Cypher as a Service

Kurosh Dabbagh Escalante

Álvaro Gómez-Arevalillo González

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

TRABAJO DE FIN DE GRADO

Madrid, 20 de Junio 2018

Director: José Luis Vázquez-Poletti

Agradecimientos

Queremos agradecer a todas las personas que nos han apoyado durante estos años en la universidad. A nuestras familias y a nuestros compañeros y amigos que hemos hecho en la facultad. A nuestros profesores con una mención especial a nuestro director de TFG José Luis Vázquez Poletti, el cual nos metió en el mundo de la seguridad informática y ha sido uno de los motivos por los que hemos desarrollado este trabajo.

¡Gracias por darnos el apoyo necesario para conseguirlo!

Prólogo

Este proyecto se inició a mediados del curso 2016/2017, cuando decidimos formar un grupo para el Trabajo Fin de Grado de cara al curso siguiente y le comunicamos a José Luis que nos encantaría tenerle como tutor. Al haber sido nuestro profesor en Redes y Seguridad queríamos hacer algo que estuviera de alguna manera relacionada con la ciberseguridad, y a la vez integrase la computación en la nube, que es una de las áreas maestras de José Luis.

Invertimos varias semanas en discutir y explorar diversos campos que podíamos tratar en nuestro proyecto hasta que finalmente llegamos a la encriptación. A parte de cumplir con los requisitos de temario que nos habíamos propuesto, la encriptación era una rama de la ciberseguridad poco tratada en las materias del Grado y a la vez una herramienta cada vez más demandada y necesaria en el mundo informático para proteger comunicaciones y recursos.

Así, y tras comunicarle nuestra decisión a José Luis y obtener su visto bueno, decidimos crear una API REST para ofrecer un servicio de cifrado en la nube.

Resumen

A día de hoy es difícil imaginarse el futuro de la tecnología informática sin los conceptos de computación en la nube y ciberseguridad. El primero nos otorga una mayor capacidad de almacenamiento y una enorme potencia de procesamiento. El segundo, a través de técnicas como la encriptación, mantiene nuestras comunicaciones y datos sensibles en secreto lejos del alcance de cibercriminales.

El objetivo de este proyecto es la creación de un servicio de cifrado en la nube a través de una API REST (Cypher as a Service) que acerque la encriptación tanto a desarrolladores como a personas sin conocimiento técnico. De esta manera, los desarrolladores tendrán una manera sencilla de encriptar sus ficheros y comunicaciones independientemente del lenguaje que estén manejando y sin tener que invertir tiempo en comprender los entresijos de la encriptación. Por otro lado, las personas sin conocimiento informático también podrán acceder a estos servicios sin tener que preocuparse por las barreras que la tecnología a menudo impone al usuario no técnico.

Palabras clave

Encriptación
Computación en la nube
API Rest
Python
AES
SHA
Plugin
Amazon EC2
Sistema Distribuido

Abstract

Nowadays is hard to think about a future of Computer Technology without the concepts of cloud computing and cyber security. Cloud gives a greater storage capacity and an enormous processing power in comparison with which we could obtain using our personal computer. Cyber security, through techniques such as encryption, keeps our sensitive data and communications from the reach of cybercriminals.

The aim of this project is the creation of an encryption service in the cloud through an API REST that brings the encryption closer to both developers and people without technical knowledge. In this way developers will have a simple way to encrypt their files and communications regardless of the language they are coding and without need to invest a lot of time to understand the intricacies of encryption. Non-expert people would be also allowed to use that service without the worries about the limits of knowledge in the field of encryption they could had.

Keywords

Encryption
Cloud computing
API Rest
Python
AES
SHA
Plugin
Hash
Amazon EC2
Distributed system

Índice

1. Introducción

Desde que el ser humano desarrolló el lenguaje como medio de comunicación entre individuos tuvo la necesidad de mantener el mensaje oculto a todas aquellas personas ajenas al destinatario del mismo. Así, aunque ya en el Antiguo Egipto, la India o Grecia se utilizaban métodos primitivos de encriptación es al emperador romano Julio César al que se le atribuye el primer método de encriptado con documentación y reglado. Este primer método de encriptación se conoce como el cifrado César o por desplazamiento y consiste en sustituir todas las apariciones de una letra en un texto por otra letra que se encuentre a un número determinado de posiciones por delante de ella en el alfabeto.

En la actualidad, la encriptación nos proporciona conexiones seguras y privadas a través de Internet por medio de protocolos como SSL, SSH o conexiones VPN, pero además también sirve para proteger ficheros y datos almacenados, siendo así un potente aliado a la hora de minimizar el daño en caso de sufrir un ciberataque. De manera muy general, podemos clasificar los algoritmos de cifrado según sean de flujo o de bloque. Estos primeros actúan modificando el mensaje original bit a bit y son sobre todo utilizados en la protección de comunicaciones, mientras que el cifrado por bloques opera en grupos de bits de longitud fija y se utilizan mayoritariamente en la encriptación de ficheros.

Cabe dedicar un espacio en esta introducción al cloud computing. Este paradigma permite ofrecer servicios de computación y almacenamiento a través de Internet. Se especula sobre si la idea de este paradigma fue desarrollada por JCR Licklider o por John Mccarthy, pero lo cierto es que fueron gigantes tecnológicos como Google, Amazon o Microsoft los que comenzaron a implementar y ofrecer este servicio allá por el año 2006. A día de hoy podemos encontrar este paradigma en servicios tan utilizados como Amazon Web Services, Dropbox o Microsoft Azure debido a la enorme potencia de computación y a la virtualmente inagotable capacidad de almacenamiento que nos ofrece a un precio muy reducido. Hoy en día es raro encontrar una gran empresa que no tenga servicios alojados en la nube y en los próximos años se pronostica la continuidad de esta tendencia al alza.

1.1. Motivación

La encriptación es un mecanismo de seguridad y privacidad cada vez más recurrente y demandado a la hora de establecer conexiones o intercambiar mensajes a través de Internet. A pesar de ello y debido a la complejidad inherente a los algoritmos de cifrado, la encriptación sigue siendo un recurso poco accesible para los desarrolladores y sobre todo para el usuario más común que es el no técnico.

Para paliar esta escasez de encriptación, en 1991 Philip R. Zimmermann desarrolló PGP (Pretty Good Privacy), software que incluso hoy en día sigue siendo muy utilizado. PGP es un software que utiliza un sistema de clave pública para cifrar ficheros

y que incluso puede ser incorporado a los sistemas de mailing para encriptar correos electrónicos. Aun así, el tiempo no se detiene [8] y PGP se ha demostrado obsoleto para los tiempos que corren, con problemas sin solucionar entre versiones, fallos de seguridad producto de la reutilización de claves, excesiva dificultad para integrar el cifrado de correos y la imposibilidad de generar nuevos pares de claves entre otras muchas cosas, lo que nos lleva a la necesidad de generar una alternativa actualizada, accesible, potente y segura.

1.2. Objetivos

El objetivo principal del proyecto es desarrollar un servicio distribuido en la nube que sea potente y seguro a la hora de cifrar ficheros y comunicaciones y que a través de una API REST, pueda ser utilizado por la comunidad de desarrolladores sin importar en qué lenguaje están desarrollando sus aplicaciones. Posteriormente decidimos implementar también un plugin para Chrome que haciendo uso de la API acerque la encriptación a usuarios no técnicos.

Para ofrecer un servicio mejorado con respecto a PGP, se considera que este servicio de encriptado ha de cumplir los siguientes requisitos:

- **Seguridad:** Las funciones hash y los algoritmos de cifrado utilizados han de ser irrompibles en la actualidad e implementarse con todas las garantías de privacidad de cara al usuario.

- **Rapidez:** El objetivo de utilizar un servicio distribuido para realizar el proceso de cifrado en la API es el de otorgar la mayor rapidez al servicio dentro de las limitaciones propias de un proceso de cifrado.

- **Fácil gestión de claves:** A diferencia de PGP, las claves de cifrado deberán tener un formato fácil de manejar y un tamaño reducido sin que ello genere fallos de seguridad. Además, se deberá evitar por completo la reutilización de claves como ocurre en PGP.

- **Perfect forward secrecy:** El servicio deberá garantizar que en caso de que un cliente sufra un robo de sus claves de cifrado la integridad y la privacidad de todo lo cifrado con anterioridad (y por supuesto, con posterioridad) no se verán comprometidas.

- **Modularidad:** El servicio deberá ser modular, permitiendo que se vayan creando nuevas funcionalidades según vayan apareciendo nuevas necesidades.

1.3. Estado del arte

Realizando una búsqueda en los principales motores de búsqueda sobre aplicaciones o herramientas que se encarguen de hacer funciones similares a nuestro proyecto se encontraron diversas opciones. Entre ellas destacamos Boxcrypto [9], Peak10 [10] o GNUPG [11].

La mayoría de estas herramientas son aplicaciones de escritorio que sirven básicamente para encriptar ficheros en un solo dispositivo. Al no ser un servicio disponible en la nube nos encontramos con que en muchas ocasiones los ficheros son imposibles de descifrar al llevarlos a otro ordenador, ya sea por incompatibilidades del sistema operativo o porque las claves de cifrado no están al alcance del usuario. Por otro lado, al ser aplicaciones de escritorio que encriptan en el mismo sistema en el que se ejecutan, no cuentan con una API que permita integrar la encriptación en el código de los desarrolladores (a excepción de GNUPG).

API REST

REpresentational State Transfer [12] es un enfoque que permite la creación de una interfaz entre sistemas utilizando HTTP para el intercambio de datos o para realizar operaciones sobre los mismos. Los formatos más utilizados en esta tecnología para el transporte de datos son JSON y XML, aunque no son los únicos. Así, una API REST es un conjunto de servicios o funcionalidades que se ponen a nuestra disposición a través de la web y a los cuales se accede haciendo uso de los métodos de HTTP: POST, GET, PUT o DELETE. Estos métodos equivalen a escribir, leer, modificar o borrar un recurso.

ENCRIPTACIÓN

La encriptación consiste en la modificación de un mensaje a través de un algoritmo determinado para ocultar su contenido a toda aquella persona que no posea el secreto. Cuando hablamos del secreto nos referimos a un elemento que permite la desencriptación del mensaje y que, por lo tanto, ha de ser guardado con celo. En el caso de AES-256 el secreto es una clave simétrica de 32 bytes.

AES

Son las siglas de **Advanced Encryption Standard,** el algoritmo de cifrado en bloque con clave simétrica más utilizado en la actualidad. Existen diferentes versiones del mismo que permiten modificar la longitud de la clave utilizada. Se llama cifrado en bloque porque no encripta bit a bit como ocurre en el cifrado de flujo sino que opera sobre bloques de una longitud predeterminada, en este caso 16 bytes.

FUNCIÓN HASH

También conocida como función resumen recibe un conjunto de caracteres y los convierte en una cadena única (si la función no contiene colisiones) de tamaño prefijado. También es un método muy utilizado para cifrar información pero al contrario que la encriptación el resultado de las funciones hash es irreversible, es decir, si el algoritmo es robusto no se puede volver a la cadena original.

SHA

Es la familia de funciones hash más utilizada en la actualidad, y comprende diversas versiones entre las que se encuentran SHA-1, SHA-256 o SHA-512.

SIGNAL

Es un protocolo de encriptado que proporciona cifrado extremo a extremo. Fue desarrollado por Open Whisper Systems en 2013 y pronto se ha convertido en uno de los protocolos más utilizados (si no el que más) para el cifrado de comunicaciones. Está presente en la aplicación a la que da nombre, Signal [7], pero también recientemente ha sido implementado en WhatsApp.

PGP

Son las siglas de Pretty Good Privacy, fue desarrollado por Phil Zimmermann y ha sido uno de los softwares de encriptación de clave pública más utilizados desde 1991 hasta la actualidad.

1.4. Plan de trabajo

El plan de trabajo que se ha seguido para realizar este trabajo de investigación y desarrollo es el siguiente. Se ha dividido el trabajo en fases como se realiza un proyecto software, teniendo en cuenta el objetivo final de presentación del proyecto y de la memoria como resultados para una mejor organización de los recursos y el tiempo disponible.

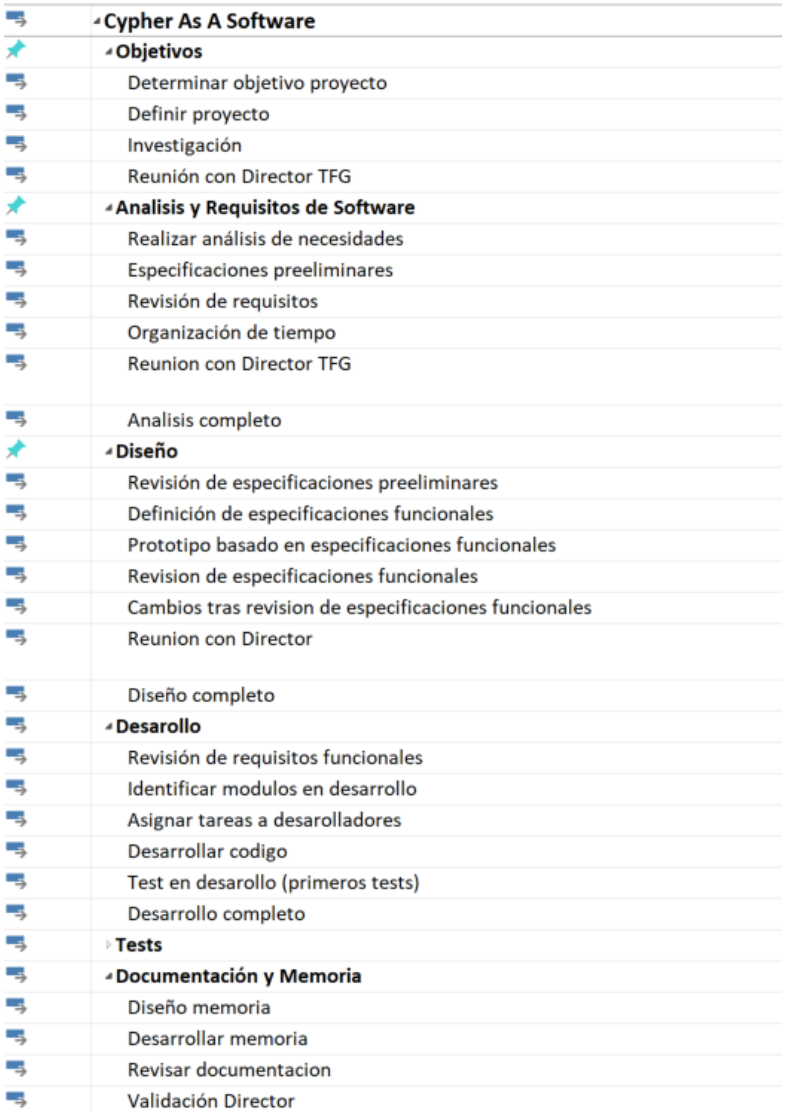


Fig. 1.Esquema de desarrollo de software que se ha utilizado durante el proyecto.

1. Introduction

Since the moment that the human developed the language as a communication method between persons, it was a need to keep the message hidden from anyone that is not the final recipient of it. Even though the primitive methods of encryption were already used in the ancient Egypt, India and Greece, was the Roman Emperor Julius Caesar who is credited with the first method of encryption ruled and documented. This first kind of encryption is known as Caesar Cipher, or shift cipher and consists of replacing all occurrences of a letter in a text with another letter that is at a certain number of positions ahead of it in the alphabet.

At present, encryption provides us with secure and private connections over the Internet through protocols such as SSL, SSH or VPN connections, but also serves to protect files and stored data, being a powerful ally in helping to minimize the impact in case of a cyber-attack happens. In a general sense, we can classify the encryption algorithms between Stream and Block. These first act by modifying the original message bit by bit and are mostly used in the protection of communications, while the encryption by blocks operates in groups of fixed-length bits and is used mostly in file encryption.

Computers Technology has also evolve noteworthy, giving to the cloud computing a very important slot in order to keep growing. This paradigm allows to offer computer and storage services through the Internet. It is speculated about the idea of this paradigm was developed by JCR Licklider or by John Mccarthy, but the truth is that it was technological giants such as Google, Amazon or Microsoft whom began to implement and offer this service back in 2006. Up until now we can find this paradigm in services such as Amazon Web Services, Dropbox or Microsoft Azure because of the enormous computing power and the virtually inexhaustible storage capacity that offers us at a very low price. Nowadays it is odd to find a large company that does not have services hosted in the cloud and in the next few years the continuity of this upward trend is forecast.

1.1. Motivation

Encryption is a security and privacy mechanism that is increasingly recurrent and demanded when establishing connections or exchanging messages over the Internet. Despite this, and due to the inherent complexity of encryption algorithms, encryption remains a resource that is not accessible to developers and, above all, to the most common user, which is the non-technical one.

To alleviate this shortage of encryption, in 1991 Philip R. Zimmermann developed PGP (Pretty Good Privacy), software that even today is still widely used. PGP is a software that uses a public / private key system to encrypt files and that can even be incorporated into the mailing systems to encrypt emails. Even so, time does not stop [8]

and PGP has proved obsolete for the times, with problems without being solved between versions, security failures due to the reuse of keys, excessive difficulty in integrating mail encryption and the impossibility of generating new key pairs among many other things, which leads us to the need to generate an updated, simple, accessible to all and powerful alternative.

1.2. Goals

The aim of the project is to develop a distributed service in the cloud that is powerful and secure when it comes to encrypting files and communications and that through an API, can be used by the developer community no matter in which language they are developing their applications. Lastly we decided to also implement a Chrome plugin that, using the API, approached encryption to non-technical users.

To offer an improved service with respect to PGP, we consider that our encryption service must meet the following requirements:

- **Security:** The hash functions and the encryption algorithms used must be currently unbreakable and implemented with all the guarantees of privacy for the users.

- **Speed:** The objective of using a distributed service to perform the process of encryption in the API is to provide the fastest service within the limitations of an encryption process.

- **Easy key management:** In contrast to PGP, encryption keys should have an easy-to-manage format and a small size without causing security failures. The reuse of keys as in PGP should be completely prevented besides.

- **Perfect forward secrecy:** The service must guarantee that in case a client suffers a loss of their encryption keys, the integrity and privacy of everything that has been encrypted beforehand (and, of course, subsequently

- **Modularity:** The service should be modular and allow new sections and utilities to be created as the needs grow.

2. Tecnologías

A continuación se detallan las tecnologías que se han empleado durante el desarrollo de Cypher As A Service, tanto para el servicio distribuido como para el plugin de Chrome.

2.1. Python

Python fue creado a finales de los ochenta, como sucesor de ABC, un lenguaje de programación que aparece como alternativa a BASIC, aunque no fue publicado por su creador hasta 1991 [13].

Python es un lenguaje de programación interpretado multiparadigma [14]. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo de la programación orientada a objetos, pero también soporta programación imperativa y programación funcional. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de este un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

A la hora de desarrollar en Python se han usado los siguientes módulos:

2.1.1. Pycrypto

Es una librería que soporta funciones para cifrado por bloques, cifrado por flujo y cálculo de hash. Soporta cifrados por bloque como AES, BLOWFISH, DES3 y también de flujo como ARC3 o XOR [15]. Además de esto, su interfaz proporciona diversas funcionalidades vitales a la hora de trabajar con encriptación. En nuestro caso particular, se han utilizado la generación de números y cadenas de bytes aleatorios, ambos en el módulo Random. Además, para verificar la integridad de los ficheros a descifrar así como para comprobar el destinatario del archivo encriptado se han utilizado las funciones resumen que nos otorga el módulo HMAC.

2.1.2. Flask

Flask es un micro framework que permite crear aplicaciones web de manera sencilla. Este framework es micro no porque necesariamente la aplicación tenga que ser un solo archivo, si no que Flask lo que pretende es tener un núcleo de aplicación simple, pero que sea extensible o modular. Por esta razón, este framework es de los más utilizados en este momento, dado que incluye lo básico para hacer funcionar una aplicación [16]. Una vez creada la aplicación básica, la cual no incluye por defecto funcionalidades como integración de base de datos de forma abstracta o manejo de cargas entre otros, te da la opción de incluir este tipo de extensiones como si el mismo las integrará, según las necesidades de la aplicación en desarrollo.

Haciendo uso de esta estructura modular, se han integrado en el proyecto varias de las funcionalidades adicionales que ofrece Flask:

2.1.2.1. HttpAuth

Implementa autenticación básica para HTTP.

2.1.2.2. SqlAlchemy

Framework que permite implementar un sistema ORM sobre SQL.

2.1.2.3. Bson

Módulo que permite utilizar el formato de fichero BSON propio de las bases de datos NoSQL.

2.1.3. Passlib

Una librería que proporciona implementaciones multiplataforma de más de 30 algoritmos para hashear contraseñas, así como un framework para administrar hashes de contraseñas existentes [17].

Los hasheos de cadenas sensibles como contraseñas u otro tipo de secretos son bastante comunes en todas las aplicaciones dado que dificultan a los posibles atacantes que quieran robar información o estén "escuchando" en alguna red comprometida la obtención de la palabra hasheada.

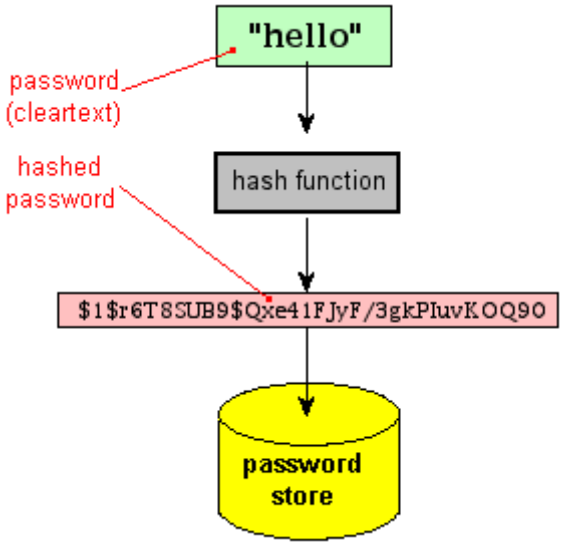


Fig. 2. Ejemplo gráfico del hasheo de una palabra. La función hash recibe como entrada el texto original y lo transforma en una cadena de caracteres irreconocible. Fuente: http://www.trafficsteed.com/why-hash-the-passwords/

2.1.4. Paramiko

Es una implementación del protocolo SSH v2 para Python que ofrece la funcionalidad de cliente y servidor. Actúa como una interfaz sobre los conceptos de las conexiones SSH [18].

Es una de las opciones preferidas a la hora de implementar en Python conexiones sobre SSH debido a que ofrece una enorme gama de posibilidades al desarrollador. Así, se puede implementar redirección de puertos local y remota, conexiones a través de credenciales o usando claves privadas, autenticación de la clave pública del otro extremo, etc.

2.1.5. PyMysql

Este paquete permite realizar transacciones con bases de datos MySQL.

2.2. Mysql

El desarrollo de MySQL comenzó en 1994, aunque no fue hasta mediados de 1995 cuando se lanzó la primera versión. Inicialmente fue creado para uso personal desde mSQL basado en ISAM, un lenguaje de bajo nivel, lo cual los desarrolladores consideraron que era difícil de aprender y poco flexible. Entonces crearon una nueva interfaz SQL y mantuvieron la misma API que mSQL, lo que permitió que los desarrolladores pudieran realizar sus aplicaciones usando MySQL en lugar del software propietario de mSQL [19].

Es la base de datos relacional open-source más popular [20]. Además de su buen rendimiento es destacable también que es muy sencillo tanto de configurar como de usar.

El modelo entidad-relación como el que usa MySQL permite que cada una de las tablas de nuestra base de datos, aparte de almacenar su información propia, pueda relacionarse con la información almacenada en otras tablas, favoreciendo la integridad de la base de datos y las consultas a la misma. De esta forma se forma un modelo relacional que es el que da nombre a estas bases de datos [21].

En la investigación de MySQL se observa que las consultas son muy rápidas, pero que en aplicaciones con alta concurrencia en modificación de datos puede haber problemas de integridad, por lo que es algo a tener en cuenta de cara a un futuro crecimiento de la aplicación.

Para mejorar estos problemas de integridad se escoge el mecanismo de almacenamiento InnoDB, soportado a partir de la versión 4.0 de MySQL. La característica más importante de este motor es que realiza transacciones de tipo ACID (**A**tomicity, **C**onsistency, **I**solation and **D**urability) y bloqueo de registros e integridad referencial. Por ello, ofrece más fiabilidad y consistencia de datos respecto al motor de base de datos MyISAM que nos ofrecería mayor rapidez si nuestra base de datos sólo fuese a ser utilizada para consulta de información [22].

La licencia de MySQL es del tipo GNU GPL (**GNU General Public License**), lo que obliga a las aplicaciones que lo utilicen a que tengan la misma licencia.

2.3. Apache

El software de apache siempre es un referente a la hora de realizar nuevos proyectos, debido a que su gran comunidad de desarrolladores disponen de buena documentación y software seguro, eficiente y de libre acceso al código. En el proyecto se han utilizado:

2.3.1. HTTP server

Basado en el servidor NCSA HTTPd, el servidor Apache ser empezó a desarrollar a principios de 1995, cuando se estancó el desarrollo de NCSA. Apache formó un papel determinante en el crecimiento de la World Wide Web y rápidamente adelantó a su predecesor como servidor dominante HTTP, y se ha mantenido como el más popular desde abril de 1996 [23]. Está estimado que sirve el 47% de todas las web activas.

Este servidor es altamente configurable, dando opciones para cambiar fácilmente el balanceo de carga, compatibilidad con IPv4 e IPv6, denegación de acceso a directorios según los permisos del usuario y comunicaciones encriptadas entre otras características que pueden ser muy costosas en otros servidores.

El servidor HTTP de apache es un producto modular y muy popular que ofrece buen rendimiento. Sus principales características son que es multiplataforma, gratuito, muy robusto y que destaca por su seguridad y rendimiento.

El servidor consta de una sección core y diversos módulos que aportan mucha de la funcionalidad que podría considerarse básica para un servidor web. En nuestro caso, se han utilizado los siguientes módulos:

2.3.1.1. SSL

Este módulo provee de los certificados SSL v3 y TLS v1 al servidor HTTP para darle una capa de seguridad (Fig. 3), evitando que las peticiones realizadas al servidor puedan ser robados o modificados por posibles atacantes [24].

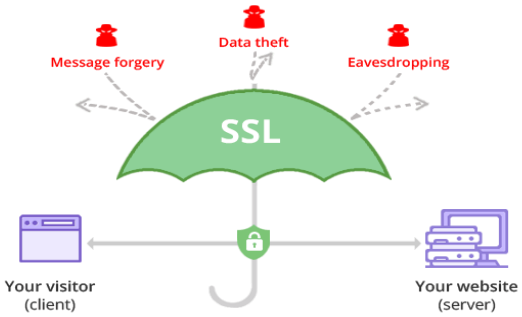


Fig. 3 Gráfico explicativo de la función de SSL. Fuente: https://www.hostinger.com/buy-cheap-ssl-certificate

2.3.1.2. WSGI

Web Server Gateway Interface es una tecnología que permite a un servidor web retransmitir las peticiones HTTP que recibe a otras aplicaciones escritas en Python (Fig. 4). Estas aplicaciones realizan las operaciones solicitadas y devuelven el resultado al servidor para que pueda comunicar la respuesta al cliente a través de protocolos web [25].

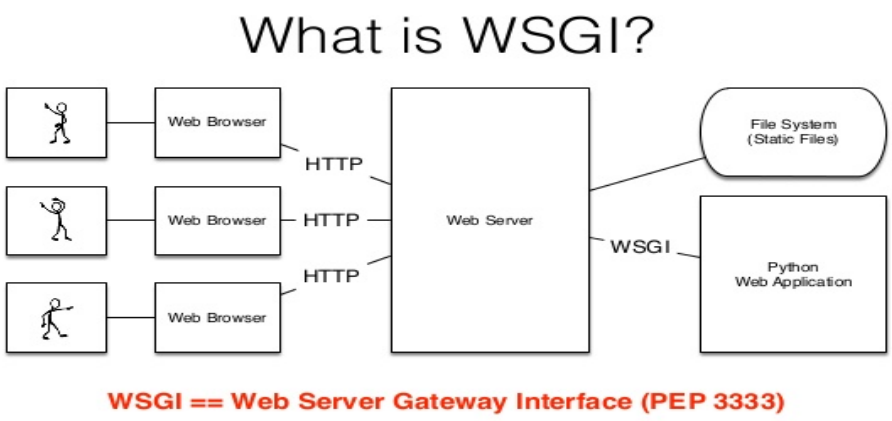


Fig. 4. Gráfico que muestra cómo queda enlazada la aplicación a través de la interfaz wsgi. Fuente: https://www.slideshare.net/GrahamDumpleton/secrets-of-a-wsgi-master

2.3.2. ApacheBench

El benchmark es una técnica que se utiliza para comprobar cuál es el rendimiento de algo que se está probando, ya sea un sistema, un servicio o una aplicación. Esto consiste en la ejecución de un conjunto de pruebas que permiten obtener el comportamiento del sujeto sometido a pruebas y así obtener la mejor configuración para el mismo [26].

Para ello existen varios programas capaces de realizar una batería de peticiones o pruebas y comprobar las respuestas. En nuestro caso se ha utilizado

ApacheBench, que es un programa monohilo que se lanza a través de consola y que se utiliza para medir el rendimiento de los servidores HTTP. Su uso es bastante sencillo y permite lanzar peticiones y escoger la concurrencia para comprobar las condiciones de carrera y bloqueos en la respuesta del servidor. Con ello se obtienen resultados que son más similares a lo que se podría dar en un entorno real y que permiten valorar la potencia de las tecnologías elegidas.

2.4. Amazon Web Services

Amazon Web Services es una plataforma de pago de servicios en la nube que proporciona tanto almacenamiento virtual (Dropbox) como servicios de computación (EC2). Se puede acceder de manera gratuita a un reducido conjunto de servicios durante un tiempo limitado y con restricciones tanto en el tráfico que tus máquinas pueden soportar, la capacidad de almacenamiento o de memoria RAM, etc.

Fig. 5 Algunos de los servicios en cloud que son ofrecidos por Amazon Web Service. Fuente: https://computationalmodelling.bitbucket.io/tools/AWS.html

AWS fue lanzado en 2006 utilizando la red global de alta capacidad de Amazon para ofrecer una alternativa barata y potente a todas aquellas personas o entidades que hasta entonces mantenían una costosa red privada física. Entre otros, en la actualidad nos encontramos con los siguientes servicios:

- **EC2**: Elastic Compute Cloud es un servicio para lanzar máquinas virtuales en el hardware de Amazon, desde instancias individuales hasta extensas redes con potentes máquinas.

- **S3**: Simple Storage Service provee almacenamiento persistente, redundante y con copia de seguridad que es accesible tanto desde EC2 como a través de una interfaz web.

- **Aplicaciones para gestión de bases de datos**: Productos como DynamoDB, ElasticCache o Redshift proveen diversos métodos para acceder y monitorear bases de datos optimizadas para diversas actividades.

- **Otros**: Servicios de email, numerosas APIS que hacen uso del cloud computing, servicios de balanceo de carga, etc.

Utilizando el servicio gratuito de AWS se puede acceder a EC2 y , a pesar de las grandes limitaciones, podemos hacernos una idea del funcionamiento de la aplicación en la nube y gestionar sus futuras implementaciones conforme a los resultados.

2.5. Jquery

JavaScript es un lenguaje interpretado de programación, orientado a objetos y débilmente tipado, utilizado principalmente en la parte cliente de una aplicación o página web [27]. Dota a estas mismas de dinamismo, permitiendo ejecutar funciones como por ejemplo llamadas a servidores para obtener información y mostrarla, modificación de los objetos que forman el DOM (Document Object Model) entre otras.

Jquery es una biblioteca multiplataforma basado en Javascript que fue lanzada por sus desarrolladores en 2006 y que permite simplificar la forma de interactuar con el árbol DOM.[28] Con Jquery, las líneas de código se reducen, obteniendo así un código más limpio, que se ejecuta en menos tiempo y que para el que se necesita menos espacio.

2.6. AJAX

El término AJAX (Asynchronous JavaScript And XML) fue creado en 2005, pero las tecnologías que aceptan el uso de ella se pueden remontar hasta diez años antes con el uso del Scripting Remoto de Microsoft.

Es una tecnología asíncrona con la que se pueden realizar peticiones para obtener o mandar información a un servidor web en segundo plano, mientras se pueden realizar otras tareas de visualización o comportamiento de la web sin que se vea perjudicado el rendimiento. Las peticiones se realizan a través de la sintaxis de Javascript o Jquery, tecnologías descritas anteriormente.

3. Arquitectura

Este apartado se centrará en discutir y argumentar las diferentes decisiones de diseño que se han ido tomando a lo largo del proyecto hasta llegar a la estructura final.

3.1. Framework

Una vez decidido que el proyecto iba a implementar una API REST, lo más razonable es utilizar un framework para aplicaciones web y así no perder demasiado tiempo en la fontanería interna que requiere este tipo de aplicaciones. En el caso de Python, existen diferentes opciones (Pyramid, Tornado) pero las dos más populares y potentes son Flask y Django.

Django [28] es un framework para aplicaciones web de código abierto escrito en Python. Dentro de este ámbito es la opción más utilizada para desarrollar proyectos web de gran envergadura debido a la enorme cantidad de componentes y funcionalidades que vienen integrados en el mismo (Fig. 6). Esta carga extra de implementación lo hace más pesado y con una curva de aprendizaje mucho más elevada que otros frameworks, pero es ideal para proyectos grandes que tengan un nivel de complejidad medio o alto. Cuando instalas Django recibes no solo una interfaz web, sino todos los componentes [29] que puedas necesitar para montar un servicio potente (autenticación, chat, anti-spam, analytics, etc).

Tan potente es Django que podemos encontrarlo en webs tan populares y concurridas como YouTube, DropBox, Google o Instagram, lo cual nos da una idea del tipo de software que tenemos entre manos. Si bien todo esto es cierto, no lo es menos que su curva de aprendizaje puede ser demasiado elevada para proyectos pequeños y que al ser tan pesado requiere invertir más recursos del servidor para suministrar los servicios. En nuestro caso se busca que la parte web simplemente sea una interfaz, una puerta de comunicación con el servicio de cifrado, y no se pretende dotarla de una excesiva funcionalidad. Si bien Django nos ofrece muchos de los componentes que necesitamos, como autenticación básica o un mapeo de bases de datos a través de ORM, se busca poder elegir qué módulos integrar y cuáles no y eso no es posible con Django.

Authentication

This is a grid of all packages for user authentication.

Features currently being evaluated

Feature
Types
Auto create users

PACKAGE	DJANGO REST FRAMEWORK	DJANGO-TASTYPIE	DJANGO-ALLAUTH	PYTHON SOCIAL AUTH	DJANGO SOCIAL AUTH (DEPRECATED, USE PYTHON-SOCIAL-AUTH)	DJANGO-FACEBOOK	DJANGO-GUARDIAN
Description	Web APIs for Django.	Creating delicious APIs for Django apps since 2010.	Integrated set of Django applications addressing authentication, registration, account management as well as 3rd party (social) account authentication.	Social auth made simple	Django social authentication made simple	Facebook open graph api implementation using the Django web framework in python	Per object permissions for Django
Category	App	App	App	App	App	App	App
# Using This	169	84	68	19	45	14	31
Python 3?	✔	✔	✔	✔	✖	✔	✔
Development Status	Production/Stable	Beta	Beta	Beta	Beta	Production/Stable	Production/Stable
Last updated	July 19, 2016, 1:39 p.m.	July 22, 2016, 11:40 a.m.	July 20, 2016, 2:57 p.m.	July 18, 2016, 4:56 p.m.	March 30, 2015, 1:14 p.m.	June 22, 2016, 3:23 a.m.	July 17, 2016, 5:54 p.m.
Version	3.3.3	0.13.3	0.26.1	0.2.19	0.7.28	6.0.3	1.4.4

Fig. 6 Diferentes módulos de autenticación integrados en Django. Fuente: https://www.codeschool.com/blog/2016/09/07/use-django-packages-dont-reinvent-the-wheel/

Por su parte, Flask es un microframework en Python para aplicaciones web y con licencia BSD que se caracteriza por ser liviano y fácil de aprender. Tiene un core básico que se sustenta en Werkzeug [30] y Jinja 2 [31] para proveer sus servicios más esenciales, pero además cuenta con una gran comunidad de desarrolladores que implementan módulos independientes. Estos módulos son integrables a placer según las necesidades de tu proyecto, lo cual evita que tengas que instalar un sistema enormemente pesado del cual no uses ni la mitad de las funcionalidades que trae consigo.

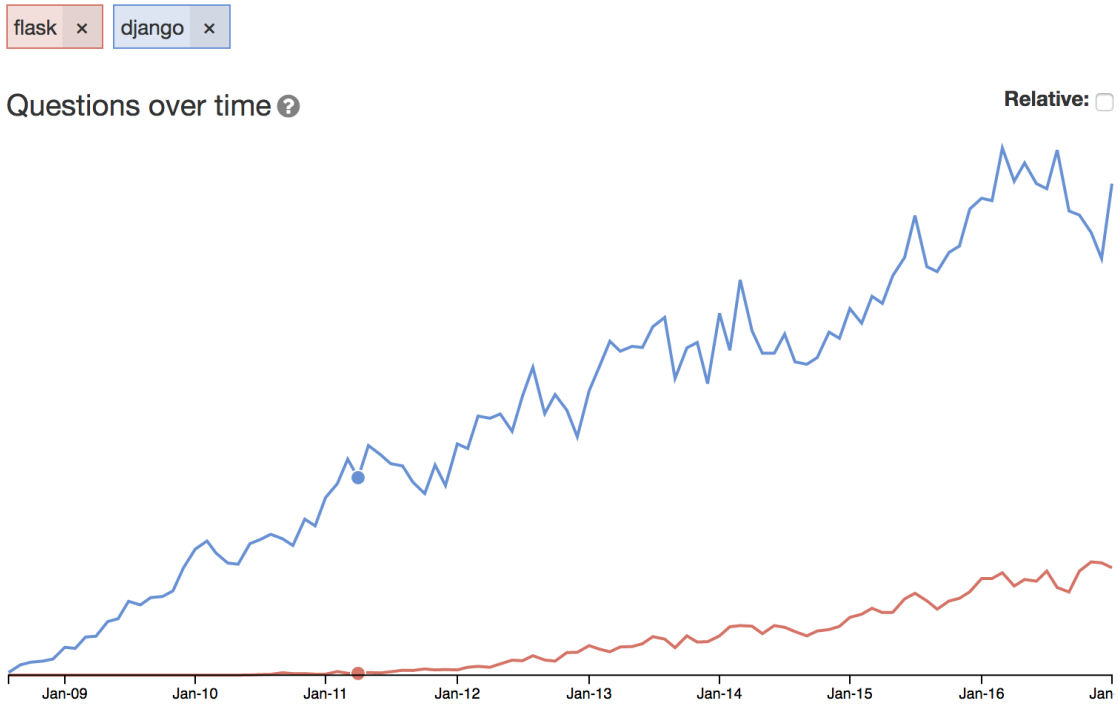


Fig. 7 Evolución del número de preguntas existentes sobre Django y Flask en StackOverflow. Este auge de preguntas coincide con la popularidad de ambos frameworks.
Fuente:https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v

Por la manera en la que está desarrollado Flask se suele usar para la implementación de microservicios, por eso no es tan sencillo saber qué grandes compañías lo utilizan. Los microservicios son aplicaciones modularizadas que se comunican entre ellas a través de HTTP y dentro de esta arquitectura encontramos las API REST, por lo que parece adecuado para nuestro proyecto.

Que ambos frameworks están en auge parece claro (Fig. 7), pero mientras Django es un software para proyectos de gran envergadura, con numerosos equipos de trabajo y experimentados tanto en Python como en aplicaciones web Flask nos ofrece una funcionalidad que poco tiene que envidiar y con una curva de aprendizaje más asequible. En nuestro caso, al carecer de experiencia en el lenguaje necesitamos una opción potente pero sencilla, que con pocas líneas de código nos permita generar los endpoints de nuestra API y además nos ofrezca variedad y modularidad, por lo que Flask es finalmente el software elegido.

3.2. Comunicación con la API

A la hora de implementar una API REST es necesario decidir qué formatos va a soportar la aplicación para recibir los datos que los clientes envían. En la mayoría de las APIS, lo que se envía suelen ser solo cadenas de texto que permiten elegir entre varias opciones, pero en nuestro caso además también hay que tener en cuenta que se

enviarán datos binarios en crudo pertenecientes a los ficheros que se quieran encriptar o desencriptar.Al estar implementada sobre HTTP son tres los formatos que se suelen utilizar para enviar datos a una API REST: JSON, XML y YAML.

3.2.1. XML

XML responde a *extensible markup language* [32] y como tal es un lenguaje de marcas parecido a HTML creado para almacenar e intercambiar datos. Es un lenguaje que como tal no realiza ninguna función más allá de servir como contenedor de datos. Su principal ventaja es que permite la utilización de etiquetas no definidas en sus estándares, por lo que los datos pueden ser almacenados de una manera sencilla de entender para otras personas y autodescriptiva. Por el contrario, comparado con otros formatos de almacenamiento de información es lento y pesado, además de que su tratamiento suele requerir una mayor cantidad de software y procesamiento.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE recipe PUBLIC "-//Happy-Monkey//DTD RecipeBook//EN"
"http://www.happy-monkey.net/recipebook/recipebook.dtd">

<recipe>

    <title>Peanut-butter On A Spoon</title>

    <ingredientlist>
        <ingredient>Peanut-butter</ingredient>
    </ingredientlist>

    <preparation>
        Stick a spoon in a jar of peanut-butter,
        scoop and pull out a big glob of peanut-butter.
    </preparation>

</recipe>
```

Fig. 8 Ejemplo de fichero XML y de cómo permite cualquier tipo de etiquetas. Fuente: https://en.wikipedia.org/wiki/Markup_language

3.2.2. YAML

A pesar de que cada vez se utiliza más, *YAML Ain't Markup Language* [33] es un formato de transporte de datos que en seguida descartamos debido a la escasez de bibliotecas y documentación existentes para su procesamiento. Básicamente es un lenguaje de serialización de datos creado con la idea de ser fácil de leer para las personas y potente computacionalmente hablando. Se basa en la idea de que todos los datos se pueden plasmar en forma de listas, arrays o tablas hash

3.2.3. JSON

Javascript Object Notation [34] posiblemente sea el formato más utilizado a la hora de intercambiar información entre servicios web. Es un formato con notación poco compleja y fácil de leer para las personas, además de ser

fácilmente validado y procesado por la gran mayoría de lenguajes web existentes. Una de las mayores ventajas que tiene es que una vez procesado suele transformarse en arrays o diccionarios, las cuales son estructuras de datos muy comunes y fáciles de manejar para los desarrolladores. Por el contrario, algunas de las desventajas son que no permite añadir comentarios y la dificultad de ampliar una sección sin añadir nuevos pares clave/valor en los arrays resultantes.


```
{
    "Rail Booking": {
        "reservation": {
            "ref_no": 1234567,
            "time_stamp": "2016-06-24T14:26:59.125",
            "confirmed": true
        },
        "train": {
            "date": "07/04/2016",
            "time": "09:30",
            "from": "New York",
            "to": "Chicago",
            "seat": "57B"
        },
        "passenger": {
            "name": "John Smith"
        },
        "price": 1234.25,
        "comments": ["Lunch & dinner incl.", "\"Have a nice day!\""]
    }
}
```


Fig. 9 Ejemplo de fichero JSON. Fuente: http://www.redversconsulting.com/cobol_json_interface.php

Una vez investigados los tres lenguajes más utilizados para el transporte y el intercambio de información entre servicios web, se llega a la conclusión de que el mejor recurso para nuestro sistema es JSON. No solo es más sencillo de utilizar y tiene un enorme abanico de bibliotecas para gestionar su procesamiento en prácticamente todos los lenguajes, sino que además es mucho más liviano que XML, lo cual se traduce en una mayor velocidad de transmisión a través de Internet de los ficheros a procesar.

El problema con JSON es que no permite integrar directamente datos binarios, que en nuestro caso es esencial para recibir los ficheros que se quieran procesar. Para poder integrar datos binarios en JSON hace falta codificarlos y la manera más habitual es a través de Base64, lo cual aumenta en alrededor de un 33% el tamaño de los datos codificados. Claramente esto no es deseable pero es necesario y realmente no existe

una mejor alternativa, dado que en XML también habría que realizar este proceso de codificación y además en ficheros de tamaños reducidos apenas se nota el aumento de tamaño.

Finalmente y de cara a dar una opción más eficiente a los usuarios se decide implementar en la API el soporte del formato BSON. BSON [35] responde a Binary JSON y es un formato utilizado en las bases de datos MongoDB para el almacenamiento y transmisión de los datos. Su estructura es idéntica a la de JSON pero permite la integración en crudo y sin codificar de datos binarios evitando así incrementar su tamaño. Además, Flask tiene un módulo para procesar BSON y otros lenguajes web como Javascript ya lo implementan también, por lo que se decide soportarlo en la aplicación. De esta manera, se busca reducir los tiempos de transmisión de datos en el caso de ficheros de tamaño medio y grande.

BASE64

Es un esquema de codificación de binario a texto que representa los datos binarios mediante una cadena ASCII. A través de dos sencillos pasos consigue transformar cada tres bytes del mensaje binario original en cuatro caracteres ASCII. Finalmente se aplica si es necesario un proceso de *padding* o relleno para completar la cadena resultante y que su tamaño en bytes sea un múltiplo de tres. Para ello, se añaden al final uno o dos bytes con valor '0' que viene representado por el carácter '='.

3.3. Backend

En relación con el backend de la aplicación hay dos decisiones de diseño principales que tomar. Por una parte, qué tipo de base de datos vamos a utilizar para almacenar los datos correspondientes a los usuarios y los mecanismos de autenticación que teníamos pensado implementar. Por otro lado, de qué manera va a estar gestionado el proceso de encriptado y desencriptado de ficheros.

Empecemos con lo referente a la base de datos. En nuestro caso, el almacenamiento de información en la base de datos se refiere exclusivamente a la información de los usuarios (identificador, credenciales de acceso, etc), los tokens de autenticación de los que se hablará después y las cadenas que permiten relacionar cada

fichero cifrado con el usuario que es dueño del mismo. En ningún momento se almacenan claves de cifrado en los servidores, lo cual es requisito imprescindible para proporcionar privacidad y seguridad de que aunque nuestro servicio sea comprometido no se verá reflejado de ninguna manera en los ficheros encriptados con anterioridad. En este punto, la disyunción es si elegir bases de datos relacionales o NoSQL.

Las bases de datos NoSQL [36] son un paradigma de almacenamiento de datos que aparece para solventar los problemas de escalabilidad y rendimiento de las bases de datos relacionales. Las bases de datos NoSQL no siguen el modelo entidad-relación y utilizan modelos de almacenamiento diferentes a las tablas comúnmente utilizadas en las bbdd SQL, como puede ser pares clave-valor, mapeo de columnas o grafos. De esta manera, se pueden ejecutar en máquinas con pocos recursos sin generar cuellos de botella y con una gran escalabilidad aún con una gran cantidad de datos, y todo ello a costa de eliminar "únicamente" la relación entre datos que nos brindan las bbdd SQL.

A pesar de ser las bbdd NoSQL una opción atractiva, los datos que se almacenan en nuestra aplicación guardan una estrecha relación entre ellos, ya que normalmente cuando accedes a los datos de un usuario no solo requieres de su perfil, sino en muchos casos también sus tokens asociados así como la información referente a los ficheros encriptados. Además, se almacena sólo lo imprescindible, lo cual conforma una cantidad de datos más bien modesta que no requiere las ventajas de escalabilidad y rendimiento de las bases de datos NoSQL. Por lo tanto, se decide implementar una base de datos MySQL como la existente en tantos otros sistemas.
Por otro lado, queda la gestión del proceso de encriptación. Esto se refiere a la manera en la que se va a distribuir el proceso de cifrado para conseguir la mayor eficiencia posible sin renunciar a la seguridad.

Como la máquina donde se ejecutará la parte web de la aplicación tiene ya de por sí una gran carga de procesamiento atendiendo a las peticiones web se decide montar un sistema distribuido. Así, la máquina principal recibe las peticiones web y divide los datos a procesar en N partes iguales, cada una de las cuales se envía a una de las máquinas subordinadas que conforman el sistema distribuido. Las máquinas subordinadas reciben su porción de los datos que procesan según el tipo de petición y devuelven el resultado final al servidor principal. De esta manera se reparte la carga de trabajo entre N máquinas y se aumenta la efectividad del servicio. El número de máquinas subordinadas depende exclusivamente de los recursos disponibles dado que el algoritmo de reparto soporta cualquier número de ellas.

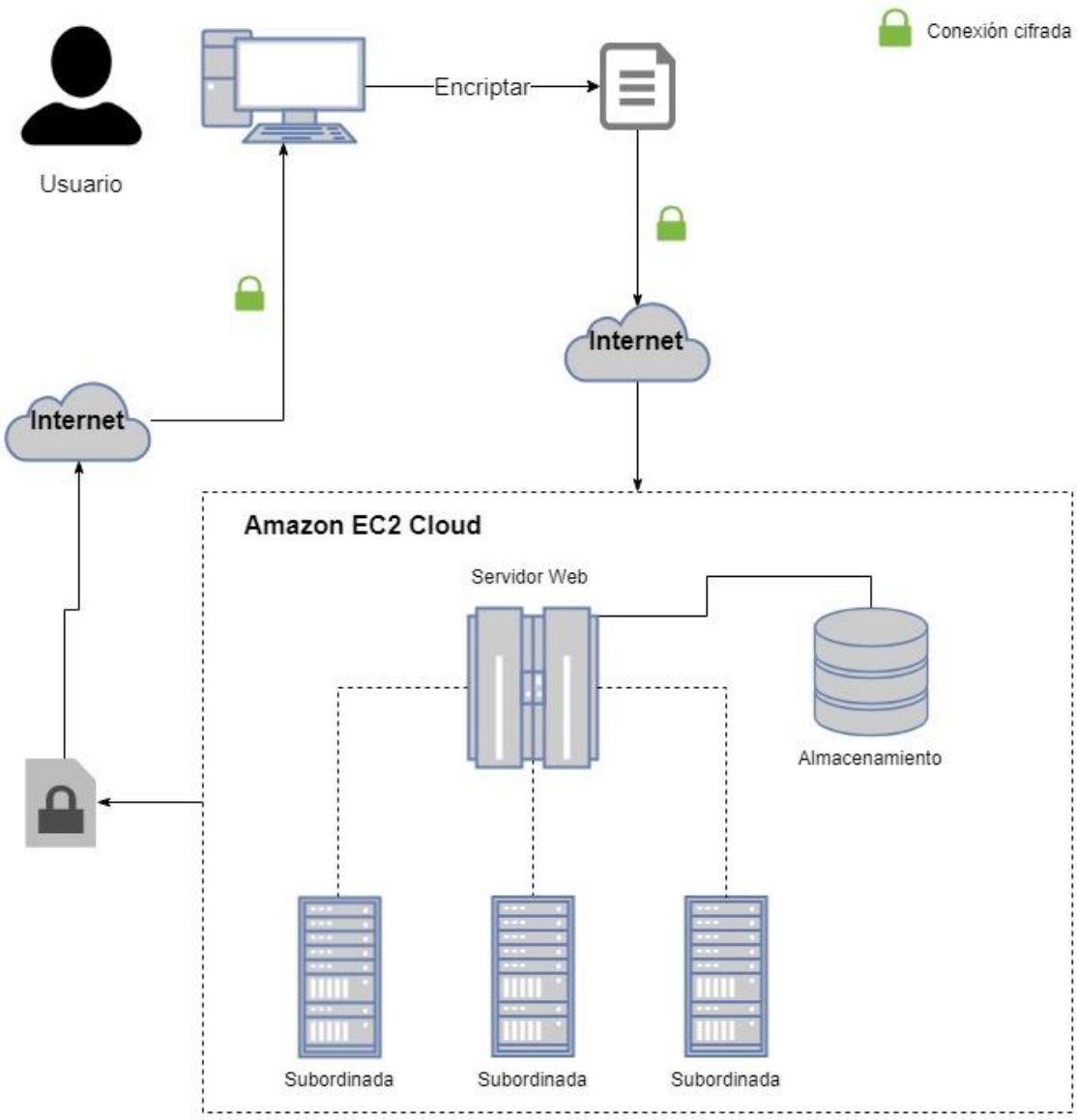


Fig. 10 Diagrama básico del proceso de encriptación. Todas las comunicaciones con la nube se realizan sobre HTTPS. El servidor web está en conexión permanente con las máquinas subordinadas a través de SSH.

Local Port Forwarding

Para conectar la máquina principal con las subordinadas se hará uso de la funcionalidad de port forwarding que ofrece el protocolo SSH. Port forwarding [37] o redirección de puertos es un mecanismo que permite conectar cliente y servidor de una aplicación cualquiera a través de un túnel SSH, el cual está encriptado. Este mecanismo es muy útil si, como en nuestro caso, se quiere conectar de manera segura a una

aplicación que corre en un puerto solamente accesible desde la red local (Fig. 11). Primero habrá que conectarse por SSH a la máquina donde se ejecuta dicha aplicación, y una vez establecida la conexión ya si será alcanzable el puerto abierto en la red local. A este mecanismo se le conoce concretamente como local port forwarding.

De esta manera, las máquinas subordinadas generan un socket escuchando en el puerto 5555 de la interfaz de red local (localhost), socket que por seguridad no será accesible desde el exterior. El servidor principal está conectado por SSH al puerto 22 de la máquina subordinada, y cuando requiere de sus servicios solicita a dicha conexión SSH ya establecida que redirija el mensaje al puerto local 5555. Este proceso se replica para cada una de las máquinas que componen el servicio distribuido.

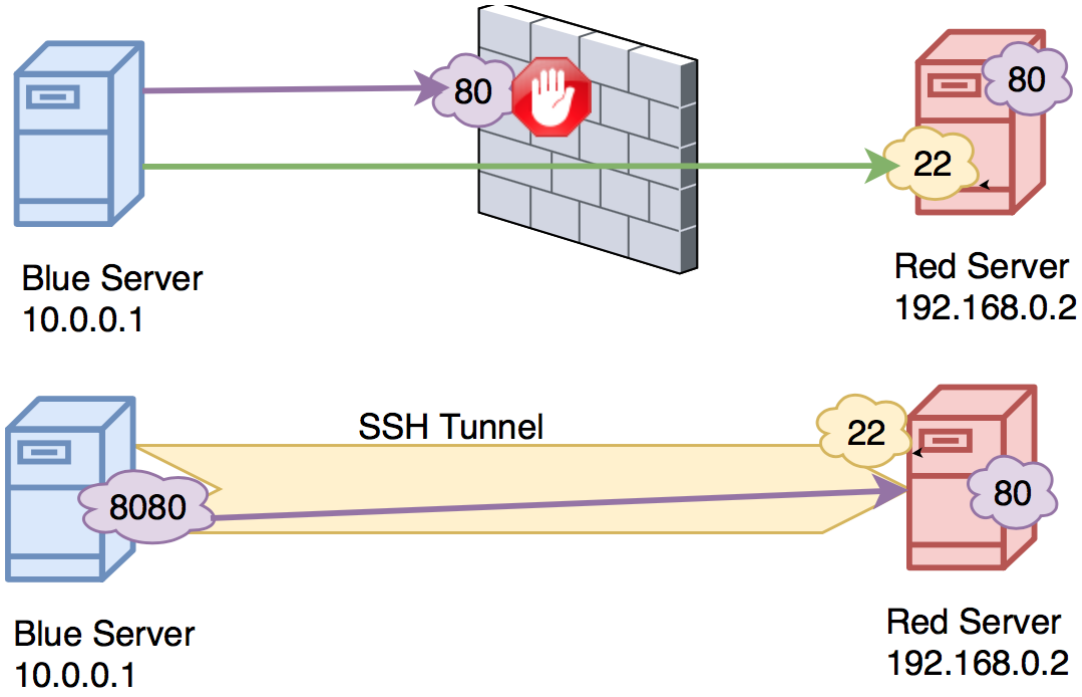


Fig. 11 Esquema básico de local port forwarding a través de SSH. Mientras que el acceso directo al puerto 80 está restringido por un firewall, se puede realizar una conexión al puerto 22 y redirigir la conexión al 80. Fuente: https://www.tunnelsup.com/how-to-create-ssh-tunnels/

3.4. Sistemas operativos

En este punto se procede a comentar cómo se llevó a cabo la elección de los sistemas operativos en la máquina principal y en las subordinadas. Dado que se decide hacer un servicio multiplataforma que pueda ser instalado en varios sistemas operativos y así facilitar su implementación en distintos ambientes, se procede a realizar varias pruebas en diversos sistemas.

El requisito básico es que la instalación de Python, Mysql y Apache así como de todas las dependencias generadas por los mismos sea factible. Como el proyecto todavía está en vías de desarrollo se ha de escoger un sistema operativo que sea gratuito y open source. Otro requisito que se ha tenido en cuenta a la hora de escoger el sistema operativo es el rendimiento que ofrezca a la hora de obtener los resultados cuando un usuario lanza una petición.

Se investiga entre los siguientes servidores: Ubuntu Server, RedHat, openSUSE, CentOs, Mandriva, ClearOS, archLinux, Fedora Server. Algunos de los sistemas investigados no cumplen con los requisitos previos, dado que por ejemplo en openSUSE y RedHat las versiones que más interesan son de pago, lo cual los descarta para este trabajo aunque son alternativas muy atractivas para implementar en el futuro.

CentOs basado en RedHat Enterprise ofrece un buen servicio. El mayor pero que tiene este SO es que no se realizan actualizaciones regularmente [38], lo que por un lado asegura que la versión instalada es estable y no hay que gestionar incompatibilidades de parcheado cada poco tiempo, pero puede perder rendimiento en comparación con otros servidores que están continuamente implementando mejoras.

Entre todas las demás alternativas, se realizan diversas pruebas y las versiones que no son implementadas como servidores, es decir las que están más orientadas a un uso de escritorio, ofrecen peor rendimiento que por ejemplo Ubuntu Server o Fedora Server. Por lo que la decisión final queda entre estas dos últimas alternativas, que ofrecen un buen rendimiento y buen soporte a nivel de servidor.

Finalmente se decide utilizar Ubuntu Server, dado que ofrece versiones LTS (Long Term Support), además de fluidez en el servicio distribuido a la hora de ofrecer una respuesta a las peticiones ejecutadas, seguridad y mucha información en Internet sobre problemas que pudiesen ocurrir debido a la gran comunidad de desarrolladores de la que dispone Ubuntu.

4. Implementación

Una vez detallada la arquitectura de la aplicación es el momento de comenzar con la implementación. En este capítulo se comentarán aspectos concretos de la implementación del proyecto que se consideran relevantes y que distan en gran medida del proceso habitual del desarrollo de una aplicación. Así mismo habrá un apartado con los problemas más significativos que se han encontrado a la hora de implementar el proyecto, detallando cómo han sido resueltos.

4.1. Encriptación

En nuestro servicio se encuentran dos maneras parecidas pero distintas de encriptar los datos. En ambos casos se utiliza AES en modo CBC en su versión de 256 bits, cuya única diferencia con sus otras versiones es que las claves de cifrado en este caso han de ser de 32 bytes de longitud.

Versión 1: Con claves

Es la versión original, la primera implementada y se basa en los procesos habituales de cifrado que existen en la actualidad combinados con la segmentación y capacidad de procesamiento que permite la computación en la nube. El proceso es el siguiente:

1.- El servidor web recibe una petición de encriptación con clave y procede a la comprobación de los datos de autenticación del usuario. En caso de ser incorrectos el proceso se detiene y se devuelve el error pertinente.

2.- El servidor web inicia el proceso de encriptación solicitando a cada una de las máquinas subordinadas del servicio distribuido que generen 50.000 claves de cifrado de 32 bytes. Una vez recibidas las 50.000 claves de cada una de las máquinas subordinadas, se juntan todas y se eligen de manera aleatoria dos de ellas. De esta manera, aumenta enormemente la seguridad del algoritmo al evitar por todos los medios posibles que las claves de cifrado sean fáciles de averiguar.

Generación de claves

Para generar cada una de las claves utilizadas en este algoritmo, primero se genera de manera aleatoria una cadena de 30 bytes a través del módulo Random de PyCrypto. Después, se utiliza la función SHA-256 para hashear esta cadena aleatoria y generar una cadena final de 32 bytes de longitud, que es el tamaño requerido por AES-256 para funcionar correctamente.

Random de PyCrypto

Este módulo es importante dado que la generación de la cadena inicial a partir de la cual se obtiene la clave de cifrado final ha de ser lo más aleatoria posible. Random consigue esto leyendo de ficheros del sistema operativo donde se captura ruido ambiental y datos generados por los distintos componentes del ordenador. En el caso de sistemas Linux, lee del fichero /dev/urandom. De esta manera se asegura tener suficiente entropía para generar un flujo continuo de bytes que permita generar cadenas completamente aleatorias evitando claves débiles o patrones que permitan descifrarlas.

3.- Se generan, de nuevo con el módulo Random, dos vectores de inicialización (IV) de 16 bytes cada uno. Los IV son cadenas aleatorias que se utilizan en algunos algoritmos de cifrado en bloque para que, aún en el caso de que se cifrasen dos veces los mismos datos con las mismas claves, al ser los IVs aleatorios y distintos en los dos procesos el resultado de ambas operaciones sería completamente distinto. De esta manera se añade una nueva medida de refuerzo al algoritmo.

4.- Se realiza un proceso de *padding* o de relleno para que el tamaño total en bytes de los datos a cifrar sea un múltiplo de 16, que es el tamaño de bloque requerido por AES. Después, se dividen los datos a cifrar de manera equitativa, y se envía a cada una de las máquinas del servicio distribuido tanto su porción de los datos como los dos IVs y las dos claves de cifrado.

5.- Cada una de las máquinas del servicio distribuido realiza una doble encriptación sobre su porción de los datos. Primero con un par IV/clave y posteriormente con el par restante. Así, se produce en paralelo una doble encriptación de los datos, aumentando enormemente su seguridad. Posteriormente se envían de vuelta los datos cifrados al servidor principal.

6.- El servidor maestro recibe los datos doblemente encriptados de cada una de las máquinas subordinadas y los reensambla. Por último, antes de devolver el fichero al usuario, se añade al final de los datos encriptados un HMAC de los mismos para asegurar la integridad y la autenticación.

HMAC [39]

Es una construcción utilizada en criptografía tanto para comprobar la integridad de los datos como para verificar la autenticación del usuario que intente descifrar el mensaje. Para ello se vale de una función hash y una clave secreta de longitud suficiente y generada de manera aleatoria. En nuestro caso, se utiliza SHA-256 como función hash y el ID del usuario que realiza la encriptación (una cadena aleatoria de 32 caracteres, única para cada usuario y secreta) como clave. Al descifrar, si el usuario que realiza el proceso no es el que realizó

la encriptación o si los datos a desencriptar han sido modificados de cualquiera manera, la verificación del HMAC fallará alertando a la aplicación y no se procederá al descifrado de los datos.

Versión 2: Sin claves

Esta segunda implementación surge como una posible mejora a la implementación original. Está basada en Signal aunque adaptando el proceso a las características del servicio que se suministra en nuestra aplicación. El objetivo es reducir el tiempo de respuesta de la encriptación eliminando la generación de las 50.000 claves en cada una de las máquinas del servicio distribuido, evitar al usuario tener que guardar y gestionar sus claves de cifrado y todo ello sin perder robustez y seguridad en el algoritmo.

El motivo de denominar a esta versión "sin claves" no es porque no se utilicen claves de cifrado dado que AES lo requiere para funcionar. Se denomina así porque las claves no se encuentran en ningún sitio concreto sino que el usuario posee una parte y el servidor otra, y solo juntando ambas porciones es posible regenerar las claves utilizadas para la encriptación. De esta manera, el usuario no se tiene que preocupar por sus claves, dado que su porción de las mismas ya viene incorporada como parte de los datos encriptados que devuelve el servidor, y a su vez si pierde o le roban dicho fichero encriptado será imposible descifrarlo porque se desconoce en todo momento la parte complementaria indispensable para regenerar las claves.

Así, en esta versión el proceso de encriptación es el mismo salvo por la manera en la que se generan las claves de cifrado (Fig. 12). Ya no se pide al servicio distribuido que genere un número ingente de claves de las que se seleccionan dos al azar. Ahora, el servidor principal va a recibir dos cadenas de longitud igual o mayor que 20 caracteres que habrá generado el usuario (o el plugin) de manera aleatoria. Cada una de estas cadenas, se complementará con otra cadena de 20 caracteres generada de nuevo aleatoriamente esta vez en el servidor, obteniendo dos cadenas finales de tamaño arbitrario. Sobre cada una de estas cadenas se realizará un proceso de hashing a través, de nuevo, de SHA-256 generando finalmente dos claves de cifrado totalmente aleatorias. Después del proceso de encriptación, el servidor almacenará las dos subcadenas que generó en el proceso, y el usuario mantendrá sus dos subcadenas además de los dos IDs que corresponden a sus cadenas complementarias en la base de datos de la aplicación.

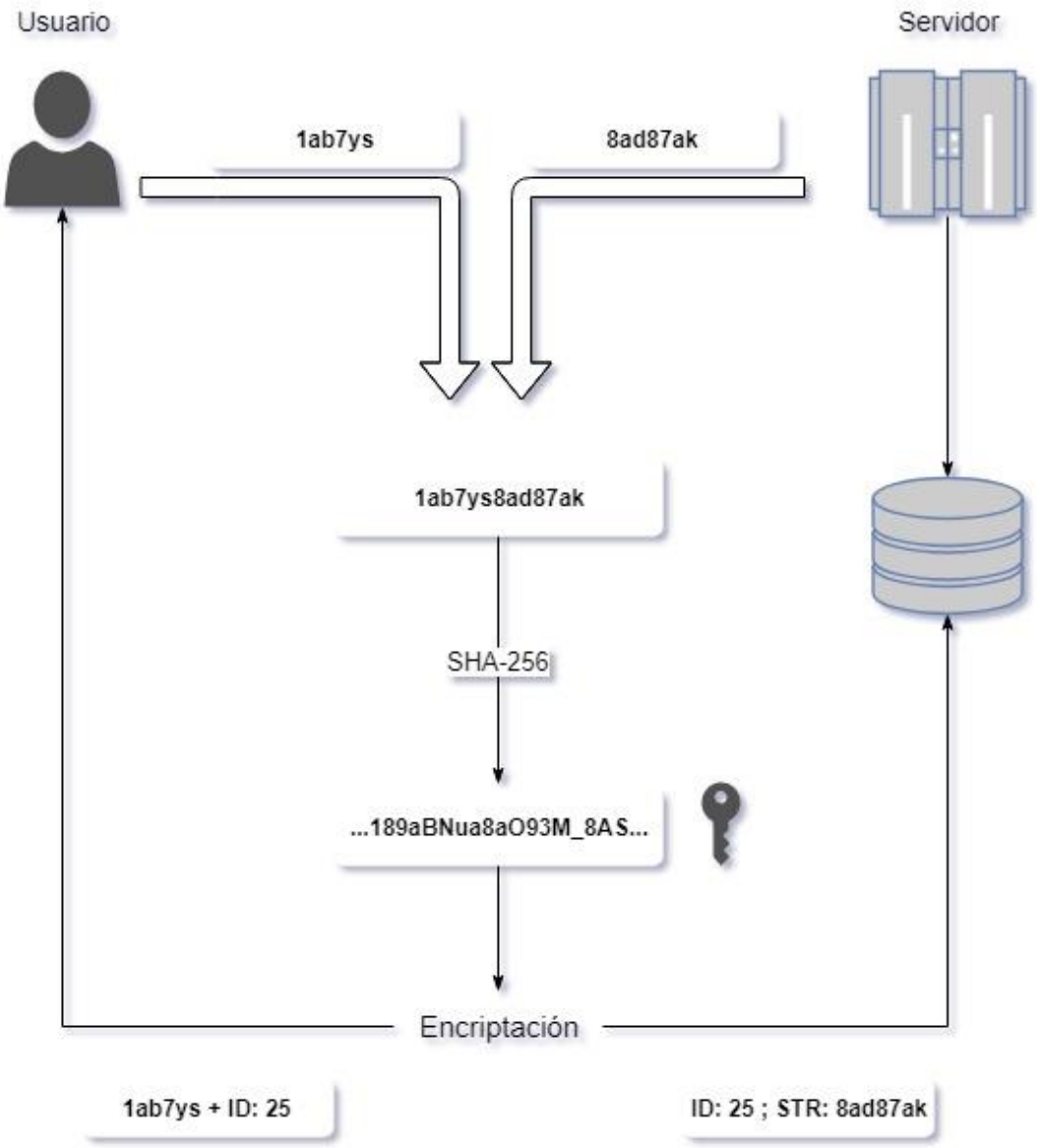


Fig. 12 Proceso de generación de una clave en la segunda versión del algoritmo de cifrado. El usuario genera una cadena aleatoria que se junta con otra creada por el servidor. Después de un proceso de hashing, la clave ya está lista para ser utilizada. Por último, cada una de las partes del proceso almacena una porción de la cadena original.

Por último, en esta segunda implementación se permite al usuario fijar quién es el receptor del mensaje cifrado, de manera que en el HMAC se usará como clave el ID de dicho receptor y solo él podrá acceder a su contenido.

4.2. Autenticación

Para interactuar con nuestra API se requiere que los usuarios de la misma se identifiquen de alguna manera. Como la gran mayoría de servicios web, nuestro sistema mantiene un registro de los usuarios en la base de datos aunque eso sí, almacenando solo los datos totalmente necesarios (como nombre de usuario, contraseña o email) y sin pedir en ningún momento otros datos más personales que puedan echar para atrás al usuario a la hora de registrarse (DNI, tarjeta de crédito, lugar de residencia, etc.). De esta manera, los clientes pueden usar su par usuario/contraseña para identificarse en nuestra aplicación y hacer uso de los servicios que ofrece.

Aunque sea el funcionamiento habitual de la mayoría de sitios web existen numerosos riesgos a la hora de enviar a través de Internet tu usuario y contraseña para autenticarse, como sufrir un ataque Man In The Middle que robe tus credenciales, que cierto tipo de malware robe la contraseña que tu navegador ha almacenado sin encriptar para que no tengas que repetirla constantemente o ataques de fuerza bruta contra contraseñas débiles. Además, si el robo se produce sin que el usuario tenga constancia del mismo el criminal tendrá acceso prácticamente ilimitado y sin restricciones al perfil dado que el par usuario/contraseña no suele cambiar habitualmente. Todo esto nos llevó a implementar un mecanismo de autenticación a través de tokens.

Se entiende como tokens a cadenas de caracteres aleatorios firmados criptográficamente por una clave secreta (en nuestro caso de nuevo utilizaremos los ID de los usuarios) y que contienen un tiempo de expiración [40], momento a partir del cual dejan de ser válidos y no podrán ser utilizados más para autenticar al usuario.

Ejemplo de token:

eyJhbGciOiJIUzI1NiIsImV4cCI6MTM4NTY2OTY1NSwiaWF0IjoxMzg1NjY5MDU1fQ.eyJpZCI6MX0.XbOEFJkhjHJ5uRINh2JA1BPzXjSohKYDRT472wGOvjc

De esta manera, un usuario puede solicitar a la API (utilizando su usuario y contraseña) la generación de un token. Este token vendrá firmado con su ID de usuario, de manera que le identificará únicamente a él y solo será válido durante un período de tiempo (por ejemplo una semana) a partir del cual dejará de funcionar. Hasta su expiración, el usuario puede enviar dicho token en lugar de su usuario y contraseña para identificarse. Cada usuario solo puede tener un token activo a la vez y los tokens no permiten modificar las credenciales de acceso del susodicho.

Este mecanismo es una clara mejora de la seguridad del usuario y su perfil personal. Por un lado, al ser cadenas de caracteres aleatorios, con fecha de expiración incluida y firmados criptográficamente con una clave secreta es prácticamente imposible averiguar los tokens con ataques de fuerza bruta. Además, en caso de robo de dicho token se puede minimizar el impacto al estar limitado el tiempo de validez del mismo e

incluso si la víctima sospecha dicho robo puede generar en cualquier momento un nuevo token invalidando a todos los anteriores. Por último, permite a un usuario compartir su perfil con personas de confianza durante un tiempo limitado y sin tener que comunicar sus credenciales de acceso, vale con generar un token e intercambiarlo con las personas deseadas.

Tokens anónimos

Los tokens de usuario son una buena medida para reforzar la seguridad del sistema, pero no incrementan de ninguna manera la privacidad y el anonimato dado que para poder generarlos has de tener un usuario registrado. Teniendo en cuenta que la encriptación es un servicio demandado por todas aquellas personas que quieren que su información personal siga siendo privada parece de sentido común implementar alguna opción que permita a dichas personas utilizar nuestro servicio tal y como lo desean.

Así y de manera muy parecida a los tokens de usuario se decide implementar los tokens anónimos. Su formato es el mismo con la salvedad de que la clave secreta que realiza la firma criptográfica es una cadena de 20 caracteres aleatorios que no son almacenados en la base de datos del servicio además de que no posee tiempo de expiración. Por otro lado, su funcionamiento es algo distinto. Si bien un token de usuario puede ser usado tantas veces como sea necesario antes de su tiempo de expiración, los tokens anónimos funcionan como un "ticket" que representan un proceso de cifrado y que solo podrán volver a ser utilizados para revertir dicho proceso. Es decir, cuando un usuario solicita sin autenticarse la encriptación de un fichero, se le expedirá de manera automática un token anónimo. Este token solo servirá para desencriptar dicho fichero, y en ningún momento podrá ser utilizado ni para encriptar ni para descifrar ningún otro mensaje. Una vez utilizado el token anónimo para el proceso de desencriptado, este deja de ser válido y se elimina de la base de datos.

De esta manera se puede acceder a los servicios de la aplicación de manera completamente anónima y con unas restricciones mínimas.

4.3. Contraseñas comprometidas

Have I Been Pwned es un sitio web creado y administrado por Troy Hunt, un Director Regional de Microsoft australiano que ha sido nombrado Most Valuable Professional por esta compañía gracias a sus aportaciones en el campo de la seguridad informática. Este servicio web recopila y almacena toda la información conocida en relación a robos masivos de información en ataques informáticos y te permite hacer búsquedas personalizadas para conocer si tu cuenta de correo electrónico o tu contraseña se han visto comprometidas en alguno de estos ciberataques y, por lo tanto, han dejado de ser seguras.

A pesar de contener información sobre más de 5 billones de cuentas hackeadas las consultas a este servicio cuentan con una optimización excelente que reduce

enormemente los tiempos de respuesta y lo hace altamente eficiente. Además, durante el desarrollo de nuestro proyecto se publicó la segunda versión de la API [41] del servicio, lo cual proporcionó la idea de integrarlo en nuestro código para comprobar las contraseñas utilizadas a la hora de registrar usuarios.

De esta manera, y aunque no revista un código demasiado extenso o llamativo, cabe destacar que cuando una persona intenta registrarse como usuario en nuestra aplicación, antes de confirmar la operación se comprueba si la contraseña utilizada es segura o se encuentra en la base de datos de Have I Been Pwned. Si se encuentra, se rechaza la petición de registro y se comunica al usuario que su contraseña ha sido comprometida. La comprobación se realiza a través de dos peticiones GET a la API que hemos comentado antes: una enviando la contraseña en texto claro y otra enviando la contraseña hasheada a través de SHA-1. De esta manera se comprueba si la contraseña se encuentra en la base de datos de Troy de las dos maneras en las que lo permite la API.

4.4. Problemas de implementación

Se procede a comentar varias de las problemáticas que se han encontrado en el transcurso del proyecto así como las soluciones ideadas para solucionarlas.

4.4.1. GIL

Son las siglas de Global Interpreter Lock [42], una estructura de Python que actúa como un mutex global (programación concurrente) y que impide que varios hilos de un mismo programa se ejecuten de manera simultánea y concurrente. De esta manera, aunque en tu código generes varios hilos y cuentes en tu equipo con varios procesadores solo uno de los hilos se ejecutará a la vez.

El motivo de implementar esta estructura es que Python utiliza contadores de referencias para el manejo de memoria. Así, cada variable que se declara almacena un contador indicando cuántas veces se hace referencia a ella en el código, y cuando esta referencia llega a cero se libera la memoria que ocupaba. Debido a esto y para evitar condiciones de carrera que pudieran modificar de manera errónea los contadores de referencia de las estructuras se implementó el GIL.

En nuestro caso esto resultó un gran problema a la hora de implementar la conexión SSH entre el servidor principal y las máquinas subordinadas del servicio distribuido. Todas estas conexiones se han de iniciar a la vez y funcionar de manera simultánea, por lo que elegimos hilos para implementar esta concurrencia en gran medida porque con su compartición de memoria son mucho más eficientes que los procesos. Se observó que algo no iba bien cuando, una vez implementadas las conexiones, se realizaron las primeras pruebas de encriptación con ficheros pequeños y se descubrió que los tiempos de respuesta eran enormemente elevados para lo que se esperaba.

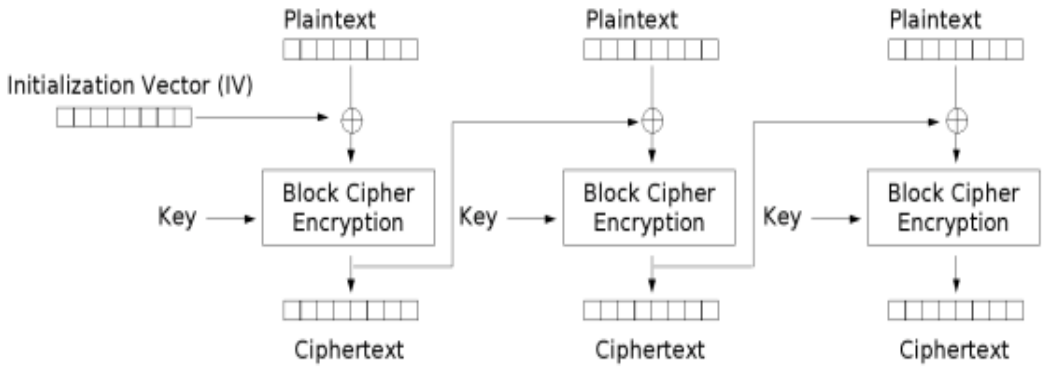
Tras muchas pruebas, investigación y *debuggeando* el código se llegó a la conclusión de que el problema residía en el GIL, así que se procedió a buscar una

manera de evitarlo. Existen diversas alternativas para programar concurrentemente sin que entre en juego el GIL, como usar bibliotecas externas al core de Python a las cuales el mutex global no tiene acceso o utilizar un intérprete no estándar para Python. Finalmente, se decidió utilizar procesos en lugar de hilos. Al crear un nuevo proceso se genera su propio espacio de memoria y por lo tanto ya no se producen esas situaciones de condición de carrera para los contadores de referencia, por lo que el GIL no entra en juego y se puede conseguir verdadera concurrencia sacrificando una pequeña cantidad de memoria.

4.4.2. Vectores de Inicialización

Ya se ha hablado por encima de los vectores de inicialización o IV en el apartado de implementación del proceso de encriptación. Los IV son cadenas aleatorias o pseudoaleatorias que se utilizan en algunas modalidades de los algoritmos de cifrado en bloque para incrementar la seguridad del proceso y evitar que posibles atacantes encuentren patrones en lo datos cifrados que les permitan o bien averiguar la clave secreta o incluso el mensaje encriptado.

En nuestro caso, se encripta usando AES en modo CBC (Fig. 13). Esta modalidad utiliza cada bloque ya encriptado para mezclarlo a través de una XOR con el siguiente bloque a cifrar. Como el primer bloque de los datos no tiene un bloque anterior con el que realizar la XOR, es susceptible de sufrir ataques de fuerza bruta que rompan la encriptación, y es por eso que se le suministra un IV que ocupe la posición correspondiente en la XOR. Por eso además los IVs son de 16 bytes, porque es el tamaño de bloque requerido por AES.



Cipher Block Chaining (CBC) mode encryption

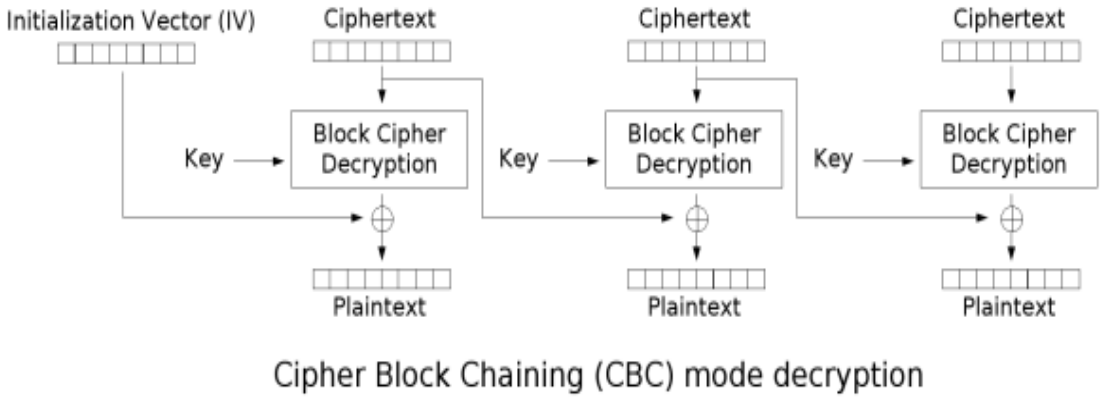


Fig. 13 Esquemas de la encriptación y desencriptación utilizando AES en modo CBC. En la ilustración se puede apreciar el uso de los IV y por qué son tan importantes. Fuente: https://defuse.ca/cbcmodeiv.htm

En una primera implementación de nuestra aplicación se generaba un IV para cada ronda de cifrado (como se produce una doble encriptación por cada fichero, dos IV en total). Este IV se utilizaba de igual manera para la encriptación de cada una de las porciones en las que es dividido el fichero a encriptar (véase *4.1 Encriptación).* El problema reside en que cada uno de los procesos de encriptación en las máquinas del servicio distribuido es independiente de los demás, es decir que lo encriptado en la máquina 1 no sirve como entrada para encriptar en la máquina 2, sino que son procesos paralelos y concurrentes. De esta manera, al utilizar el mismo IV para todas las porciones de un mismo fichero podía darse la situación en la que el primer bloque de todas las porciones (que es el que en CBC recibe como entrada el IV) fueran iguales o parecidos, dando una salida muy similar y por lo tanto generando patrones de repetición en los datos encriptados que se podían utilizar para desencriptar el fichero de manera maliciosa. Esto es inadmisible en una aplicación como la nuestra que enarbola la seguridad como punto fuerte, por lo que se tuvo que buscar una solución.

Una alternativa para solucionar el problema es generar un IV para cada una de las porciones. Nuestro servicio no almacena los IVs en la base de datos, sino que se añaden a la cabecera del fichero encriptado para que a la hora de desencriptar el mismo ya contenga dicha información incrustada sin que el usuario se tenga que preocupar por ello. Por lo tanto, resulta poco eficaz generar tantos IVs y aumentar de manera injustificada el tamaño de los ficheros encriptados, así que se descartó esta opción.

Finalmente se dio con una solución elegante: generar un solo IV y realizar operaciones a nivel binario sobre el mismo para obtener el resto de IVs necesarios. Cuando llegue un fichero para ser encriptado se generará un IV y, en caso de que el fichero por su tamaño tenga que ser dividido en varias porciones, se desplazarán

dos posiciones a la izquierda los bits del IV y posteriormente se invertirán, generando así un nuevo vector de inicialización. Este proceso se repetirá para cada una de las porciones en las que se divida el fichero, generando varios IVs pero solo teniendo que concatenar el original una vez se termine el proceso de encriptación dado que los demás se pueden obtener en cualquier momento repitiendo el mismo proceso. A continuación, se muestra un ejemplo simplificado de generación de IVs para la encriptación de un fichero que se divide en 3 porciones:

- IV original: **1001010101110110**.
- IV segunda porción: 1001010101110110 << 2 = 0101010111011000
  inv(0101010111011000) = **1010101000100111**.
- IV tercera porción: 1010101000100111 << 2 = 1010100010011100
  inv(1010100010011100) = **0101011101100011**.

De esta manera se generan todos los IVs a partir del primero, solucionando el fallo de seguridad que se había producido en la implementación original.

4.4.3. Encriptado de ficheros grandes

Durante varios meses después de terminar el core de la API que incluía la encriptación y desencriptación de ficheros a través del método de encriptación con clave se produjo una situación que provocó numerosos quebraderos de cabeza. El proceso de encriptación fallaba cuando se intentaban cifrar ficheros que excedieran los 30 MB de tamaño, mientras que por debajo de ese peso funcionaba correctamente.

Se barajaron distintas causas que podrían estar provocando esta situación, desde que el servidor de pruebas integrado en Flask (que era el que usábamos por aquel entonces) no tenía capacidad suficiente para soportar dicha carga, pasando porque Python restringía la cantidad de memoria RAM que se podía utilizar o incluso que las máquinas virtuales utilizadas no tuviesen suficiente capacidad para soportar dicho proceso. Aun así, ninguno de esos motivos terminaba de encajar ni de solucionar el asunto.

Después de muchas pruebas se descubrió que no era un problema de memoria sino de la gestión de los datos. Los datos que se envían para cifrar al servicio distribuido no se envían solos, sino que se les añade al principio una cadena de caracteres no binarios que indican a la máquina subordinada qué acción ha de realizar sobre ellos, además de los IVs y las claves de cifrado (que de nuevo son datos binarios). De esta manera lo que en realidad se enviaba era un string combinado formado por una cabecera de datos no binarios en formato ASCII que indicaba la orden a ejecutar unido a los datos binarios en crudo que no poseían

ningún formato en concreto. Al enviar esta cadena combinada a través de SSH se producían errores en el socket receptor, de manera que los datos quedaban corruptos e incluso se perdía información, por lo que el proceso fallaba al intentar encriptar con AES un conjunto de datos cuyo tamaño no era múltiplo de 16.

La solución a este problema fue sencilla una vez encontrado el error: codificar en base64 la cadena conjunta que forman los IVs, las claves de cifrado y los datos binarios de cada porción, y a estos datos codificados añadirles ya la cadena en ASCII que indica la orden. De esta manera y al tener los datos en formatos manejables por Python y fáciles de enviar a través de un túnel SSH no se producen los errores mencionados y se pueden encriptar ficheros de cualquier tamaño que permitan los recursos de las máquinas donde se ejecuta la aplicación.

5. Especificación

Características generales

Se procede a especificar las características de la API, tanto los parámetros que espera cada endpoint como los resultados producidos. El formato básico sobre el que se enviarán y recibirán los datos es JSON, y se especificará aquellas funcionalidades que permitan a su vez el formato BSON. Siempre se deberá indicar el formato elegido con la cabecera HTTP Content-Type (application/json o application/bson).

Todos los datos binarios enviados o recibidos de la API a través de JSON irán codificados en Base64 y se especificará con el símbolo * al lado del nombre de su parámetro.

A su vez, la autenticación que se requiere para acceder a algunos de los servicios se realizará a través de autenticación HTTP básica, ya sea con usuario/contraseña o utilizando un token y rellenando el campo contraseña con una cadena cualquiera (por ejemplo “unused”).

Registro de usuarios:

Este método crea un nuevo usuario en la base de datos para poder posteriormente acceder autenticado a los servicios.

POST	https://[url]/api/register
Parámetros	
username	Nuevo nombre de usuario.
password	Contraseña para el usuario.
Resultados	
username	Nombre de usuario registrado.
uri	Dirección del perfil del usuario recién registrado.
Códigos de respuestas	
201	Se ha registrado el usuario correctamente.

400	Faltan argumentos en el Json enviado o el formato del Json es incorrecto.
422	La contraseña ha sido rechazada al encontrarse en la base de datos de Have I Been P0wned.
500	Error en el procedimiento.

Búsqueda de usuarios:

Este método realiza una búsqueda en la base de datos y devuelve la información del usuario.

Requiere estar autenticado para realizar esta acción.	
GET	https://[url]/api/users/<username>
Parámetros	
<username>	Nombre (o token) del usuario que se quiere buscar.
Resultados	
username	Nombre del usuario encontrado.
Códigos de respuestas	
200	La búsqueda se ha realizado correctamente.
400	El usuario o la contraseña son incorrectos.
403	El usuario autenticado es distinto del cual sobre el que se quiere obtener la información. No autorizado.
500	Error en el procedimiento.

Obtener token de usuario:

Este método genera un nuevo token para el usuario que lo solicita.

Requiere estar autenticado para realizar esta acción.	
GET	https://[url]/api/get_token
Resultados	
token	Valor del nuevo token generado.
Códigos de respuestas	
200	El token se ha generado correctamente.
500	Error en el procedimiento.

Cifrado con claves:

Método que cifra los datos binarios que recibe y devuelve los mismos encriptados junto a las claves utilizadas en el proceso.

Requiere estar autenticado para realizar esta acción. Acepta formato BSON.	
POST	https://[url]/api/encrypt_file
Parámetros	
username	Nombre de usuario (o token) que realiza la encriptación. Se puede dejar en blanco para encriptación anónima.
password	Contraseña del usuario que realiza la encriptación. Vacío en caso de utilizar un token o realizar encriptación anónima.
alg	Algoritmo de cifrado requerido, por ahora 'aes256'.
file*	Contenido binario del archivo.
Resultados	
file*	Resultado de la encriptación.

key_1*	Primera clave utilizada en el proceso de encriptación.
key_2*	Segunda clave utilizada en el proceso de encriptación.
Códigos de respuestas	
200	La encriptación se ha completado satisfactoriamente.
400	El formato del fichero es incorrecto o faltan parámetros.
500	Error en el procedimiento.

Descifrado con claves:

Método que se encarga de desencriptar un fichero cifrado a través del método con claves.

Requiere estar autenticado para realizar esta acción. Aceptan formato BSON.	
POST	https://[url]/api/decrypt_file
Parámetros	
username	Nombre de usuario (o token) que realiza la desencriptación. Se puede utilizar un token anónimo válido.
password	Contraseña del usuario que realiza la desencriptación. Vacío en caso de utilizar un token o realizar encriptación anónima.
key_1*	Primera clave secreta.
key_2*	Segunda clave secreta.
alg	Algoritmo de cifrado, por ahora 'aes256'.
file*	Datos binarios que se quieren desencriptar.
Resultados	

file*	Resultado de la desencriptación.
Códigos de respuestas	
200	La desencriptación se completó satisfactoriamente.
400	El formato del fichero es incorrecto o faltan campos.
500	Error en el procedimiento.

Cifrado sin claves:

Este método permite encriptar ficheros o mensajes especificando un usuario receptor del contenido encriptado.

Requiere estar autenticado para realizar esta acción.	
POST	https://[url]/api/encrypt_to
Parámetros	
str1	Cadena aleatoria de al menos 20 caracteres.
str2	Cadena aleatoria de al menos 20 caracteres.
alg	Algoritmo de cifrado, por ahora 'aes256'.
to	Usuario (o token de usuario) destinatario de la encriptación.
file*	Datos binarios a encriptar.
Resultados	
file*	Resultado de la encriptación.
to	Usuario destinatario de la encriptación.
Códigos de respuestas	

200	La encriptación se ha completado satisfactoriamente.
400	El formato del fichero incorrecto o faltan campos.
500	Error en el procedimiento.

Descifrado sin claves:

Este método permite al destinatario de una encriptación sin claves proceder a desencriptar el mensaje.

Requiere estar autenticado para realizar esta acción.	
POST	https://[url]/api/decrypt_to
Parámetros	
username	Nombre de usuario (o token) que procede a desencriptar.
password	Contraseña del usuario que realiza la desencriptación. Se puede dejar vacío en caso de utilizar un token.
alg	Algoritmo de cifrado, por ahora 'aes256'
file*	Contenido sobre el que realizar la desencriptación.
Resultados	
file*	Resultado de la desencriptación.
username	Usuario destinatario del mensaje.
Códigos de respuestas	
200	La desencriptación se ha completado satisfactoriamente.
400	El formato del fichero incorrecto o faltan campos.
500	Error en el procedimiento.

6. Pruebas de concepto y medidas de tiempos

Metodología de evaluación

En este apartado se muestra cómo se ha realizado el análisis del rendimiento del servicio para los casos de encriptado y desencriptado, además de mostrar los resultados obtenidos y el software utilizado para ello.

A través de las pruebas lanzadas se han encontrado partes de la aplicación que han tenido que ser modificadas debido a la cantidad de recursos y de tiempo que era necesario para la obtención de los resultados. Esto es algo clave dado que nuestro producto, además de perseguir la seguridad como punto fuerte, busca que los usuarios realicen sus gestiones en el menor tiempo posible.

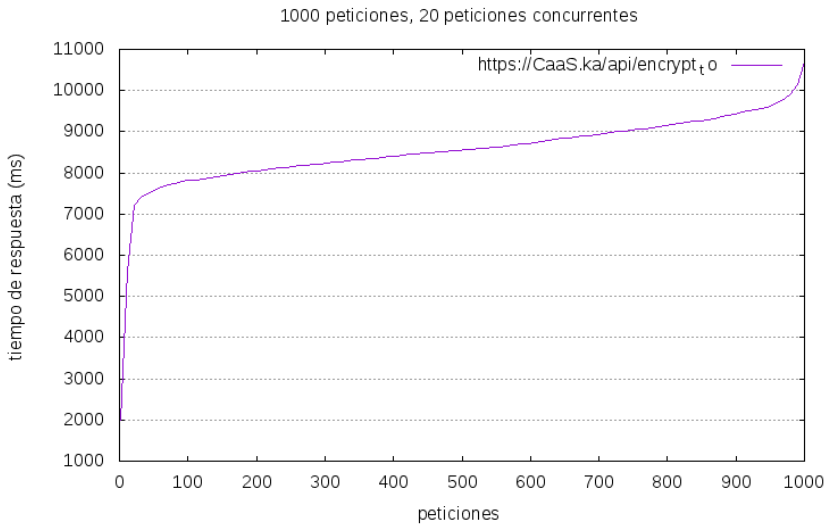
Entorno de pruebas

Como entorno de pruebas se ha escogido el que será utilizado de momento como entorno real en producción. Este mismo consta de 3 instancias t2.micro del servicio EC2 de Amazon, una realizando las tareas de máquina principal y servidor web, y las otras dos actúan como máquinas subordinadas. Hay que recalcar que estas máquinas sólo disponen de una CPU y 1GB de memoria para cada una de las instancias, por lo que los resultados obtenidos deberán ser juzgados teniendo en cuenta dichas características.

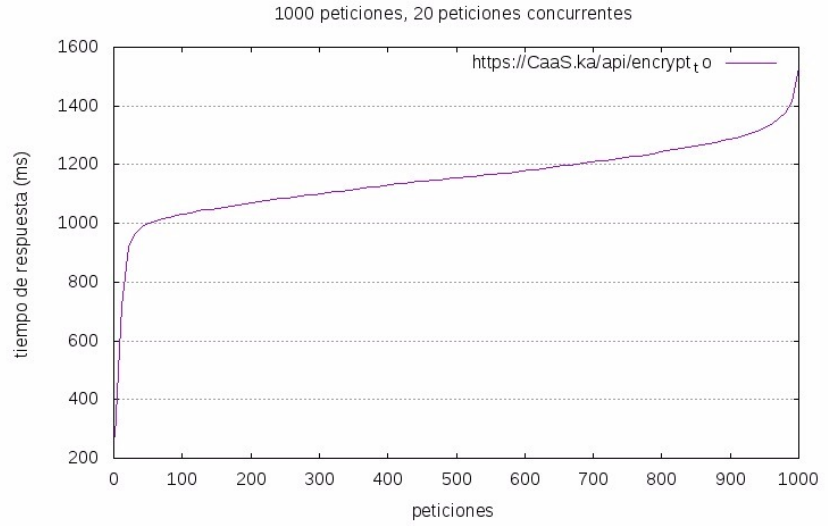
Para valorar con más conocimientos los resultados también se dispone de otro entorno de pruebas local. Este se compone de una maquina con 32 GB de memoria y 4 núcleos, lo que aporta un rendimiento mayor que el de las máquinas del servicio en la nube.

Encriptado

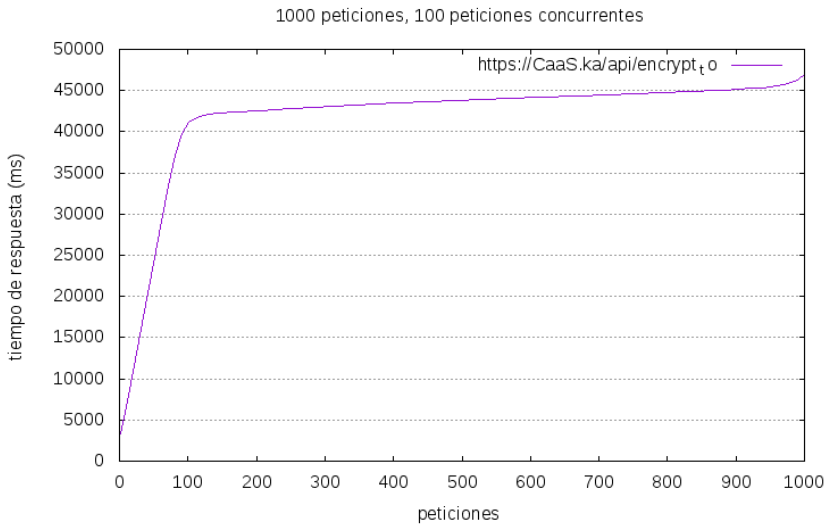
En este primer gráfico se lanza una batería de pruebas que consta de hasta 1000 peticiones lanzadas en tandas de 20 peticiones concurrentes. Se puede comprobar que el tiempo de respuesta en este caso se situaría en una media de entre 8 a 10 segundos. Analizando también la forma que toma la gráfica puede comprobarse que se asemeja a la de una función logarítmica, que crece rápido para valores pequeños pero que para valores medios y grandes mantiene un aumento muy suavizado. Esto es positivo porque demuestra que existe una concurrencia eficaz en la aplicación que permite gestionar gran número de peticiones sin que se vean afectados los tiempos de respuesta.



Para comprobar la diferencia entre las máquinas del servicio distribuido de las que se dispone en la nube y un ordenador con mayores prestaciones, lanzamos la misma batería de pruebas de 1000 peticiones de 20 peticiones concurrentes pero esta vez en local. La diferencia de rendimiento es notable. El tiempo medio de respuesta es de un 1,2 segundos mientras que como se ha comprobado anteriormente, en la prueba lanzada en la nube el tiempo medio de respuesta es de 9 segundos.

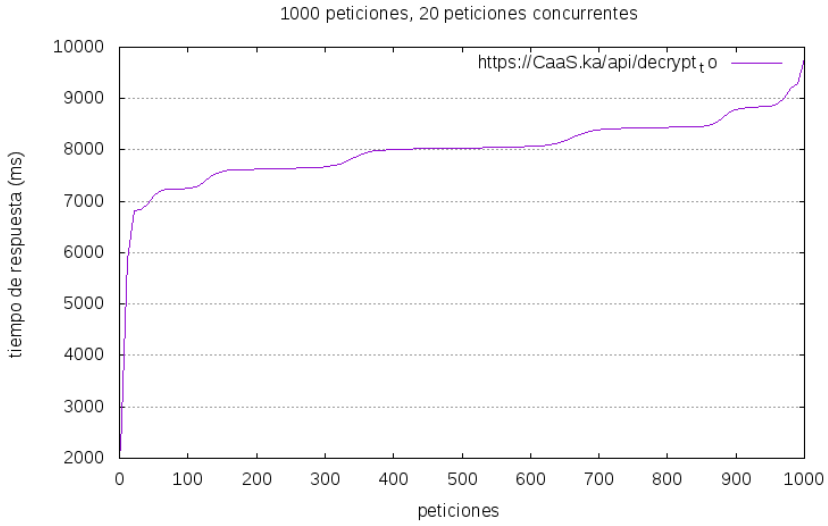


Tras lanzar esta prueba, se lanza otra batería de peticiones de encriptación pero esta vez las 1000 peticiones se lanzan en tandas de 100 peticiones concurrentes. En este caso se aprecia que, aunque la forma de la gráfica sigue asemejándose a una función logarítmica, la media del tiempo de respuesta sube a alrededor de 45 segundos. Este tiempo de respuesta por petición es excesivo y en absoluto aceptable, pero responde a las restricciones de CPU y memoria RAM impuestas por las instancias gratuitas del servicio cloud que hemos utilizado. En caso de instalar el servicio en ambientes reales, sería necesario contar con máquinas más potentes equipadas con mayor capacidad de memoria y un número de cores superior a 4, obteniendo así resultados mucho mejores.

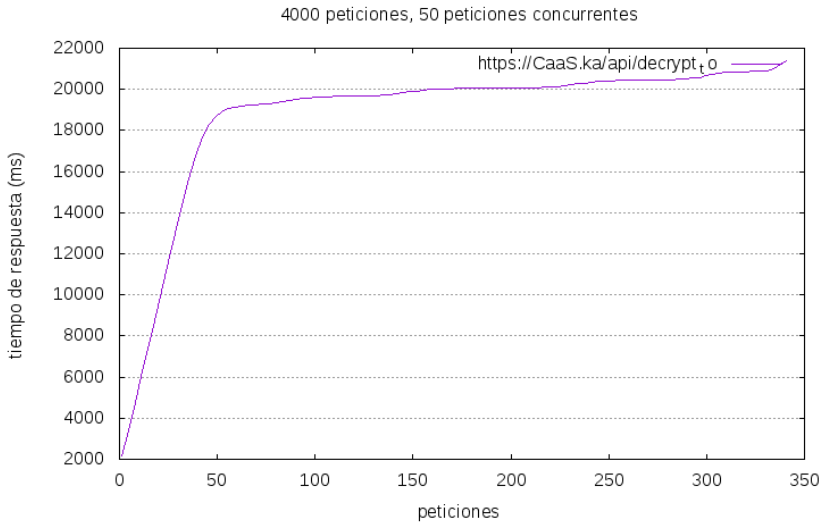


Desencriptado

En este gráfico se lanza de nuevo una batería de pruebas que consta de 1000 peticiones en tandas de 20 peticiones concurrentes, pero en esta ocasión se lleva a cabo sobre el desencriptado. Se puede comprobar que el tiempo de respuesta en este caso se situaría en una media entre 7 a 9 segundos, pero esta vez no sigue un patrón creciente uniforme si no que entre algunas peticiones el tiempo de respuesta no crece y se mantiene. Esto responde a que el proceso de desencriptado por lo general es más ligero y menos costoso que el de encriptación.



Al lanzar la misma prueba con 40000 peticiones en tandas de 50 peticiones concurrentes se detecta un error de timeout y solo se resuelven un total de 1197 peticiones. El error obtenido es “apr_pollset_poll: The timeout specified has expired (70007)” y se produce debido a que se exceden los 30 segundos configurados por defecto como tiempo de espera a una petición realizada al servidor. Esto es consecuencia de los pocos recursos de los que dispone el servidor principal, los cuales no son suficientes para gestionar más peticiones y provoca un colapso del servicio.



Conclusión sobre las pruebas

Estas pruebas demuestran que, a pesar de un rápido incremento inicial de los tiempos de respuesta, a partir de cierto número de peticiones el aumento de tiempos de respuesta se estabiliza permitiendo realizar muchas peticiones sin que se perjudiquen entre ellas. Hay que tener en cuenta que las máquinas utilizadas disponen de unos recursos enormemente limitados y se ha demostrado que las mismas pruebas en local sobre máquinas virtuales con más recursos devuelven resultados mucho más satisfactorios. En un ambiente real cabe imaginar mayor número de máquinas independientes y cada una de ellas con un mayor número de recursos, de manera que los tiempos de respuesta se pronostican muy satisfactorios en tales circunstancias.

7. Conclusiones y trabajo futuro

A pesar de haber cumplido la gran mayoría de los objetivos fijados para el proyecto y haber alcanzado un nivel de desarrollo enormemente satisfactorio existen funcionalidades y secciones de la aplicación que, o bien aún requieren de mejoras para alcanzar su máximo nivel de optimalidad, o bien su implementación aún no ha sido completada del todo. Todos estos aspectos se recogen en este apartado el cual va a dividirse en dos secciones: cliente y servidor.

7.1. Cliente

Una vez finalizada la fase principal de desarrollo del plugin para Chrome podemos centrarnos en las mejoras e implementaciones futuras que se pueden llevar a cabo sobre el mismo.

Mejoras

Actualmente al autenticarse en el plugin se guardan las credenciales en los localStorage del navegador, una forma de almacenar contenido común para todos los complementos del navegador provenientes de un mismo origen. Este sin duda sería uno, sino el apartado más importante a mejorar de cara al futuro, dado que es una brecha de seguridad considerable que podría permitir a otros plugins instalados acceder a las credenciales de acceso de nuestro servicio. Por lo tanto, es totalmente necesario desarrollar otro método para almacenar las sesiones que no comprometa la seguridad de los usuarios.

Otro apartado de mejora podría ser lanzar mensajes más descriptivos en cada caso de error, para facilitar al usuario toda la información de los problemas que ocurren en cualquier situación en lugar de utilizar mensajes genéricos que no permiten localizar el fallo.

Novedades

En la actualidad existen modos de utilizar complementos de Google Chrome en otros navegadores, como por ejemplo Firefox Quantum a través de Chrome Store Foxified. Esto, aunque puede servir de momento, puede que en ciertos apartados futuros del cliente cause problemas de compatibilidad, debido a que cada navegador puede disponer de funciones que aun realizando la misma función tenga apelativos diferentes y se llamen de otra manera. Por ello sería recomendable crear extensiones como el plugin desarrollado para cada navegador para no forzar al usuario a utilizar un navegador con el que no está familiarizado.

Otra de las novedades a implementar podría ser realizar más clientes y no centrarnos únicamente en versiones de navegador. En este caso hemos pensado en no basarnos únicamente en una sola plataforma, sea móvil o escritorio, sino hacer una aplicación multiplataforma que sea de fácil acceso y que ofrezca todas las funcionalidades que se tienen actualmente más las que se vayan desarrollando en la API.

Se puede también comunicar con actuales servicios de mensajería para promocionar nuestro servicio. Creemos que nuestra propuesta podría ser de buen gusto para algunos servicios en los que se envían mails y se podría incluir la opción de cifrar con CaaS directamente desde su aplicación.

Por último, una opción bastante interesante sería incluir algún tipo de relaciones entre los usuarios, de manera que se pudiesen crear canales de comunicación entre usuarios "amigos" totalmente cifrados y se puedan enviar ficheros, texto o comunicarse de un modo seguro a través de nuestro servicio.

7.2. Servidor

Si bien tanto el proceso de encriptado como el de desencriptado están completamente desarrollados y se consideran terminados, si es cierto que de cara a montar la API en ambientes reales se requieren ciertas mejoras en el código que permitan alcanzar el máximo nivel de potencial y seguridad que ofrece el servicio.

En primer lugar, se requiere implementar medidas extra de seguridad en el código que reduzca el riesgo de sufrir los ataques web más comunes, como pueden ser el SQL Injection o el Cross Site Scripting. Este tipo de ataques se benefician de aplicaciones que no comprueban adecuadamente la información que los usuarios envían al servicio. De esta manera, todos los endpoints de la aplicación expuestos a Internet han de comprobar (y filtrar) la información recibida antes de iniciar cualquier proceso en el backend de la aplicación. Una vez implementadas estas medidas sería conveniente realizar una auditoría de seguridad para comprobar la robustez de las mismas.

Por otra parte, es imprescindible mejorar el sistema de excepciones y errores de la API. En la actualidad, es un sistema totalmente funcional pero demasiado genérico, de manera que muchos errores de fuentes muy distintas devuelven el mismo tipo de excepción, lo cual impide que se conozcan de manera sencilla y precisa los motivos reales por los que se ha producido un error en el sistema. Cada error capturado debería lanzar una excepción distinta con una descripción precisa del problema, de manera que se pueda devolver al usuario un código de error HTTP adecuado (en lugar de un error 500 genérico como ocurre en la actualidad) además de facilitar a los administradores del servicio la corrección de problemas en la aplicación.

Por último, el sistema de autenticación actual utiliza autenticación básica HTTP [43], que aun siendo efectiva y sencilla de implementar no es la más segura ya que las credenciales se almacenan en el navegador y se envían codificadas en Base64. Por ello es aconsejable en ciertas circunstancias y para determinados ambientes implementar un sistema más robusto de autenticación que blinde las credenciales de los usuarios y a la vez no interfiera con el mecanismo de autenticación por tokens que integra la API.

Estas son las mejoras cuya implementación se considera necesaria de cara a integrar la aplicación en un ambiente real. Además de ello, la modularidad de la aplicación permite que conforme avance la criptografía se puedan implementar e integrar nuevos algoritmos, técnicas y funcionalidades que refuercen el proceso de encriptación. Entre estas nuevas funcionalidades sería ideal implementar un mecanismo de encriptación para canales de comunicación de flujo (como por ejemplo un chat o llamadas telefónicas) que complemente la encriptación de mensajes de tamaño estático que ofrece la aplicación en la actualidad.

7. Conclusions and future work

In spite of having fulfilled most of objectives set for the project at the beginning and having reached a very satisfactory level of development, there are functionalities and sections of the application that still requires improvements to reach their maximum level of optimality, or else their implementation has not yet been fully completed. All these aspects are collected in this section which will be divided into two sections: client and server.

Client

Once the main phase of development of the Chrome plugin is finished, we can focus on the improvements and future implementations that can be carried out on it.

Improvements

Currently, when login CaaS plugin, credentials are saved in the localStorage of the browser, a way to store common content for all browser add-ons from the same source. This would undoubtedly be one, but the most important section to improve for the future, considering that it is a big security breach that could allow other installed plugins to access the login credentials of a determined user of our service. Therefore, it is absolutely necessary to develop another method to store the sessions that does not compromise the security of the users.

Another improvement section could be to send more descriptive messages in each case of error, to provide the user all the information of the problems that occur in any situation instead of using generic messages that do not allow locating the failure.

Future Implementations

At the moment we know that there are ways to use Google Chrome add-ons in other browsers, such as Firefox Quantum through Chrome Store Foxified, another plugin, but this time a native one from this server. This, although it may be useful at the moment, may cause compatibility problems in certain future sections of the client, because each browser can have functions that, even though performing the same function, have different names and are called differently. Therefore, it would be advisable to create extensions such as the plugin developed for each browser so as not to force the user to use a browser which he is not familiar with.

Another novelty to implement could be to make more customers and not only focus on browser versions. In this case we have thought about not only being based on a single platform, whether mobile or desktop, but making a multiplatform app that would be easy to access and that offers all the functionalities that are currently available, plus those that would be developed in the API.

Also, we have thought about the possibility to communicate with current email clients to promote our service. We believe that our proposal could be good for some services in which emails are sent and could include the option to encrypt and decrypt with CaaS directly from their application.

Finally, a very interesting option would be to include some kind of relationships between the users. There would be some fully encrypted communication channels allowing the users to send files, text or communications in chats being sent in a safe way through our service.

Server

Although both the encryption process and the decryption process are fully developed and considered that it has been completed, if it is true that in order to mount the API in real environments, certain improvements in the code are required to reach the maximum level of potential and security that the service offers.

First, it is necessary to implement extra security measures in the code that reduces the risk of suffering the most common web attacks, such as SQL Injection or Cross Site Scripting. This type of attacks benefit from applications that do not adequately check the information that users send to the service. In this way, all application endpoints exposed to the Internet must check (and filter) the information received before starting any process in the backend of the application. Once these measures are implemented, it would be advisable to carry out a security audit to verify the robustness of the same.

Also, it is essential to improve the system of exceptions and errors of the API. At present, it is a fully functional but too generic system, so many errors from very different sources return the same type of exception, which prevents the simple and precise knowledge of the real reasons for which a system error occurred. Each error captured should throw a different exception with an accurate description of the problem, so that the user can be given an adequate HTTP error code (instead of a generic 500 error as is currently the case) as well as facilitating the administrators the correction of issues in the application.

Finally, the current authentication system uses HTTP basic authentication [43], which, while being effective and simple to implement, is not the most secure since the credentials are stored in the browser and sent encoded in Base64. Therefore, it is advisable in certain circumstances and for certain environments to implement a more robust authentication system that shields the user's credentials and at the same time does not interfere with the token authentication mechanism that integrates the API.

These are the improvements whose implementation is considered necessary in order to integrate the application in a real environment. In addition, the modularity of the application allows to implement new algorithms, techniques and functionalities. Among these new functionalities, it would be ideal to implement an encryption mechanism for communication channels such as a chat or telephone calls that complements the encryption of static-sized messages that the application currently offers.

8. Contribución al proyecto

En este apartado se procede a detallar la contribución de cada uno de los miembros del grupo al proyecto.

Kurosh Dabbagh Escalante

Investigación

Es imprescindible realizar un proceso de investigación y documentación de las tecnologías a utilizar antes de comenzar cualquier desarrollo, pero además es una actividad que perdura durante prácticamente todo el proyecto.

Este proceso de investigación comenzó con la selección de Python como lenguaje para la API y de Flask como el framework web a utilizar. Una vez seleccionadas ambas tecnologías se investigó sobre los diferentes paradigmas para implementar bases de datos que concluyó con la elección de MySQL y de PyMySQL como módulo de Python utilizado como framework ORM para la comunicación con la base de datos.

También hubo que investigar sobre los diferentes algoritmos de encriptación que existen en la actualidad, su robustez y su validez en los tiempos actuales. Una vez seleccionado AES-256 como algoritmo a utilizar se investigó sobre los diferentes módulos existentes en Python que otorgan las funcionalidades requeridas para implementar este algoritmo de manera eficiente y segura. Por supuesto, dado el gran nivel de complejidad intrínseco a la criptografía fue del todo necesario actualizar y recabar conocimiento relacionado con temas tan complejos como IV's, HMAC, creación de claves de cifrado, modos de cifrado en bloque o protocolo Signal para proporcionar una implementación correcta de todas estas técnicas.

La investigación no solo facilita la selección de tecnologías sino que también ayuda a descartar aquellas que no encajan correctamente en los requisitos del proyecto. Este es el caso por ejemplo de Chilkat, un módulo de Python que permite conexiones SSH y que finalmente fue descartado en favor de Paramiko para generar la conexión entre el servidor principal y las máquinas del servicio distribuido. También es el caso de formatos como XML o YAML que fueron descartados como mecanismo de comunicación con la API, eligiéndose finalmente tanto JSON como BSON.

Durante el proceso de desarrollo surgieron diversos problemas detallados en el apartado de Implementación cuya solución superaba los conocimientos de los componentes del proyecto, por lo que de nuevo se requirió de investigación y lectura de numerosa documentación para solventarlos. Este es el caso del GIL y los problemas con la concurrencia o el problema de codificación de cadenas que provocaba el fallo en la encriptación de ficheros superiores a 30MB.

Decisiones de diseño

Una vez realizada la investigación inicial y seleccionadas las tecnologías a utilizar en primera instancia, se decidió la manera en la que estaría diseñada la aplicación. En este caso, se buscaba explotar el potencial de la computación en la nube para incrementar la eficiencia en el cifrado y además mantener unos niveles altos de seguridad en el proceso del mismo.

De esta manera, la estructura finalmente seleccionada es la de una máquina principal que actúa como servidor web y que está conectada a una serie de máquinas subordinadas que se encargan de realizar concurrentemente tanto la encriptación como el desencriptado. Para no comprometer la seguridad de los datos que se transmiten entre el servidor principal y las máquinas del servicio distribuido se hace uso de la redirección de puertos, funcionalidad que nos ofrece el protocolo SSH. Así aunque se envíen entre distintas máquinas datos aún sin procesar en todo momento se realiza de manera segura.

Otro aspecto a tener en cuenta a la hora de diseñar la arquitectura de la API es la base de datos, que aun cuando en este proyecto almacenan únicamente los datos imprescindibles para poder funcionar, se tuvo que diseñar el modelo entidad relación adecuado para el servicio. Esto es importante debido a que, al hacer uso de un mapeo ORM a través de PyMySQL, las tablas de la base de datos tienen su correspondencia directa (y por lo tanto su impacto directo) en el código y el funcionamiento de la aplicación. Además un diseño adecuado es el que permite realizar la autenticación por tokens tanto anónimos como de usuario comentado en el apartado de Implementación.

Por último, tal vez el diseño más importante realizado es el de los algoritmos de cifrado utilizados. De esta manera se puede ofrecer tanto la encriptación con claves, que se basa en los sistemas más habituales de encriptación en la actualidad, así como la encriptación sin claves que sin duda es una de las grandes apuestas del proyecto. El diseño de dichos algoritmos comprende desde seleccionar qué datos se van a solicitar al usuario o cómo va a procesar los datos el servidor web hasta cómo el controlador va a gestionar la división y envío de los datos al servicio distribuido y cómo van a realizar concurrentemente el proceso de encriptación (o de desencriptado) las máquinas subordinadas.

Cabe destacar que el diseño de la arquitectura de la aplicación cumple con el principio de modularidad para facilitar su ampliación en el futuro.

Implementación de la API

La API es el core del proyecto, dado que es la que realiza toda la funcionalidad y alrededor de la cual se pueden implementar posteriormente otra serie de aplicaciones de gran utilidad como por ejemplo el plugin que se ha desarrollado para Chrome.

Su implementación comenzó nada más se llevaron a cabo las decisiones de diseño pertinentes y abarca todo el código en Python generado para el proyecto.

De esta manera, se generó la interfaz web que sirve como puerta de enlace con el servicio, los dos algoritmos de encriptación con sus correspondiente algoritmos de desencriptación, la generación de IVs y claves de cifrado, el algoritmo de división equitativa de los datos a procesar, el código del servicio distribuido que se encarga del procesamiento en paralelo de los datos y también las firmas criptográficas que podemos encontrar tanto en el HMAC que se añade a los datos encriptados como en los tokens utilizados para la autenticación.

Para poder conectar el servidor principal con las máquinas subordinadas se usa Paramiko, que permite realizar una conexión SSH con redirección local de puertos en Python.

También se generó el sistema básico de excepciones que captura los distintos errores que pueden producirse en la aplicación y les asigna un código HTTP que es enviado en respuesta al usuario.

Cabe destacar las últimas implementaciones llevadas a cabo en la API como puede ser la generación, verificación y almacenamiento tanto de los tokens de usuario como de los tokens anónimos, la verificación de las contraseñas utilizadas en el registro de usuarios contra la base de datos del servicio Have I Been P0wned o el módulo de validación de los Json (o Bson) recibidos por los diferentes endpoints de la API.

Todo ello unido al desarrollo de soluciones de los problemas detallados en el capítulo de Implementación conforma el desarrollo de la API.

Pruebas

Al tener una envergadura considerable, no ha sido sencillo integrar directamente código a la API de manera que funcione correctamente. De esta manera, muchas de las diferentes secciones que conforman la API han sido generadas o puestas a prueba por separado a través de una serie de pequeños ficheros escritos en Python. Estos ficheros han sido fundamentales a la hora de probar el correcto funcionamiento de, por ejemplo, la generación de IVs a través de operaciones binarias sobre una cadena original, el establecimiento de una conexión SSH con redirección local de puertos, la generación de cadenas binarias utilizadas para la creación de las claves de cifrado, etc. También han sido utilizados para interactuar con la API de manera cómoda antes de contar con el plugin, lo cual permitió ganar en eficacia y tiempo invertido.

Estos ficheros no son parte del proyecto como tal dado que una vez cumplida su funcionalidad su código es integrado en la aplicación y por lo tanto dejan de ser útiles. Aun así, han sido uno de los recursos clave más utilizados a la hora de la generación de la API porque ha permitido probar de manera independiente cada una de las secciones.

Amazon Web Services

Una vez estuvo terminado la mayoría del código del proyecto, se procedió a montar la API en la nube de Amazon. Para ello primero se generó y configuró las instancias de máquinas virtuales necesarias para el funcionamiento de la aplicación.

Para este caso concreto se utilizó una instancia como servidor principal y luego otra instancia que se duplicó una vez configurada para las máquinas del servicio distribuido.

Debido a que la comunicación con la API en todo momento ha de ser segura y encriptada, se generó también un certificado digital con OpenSSL para que en todo momento se haga uso del protocolo HTTPS en las comunicaciones entre cliente y servidor.

Memoria

El último paso de cara a completar el proyecto ha sido recopilar toda la información y todos los procedimientos llevados a cabo y plasmarlos en la memoria de manera estructurada. El proceso de escritura de la memoria ha sido dividido entre los componentes del grupo, y en este caso comprende la redacción y validación íntegra de los siguientes capítulos: Prólogo, Resumen, Introducción, Arquitectura e Implementación. También se ha redactado subapartados de capítulos cuyo trabajo ha sido dividido entre los componentes del grupo, como por ejemplo el de Conclusiones y trabajo futuro.

También se ha llevado a cabo un proceso de corrección y reestructuración de prácticamente la totalidad del resto de capítulos, reescribiendo todas aquellas partes que pudieran ser sujeto de mejora, de cara a adecuar el contenido a lo desarrollado durante el proyecto además de especificar de la manera más clara posible todo el trabajo realizado.

Álvaro Gómez-Arevalillo González

Ayuda en Investigación

Durante la fase de investigación, se plantearon varias cuestiones. Una de ellas fue conocer cuál sería el sistema operativo en el que se ejecutase finalmente el servicio distribuido. Esto sobretodo nos ha dado información de cuáles pueden ser los mejores servidores en los que se pueden mantener aplicaciones de nuestro tipo dado que necesitábamos seguridad, eficiencia y un buen soporte para mantener la integridad de nuestra aplicación en todo momento por si hay brechas de seguridad.

Para ello, además de buscar información sobre los sistemas operativos, se realizaron pruebas en varios de los sistemas que ofrecían una versión gratuita, como por ejemplo Debian, Ubuntu Server, CentOS para comprobar cuál sería la eficiencia con un fichero de prueba que utiliza nuestro servicio distribuido y que fue creado por mi compañero.

Una vez lanzado este fichero en los distintos sistemas, y en distintas máquinas, se procedió a la investigación de los resultados, con toda la información recabada tanto con las pruebas como con la primera recogida de datos en internet.

Migración a servidor apache

Durante la fase de desarrollo comprobamos que el servidor WSGI que venía por defecto con Flask, Werkzeug no tenía el mismo soporte de cara a una futura implementación en un entorno real y encontrar soluciones a problemas que iban surgiendo era más complicado que en otros. Por ello decidimos realizar una migración a uno de los servidores con mayor soporte debido a la comunidad de desarrolladores que utilizan este servidor web.

Para ello primero se ha realizado previamente una fase de investigación para conocer cuál sería el modo de utilizar código Python en este servidor y que módulos serían necesarios para poder implementarlo correctamente.

Una vez recopilada toda la información necesaria se procedió a la migración del servidor.

Amazon Web Services

Una vez terminado el servicio distribuido se procedió a la subida al entorno cloud en el que se va a presentar finalmente. Esta tarea fue repartida más o menos entre los dos integrantes del grupo. Durante la fase de desarrollo de la aplicación del API, se procedió a la lectura de la documentación de varios entornos cloud, y finalmente nos decantamos por este servicio gratuito.

Tras crear la cuenta, proporcione a mi compañero las credenciales para que configurarse las instancias del servicio. Tras su intervención, yo procedí a la instalación del servicio distribuido y la configuración de la máquina principal.

Benchmark

A día de hoy es normal que la mayoría de productos que ofrecen servicios por internet sean puestos a prueba para probar cual es el rendimiento real del mismo y así poder mejorar la calidad del servicio y así superar a sus posibles competidores en el mercado.

Para realizar las comprobaciones pertinentes a cómo se comportaría nuestra API de cara a un uso normal por parte de usuarios de la misma, he realizado varias baterías de pruebas para obtener los resultados de cuál sería el modo de actuar el servidor actual, el de menores prestaciones de los ofrecidos por Amazon Web Services por el uso gratuito de cara a las peticiones lanzadas por los usuarios. Estas pruebas, como se detallan en dicho apartado han sido lanzadas con el software de Apache Bench.

También, para recopilar más información en distintos entornos, he lanzado las mismas pruebas en un servidor local. De este modo hemos podido obtener más resultados y hemos realizado una comparación de los resultados y valorar más objetivamente el rendimiento del servicio distribuido.

Tras obtener los resultados también se realiza un análisis de los mismos, para identificar posibles mejoras en ciertas partes de la aplicación, como por ejemplo pueden ser tiempos de espera demasiado largos o incluso bloqueo de los servidores a la hora de ofrecer respuestas a peticiones.

Desarrollo de complemento Chrome

Pensamos en hacer un ejemplo práctico de cómo se puede usar nuestra API y para ello decidimos hacer el complemento para navegadores Chrome.

Sobre todo se pensó en darle un uso sencillo para que personas que no tuviesen grandes conocimientos informáticos pudiesen utilizarlo y hoy en día, el uso de complementos en el navegador para diversas funcionalidades, ya sea como traductores, bloqueador de anuncios o como recordatorios, muchos usuarios son capaces de instalarlos y darles uso sin dificultad.

Este ha sido desarrollado utilizando lenguajes de programación web. Principalmente escrito en lenguaje HTML y JavaScript también hemos incluido el uso de frameworks como Jquery.

Para esta parte del cliente se realiza también una investigación para conocer de primera mano cómo se realizan este tipo de complementos, dado que ninguno de los dos componentes del grupo conocía previamente sobre ello.

Memoria

Para finalizar este Trabajo de fin de grado, quedaba tan sólo redactar la memoria de forma que se cumplan todos los requisitos incluidos en las normas. Para ello el proceso de escritura de la memoria ha sido dividido entre los componentes del grupo, y en este caso comprende la redacción de los siguientes capítulos: Tecnologías, algún subapartado como Sistemas Operativos en Arquitectura, Especificación y Pruebas de concepto y medidas de tiempo. Se ha redactado subapartados de capítulos cuyo trabajo ha sido dividido entre los componentes del grupo, como por ejemplo el de conclusiones y trabajo futuro.

También se ha realizado un documento anexo a esta memoria para mostrar cuál sería el procedimiento para instalar el servicio distribuido en el servidor cloud Amazon Web Services y los pasos para instalar el complemento para el navegador Google Chrome.

Además de este trabajo, se ha procedido a la traducción de las partes que eran necesarias para esta memoria como son tanto el Abstract, como los apartados de Introduction y Conclusions and Future work y se ha dado el formato correcto a todo el documento y corregido ortográficamente para que este pueda ser objeto de entrega.

Anexo A. Manual de instalación

Instalación en cloud

Para instalar el servicio distribuido se ha de tener en cuenta que se necesitará como mínimo 3 máquinas para hacerlo funcionar de manera eficiente. Por motivos de rendimiento, y a no ser que todas las máquinas o instancias de las que se disponga tengan las mismas características técnicas, se utilizará la máquina más potente y que disponga de más memoria RAM como servidor principal.

Configuración inicial

Creación de usuario e inicio de sesión en AWS.

Para este paso se debe escoger un usuario y una contraseña para la cuenta. Si se decide hacer la cuenta gratuita se dispondrá de la máquina más simple por un máximo de 750 horas sin coste. (Fig. 14)

Fig. 14 Formulario de inicio de sesión en AWS (Amazon Web Services).

Creación de instancias

Una vez dentro del panel de configuración de la cuenta de AWS, en el dashboard de EC2 aparece la opción de Instancias. (Fig. 15)

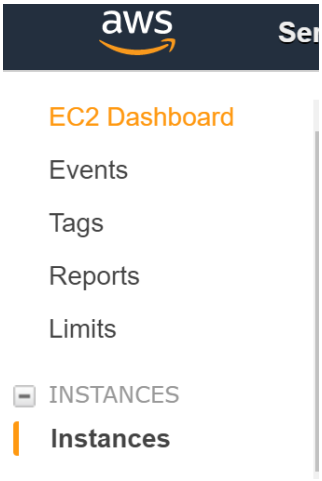
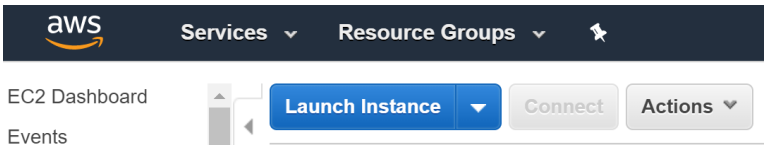


Fig. 15 Dashboard o menú lateral que aparece tras iniciar sesión.

Ahí se puede crear nuevas instancias para nuestro servicio, pulsando el botón de Launch Instance. (Fig. 16)



En nuestro caso, se ha escogido una imagen de Ubuntu Server (Fig. 16) tanto para el servidor principal como para las máquinas del servicio distribuido.

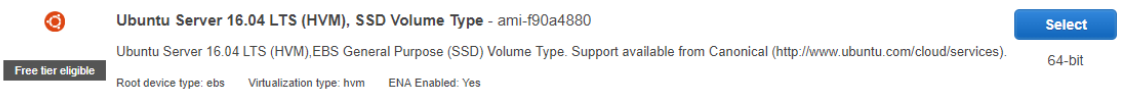


Fig. 16 Imagen de Ubuntu Server seleccionada para nuestro servicio.

Se escoge el tipo de instancia que se va a lanzar. (Fig. 17) Dado que la única opción gratuita que nos ofrece Amazon Web Services es la t2.micro con 1 CPU y 1 Gb de memoria, se elige esa misma.

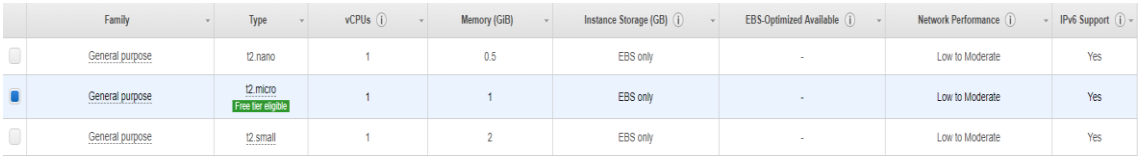


Fig. 17 Tipos de instancias que pueden ser seleccionadas en Amazon EC2.

Por último nos da la opción de crear un par de claves pública y privada para las conexiones ssh o de utilizar una que ya esté creada. (Fig. 18) En nuestro caso, se

crea el par de claves una única vez y se utiliza para la conexión con todas las instancias.

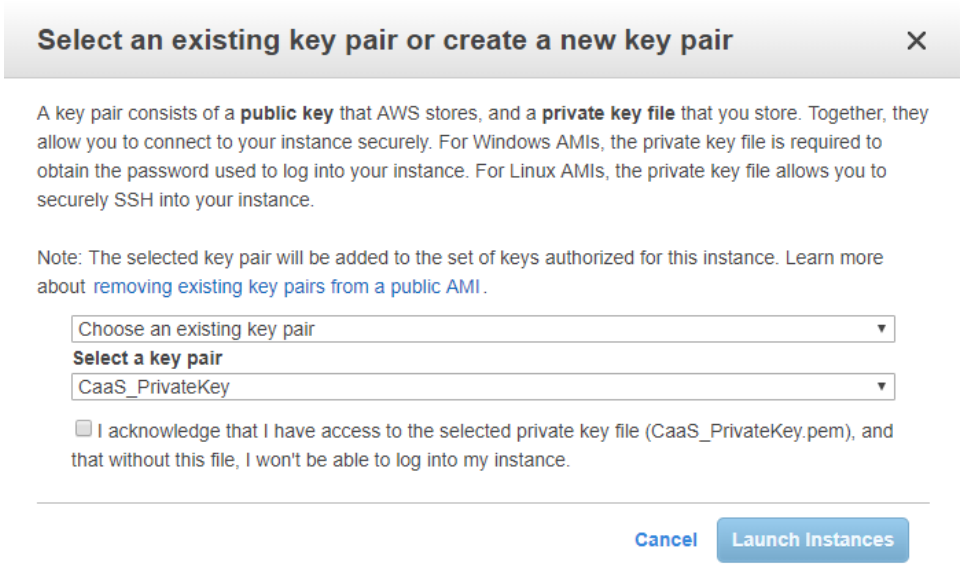


Fig. 18 Formulario de creación de par de claves privada pública para conectarse por SSH.

Configuración básica de instancias

La siguiente configuración es la que se tiene que realizar en todas las máquinas que componen el servicio distribuido, por lo que se podría definir como la puesta a punto por defecto.

Una vez que se dispone de la primera instancia en nuestro panel de control, se obtiene la información con la que se podrá conectar a la máquina. (Fig. 19)

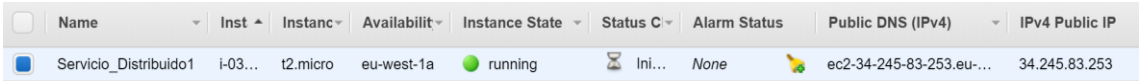


Fig. 19 Menú en el que aparecen todas las instancias creadas y la información de estado, IP, DNS entre otros datos.

Para realizar la conexión por SSH se necesita la clave privada que se ha generado anteriormente.

Se puede realizar la conexión de diferentes formas, por ejemplo por consola utilizando el siguiente comando:

[ssh -i “claveprivada.pem” usuario@ip]

```
root@ubuntu:ssh  -i ‘CaaS_PrivateKey.pem’ ubuntu@ip
```

También se puede realizar la conexión a través de algún cliente SSH como podría ser Putty (Fig. 20). Para esto necesitamos primero generar una Putty Private Key con el programa PuTTyGen que viene incluido en la instalación de Putty.

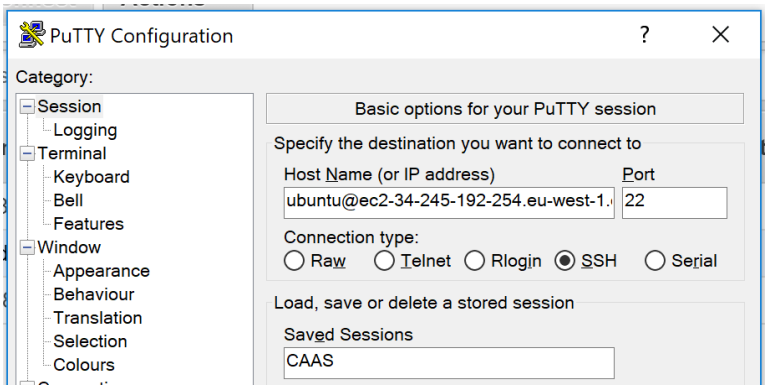


Fig. 20 Aplicación PuTTy. Cliente para conexión SSH.

Bajar el repositorio de github

Se conecta a la instancia y se instala git en el servidor con el comando "apt-get install git". Posteriormente se procederá a descargar el código en nuestra máquina a través del siguiente comando :

```
root@ubuntu: git clone https://TFG-CaaS@github.com/TFG-CaaS/RestApi.git
```

Instalar Python

Para instalar Python en nuestra instancia se debe lanzar el siguiente comando. Para nuestro servicio hemos escogido la versión 2.7 que es la versión RELEASE más actual.

```
root@ubuntu: apt-get install python
```

Instalar pip y módulos necesarios

En Python los módulos que se utilizan los hemos instalado a través de la herramienta pip. Para instalarla se ejecuta el siguiente código:

```
root@ubuntu: apt-get install pip
```

Después se instalan todos los módulos que se utilizan en el servicio:

```
root@ubuntu: pip install pycrypto
```

Duplicar instancias

Una vez configurada la primera instancia de las máquinas subordinadas que componen el servicio distribuido, se procede a exportar una imagen de la

misma. Con esto se puede duplicar la máquina en su estado actual y no tener que repetir el proceso de instalación de dependencias una y otra vez.

Configuración del servidor principal

Se conecta a la máquina que se ha elegido como servidor principal del mismo modo que se ha conectado a la primera instancia y se lleva a cabo la configuración avanzada, ya que en esta instancia se requiere instalar más módulos, mysql y el servidor apache.

Instalar módulos del servidor con pip

```
root@ubuntu: pip install flask
root@ubuntu: pip install flask_sqlalchemy
root@ubuntu: pip install flask_httpauth
root@ubuntu: pip install flask-bson
root@ubuntu: pip install passlib
root@ubuntu: pip install pymysql
```

Instalar MySQL e importar BBDD

En las siguientes líneas de comando se realiza la instalación de MySQL en nuestra máquina. Se accede como root a la consola de administración de Mysql, se crea una base de datos llamada “tfg” y se importa el script de creación de tablas.

```
root@ubuntu:cd RestApi
root@ubuntu:sudo apt-get install mysql-server
root@ubuntu: mysqladmin -p -u root version
mysql>CREATE DATABASE tfg;
mysql>use tfg;
mysql>source tfg.sql;
```

Instalar apache y módulos

Una vez instalado MySQL se procede a la instalación del servidor web Apache. Además de instalar el servidor hay que activar el módulo de ssl e instalar el WSGI para poder ejecutar el código de Python en el servidor web.

```
root@ubuntu: apt-get install apache2
root@ubuntu: a2enmod ssl
root@ubuntu: apt-get install libapache2-mod-wsgi
```

Configurar Apache

Una vez instalados todos los módulos de Apache se añade un *site*.

Se crea también una carpeta en el apartado **/var/www/** , lugar en en el que se dispone todo el código del servicio. En esta carpeta que vamos a llamar CaaS, se deberá crear a su vez dentro de ella una subcarpeta, que será el contenedor de toda la aplicación y que nombraremos como "app" y un archivo que llamaremos "app.wsgi" que servirá de enlace entre el servidor web y la aplicación en Python.

```
root@ubuntu: cd /var/www/
root@ubuntu: mkdir CaaS
root@ubuntu: cd CaaS
root@ubuntu: mkdir app
```

Se mueven todos los archivos del servicio Rest a el servidor:

```
root@ubuntu: mv /RestApi/ /var/www/CaaS/app
```

Y también hay que mover la configuración del wsgi a la ruta **/var/www/CaaS/:**

```
root@ubuntu: mv /RestApi/conf/app.wsgi /var/www/CaaS/app.wsgi
root@ubuntu: a2ensite app.conf
```

Por último, se debe configurar el *site* para que pueda accederse a nuestro servidor. Este apartado se configura en **/etc/apache2/sites-available.**

```
root@ubuntu: mv /RestApi/conf/App.conf /etc/apache2/sites-available/app.conf
root@ubuntu: a2ensite app.conf
```

Puesta en marcha

Máquinas subordinadas

Para que las instancias creadas como máquinas subordinadas se queden escuchando el tráfico a la espera de las peticiones de la máquina principal, se ha de lanzar en ellas el siguiente comando:

```
root@ubuntu: python /RestApi/serviciodistribuido/main_socket.py
```

Máquina principal

Una vez lanzadas las dos instancias mínimas que se necesitan para el funcionamiento correcto del servicio distribuido, se buscan las direcciones IP que han sido asignadas a cada una de las máquinas y se modifica el archivo de configuración situado en /var/www/CaaS/app/conf, el cual contiene la información de las máquinas a las que se va a conectar el servicio distribuido. Esta conexión se realiza por ssh con la clave privada que hemos generado al principio de este manual, con la creación de la primera instancia.

Además, hemos de indicar en la variable "ips" del archivo *controler.*py el número de máquinas que conforman el servicio distribuido.

```
root@ubuntu: sudo su
root@ubuntu: nano /var/www/CaaS/app/conf/config
```

Plugin en Chrome

Para este paso es necesario que el cliente tenga instalado el navegador Google Chrome en su ordenador.

Cuando la aplicación esté disponible en el Chrome Web Store, tendrá que añadirse el plugin de la siguiente manera:

Buscar el Chrome Web Store en el navegador Google Chrome (Fig.21):

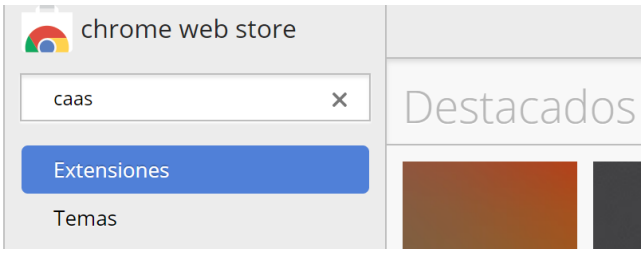


Fig. 21 Campo de búsqueda en Google Chrome Store, a través del cual se puede filtrar por las extensiones o los temas que se pueden añadir al navegador.

Tras haber finalizado la búsqueda y una vez tengamos en la sección derecha el complemento de CaaS, nos metemos en él y pulsamos en el botón añadir a Chrome (Fig.22):

Fig. 22 Apariencia de un complemento sin instalar en Chrome Store.

Una vez instalado el complemento, se habrá añadido al navegador en la barra de herramientas un pequeño icono que representa a nuestro plugin (Fig.23). En caso de no que no aparezca, posiblemente haya que reiniciar el navegador.

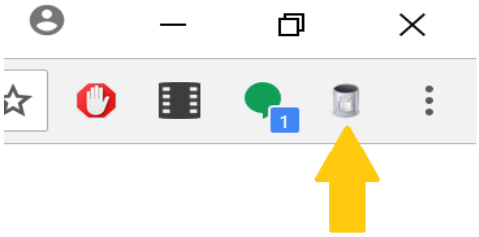


Fig. 23 Ejemplo de una barra de herramientas en el navegador Google Chrome en el que aparecen varias extensiones y la extensión de CaaS marcada por una flecha amarilla.

Una vez que pinchamos en el icono encontraremos algo como la siguiente imagen en la que nos aparecen todas las opciones disponibles. Al no tener una sesión activa, primero se debe hacer login. (Fig. 24)

Fig. 24 Menú desplegable que aparece al pulsar el icono del plugin en la barra de herramientas del navegador sin tener una sesión activa en CaaS.

Menú desplegable que aparece al pulsar el icono del plugin en la barra de herramientas del navegador sin tener una sesión activa en CaaS.

Una vez realizado el inicio de sesión (Fig. 25) se dispondrá de todas las funcionalidades del plugin, como por ejemplo encriptar o desencriptar ficheros o correos.

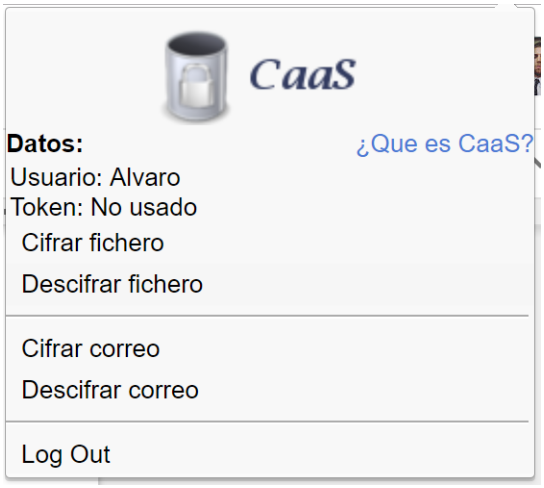


Fig. 25 Menú desplegable que aparece al pulsar el icono del plugin en la barra de herramientas del navegador cuando la sesión está iniciada en CaaS.

Bibliografía

[1] Tipos de encriptado seguro
http://www.toptenreviews.com/software/articles/secure-encryption-methods/

[2] Rendimiento de encriptados
http://thesai.org/Downloads/Volume8No11/Paper_41-A_Survey_on_the_Cryptographic_Encryption_Algorithms.pdf

[3] Documentación de serpent con python
http://legacy.python.org/workshops/1998-11/proceedings/posters/stajano/serpent.html

[4] Autenticación en servidor con WSGI
https://code.google.com/archive/p/modwsgi/wikis/ConfigurationDirectives.wiki#WSGIPassAuthorization

[5] Documentación del mod_wsgi
http://modwsgi.readthedocs.io/en/develop/user-guides/access-control-mechanisms.html

[6] Autenticación de usuarios en api Rest con Flask
https://blog.miguelgrinberg.com/post/restful-authentication-with-flask

[7] Documentación oficial de signal
https://signal.org/
https://signal.org/blog/asynchronous-security/

[8] Estado actual de PGP
https://blog.cryptographyengineering.com/2014/08/13/whats-matter-with-pgp/
https://arstechnica.com/information-technology/2016/12/op-ed-im-giving-up-on-pgp/

[9] Boxcryptor
https://www.boxcryptor.com/

[10] Peak10
http://www.peak10.com/

[11] GNUPG
https://www.gnupg.org/.

[12] Qué es una API REST
https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[13] Historia de python
Mark Pilgrim (2004). Dive into Python. Editorial Apress

[14] Python
http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf

[15] Pycrypto
http://www.ellaberintodefalken.com/2014/04/criptografia-python-pycrypto.html

https://www.dlitz.net/software/pycrypto/api/2.6/

[16] Flask
http://flask.pocoo.org/docs/0.12/

[17] Passlib
https://passlib.readthedocs.io/en/stable/

[18] Paramiko
http://www.paramiko.org/

[19] MySQL
http://ebooks.wtbooks.com/static/wtbooks/ebooks/9781978914490/9781978914490.pdf

[20] MySQL
https://www.mysql.com/about/

[21] Entidad Relación
https://www.campusmvp.es/recursos/post/Disenando-una-base-de-datos-en-el-modelo-relacional.aspx

[22] MyISAM vs InnoDB
https://www.webreunidos.es/blog/myisam-vs-innodb/

[23] Historia de Apache HTTP
http://www.drupaldocker.org/images/APACHE.html

[24] Módulo ssl de apache
https://httpd.apache.org/docs/trunk/es/mod/mod_ssl.html

[25] WSGI
http://www.python.org.ar/wiki/WSGI

[26] Definición de Benchmark
https://computerhoy.com/noticias/moviles/que-es-benchmark-que-sirve-40273

[27] Definición de javascript
https://medium.com/elblogdejavascript/historia-y-fundamentos-b8db8d838946

[27] Definición de jQuery
http://www.andrescampanario.es/es/p/tecnologias/jquery

[28] Django
https://www.djangoproject.com/

[29] Módulos integrados en Django
https://djangopackages.org/grids/
[30] Werkzeug
http://werkzeug.pocoo.org/

[31] Jinja 2
http://jinja.pocoo.org/docs/2.10/

[32] XML
https://www.w3schools.com/xml/xml_whatis.asp

[33] YAML
http://yaml.org/

[34] Por qué es tan popular JSON
https://www.quora.com/Why-is-JSON-so-popular

[35] MongoDB y BSON
https://www.mongodb.com/json-and-bson

[36] Características de las bases de datos NoSQL
https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf
https://blog.pandorafms.org/es/bases-de-datos-nosql/

[37] Port forwarding con SSH
https://www.ssh.com/ssh/tunneling/example
https://blog.trackets.com/2014/05/17/ssh-tunnel-local-and-remote-port-forwarding-explained-with-examples.html

[38] CentOS vs Ubuntu Server
https://thishosting.rocks/centos-vs-ubuntu-server/

[39] HMAC (RFC 2104)
http://www.ietf.org/rfc/rfc2104.txt

[40] It's dangerous (tokens)
https://pythonhosted.org/itsdangerous/

[41] Api Have I Been Pwned
https://haveibeenpwned.com/API/v2

[42] GIL
https://realpython.com/python-gil/

[43] Autenticación HTTP básica
https://tools.ietf.org/html/rfc7235

[44]OpenSSL
https://www.openssl.org/