CRIPTOGRAFÍA LIGERA APLICADA A IOT

José Luis Caro Bozzino

TFM Master en Ciberseguridad

Contenido

1. Resumen 3
2. Introducción 4
3. Incidentes destacados 8
   3.1 Botnet Mirai 8
   3.2 Ataque a coches Jeep 9
   3.3 Vulnerabilidades en los monitores de frecuencia cardiaca para bebés Owlet 11
4. State of the art 13
   4.1 Cifrado en bloque 14
      4.1.1 Present 14
      4.1.2 SIMON 15
      4.1.3 SPECK 16
   4.2 Funciones Hash 18
      4.2.1 PHOTON 18
      4.2.2 QUARK 19
   4.3 Cifrado en flujo 19
      4.3.1 Grain 20
      3.3.2 Trivium 20
      4.3.3 MICKEY 21
   4.4 Cifrados MAC 21
      4.4.1 Chaskey 22
      4.4.2 LightMAC 23
5. LightCipher 24
   5.1 Ejecutando LightCipher en una instancia local 28
   5.2 Estructura del proyecto 31
6. Conclusiones 34
7. Bibliografía 36
   7.1 Recursos online 36
   7.2 Imágenes 37

1. Resumen

El presente Trabajo Final de Máster busca concienciar sobre los peligros del uso de dispositivos IoT que no han sido diseñados con la seguridad en mente, demostrando mediante eventos ocurridos en los últimos años las consecuencias que esto puede tener, así como enumerando algunos de los algoritmos criptográficos ligeros que podrían ser usados para evitarlo.

Para esto se tratarán algunos incidentes de seguridad relevantes en el ámbito de los dispositivos IoT, así como formas de mitigar los problemas derivados de la ausencia de cifrado de información.

Con el fin de poder ilustrar el funcionamiento de algunos de estos algoritmos, se ha desarrollado la aplicación web LightCipher, con la que poder cifrar y descifrar texto mediante las variantes más importantes de estos algoritmos, ya que existe una ausencia de herramientas criptográficas que ofrezcan la posibilidad de trabajar con algoritmos de criptografía ligera mediante una interfaz gráfica.

2. Introducción

El término IoT ó *Internet of Things* se acuñó en 1999, cuando Kevin Ashton, trabajador por aquel de entonces de la multinacional Procter & Gamble, quien ideó un sistema de etiquetas RFID por el cual los productos podrían ser localizados a lo largo del proceso de fabricación, venta y distribución[1].

Este término se utiliza para referirse a una red de dispositivos interconectados, que poseen un identificador único ó UID, y que pueden interactuar entre ellos sin necesitar de interacción humana, como pueden ser chips localizadores, relojes inteligentes, sensores de temperatura, etc… [2]

Según IoT Analytics, a día de hoy existen alrededor de 14.000.000.000 dispositivos IoT, y se espera que para 2025 esta cifra haya crecido hasta los 27 billones (americanos) de dispositivos conectados (Figura 1).

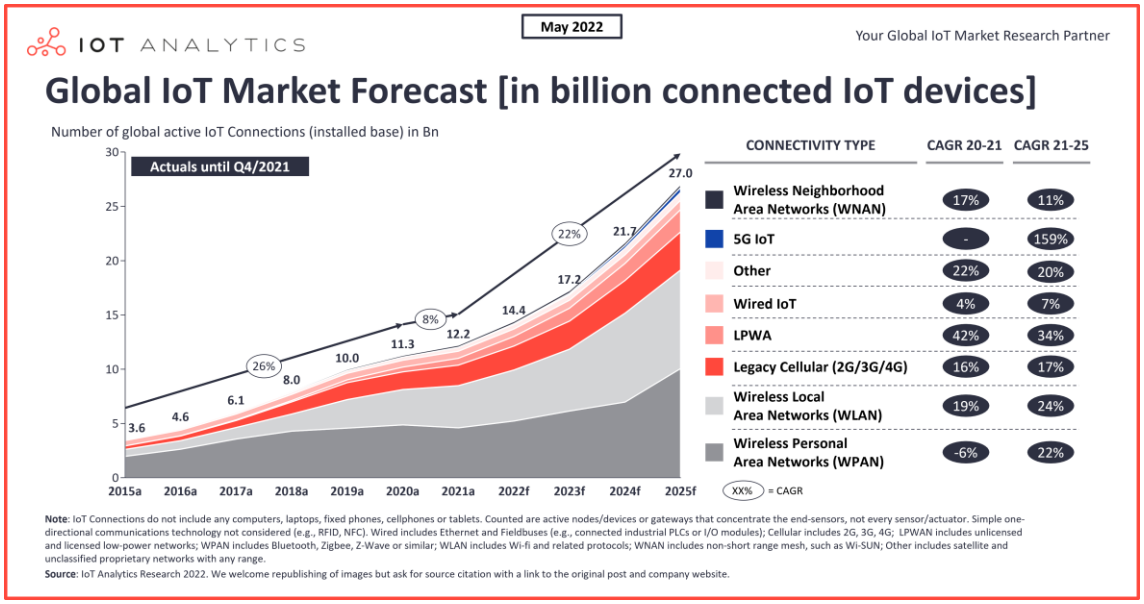


Figura 1. Previsión de dispositivos IoT en el mercado entre 2015-2025

Este volumen de dispositivos interconectados, sumado a que cada vez más objetos que utilizamos diariamente han pasado a ser dispositivos IoT (relojes, neveras, cafeteras, etc…) , hace que las brechas de seguridad que puedan existir en estos puedan acarrear consecuencias más graves, que van de fugas de información, secuestros de vehículos a directamente convertir nuestro

sistema doméstico de seguridad en una cámara que emite 24 horas al día en abierto para cualquier atacante que decida interceptarla.

En el siguiente capítulo se tratarán algunos de estos sucesos y sus repercusiones, demostrando que estos problemas no solo se dan en dispositivos de fabricación doméstica o de pequeñas marcas, sino incluso en aquellos fabricados por empresas más conocidas y que consumimos de forma inconsciente al relacionar mentalmente su imagen con productos de calidad y seguros, a pesar de que en muchos casos se ha demostrado no ser así, como se expondrá en futuros capítulos.

Como es lógico, la irrupción de estos aparatos en el mercado, así como su explosiva expansión, han venido acompañadas de nuevas formas de realizar ciberataques, dando lugar a problemas de seguridad que no existían antes, y unas cifras de dispositivos de uso diario que son vulnerables a ataques básicos que no hace más que escalar.

La fundación OWASP ha creado y difundido un Top 10 [3] de las mayores amenazas y vulnerabilidades más comunes en dispositivos IoT (figura 2), que contempla los siguientes problemas:

1. Contraseñas débiles ante ataques de fuerza bruta, o en muchos casos, directamente *hardcodeadas* en el código del dispositivo. En muchos casos estas contraseñas directamente no son modificables por el usuario o no se le insiste en la importancia de cambiarlas.
2. Los servicios en internet de estos dispositivos muchas veces no se securizan, haciendo que el atacante pueda acceder al dispositivo de forma remota e interactuar con él a través de su endpoint.
3. En muchos casos, las propias interfaces gráficas que deberían ser sencillas y cómodas para el usuario final son tan básicas que no realizan correctamente las tareas de autenticación o no filtran las entradas o salidas de información.
4. Muchos de estos dispositivos no incluyen mecanismos de validación de firmware, lo que permite que un atacante pueda instalar versiones no firmadas que comprometan el sistema operativo y el dispositivo en sí.
5. Estos dispositivos en muchos casos son difíciles de actualizar y mantener, haciendo que, en algunos casos, tras descubrir una vulnerabilidad a nivel de software o hardware, sea muy complicado poder parchear o mitigar estos problemas.
6. La protección de datos privados es otro problema común en el mundo de los dispositivos IoT, ya que muchos de estos manejan altas cantidades de información sensible, lo que sumado al resto de problemas que hemos visto, puede causar fugas de información con nefastas consecuencias.

7. La ausencia de mecanismos de cifrado a la hora de transmitir la información hace que muchos dispositivos simplemente la envíen en claro, facilitando a un atacante interceptar estas transacciones y acceder fácilmente a datos privados.
8. Muchos dispositivos no permiten una modificación de su configuración por parte del usuario final, lo que significa que, si la configuración inicial del dispositivo se ve comprometida, el usuario no pueda hacer nada por modificarla ni mitigar el impacto de la vulnerabilidad.
9. Dentro de un ecosistema IoT a veces es muy complicado poder securizar todos los dispositivos que lo conforman, haciendo que puedan darse vulnerabilidades que se extiendan desde los dispositivos más básicos a todos los demás que se encuentran interconectados dentro de una red Wi-Fi por ejemplo.
10. A nivel de hardware, en muchos casos estos dispositivos siguen incluyendo puertos para depuración accesibles con solo quitar un tornillo, y demás elementos que hacen que un atacante pueda adquirir un dispositivo, desmontarlo y analizarlo fácilmente en busca de vulnerabilidades que usar en futuros ataques.

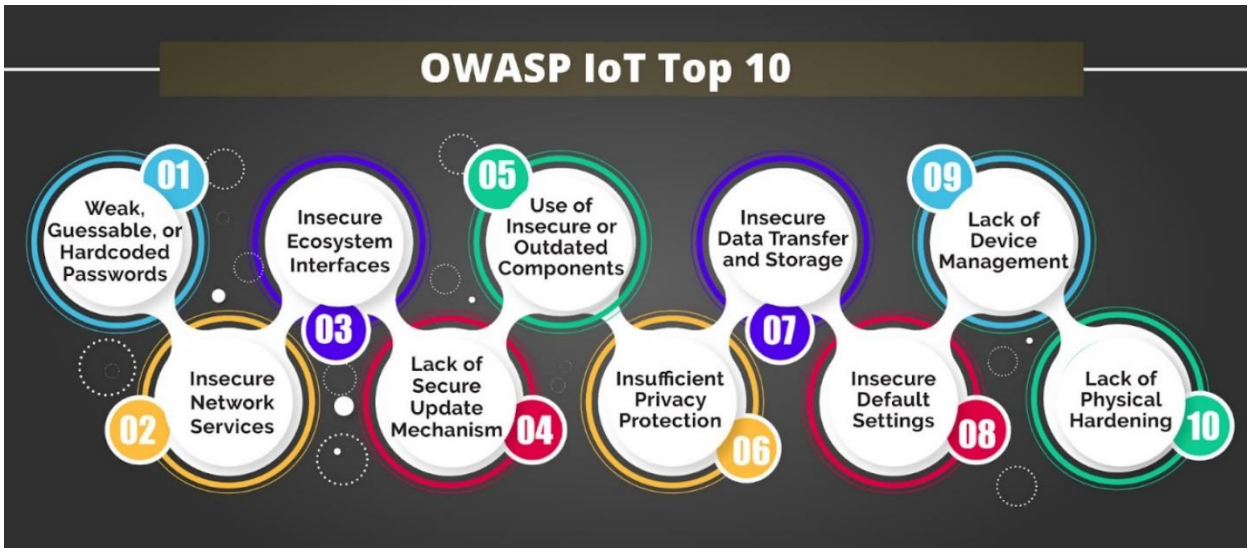


Figura 2. Top 10 vulnerabilidades IoT según OWASP

Los dispositivos IoT deben trabajar bajo la premisa de ser ligeros y no sacrificar a cambio su rendimiento, lo que crea un caldo de cultivo perfecto para que las empresas encargadas de diseñarlos, en muchos casos ignoren el aspecto de la seguridad o no lo tomen tan en cuenta como deberían, lo que ocasiona dispositivos que transmiten información en abierto, cifrados anticuados y poco efectivos, ecosistemas domésticos que permiten propagar ataques desde dispositivos básicos a los más sofisticados…

Sin embargo, como se tratará más adelante en este trabajo en el apartado 4, existe actualmente una interesante corriente de expertos en ciberseguridad que día a día se esfuerzan en investigar, diseñar e implementar algoritmos criptográficos ligeros orientados a estos dispositivos, que, si bien en muchos casos son conceptos en fase aún muy temprana, son una línea de investigación cada vez más necesaria e interesante, sobre todo para aquellos involucrados en el diseño de estos dispositivos.

La criptografía ligera es una vertiente que tiene como objetivo el desarrollo de algoritmos criptográficos lo suficientemente livianos para poder aportar una capa de seguridad robusta sin repercutir de forma negativa en el rendimiento. Está muy relacionada con la seguridad en IoT, ya que estos dispositivos suelen contar con procesadores menos potentes que permitan abaratar costes y tamaño en los dispositivos.

La correcta implementación de algoritmos de criptografía ligera en dispositivos IoT podría ser muy útil para mitigar los problemas ocasionados por ausencia de cifrado que hemos visto en el Top 10 anterior.

3. Incidentes destacados

Para ponernos en contexto sobre la importancia de aplicar medidas de ciberseguridad a la hora de interactuar con dispositivos IoT, comentaremos una serie de incidentes de seguridad ocurridos en distintas partes del mundo, que ponen sobre la mesa las nefastas consecuencias que puede acarrear el ignorar un aspecto tan importante como es la seguridad en estos dispositivos de uso diario.

3.1 Botnet Mirai

El caso de la botnet Mirai es uno de los más conocidos y al mismo tiempo uno de los más inquietantes.

Mirai se trata de un malware cuyo funcionamiento se basa en realizar amplios barridos de direcciones IP en busca de dispositivos (principalmente IoT) vulnerables a ataques de fuerza bruta, utilizando para ello un diccionario de contraseñas.

Una vez un dispositivo es infectado por Mirai, este no cambia su comportamiento, pero alberga el malware "latente", esperando órdenes, lo que hace que en el momento en el que el atacante con acceso a esta botnet lo desee, pueda utilizar de forma simultánea millones de dispositivos para ejecutar ordenes, pudiendo realizar ataques DDoS desde todo el mundo (Figura 3).

El código de este malware se ha publicado en foros de hacking en numerosas ocasiones, y se ha vuelto bastante sencillo de encontrar.

Esta pieza fue construida por Paras Jha Fanwood, Josiah White y Dalton Norman, quienes fueron sentenciados a servicio comunitario y a pagar una indemnización [4].

Curiosamente, el malware Mirai tiene una serie de tablas configuradas con máscaras de red a las que no infecta, entre las cuales se encuentran las pertenecientes al Servicio Postal de Estados Unidos, el Departamento de Defensa o IANA.

Uno de los ataques DDoS más destacados en los que se empleó esta botnet fue el llevado a cabo el 21 de Octubre de 2016, que tuvo como objetivo al proveedor de servicios DNS Dyn, y

que causó que algunas webs tan importantes como Netflix, GitHub o Twitter fueran inaccesibles.[5]

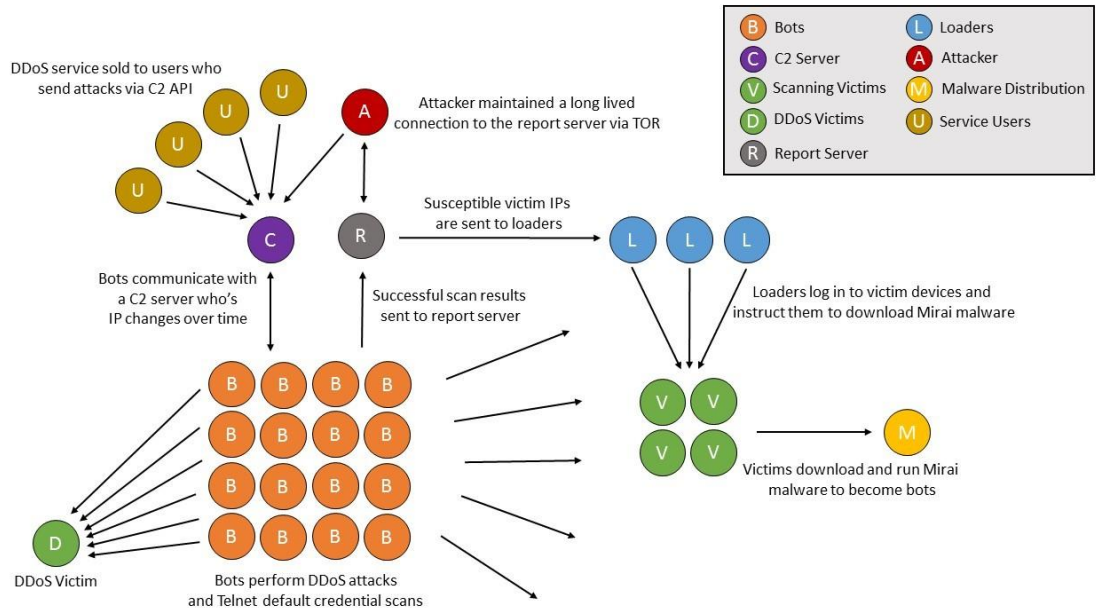


Figura 3. Estructura de la Botnet Mirai

3.2 Ataque a coches Jeep

En el año 2015, durante la feria sobre ciberseguridad y hacking Black Hat USA, se comprobó que ciertos vehículos Jeep podían ser hackeados para poder manejarlos de forma remota, lo que podría ocasionar que un atacante provocase un accidente al secuestra un vehículo durante un trayecto.

Los autores; Charlie Miller y Chris Valasek, explicaron y demostraron el ataque que habían elaborado, impactando al público con la sencillez de este.

Este ataque se basa en conectarse a la señal WiFi del vehículo, que en circunstancias normales de activaba mediante una suscripción contratada por el propietario del vehículo.

La contraseña de esta señal consistía en una clave generada a partir de la fecha de puesta en marcha del vehículo y su centralita, con precisión de segundos.

Este método de generación de claves hace que un atacante que conozca el año de fabricación del vehículo y acierte el mes, solo tendría que enfrentarse a unos 15 millones de combinaciones posibles; una cifra bastante pequeña a la hora de llevar un ataque de fuerza bruta con las herramientas adecuadas.

Con ese conocimiento, el siguiente paso fue tratar de acelerar el proceso para evitar tener que mantenerse cerca del vehículo durante el tiempo que tarda en realizarse el ataque de fuerza bruta.

Fue entonces cuando Charlie y Chris descubrieron que la contraseña se generaba antes de que la centralita configurase la fecha y hora reales, en su lugar usando la fecha del sistema y sumándole los segundos que tardaba en inicializarse la centralita.

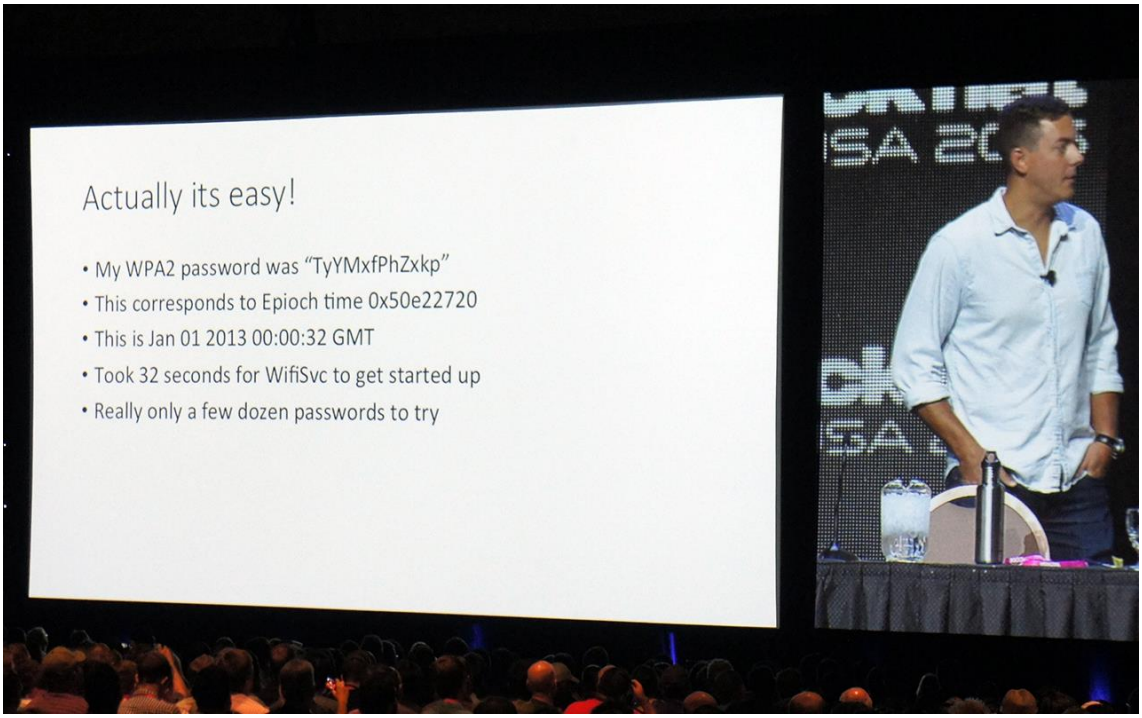


Figura 4. Ejemplo de generación de contraseña

Con esto lograron el acceso a la unidad central del vehículo y pudieron extender el ataque al reproductor multimedia del vehículo, lo que permitía alterar la radio, así como el reproductor de música o el volumen de éste.

Una vez consiguieron interceptar el reproductor multimedia, pudieron generar un payload camuflado como actualización del firmware del coche, con el que poder infectar el Bus CAN (red interna que gestiona las conexiones de todas las piezas). Esto no fue difícil ya que las actualizaciones no tenían comprobaciones de seguridad ni requerían ninguna clase de firma.

Con dicha actualización pudieron tomar el mando del controlador V850, lo que les permitió controlar la dirección, el motor, el sistema de bloqueo de puertas e incluso el termostato del vehículo. [6]

Este caso resulta especialmente perturbador, ya que pone de manifiesto como, si bien los dispositivos IoT nos ofrecen cantidad de opciones y personalización que hace unas décadas hubieran resultado impensables, también hacen que si la empresa encargada del diseño y fabricación del dispositivo no toma las suficientes medidas de seguridad, puede convertir algo tan familiar y a lo que estamos acostumbrados como un coche, en una trampa mortal si es interceptado por un atacante con suficiente conocimiento.

Figura 5. Demostración del hack en la Black Hat USA 2015

3.3 Vulnerabilidades en los monitores de frecuencia cardiaca para bebés Owlet

En 2016 saltó la voz de alarma debido a una serie de vulnerabilidades descubiertas en los monitores de frecuencia cardiaca de la marca Owlet, que se colocan en el calcetín del recién nacido y se sincronizan con un hub, y en caso de detectar alguna anomalía, envían una alerta al smartphone de los padres.

El investigador especializado en seguridad informática Jonathan Zdziarski fue quien dio la voz de alarma, al descubrir que el monitor cifraba la información del dispositivo, que era enviada a los servidores de la empresa propietaria, que serían quienes contactarían con los padres en caso de ser necesario. El problema es que la red Wi-Fi generada para que el dispositivo se

conectase al hub estaba totalmente abierta, permitiendo a cualquier persona dentro del rango poder interceptar la información.

Además de esto, si el atacante envía comandos HTTP simples al hub, puede hacer que éste se desconecte de la red Wi-Fi doméstica, volviéndolo inútil, así como conectarlo a otra red creada por el atacante, permitiendo que no sólo éste pueda interceptar la información enviada por el dispositivo, sino que puede hacer que la conexión nunca llegue a los servidores de la empresa, inutilizando el dispositivo y dejando expuesto al bebé en caso de que sufriera alguna complicación, ya que la alerta nunca sería enviada.

Este ataque es tan sencillo de realizar como realizar un escaneo de dispositivos con el puerto 80 abierto dentro de nuestra red doméstica, y una vez localizado, conectarnos a él, ya que no implementa ninguna medida de autentificación ni de autorización, ni a la hora de conectarnos a él ni a nivel de interfaz, lo que permite que un atacante pueda fácilmente eliminar la información sobre la red inalámbrica a la que este se conecta, o modificarla para vincularla a la red deseada. [7]

Este caso demuestra una vez más lo que hemos visto en durante esta sección, que un dispositivo que debería hacernos la vida más fácil, si no es diseñado con la seguridad en mente por parte del fabricante, puede acarrear nuevos problemas, que en casos como este son más grandes e importantes que las facilidades y beneficios que aporta su existencia.

Figura 6. Revisión actual del Owlet Sock

4. State of the art

En un mundo cada vez más dominado por un IoT que, si bien podríamos considerar en fase relativamente temprana, ya ha avanzado hasta cuotas que hace unos años eran inimaginables (se calcula que en 2022 existen alrededor de 14.000.000.000 dispositivos IoT), nos encontramos con ciertas convenciones sobre criptografía ligera aplicada a estos dispositivos de usos cotidianos.

Estos dispositivos tienen la necesidad de funcionar con procesadores reducidos y de baja potencia con el fin de primar su utilidad cotidiana por encima de su complejidad, lo que deja poco espacio para desarrollar medidas de seguridad en la mayoría de los casos. Cabe destacar el caso de los dispositivos basados en etiquetas RFID como podrían ser las pulseras de acceso a centros deportivos; estos dispositivos contienen información que debe protegerse, a pesar de ser tan sencillos en su construcción que no poseen ni batería propia, sino que se alimentan de la propia energía del lector cuando las acercamos.

Hay que tener en cuenta que los sistemas criptográficos modernos, en muchos casos requieren de una potencia computacional demasiado elevada para poder implementarse en estos dispositivos, que, por otra parte, necesitan poder cifrar información sensible y personal con el fin de evitar filtraciones de datos, suplantación de usuarios…

Esta tarea es compleja cuando tratamos de aplicarla a dispositivos basados en microprocesadores que en muchos casos no superan los 16 bytes de memoria RAM para un procesador de 4, 8 o 16 bits.

De esta forma, un algoritmo AES o RSA sería prácticamente imposible de implementar en uno de estos dispositivos, lo que nos sitúa constantemente en la tesitura de tener que encontrar un equilibrio entre ligereza y seguridad. A más seguro sea el algoritmo, peor será el funcionamiento del dispositivo o más potencia necesitará, lo que repercute directamente de forma negativa en el producto, que, por otro lado, mientras menos potente sea su cifrado, más peligroso se vuelve su uso y peor imagen dará de él.

Esta necesidad de seguridad, que a primera vista puede pensarse comúnmente que solo afecta a dispositivos como relojes digitales, hay que sumarle que cada día más dispositivos como bombas de insulina, marcapasos, wearables de todo tipo, sistemas de peaje, lectores de tarjetas de crédito contactless…

Es por ello que la necesidad de proteger la información manejada por dichos dispositivos, así como controlar el acceso a ésta, sin disminuir la eficiencia ni aumentar costes, se ha convertido en uno de los mayores desafíos actuales de la criptografía.

Para empezar a profundizar podemos enumerar los algoritmos de cifrado ligero más utilizados actualmente para dispositivos enfocados al IoT, que serán analizados en el siguiente apartado.

- Cifrado en bloque
  - PRESENT
  - SPECK
  - SIMON
- Funciones Hash
  - PHOTON
  - QUARK
- Cifrado en flujo
  - Grain
  - Trivium
  - MICKEY
- MAC (Código de autenticación de mensajes)
  - Chaskey
  - LightMAC

4.1 Cifrado en bloque

Estos cifrados ligeros se han diseñado con el objetivo de poder crear un sustituto de AES que pueda funcionar de forma eficiente en dispositivos con poca capacidad de procesamiento, pero siempre con el objetivo de poder mantener un nivel de seguridad lo más cercano posible a pesar de trabajar con bloques más pequeños.

En esta sección trataremos algunos algoritmos ligeros de cifrado en bloque que he considerado interesantes a la hora de una posible implementación en dispositivos inteligentes de baja potencia.

4.1.1 Present

El algoritmo de cifrado Present fue desarrollado en 2007 por Orange Labs junto a la Universidad de Bochum y la Universidad Técnica de Dinamarca y está enfocado a etiquetas RFID.

La seguridad de estas etiquetas en el IoT es muy importante, ya que son estas las que permiten la identificación inequívoca de un equipo, y el no preocuparse por protegerlas puede dar lugar a ataques de suplantación.

Present es casi tres veces más ligero que AES y está pensado para aparatos con un consumo de energía muy bajo que necesiten una gran eficiencia. Ha sido incluido en el nuevo standard internacional de métodos criptográficos ligeros.

Se trata de un algoritmo compuesto por una red SP (Sustitución-Permutación) de 31 rondas. El bloque es de longitud 64 bits, y admite dos tamaños distintos de clave; 80 y 128 bits. [8]

generateRoundKeys()
for $i = 1$ to 31 **do**
  addRoundKey(STATE,K_i)
  sBoxLayer(STATE)
  pLayer(STATE)
end for
addRoundKey(STATE,K_{32})

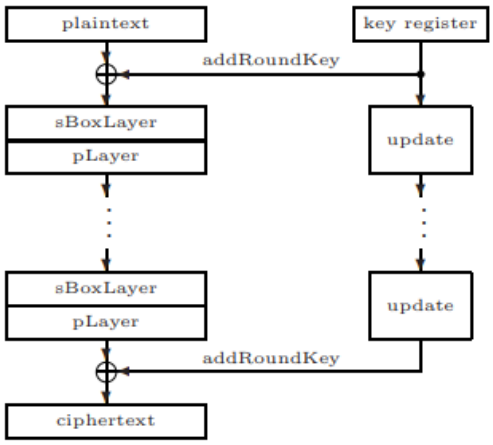


Figura 7. Esquema de alto nivel de Present

4.1.2 SIMON

La familia de algoritmos de cifrado SIMON fue desarrollada y lanzada por la NSA en el año 2013. SIMON está optimizado para su uso en hardware, mientras que SPECK, del que hablaremos a continuación, está enfocado al software.

Estos cifrados comenzaron su desarrollo en 2011 con una serie de necesidades para dispositivos IoT en mente, y la NSA presionó mucho para intentar incluirlos en el standard internacional.

Esto se ralentizó ya que países como Alemania, Japón o Israel se opusieron, alegando que la NSA estaba tratando de estandarizarlos a sabiendas de sus debilidades, cosa que la NSA sigue negando hoy en día. A pesar de esto, en 2018 fueron aceptados como standard para RFID.

Este cifrado se basa en una red de Feistel con una palabra de *n* bits, por lo que su longitud de bloque es de *2n*. Su clave *m* tiene longitud múltiplo de 2, 3 o 4 por *n*. A la hora de referirnos a un cifrado de SIMON también se suele hacer como Simon*n/nm*. Así, por ejemplo, un cifrado Simon64/128 tendría una palabra de 32 bits y una clave de 128 bits. [9]

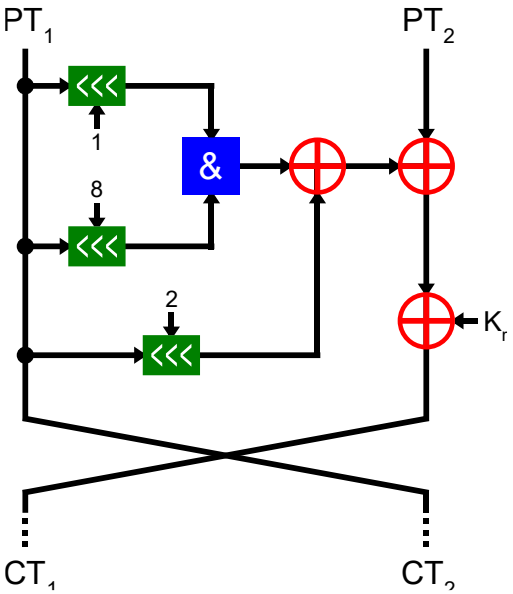


Figura 8. Ronda de cifrado SIMON

4.1.3 SPECK

Como hemos mencionado antes, el cifrado SPECK es la contraparte de SIMON, con un enfoque en el software.

En este caso, el algoritmo consta de un bloque formado siempre por dos palabras, que pueden variar su longitud en bits entre 16, 24, 32, 48 o 64. Por otra parte, su clave puede ser de 2, 3 o 4 palabras. Cada ronda consiste en dos rotaciones, sumándole la palabra derecha a la izquierda, realizando una operación XOR entre la clave y la palabra izquierda y otro XOR de la palabra izquierda con la palabra derecha. [10]

El número de rondas depende de los parámetros seleccionados:

Tamaño del bloque en bits	Tamaño de la clave en bits	Rondas
2 × 16 = 32	4 × 16 = 64	22
2 × 24 = 48	3 × 24 = 72	22
	4 × 24 = 96	23
2 × 32 = 64	3 × 32 = 96	26
	4 × 32 = 128	27
2 × 48 = 96	2 × 48 = 96	28
	3 × 48 = 144	29
2 × 64 = 128	2 × 64 = 128	32
	3 × 64 = 192	33
	4 × 64= 256	34

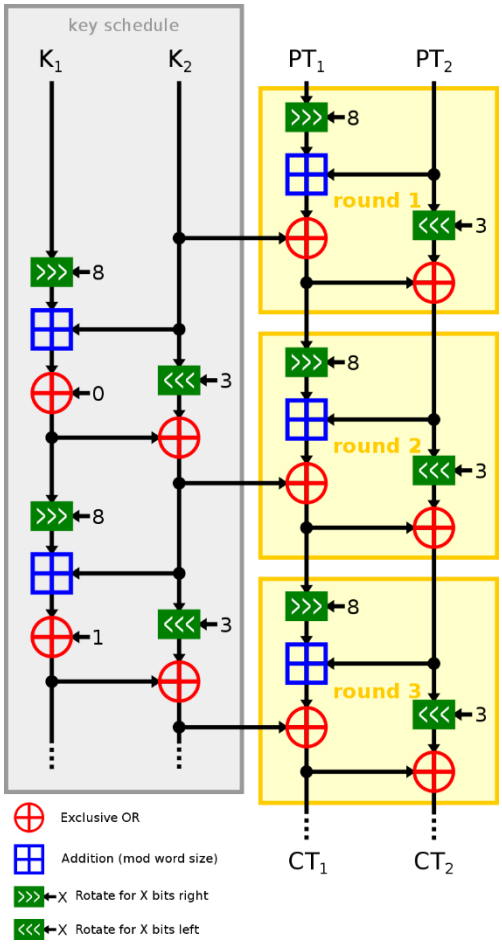


Figura 9. Tres rondas de Speck con 2 palabras por clave

4.2 Funciones Hash

Las funciones hash son algoritmos criptográficos que reciben un dato de entrada en una salida en forma de texto que, dependiendo de la función, puede tener tamaño variable o siempre fijo independientemente del tamaño de los datos recibidos.

Son comúnmente utilizadas en criptografía para almacenar contraseñas en bases de datos sin necesidad de guardarlas en limpio, y a continuación vamos a tratar dos de las funciones hash ligeras más comunes en IoT.

4.2.1 PHOTON

Esta familia de funciones hash ligera fue diseñada por Jian Guo, Thoman Peyrin y Axel Poschmann como respuesta a la necesidad de una función hash ligera aplicable a etiquetas RFID.

El diseño de esta familia de funciones, al menos en el momento de su publicación, la convirtió en la función hash más compacta conocida hasta entonces.

Este algoritmo tiene una estructura inspirada en las construcciones llamadas “de esponja” o “sponge functions” en inglés (funciones criptográficas con un estado interno que pueden tomar como entrada un stream de bits de cualquier tamaño y producir una salida de un tamaño deseado), siendo su principal diferencia la capacidad de utilizar un bitrate distinto para la salida que el empleado en la entrada. [11]

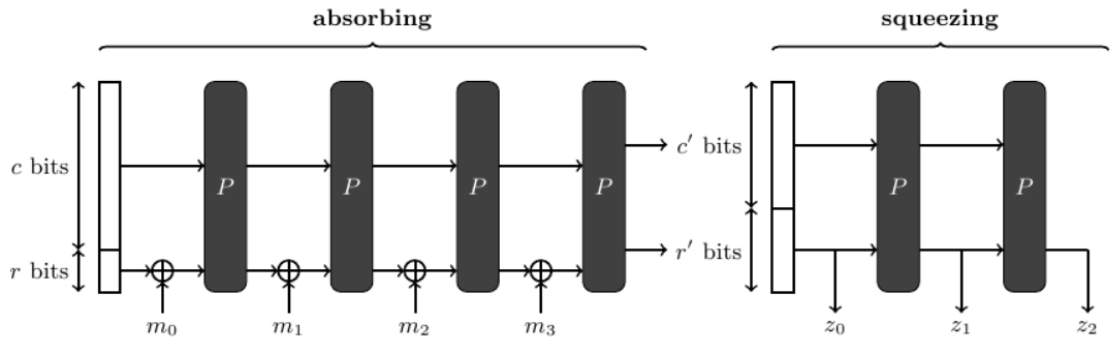


Figura 10. Sponge function

Por otra parte, sus permutaciones están muy inspiradas en la estructura de cifrado AES, donde el estado interno puede ser representado por una matriz cuadrada de tamaño *d.d*, donde cada permutación interna se compone de 12 rondas.

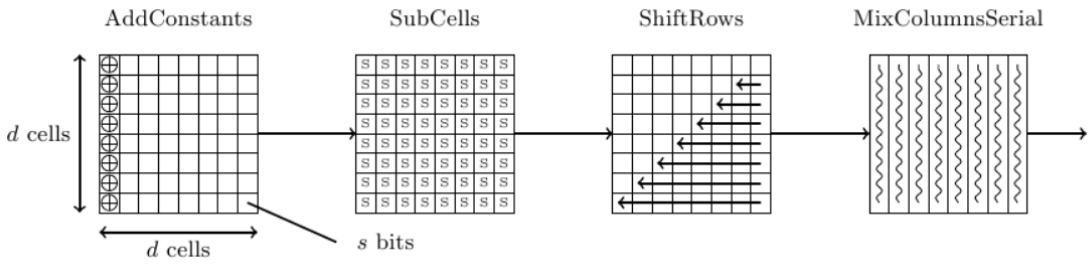


Figura 11. Esquema de PHOTON

4.2.2 QUARK

La familia de funciones Quark fue desarrollada por Jean-Philippe Aumasson, Luca Henzen, Willi Meier y María Naya-Plasencia, y nació con la mente puesta en las etiquetas RFID.

Esta familia se basa en una construcción en esponja, lo que significa que tiene un número finito de estados internos que toman como entrada un flujo de bits de cualquier tamaño y producen otro flujo de bits con un tamaño deseado como salida.

Tiene un único nivel de seguridad, con el fin de reducir las necesidades de memoria.

Está inspirada por otros protocolos ligeros como Grain o KATAN y se compone de tres instancias: u-Quark, d-Quark, and t-Quark.

Esta familia de funciones hash ha sido muy comparada con la familia anteriormente mencionada, PHOTON. En estas comparaciones se comprobó que, si bien ambas familias de algoritmos son muy similares, Quark está menos optimizada para un uso a nivel de software, aunque esto no tiene demasiada importancia ya que su implementación está pensada para ser realizada a nivel de hardware. En conclusión, ambos tienen un buen equilibrio entre rendimiento y seguridad.

Cabe también destacar que, en el momento de dicha comparación, ninguno de los dos algoritmos ha sido roto aún. [12]

4.3 Cifrado en flujo

Los algoritmos de cifrado en flujo son cifrados de clave simétrica en los que la entrada pasa por un flujo de claves o *keystream* generada a partir de la clave de cifrado, en las que se opera dígito por dígito para obtener una salida codificada.

Estos algoritmos suelen tener el problema de depender de un generador de números pseudoaleatorios (PRNG) que cumpla unos mínimos de calidad en lo que a ser criptográficamente seguros se refiere.

4.3.1 Grain

Este algoritmo de cifrado fue subido a eSTREAM en 2004, y está diseñado para funcionar en entornos de hardware restrictivos, lo que lo vuelve una buena opción para su aplicación en IoT.

Grain tiene un estado interno de 160 bits que consiste en 80 bits de LSFR y 80 bits de NLSFR.

A día de hoy, se le han encontrado númerosas vulnerabilidades, las cuales han sido en su mayoría corregidas en Grain 128a, que es la versión recomendada y que aporta seguridad en 128 bits y mecanismos de autenticación. [13]

3.3.2 Trivium

Algoritmo de cifrado en flujo síncrono diseñado por Christophe De Cannière y Bart Preneel.

Fue subido a eSTREAM y a pesar de no estar patentando, ha sido incluido en el standard internacional ISO/IEC 29192-3.

Posee un estado interno de 288 bits que consiste en 3 registros de desplazamiento de longitud variable. [14]

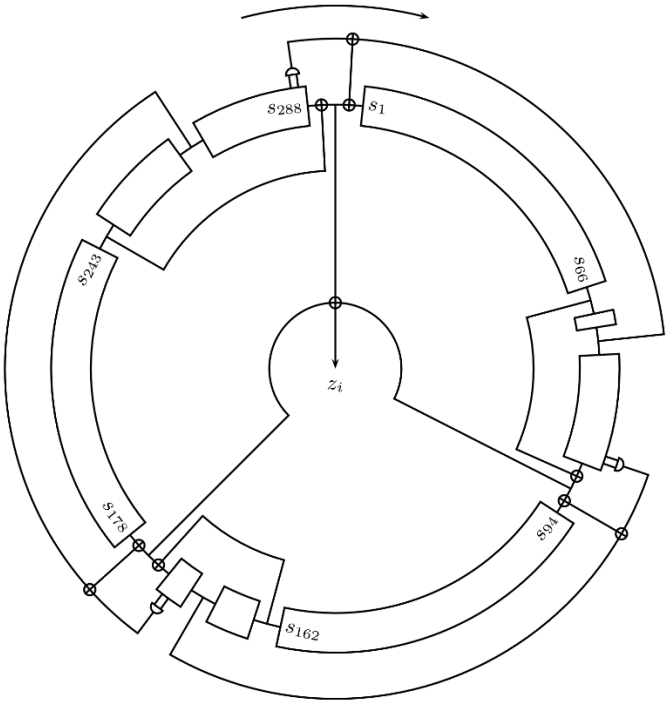


Figura 12. Esquema de Trivium

4.3.3 MICKEY

El cifrado **Mutual Irregular Clocking KEYstream generator** ó Generador de keystream de irregularidad de reloj mutua en castellano, es un algoritmo de cifrado en flujo diseñado con el hardware en mente y enfocado a dispositivos con pocos recursos.

Es uno de los tres cifrados de hardware aceptados por eSTREAM junto a los mencionados anteriormente; Grain y Trivium.

Este algoritmo convierte una clave de 80 bits y un vector de inicialización con un tamaño variable, entre 0 y 80 bits, en un generador de claves con tamaño de hasta 2^{40} bits. [15]

4.4 Cifrados MAC

Los cifrados MAC son aquellos que se encargan de cifrar los códigos de autenticación de mensaje o MAC por sus siglas en inglés.

Estos valores se crean a partir de una función hash con clave secreta K, que debe ser criptográficamente segura para poder resistir ante ataques.

Esta clave secreta solo es conocida por el emisor y el receptor, por lo que, con ella, el receptor puede recalcular el hash del mensaje y comprobar que cumple el principio de no repudio y que la información contenida en éste no ha sido alterada.

Dentro de este tipo de funciones, actualmente podemos distinguir tres categorías para clasificarlas:

1. **CBC-MAC**: Estas funciones cifran el mensaje mediante algún algoritmo de cifrado en bloque en modo CBC (Cipher block chaining). De este modo se crea una cadena de bloques en los que el cifrado de cada uno depende del resultado de cifrar el bloque anterior, haciendo que cualquier cambio en un bit cambie completamente el resultado del cifrado, impidiendo que el atacante pueda modificar el mensaje sin ser detectado.

2. **HMAC**: Siglas de "Código de autenticación de mensaje basado en hash" son aquellas funciones MAC en las que se utiliza una función hash y una clave secreta para obtener como resultado un hash que permita demostrar el origen y contenido del mensaje. Se pueden construir en base a cualquier función hash, haciendo que su seguridad dependa de la función escogida, el tamaño de salida escogido para este hash y la clave privada utilizada.

3. **UMAC**: Por último, tenemos los cifrados MAC basados en un hash universal. Estas funciones utilizan una función hash escogida de entre varias mediante un proceso de selección desconocido para el atacante. Acto seguido, el código vuelve a cifrarse para tratar de dificultar la identificación del cifrado utilizado. De esta forma, solo el receptor es capaz de averiguar qué función se ha utilizado para realizar el hash y usarlo para comprobar la integridad y el origen del mensaje.

Dentro de este tipo de cifrado, vamos a tratar algunos que en los últimos años han sido bastante relacionados con el mundo de la seguridad aplicada al Internet of Things.

4.4.1 Chaskey

Chaskey es una función MAC basada en permutaciones diseñada por Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel e Ingrid Verbauwhede.

Esta función se compone de una clave de 128 bits K y, usando una permutación de 128 bits π, convierte un mensaje m en bloques de 128 bits.

Estas permutaciones están basadas en ARX (Addition-Rotation-XOR).

Chaskey está diseñado para microcontroladores de 32 bits, es implementable entre distintas plataformas y es resistente a ataques de temporización.

Además de todo esto, Chaskey no está patentado, lo que permite su uso de forma gratuita[16].

4.4.2 LightMAC

LightMAC es una función MAC diseñada por Atul Luykx, Bart Preneel, Elmar Tischhauser y Kan Yasuda, publicada en 2016.

Dentro de las funciones MAC la podríamos englobar dentro de las CBCMAC, ya que se basa en bloques de cifrado.

Esta función permite usar PRESENT o SPECK como cifradores, por lo que su uso en dispositivos IoT está demostrado como viable y eficiente. [17]

Algorithm 1: $\mathrm{LightMAC}_{K_1,K_2}(M)$

Input: $K_1, K_2 \in \{0,1\}^k$, $M \in \{0,1\}^{\leq 2^s(n-s)}$
Output: $T \in \{0,1\}^t$

$V \leftarrow 0^n \in \{0,1\}^n$
$M[1]M[2]\cdots M[\ell] \xleftarrow{n-s} M$
for $i = 1$ **to** $\ell - 1$ **do**
  $V \leftarrow V \oplus E_{K_1}(i_s M[i])$
end
$V \leftarrow V \oplus (M[\ell]10^*)$
$T \leftarrow \lfloor E_{K_2}(V) \rfloor_t$
return T

Figura 13. Función LightMAC

5. LightCipher

Con el fin de implementar el funcionamiento de algunos de estos algoritmos de criptografía ligera aportando una interfaz gráfica, se ha desarrollado la aplicación web LightCipher, la cual es accesible de forma pública tanto a través de un repositorio de código en Github [18], como de un despliegue en un servidor en Heroku [19].

Esta aplicación está programada en el lenguaje Java, con el framework Spring, y se nutre de las implementaciones de los algoritmos de las familias Speck y Simon realizadas por el usuario GaloisInc dentro del proyecto SAWScript [20], el cual implementa estos algoritmos en todas sus variaciones, pero no aporta una interfaz gráfica, sino que se utiliza mediante la línea de comandos.

LightCipher permite experimentar con las variaciones más comunes de ambas familias de algoritmos, de forma que se puede comprobar el resultado de cifrar un texto con una clave concreta, así como realizar el camino inverso.

Las versiones de Simon implementadas son aquellas para bloques de 32, 48, 64, 96 y 128 bits, lo que permite el uso de claves desde 64 hasta 256 bits en función de la versión utilizada del algoritmo:

Tamaño del bloque en bits	Tamaño de la clave en bits	Rondas
32	64	32
48	72	36
	96	36
64	96	42
	128	44
96	96	52
	144	54
128	128	68
	192	69
	256	72

Para Speck, las versiones implementadas permiten trabajar con pares de bloques de 16, 24, 32, 48 y 64 bits. Los parámetros de esta familia de algoritmos se encuentran desglosados anteriormente en el apartado 4.1.3

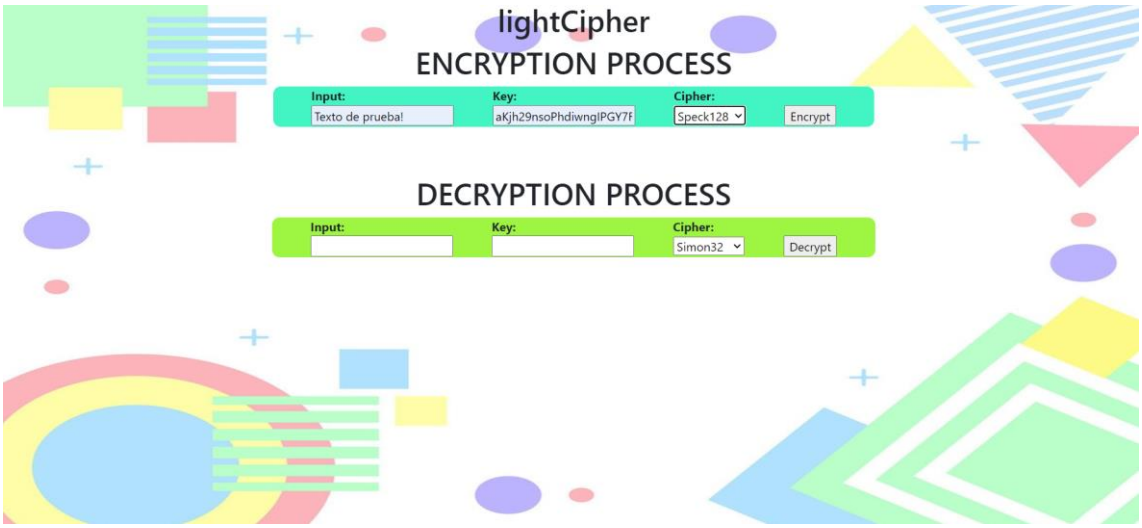
Una vez dentro de la aplicación web, se encuentra una página de inicio con la información de las funcionalidades actuales de LightCipher:

Pantalla de inicio

En el menú principal de la aplicación es donde se pueden observar las distintas opciones con las que experimentar.

Con el fin de mostrar un ejemplo, se realizará el cifrado y descifrado de una palabra mediante el algoritmo Speck, en su variante de 128 bits, para un texto de 16 caracteres y una clave de 24.

El texto a cifrar será “Texto de prueba!”, y la clave usada será la cadena de texto “aKjh29nsoPhdiwngIPGY7Fvs”:



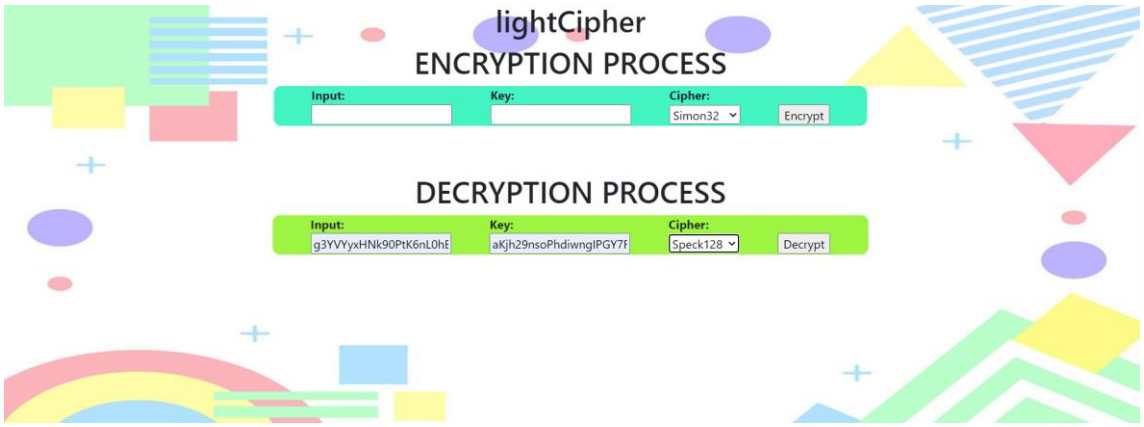
Cifrado desde el menú principal

El texto introducido es encriptado a nivel de bytes, y produce como salida dichos bytes encriptados y codificados en Base64.

Si realizamos el cifrado para estos valores obtenemos que el resultado es la cadena de texto “g3YVYyxHNk90PtK6nL0hBw==”

Resultado de cifrar el texto

A continuación, se puede deshacer dicha transformación para descifrar el texto anteriormente cifrado:

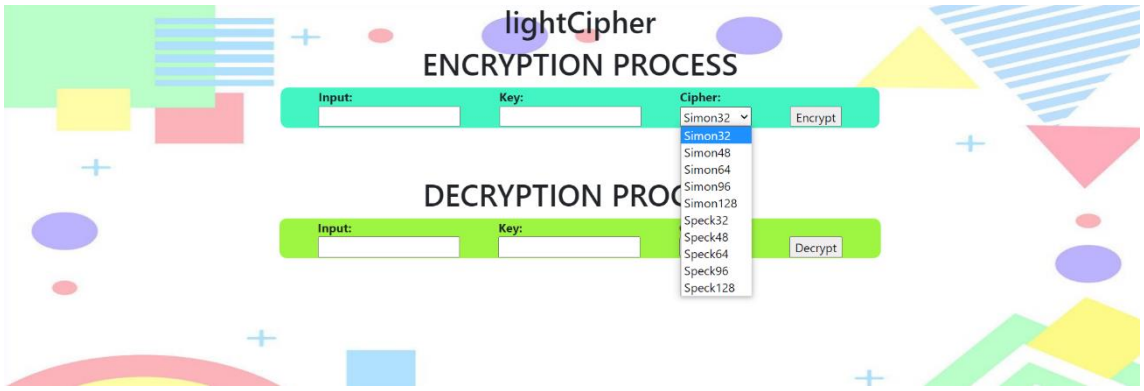


Descifrado desde el menú principal

De esta forma se demuestra el funcionamiento del algoritmo en ambos sentidos:

Resultado de deshacer el cifrado anterior

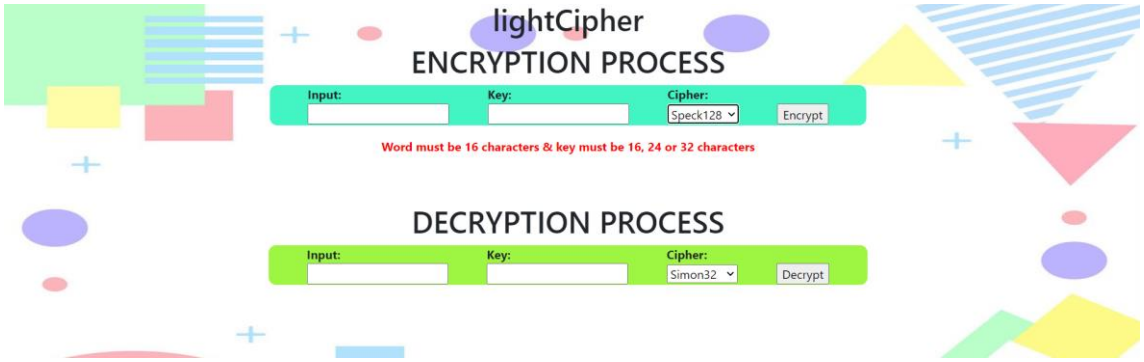
Esta aplicación permite el cifrado y descifrado para las siguientes variantes de dichos algoritmos:



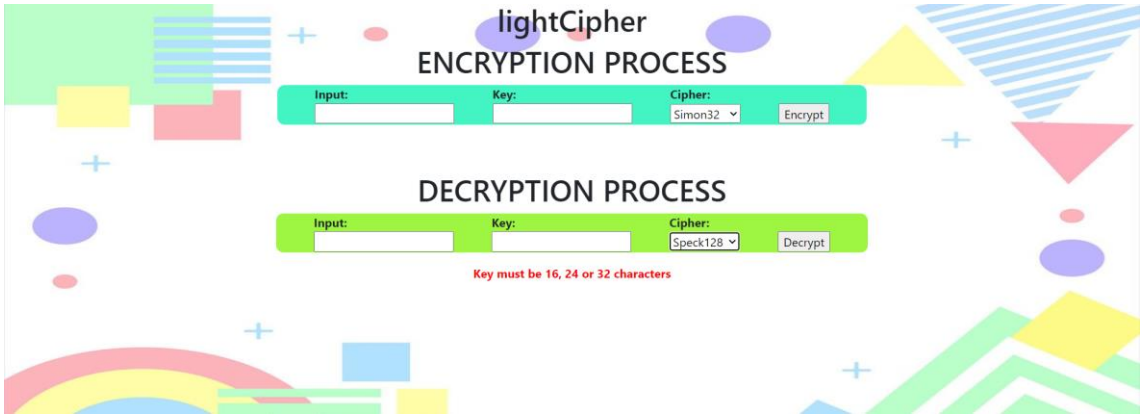
Desplegable con las opciones posibles

Ya que el fin de esta aplicación es proporcionar una herramienta que implemente algoritmos de criptografía ligera y permita utilizarlos mediante una interfaz gráfica , se ha implementado su funcionalidad desde el nivel más básico, es decir, solo pueden cifrarse/descifrarse las palabras de tamaño más pequeño aceptadas por el algoritmo.

Para ilustrar esta restricción se han implementado mecanismos de validación en los campos de entrada, que devuelven un mensaje de error con el tamaño aceptado por campo para cada uno de los algoritmos:



Mensaje de error para cifrado Speck128

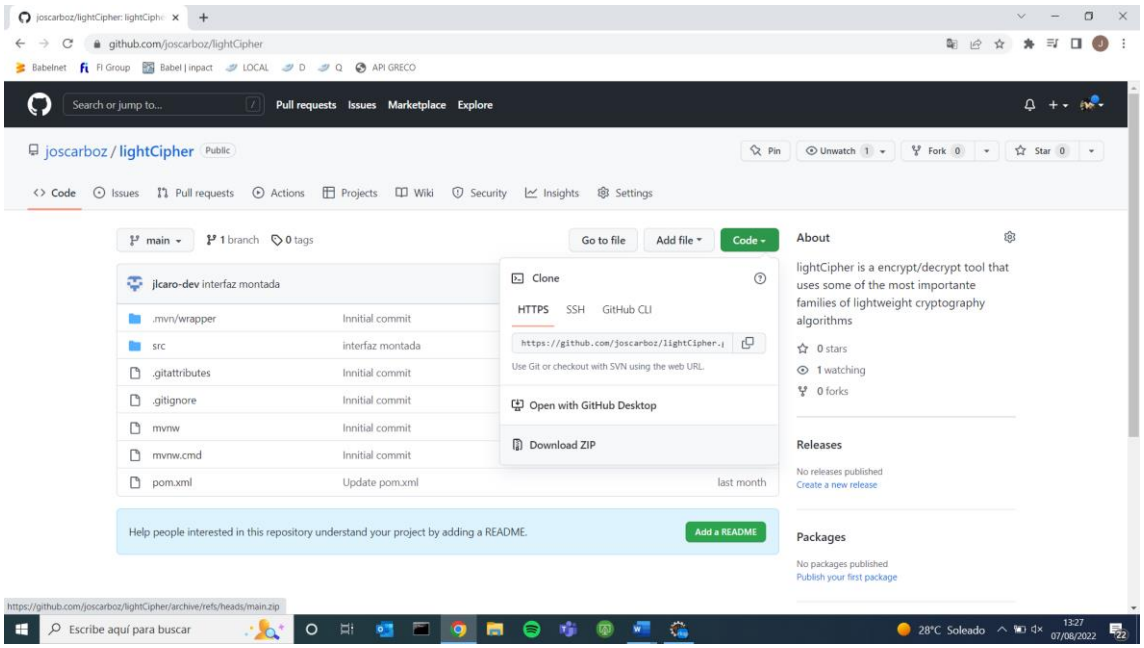


Mensaje de error en descifrado Speck128

5.1 Ejecutando LightCipher en una instancia local

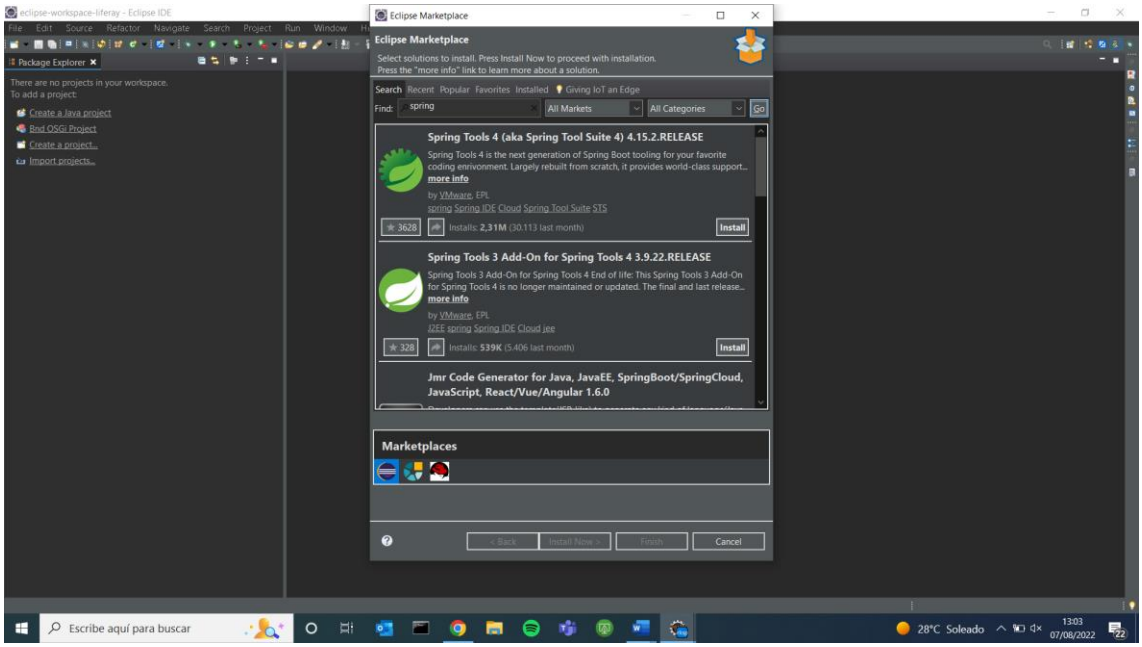
Para poder ejecutar una instancia de LightCipher en local, se debe tener instalado un IDE (en este caso se ha utilizado Eclipse), configurado un JDK de Java 8 o superior.

El proyecto se puede descargar desde su repositorio en GitHub:

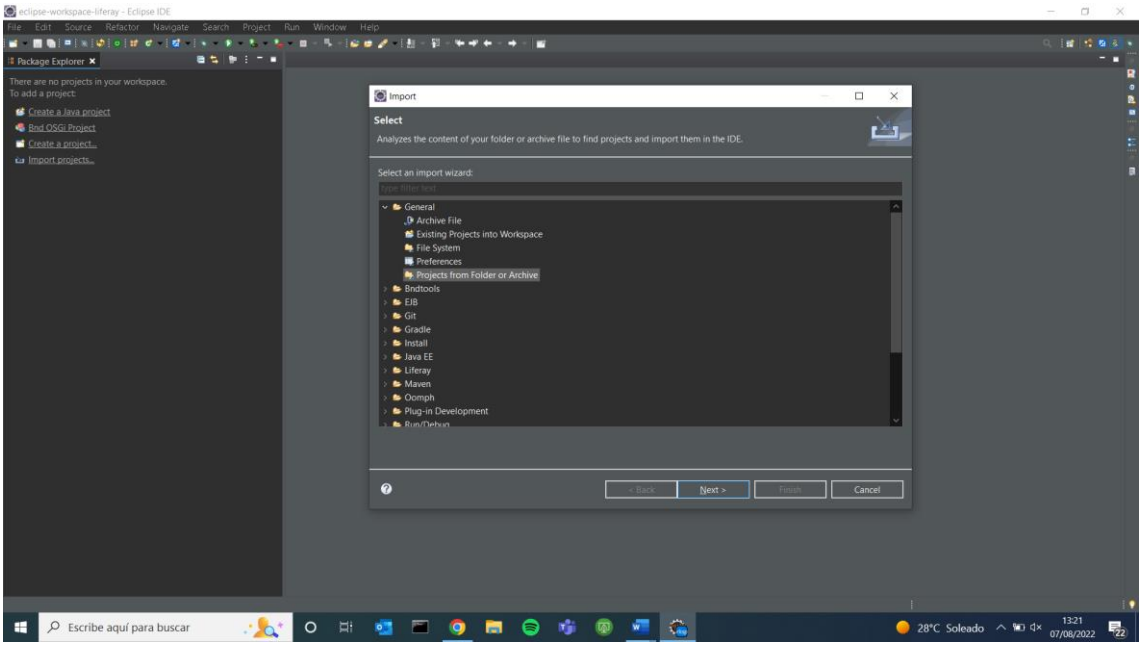


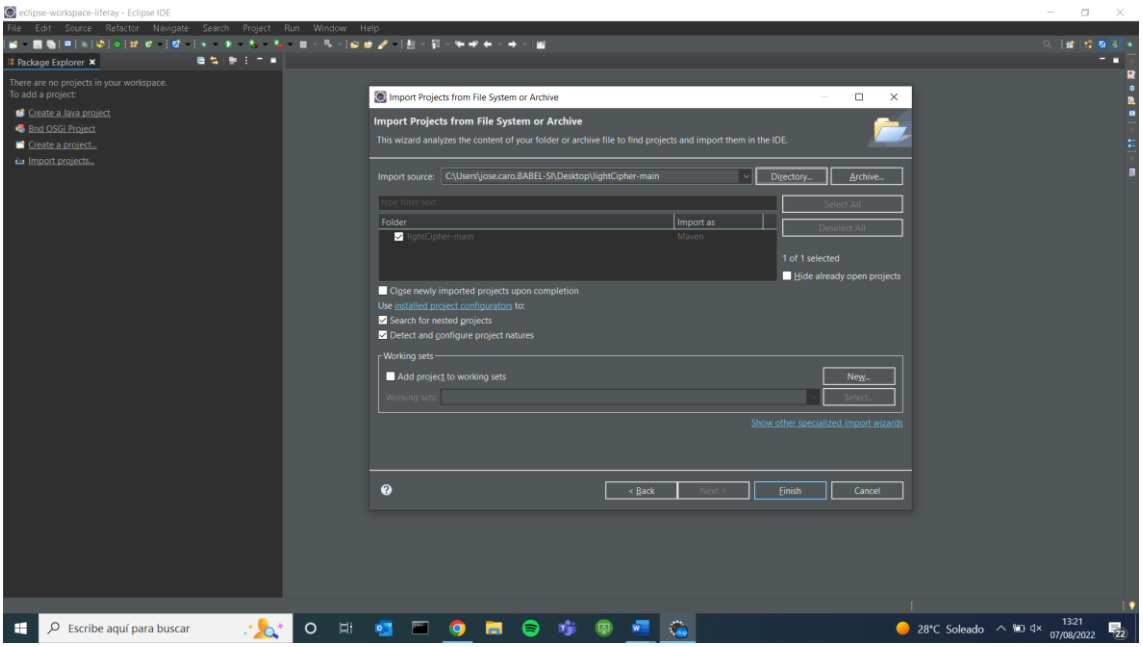
Tras esto, se instala el plugin de Spring Tools 4.

Desde Eclipse esto se puede hacer desde Help, accediendo al Eclipse Marketplace:

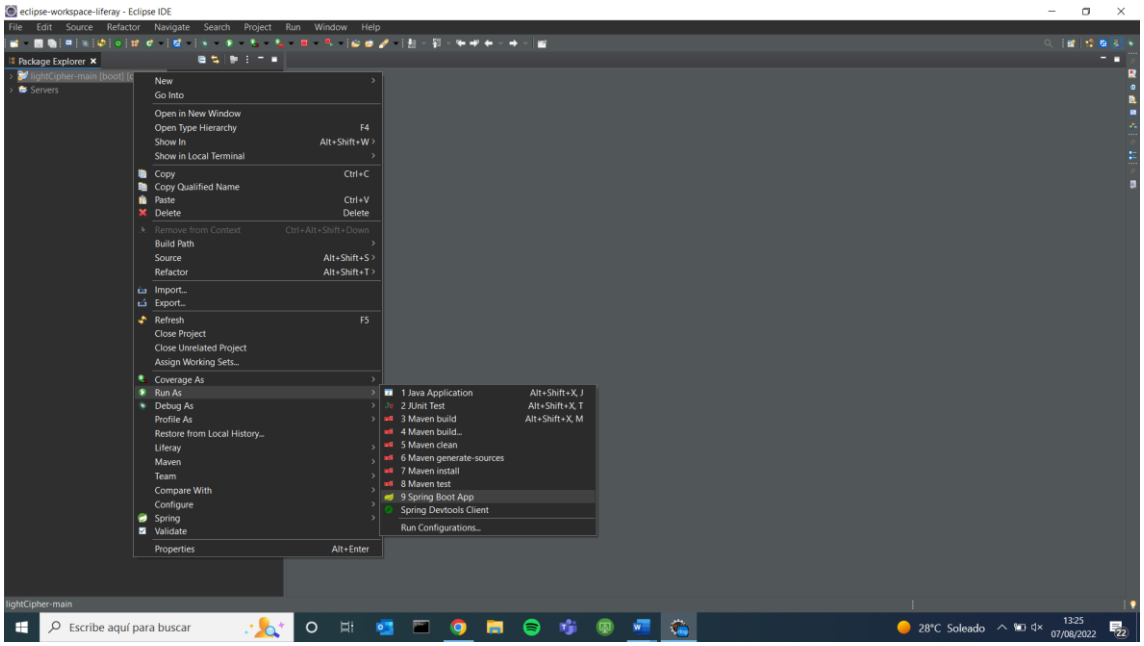


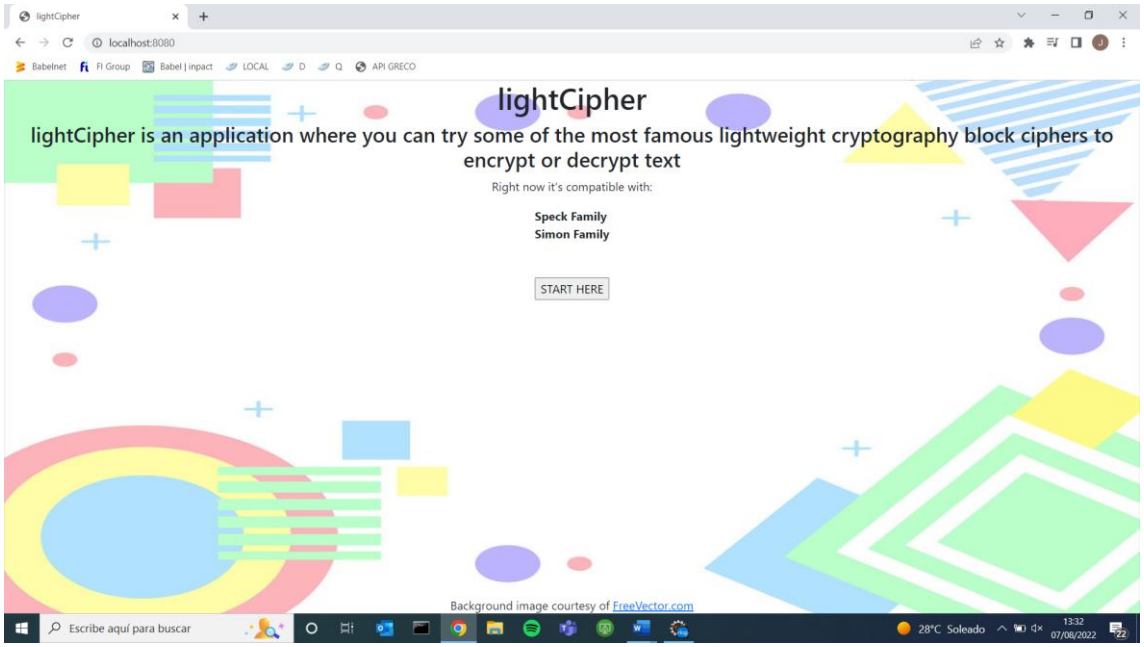
Con Spring Tools 4 instalado, se descarga el proyecto desde el repositorio de GitHub, y se importa desde Eclipse mediante la opción de menú File, en las opciones de Import, seleccionando para importarlo desde la opción Import Project From Folder or Archive





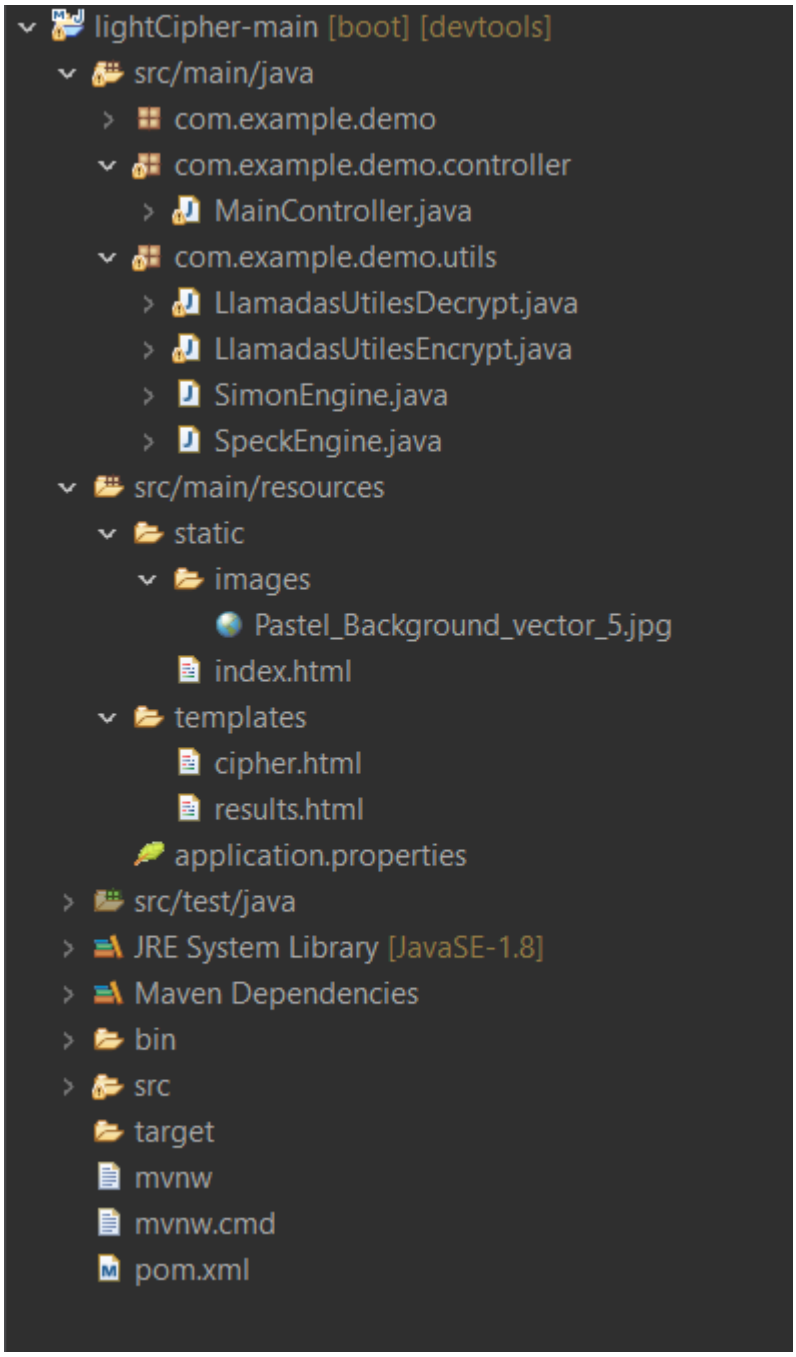
A continuación, se ejecuta el proyecto como Spring Boot App, lo que desplegará una instancia local de LightCipher en localhost:8080





5.2 Estructura del proyecto

La estructura del proyecto una vez importado es la siguiente:



Dentro de la carpeta src/main/java existen dos paquetes principales:

1. com.example.demo.controller: Dentro de este paquete se encuentra el controlador principal de la aplicación:

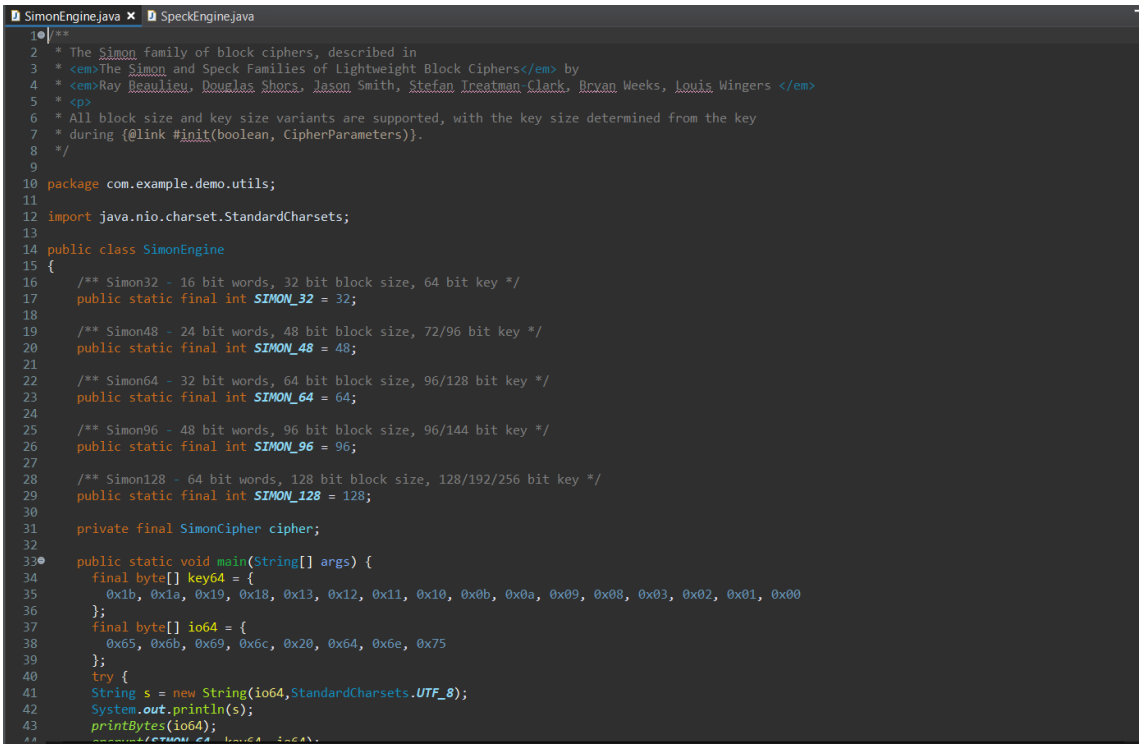
```java
@RequestMapping("/cipher")
public String cipher(Model modelo) {
    modelo.addAttribute("outputEncrypt", "");
    modelo.addAttribute("outputDecrypt", "");
    modelo.addAttribute("outputEncryptError", "");
    modelo.addAttribute("outputDecryptError", "");
    return "cipher";
}

public static byte[] hexStringToByteArray(String s) {
    int len = s.length();
    byte[] data = new byte[len / 2];
    for (int i = 0; i < len; i += 2) {
        data[i / 2] = (byte) ((Character.digit(s.charAt(i), 16) << 4) + Character.digit(s.charAt(i + 1), 16));
    }
    return data;
}

public String toHex(String arg) {
    return String.format("%040x", new BigInteger(1, arg.getBytes(/* YOUR_CHARSET? */)));
}

@RequestMapping("/encrypt")
public String encrypt(@RequestParam(name = "inputText", required = false, defaultValue = "") String inputText,
        @RequestParam(name = "keytext", required = false, defaultValue = "") String keytext,
        @RequestParam(name = "selectEncrypt", required = false, defaultValue = "") String select, Model modelo) {
    String res = "results";
    String selector = select;
    switch (selector) {
    case "Simon32":
        if (inputText.length() == 4 && keytext.length() == 8) {
            LlamadasUtilesEncrypt.Simon32(inputText, keytext, modelo);
        } else {
            String message = "Word must be 4 characters & key must be 8 characters";
            modelo.addAttribute("outputEncryptError", message);
            res = "cipher";
        }
        break;
    case "Speck32":
        if (inputText.length() == 4 && keytext.length() == 8) {
            LlamadasUtilesEncrypt.Speck32(inputText, keytext, modelo);
        } else {
            String message = "Word must be 6 characters & key must be 8 characters";
```

MainController.java

2. com.example.demo.utils: En este paquete se encuentran las clases que contiene las implementaciones de los algoritmos Simon y Speck realizadas por GaloisInc:

```java
/**
 * The Speck family of block ciphers, described in
 * <em>The Simon and Speck Families of Lightweight Block Ciphers</em> by
 * <em>Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, Louis Wingers </em>
 * <p>
 * All block size and key size variants are supported, with the key size determined from the key
 * during {@link #init(boolean, CipherParameters)}.
 */

/*
 From: https://github.com/timw/bc-java/tree/feature/simon-speck/core/src/main/java/org/bouncycastle/crypto/engines
 License: https://github.com/timw/bc-java/blob/feature/simon-speck/LICENSE.html

 Modifications:

 * Removed Bouncy Castle dependency

*/
package com.example.demo.utils;
public class SpeckEngine
{
    /** Speck32 - 16 bit words, 32 bit block size, 64 bit key */
    public static final int SPECK_32 = 32;

    /** Speck48 - 24 bit words, 48 bit block size, 72/96 bit key */
    public static final int SPECK_48 = 48;

    /** Speck64 - 32 bit words, 64 bit block size, 96/128 bit key */
    public static final int SPECK_64 = 64;

    /** Speck96 - 48 bit words, 96 bit block size, 96/144 bit key */
    public static final int SPECK_96 = 96;

    /** Speck128 - 64 bit words, 128 bit block size, 128/192/256 bit key */
    public static final int SPECK_128 = 128;

    private final SpeckCipher cipher;

    public static void main(String[] args) {
      final byte[] key64 = {
        0x1b, 0x1a, 0x19, 0x18, 0x13, 0x12, 0x11, 0x10, 0x0b, 0x0a, 0x09, 0x08, 0x03, 0x02, 0x01, 0x00
      };
      final byte[] io64 = {
```

SpeckEngine.java



SimonEngine.java

En este paquete también se incluyen las clases LlamadasUtilesDecrypt.java y LlamadasUtilesEncrypt.java, que contienen métodos que llaman a las clases anteriormente mencionadas para cada versión de los algoritmos.

En la carpeta src/main/resources se encuentran la imagen de fondo utilizada en la aplicación, así como los archivos HTML con cada una de las vistas de la aplicación.

6. Conclusiones

Durante la elaboración de este trabajo ha quedado demostrada la importancia de los dispositivos IoT en el día a día de la sociedad actual, así como el enorme crecimiento de estos dispositivos en nuestros hogares.

Si bien las vulnerabilidades y problemas de seguridad que vienen con estos dispositivos son ampliamente conocidas por la comunidad especializada y los desarrolladores, pocas son las empresas que se esmeran por mitigar o evitar estos problemas.

Esto ha dado lugar a incidentes como los que hemos visto, que, si bien son bastante representativos, son una mínima parte de todos los que tienen lugar a diario, cada vez que se lanza un nuevo producto al mercado o una nueva actualización para un dispositivo ya existente.

Por otra parte, tenemos a un sector de profesionales en el mundo de la ciberseguridad muy enfocados en desarrollar nuevos algoritmos y métodos de cifrado ligeros que permitan a estos dispositivos contar con una buena capa de seguridad. Todo esto sin tener que renunciar a la eficiencia ni requerir un aumento de costes a la hora de la fabricación.

Durante la elaboración de este trabajo se ha podido comprobar no sólo la cantidad de algoritmos de criptografía ligera existentes, sino las investigaciones que hay en curso, las implementaciones que se están realizando de estos y las mejoras en su eficiencia que se están buscando.

También hay que incidir en que existe un problema serio a la hora de aplicar estas capas de seguridad a los dispositivos IoT que salen al mercado día a día, así como insistir en que existe una parte de responsabilidad en los consumidores que se lanzan a las ofertas y a los precios bajos sin pararse a pensar en qué existe detrás, produciéndose un efecto recíproco entre el desarrollo de dichos dispositivos y los hábitos de consumo de la población en el contexto del mercado tecnológico.

Este fenómeno puede llegar a traducirse un riesgo directo para la seguridad sectores especialmente vulnerables. Así, como por ejemplo hemos visto anteriormente, se están comercializando productos enfocados a la seguridad de los más pequeños cuyas medidas de seguridad son deficientes, y que pueden estar causando más problemas de los que solucionan.

Podemos concluir en que existen varios proyectos interesantes en el ámbito de la criptografía ligera aplicada a IoT, pero conviven con una ausencia de implementaciones en código de estos, así como los que se encuentran desarrollados e implementados, no suelen venir acompañados de documentación o ejemplos prácticos de cómo utilizarlos.

Por este motivo se ha desarrollado la aplicación LightCipher, con el objetivo de aportar usabilidad algunas de estas implementaciones en una aplicación web con interfaz gráfica, que permita utilizar y experimentar con dichos algoritmos, ya que actualmente no ha sido posible durante el desarrollo de este trabajo encontrar ninguna aplicación que ofrezca una funcionalidad similar.

Esto, sirviendo también para darlos a conocer y probar que ya existen algoritmos de criptografía ligera que pueden ser implementados en entornos industriales, y que pueden servir para evitar ataques como los anteriormente descritos sin necesidad de sacrificar el rendimiento del producto.

7. Bibliografía

7.1 Recursos online

[1] https://www.visionofhumanity.org/what-is-the-internet-of-things/#:~:text=The%20term%20'Internet%20of%20Things,them%20through%20a%20supply%20chain.

[2] https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT

[3] https://owasp.org/www-project-internet-of-things/

[4] https://www.justice.gov/usao-nj/pr/computer-hacker-who-launched-attacks-rutgers-university-ordered-pay-86m-restitution

[5] https://www.cloudflare.com/es-es/learning/ddos/glossary/mirai-botnet/

[6] https://www.kaspersky.es/blog/blackhat-jeep-cherokee-hack-explained/6552/

[7] https://www.theregister.com/2016/10/13/possibly_worst_iot_security_failure_yet/

[8] http://lightweightcrypto.org/present/present_ches2007.pdf

[9] https://en.wikipedia.org/wiki/Simon_(cipher)

[10] https://www.cryptopp.com/wiki/SPECK

[11] https://sites.google.com/site/photonhashfunction/design

[12] https://link.springer.com/article/10.1007/s00145-012-9125-6

[13] https://www.ecrypt.eu.org/stream/e2-grain.html

[14] https://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf

[15] https://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf

[16] https://mouha.be/chaskey/

[17] https://tinycrypt.wordpress.com/2017/03/22/asmcodes-lightmac/

[18] https://github.com/joscarboz/lightCipher

[19] http://lightcipher2022.herokuapp.com/

[20] https://github.com/GaloisInc/saw-script

7.2 Imágenes

1. https://iotbusinessnews.com/2022/05/19/70343-state-of-iot-2022-number-of-connected-iot-devices-growing-18-to-14-4-billion-globally/
2. https://www.appsealing.com/owasp-iot-top-10/
3. https://news.softpedia.com/news/there-are-almost-half-of-million-iot-devices-infected-with-the-mirai-iot-malware-509432.shtml
4. https://www.kaspersky.es/blog/blackhat-jeep-cherokee-hack-explained/6552/
5. https://www.kaspersky.es/blog/blackhat-jeep-cherokee-hack-explained/6552/
6. https://www.amazon.es/Owlet-Smart-Vigilabeb%C3%A9s-card%C3%ADaco-ox%C3%ADgeno/dp/B08CY71MZM
7. http://lightweightcrypto.org/present/present_ches2007.pdf
8. https://en.wikipedia.org/wiki/Simon_%28cipher%29
9. https://en.wikipedia.org/wiki/Speck_%28cipher%29
10. https://sites.google.com/site/photonhashfunction/design
11. https://sites.google.com/site/photonhashfunction/design
12. https://en.wikipedia.org/wiki/Trivium_%28cipher%29
13. https://tinycrypt.wordpress.com/2017/03/22/asmcodes-lightmac/