843	2632672	73631797
17297	259704	3109219	32949906
6453	182921	3605906	101342860
17422	155985	2547282	51491781

14828	377109	270890	20131734
20156	5031	6329281	94308781
25016	104375	5186032	172641
2047	209016	5224078	23252531
5094	82953	2418359	77130500
750	227469	5112641	83079047
12515	135797	5765109	31574265
391	351781	6224188	2831657
25328	309297	2707234	80361078
21531	86625	977641	66572188
12656	228250	452828	17634344
21844	222031	5758312	61834781
19922	166922	4804719	15066437
17484	372015	3151953	28974188
23625	38610	6104469	80931000
18750	205906	6300750	56181703
13391	205406	6120312	33880500