UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Generación de contraseñas de un sólo uso mediante móviles Android

Alberto Cabello Álvarez
Tutor: David Arroyo Guardeño

Junio 2016

Generación de contraseñas de un sólo uso mediante móviles Android

AUTOR: Alberto Cabello Álvarez
TUTOR: David Arroyo Guardeño

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2016

Resumen

En la actualidad, el número de contraseñas que utilizamos para identificarnos en distintos sitios web está en constante crecimiento. La dificultad para recordar tal cantidad de contraseñas y nombres de usuario desemboca en multitud de malas prácticas por parte de los usuarios, como el empleo de contraseñas cortas y sencillas, más fáciles de recordar, así como la repetición de contraseñas entre sitios diferentes, para solo tener que recordar un número pequeño de contraseñas diferentes.

Este comportamiento constituye un problema de seguridad importante, es más sencillo que la *password* quede comprometida, y una vez que ocurra el daño es mayor. Un atacante que disponga de las credenciales de un sitio, podría acceder a otras cuentas del usuario con facilidad, si se repiten estos datos de acceso.

La seguridad, en particular la autenticación como pieza fundamental, es un aspecto clave en el modelo actual de internet, donde cada vez más datos están asociados a nuestras cuentas personales, siendo aún más peligroso y difícil de remediar el problema de un acceso ajeno una vez una cuenta queda comprometida.

Por estos motivos se pretende dar una solución a este problema. Mediante un gestor de contraseñas disponible en el teléfono Android del usuario, se dispone de una herramienta portátil desde la cual se puede acceder a un sistema que le permite el almacenamiento seguro de claves. La aplicación Android contacta con un servidor web programado utilizando Flask, un web framework escrito en Python y licenciado BSD. Este servicio facilita una manera de mantener contraseñas fuertes, mediante generación automática aleatoria de las mismas. Además, para mayor conveniencia, este servicio presenta una forma de cambiar automáticamente las contraseñas de las cuentas del usuario, en el propio servicio, utilizando el framework Selenium así como otras herramientas de web *scraping* para *logear* en las cuentas del usuario y cambiar las contraseñas cuando se desee.

En este documento se detalla el proceso de diseño y desarrollo de este sistema. Se analiza el estado de estandarización de las páginas web habituales, considerando el impacto de su diseño en herramientas como esta. Por otro lado, se analizan mecanismos de seguridad se pueden aplicar en el tráfico entre aplicación y servidor web, así como en el almacenaje en base de datos. Tras realizar el análisis y diseño de sendas partes, se procede a la creación de ambas.

Palabras clave

Contraseñas, seguridad, criptografía, Android, cliente móvil, Flask, servidor web, gestor de contraseñas, generación de números aleatorios.

Abstract

Currently, the amount of passwords that we use to identify in different websites is constantly growing. To remember that amount of passwords and usernames results in plenty of bad usage from the users, such as using short and simple passwords, easier to remember, as well as repeating said passwords between different sites, in order to only have to remember a small number of different passwords.

This behaviour consists an important security issue, it is easier for the password to become compromised, and once that happens, the damage is greater. An attacker that gained the login information of a website, could access other accounts of that same user with ease, if the login data is repeated.

Security, and authentication as the main concern, is a key factor in the current internet model, where the amount of data linked to our personal accounts is increasing. This makes even more painful and hard to fix this problem, the moment that one of our accounts becomes compromised.

Considering these factors, we attempt to provide an answer to this problem. A password manager, available in Android phones, is a powerful and portable tool, that can connect to a service that allows safe storage and generation of passwords. The Android application contacts with a Python web service programmed using Flask, a BSD licensed micro web framework. This service provides a way to maintain string, randomly generated passwords. Furthermore, for more convenience, a way of automatically changing passwords from the user accounts, from within the web service is available. This system uses the framework Selenium as well as other web scraping tools to login in the user's accounts and change the passwords when requested.

In this document it is detailed the process of design and development of this project. We analyse the current state of standardisation of web pages, considering the impact of their design in tools like this one. On top of that, we analysed security protocols to be applied in the communication between application and web server, as well as the data base storage. After the analysis and design of both parts, we proceed with their creation.

Keywords

Passwords, security, cryptography, Android, mobile client, Flask, web server, password manager, random numbers generation.

Agradecimientos

Mis agradecimientos a todas las personas que me han acompañado este tiempo, en particular a David Arroyo, del que he aprendido gran cantidad sobre el mundo de la seguridad informática, y muchas otras cuestiones.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

La seguridad de las cuentas del usuario es un factor imprescindible para el tipo de uso de internet que se realiza actualmente. En el campo de la seguridad, las contraseñas juegan un papel como un **secreto conocido** dentro de un sistema de autenticación, tal que únicamente el dueño conocería la contraseña, lo que le permitirá identificarse. Los sistemas de autenticación permiten establecer diferentes niveles de acceso pudiendo trazar la relación usuario/entidad – cuenta/identidad. Cabe destacar asimismo, que en sistemas de alta importancia se pueden combinar diferentes pasos de autenticación para reforzar la seguridad, dificultando a intrusos suplantar las cuentas de los usuarios legítimos. Este sistema resulta en el interés en encontrar soluciones para el aumento de seguridad comprometiendo lo menos posible la usabilidad y la molestia que tenga que tomarse el usuario para mantener un esquema seguro. Referente a la molestia, existe el concepto de *fatiga de contraseña* [1], que ya no únicamente en el ámbito digital si no también en general, consiste en la necesidad de recordar e introducir una gran cantidad de contraseñas como parte de la rutina diaria.

Está claro que el nivel de usabilidad o funcionalidad está reñido con el nivel de seguridad. No queda más remedio que tener que tomar medidas adicionales si se quiere garantizar la seguridad de las cuentas de uno mismo. Se busca entonces poder aumentar la seguridad limitando lo menos posible la usabilidad.

Han existido y existirán múltiples problemas de seguridad relacionados con contraseñas. En 2012, la red social orientada a perfiles profesionales Linkedin fue víctima del robo de 6,5 millones de *passwords hasheadas* [2]. Después de pocas horas, pudieron romper gran cantidad de estas contraseñas, más de dos terceras partes de estas contraseñas tenían longitud entre 1 y 8 caracteres. Más del 35% finalizaban con 1 a 3 dígitos, siendo el número '1' el más frecuente como último número [3]. Es evidente que estas contraseñas siguen una serie de estadísticas y reglas que facilitan el trabajo a atacantes. Los procedimientos habituales de los usuarios para completar requerimientos de contraseñas, resultan de esta manera, bastante predecibles, y pueden llevar a heurísticas que faciliten el trabajo de un ataque por fuerza bruta sobre este tipo de bases de datos filtradas. En lugar de tener que probar todas las combinaciones de caracteres posibles como sería en el caso de un ataque sobre un texto completamente desconocido, se puede priorizar palabras conocidas, o estructuras repetidas con más frecuencia. De esta manera, los atacantes reducen las combinaciones que tienen que probar en este tipo de ataques. Con una generación aleatoria es posible dificultar enormemente el trabajo de romper estas contraseñas, ya que no se dispondría de una manera de reducir la búsqueda.

Por otro lado, una motivación más personal para afrontar este proyecto es el poder trabajar con un ámbito no muy abundante en la trayectoria académica presente en el grado de Ingeniería Informática, pero que es de gran interés y de alta importancia actualmente. Este trabajo resulta de una aplicación combinada de diferentes y variados factores y técnicas, poniendo en conjunto lo aprendido para construir un sistema completo siguiendo la estrategia *security by design,* llevando a cabo desde el principio consideraciones de seguridad para una aplicación práctica [4]..

1.2 Objetivos

El objetivo principal del proyecto es construir una aplicación generadora de contraseñas aleatorias que puedan ser empleadas de manera cómoda y *usable* en las cuentas del usuario.

Para ello será necesario un esquema fiable y seguro para la transmisión y almacenamiento de las contraseñas, lo que será fundamental para el éxito de la aplicación.

Se propondrá un modelo para facilitar al usuario obtener una contraseña aleatoria con la que podrá renovar su contraseña de sus cuentas, así como una interfaz a través de la cual poder acceder a ellas y no tener que recordarlas. El objetivo será ofrecer una arquitectura posible para solventar la cuestión propuesta en [5] sobre contraseñas con mecanismo de auto-destrucción. Además, se trata de un modelo de aplicación móvil, de tal manera que no perderá acceso a sus cuentas por no disponer de su ordenador principal.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

El presente documento se estructura aproximadamente, de la manera en la que se realizó el proyecto.

El primer apartado es la presente introducción, dónde se enmarca el trabajo en un contexto, así como se definen los principales objetivos y metas a alcanzar.

En el segundo apartado se realiza un estudio sobre el estado del arte. Se considera la fiabilidad de identificación basada en contraseña, según diferentes modelos de contraseña, proporcionando diferentes niveles de seguridad y usabilidad. Por otro lado, se estudian otras soluciones similares para el problema en cuestión.

En el tercer apartado se realiza todo el análisis de requisitos, definidos según los objetivos que queremos conseguir en la aplicación final y en alcance del proyecto. Se describe la funcionalidad a partir de un catálogo de requisitos y casos de uso, además de las herramientas que serán precisas para construir las partes del sistema en su conjunto.

El cuarto apartado detalla la fase de diseño, donde se consideran las estructuras de datos empleadas, el protocolo de comunicación y cifrado de datos seguido, el flujo de navegación de la aplicación en sí y detalles estructurales.

El quinto apartado describe los pasos seguidos en la fase de implementación, esto es, todo lo relacionado con el proceso de desarrollo, organización, pruebas y resultados obtenidos.

El sexto capítulo traza una conclusión sobre el proyecto, considerando el impacto y la usabilidad que pueden llegar a tener este sistema, así como mejoras posibles que se podrían llegar a realizar, en caso de querer expandir más las funciones que se quieren abarcar.

2 Estado del arte

2.1 Acerca de la seguridad de las contraseñas

Por lo general, la gestión de contraseñas que realiza el usuario general, se basa fundamentalmente en el difícil compromiso y equilibrio comodidad-seguridad. El empleo de diferentes contraseñas complejas, con diferentes tipos de caracteres resulta difícil de recordar, y es una molestia renovar las contraseñas frecuentemente. Estos factores conllevan problemas de seguridad.

Para hacer un intento de mantener un mínimo de complejidad en contraseñas, muchos proveedores de servicios requieren ciertos criterios bajo los cuales restringir la contraseña, criterios de longitud, de uso de diferentes tipos de caracteres, uso de mayúsculas.

Estos criterios, realmente pueden resultar arbitrarios, es más, si estas condiciones son presentadas a modo de lista o como serie de reglas, pueden causar simplemente que el usuario emplee técnicas de generación de contraseñas como el concatenar bloques de letras con bloques de números, o utilizar patrones predecibles, como comenzar por mayúscula, lo cual puede incluso facilitar el trabajo a un atacante.

Deberemos entender, en primer lugar, el tipo de ataques que pueden realizarse contra las cuentas de una víctima. En primer lugar, se puede considerar el tipo de ataque *online*, dónde un atacante realiza intentos de acceso en una página de *login* en activo. El servidor puede detectar estos intentos, y bloquearlo después de cierto número de accesos fallidos. Un ejemplo de este tipo de funcionalidad es Fail2Ban [6]. Fail2Ban es un servicio dedicado a detectar y evitar este tipo de ataques, con múltiples opciones de configuración [7]. Por ejemplo, para el caso de aplicarlo a un servidor SSH, este programa detectará los intentos de login fallidos, y permite bloquear las conexiones de entrada para esa ip atacante tras cierto número de intentos. También dispone de funcionalidad de alerta vía email, quedando informado de esta manera sobre los bloqueos realizados a conexiones entrantes. Sin embargo, esto se puede evitar alargando en el tiempo el ataque.

Otro modo a seguir sería el ataque *offline*, donde el atacante se hace con la posesión de una serie de archivos o bases de datos con los *hashes* de contraseñas. Por lo general, las contraseñas se almacenan tras pasar por una función hash, que transforma una entrada de tamaño arbitrario en una salida de tamaño fijo de forma que la transformación no es reversible. De esta manera al comprobar la validez de esa contraseña, se compara el hash de la contraseña introducida contra el contenido almacenado. Como ejemplo, en los sistemas UNIX se almacenan las contraseñas del sistema como hashes en el fichero **/etc/shadow** [8], en un formato particular.

```
Aychedee:$6$vb1tLY1qiY$M.1ZCqKtJBxBtZm1gRi8Bbkn39KU0YJW1cuMFzTRAN
cNKFKR4RmAQVk4rqQQCkaJT6wXqjUkFcA/qNxLyqW.U/:15405:0:99999:7:::
```

Se trata de una serie de campos separados por ':', el segundo campo, destacado en negrita es el referente al hash. Dentro de esta serie de caracteres se distinguen otros campos separados por $. $6 se refiere al algoritmo utilizado, en este caso SHA-512. El siguiente bloque es una sal aleatoria. Finalmente, el tercer campo es el hash resultante

de combinar la sal con la contraseña y procesandolo a través de la función hash especificada. El resto de campos son referentes a la validez de la contraseña.
En este modelo de ataque, se realizan un gran número de comprobaciones *hasheando* diferentes entradas y buscando ese hash en la serie de contraseñas, posiblemente utilizando una *rainbow table* para este fin. Si se encuentra una coincidencia, se ha encontrado la *password* de ese usuario.

Una forma de mitigar estos ataques, es el hash de las *passwords* junto con una sal, una serie de bits aleatorios que al mezclarse con la password pre-hash, aumentan el tiempo computacional requerido enormemente para romper múltiples *passwords* simultáneamente, y hace inviable el uso de *rainbow tables* [9] . Con este modelo, se incrementa computacionalmente en un factor **n** el procedimiento anterior. Ahora, para comprobar si entre las contraseñas, está una determinada cadena, se tendría que realizar el hash n distintas veces, una por cada usuario con una sal diferente. Además, se evita que encontrando la password de un usuario se puedan detectar todos los demás usuarios con la misma password, siendo, en este caso, los hashes distintos debido a la sal.

Según estas consideraciones, la manera de evitar ataques de estos tipos es el cambio semi-frecuente de contraseñas, así como el uso de contraseñas fuertes, entendiendo como fuerte, una clave tal que sea poco probable que el atacante averigüe, utilizando técnicas como ataques de diccionario.

Por otro lado, nótese que existen tipos de ataques que no se ven afectados por la complejidad de la contraseña. Es el caso de la ingeniería social, obtener información a través de la manipulación del usuario por diversos métodos, como *phishing*, estafas a través del correo electrónico, o introducir *keyloggers* a través de dispositivos de almacenamiento abandonados a la vista, entre otros. Otra ocurrencia para que datos sean comprometidos es el caso de los filtrados desde dentro, los *insider attacks* [10]. Mediante un acceso legítimo a los datos o los sistemas de una organización realizado por un empleado, o similar, puede darse casos de fraude, venta de información valiosa y similares. De este tipo de ataque es difícil protegerse. Algunas opciones para esta protección es restringir niveles de acceso o registrar los movimientos realizados en el sistema de manera adecuada.Un caso en el 2012 de este tipo de ataque consistió en el robo de discos duros con información del servicio de inteligencia suizo, sacándolos de edificios del gobierno. [11]

Otros sistemas de autenticación alternativos son los basados en pruebas de conocimiento nulo, como el protocolo SRP, Secure Remote Password [12]. Este protocolo es resistente a ataques vía *keyloggers* así como de ataques *Man In the Middle* (MiTM), o de observadores en el intercambio de paquetes. Esencialmente consiste en dadas dos partes que conocen un secreto, la contraseña, una parte, el cliente, prueba a la otra, el servidor que la conoce, sin enviar la contraseña en sí. Para este protocolo, la manera de conseguir este objetivo es mediante esquemas de clave pública-privada, generados a través de los datos que tienen ambas partes en su lado.

Por otro lado, existen protocolos como el **OAuth** [13], que permiten la identificación a webs de terceros utilizando identidades existentes de otras páginas, como cuentas de Google o Twitter. El mecanismo consiste en un intercambio de tokens de identificación, tal que la página referencia al sitio de login externo, el usuario se identifica y esta página concede un token de acceso al primer sitio, al que se desea

acceder. De esta manera se puede identificar al usuario mediante una de sus cuentas existentes sin exponer su contraseña, ya que el intercambio realizado es únicamente mediante tokens de identificación. Además, con una identificación de esta manera, sólo precisa recordar una contraseña, potencialmente para poder identificarse en múltiples sitios que sigan este protocolo u otros similares. Esto es una ventaja y un inconveniente, porque resulta en un *single point of failure,* si queda comprometida una sola cuenta, el resto pueden quedar igual.

2.2 Acerca de la generación automática de contraseñas

La mayoría de las páginas no cuentan con todos los mecanismos auxiliares que mejoren su paradigma de seguridad, o con protocolos alternativos a la identificación vía contraseña estándar. Según esto, el método de acción para mantener un sistema seguro es mantener contraseñas robustas, distintas y renovarlas cada cierto tiempo. Aquí entran en juego los gestores de contraseñas. Un generador de *passwords* aleatorio, es una pieza de software que nos puede servir para obtener contraseñas lo más "fuertes" posible. Según un análisis de contraseñas utilizadas por usuarios, en 3 millones de contraseñas de 8 caracteres, la letra 'e' fue empleada más de 1,5 millones de veces, siendo la más común, frente a, por ejemplo, la 'f', siendo utilizada sólo 250.000 veces [14]. Además, raramente se utilizan símbolos no alfanuméricos, a no ser que sean expresamente requeridos.

Según esto, uno de los objetivos de un generador de contraseñas sería producir contraseñas con una distribución más próxima a la distribución uniforme. En el ejemplo anterior, cada carácter debería haber sido empleado aproximadamente 900.000 veces.

Un método "manual" de generación aleatoria podría ser a través de tiradas de dados. Usando una tabla 6x6 y asignando diferentes caracteres a casillas, se pueden seleccionar 36 caracteres de forma razonablemente aleatoria. Este método se puede ampliar incluso con otros tipos de dados, tablas más grandes, o añadiendo la tirada de una moneda para seleccionar entre mayúscula/minúscula, etc. La siguiente tabla podría ser una variante sencilla de este método. Otro sistemas más sofisticados de este mismo tipo es el que realiza **Diceware** [15] , que a partir de una lista de palabras indexadas con dígitos del 1-6, tira 6 dados y selecciona las palabras correspondientes, de manera aleatoria.Esto tiene la ventaja de ser más fácil de recordar, puesto que es una serie de palabras.

	1	*2*	*3*	*4*	*5*	*6*
1	**a**	**b**	**c**	**d**	**e**	**f**
2	**g**	**h**	**i**	**j**	**k**	**l**
3	**m**	**n**	**o**	**p**	**q**	**r**
4	**s**	**t**	**u**	**v**	**w**	**x**
5	**y**	**z**	**0**	**1**	**2**	**3**
6	**4**	**5**	**6**	**7**	**8**	**9**

Tabla 1: Generación manual aleatoria

Métodos computacionales para este objetivo se topan directamente con la cuestión de diseño de un generador de números aleatorios seguro.

Se pueden considerar dos tipos de generadores de números aleatorios [16], los que utilizan información realmente aleatoria para generar números, como a partir de condiciones atmosféricas, nivel de ruido... Por otro lado, están los generadores pseudoaleatorios, que aplican funciones deterministas junto con una inicialización o semilla desde la que partir.

Una propiedad fundamental es que los números parezcan realmente aleatorios [17], a pesar de estar generados de forma determinista, esto es, que la secuencia de números generados estén uniformemente distribuidos. Además, no se debería poder conocer que número se generará siguiente, a partir de una secuencia anterior, ni debería ser posible encontrar la semilla de inicialización.

La elección de la semilla también es un factor fundamental para la seguridad criptográfica del generador. Una implementación habitual en de números aleatorios en C es inicializar **srand** con el tiempo actual. Esto resulta problemático: para recrear esta secuencia basta con probar con todas las semillas posibles en el intervalo de tiempo que se piense que se ha generado. Como en principio, las contraseñas deberían tener un espacio de vida corto, se acota mucho el espacio de búsqueda. Si el atacante conoce el algoritmo usado para generar las contraseñas, nos encontramos con un problema importante, especialmente en el caso de ataques *offline*.

Teniendo en cuenta esto, para la selección de la semilla se pueden usar generadores aleatorios de la primera categoría, los verdaderamente aleatorios, y a partir de esa semilla, utilizar un algoritmo determinista para obtener números pseudoaleatorios.

En sistemas tipo Unix, se puede leer de /dev/random y **/dev/urandom** para acceder a ruido procedente de *drivers* y otras fuentes, almacenado en forma de un *'pool de entropía'* [18]. Su propósito es funcionar como una interfaz al generador de números aleatorios del kernel, produciendo una pequeña cantidad de bits para usar como semilla de un generador pseudoaleatorio.

Otra opción podría ser utilizar un algoritmo de cifrado como AES como método para generar números aleatorios. En este caso, la clave de cifrado tomaría el papel de semilla, y la entrada al algoritmo podría ser una secuencia incremental de números [19]. Debido al criterio de avalancha [16] presente en este algoritmo de cifrado u otros, es posible a partir de una secuencia de números incremental, obtener un cambio significativo en la salida, consiguiendo un efecto de generación de números aleatorios.

En definitiva, para la generación segura de contraseñas aleatorias, se precisa un algoritmo de generación de números pseudoaleatorios lo suficientemente efectivo, así como una manera confiable y necesariamente aleatoria con la que obtener una semilla para el algoritmo anterior.

Por otro lado, los generadores de números aleatorios están directamente relacionados con los algoritmos de cifrado de flujo, como RC4 o A5 [20]. En general, en estos algoritmos se combina cada bit del mensaje con un bit de un stream de números aleatorios, mediante operaciones XOR. De nuevo se vuelve a la cuestión de fuentes de flujo de números aleatorios como pieza central. En 2004, un proyecto de la Union Europea, eSTREAM [21] se llevó a cabo, con el fin de solicitar propuestas para nuevos cifrados de flujo. El objetivo era que fuesen adoptados ampliamente, estableciendo una serie de algoritmos como herramientas que poder utilizar con

diferentes características y ventajas. En la siguiente tabla se pueden observar los algoritmos aceptados por esta entidad [22].

Profile 1 (software)	Profile 2 (hardware)
HC-128	Grain
Rabbit	MICKEY
Salsa20/12	Trivium
SOSEMANUK	

Tabla 2: Portfolio de cifrados de flujo eSTREAM

2.3 Acerca de los gestores de contraseñas

Un gestor de contraseñas es una aplicación software que permite al usuario mantener *passwords* seguras para utilizarlas en el resto de sus aplicaciones.

En la forma más básica, se trata de, simplemente una manera de organizar claves usuario-contraseña, generalmente tras una contraseña maestra que regula el acceso a la aplicación.

Estos gestores pueden presentar diferentes particularidades en cuanto a su diseño y funcionalidad, cada cual con ventajas e inconvenientes. Una opción es realizar todo el proceso y almacenamiento mediante un software en el ordenador del usuario, bien sea completamente independiente, o como plugin del navegador. Actualmente los navegadores habituales como Chrome o Firefox vienen equipados con sus propias versiones de gestores de contraseñas. Normalmente permiten el relleno de campos en los formularios de login. Otros gestores independientes comerciales, son aplicaciones como LastPass, Dashlane o Zoho Vault. Cada uno con características determinadas como completar formularios, guardar las contraseñas al ingresar con ellas, autenticación en dos pasos y otras [23].

El principal inconveniente de este modelo, es que si un atacante toma acceso del ordenador, dispone de todas las contraseñas guardadas. Chrome, por ejemplo, tiene una opción para revelar las contraseñas guardadas, tras solicitar la contraseña del usuario. Por otro lado, tienen ventajas como la protección contra *keyloggers*, al no tener que introducir la contraseña manualmente, de manera que no pueden capturar las teclas. Además, evitan problemas de *phishing*, ya que si no reconocen la página como a la que verdaderamente pertenece la contraseña, no ofrecerán hacer el login.

Una variante consistiría en una aplicación portable que resida en un dispositivo portable, como una memoria USB. Esto permitiría mantener el almacenamiento fuera de la red, salvo en los momentos en los que se use, de manera que el acceso por parte de un atacante sería difícil.

Otro modelo es el basado en web. Mediante servicios web ofrecen el almacenaje de información cifrada. Una ventaja es que se puede separar el contenido encriptado de la clave de cifrado. Como en los servidores no se encuentra la clave de cifrado, aún habiendo un *leak* de los datos, no sería posible recuperar el texto plano directamente.

En definitiva, el uso de un gestor de contraseñas de algún tipo es un neto positivo, pues permite mantener contraseñas distintas y de fuerza superior a las que pueda

recordar una persona mentalmente. Este uso no implica aún así un modelo infranqueable, con *malware* en el sistema se puede capturar la contraseña una vez introducida, bien mediante *keyloggers* o criptoanálisis acústico [24]. Otra fuente de debilidades es un generador de números aleatorios débil, como comentado en el apartado anterior. Existen, por otro lado, problemas específicos a los gestores que utilizan mecanismos de relleno automático de las credenciales [25]. Un ejemplo de este caso son páginas malintencionadas, posiblemente usadas como página de confirmación al entrar en una red WiFi pública, que contienen frames invisibles que referencian a páginas de login, engañando posiblemente de esta manera al gestor para que los complete y posteriormente extraer esos campos vía JavaScript.

Es necesario un uso con cautela de gestores de contraseña, y un diseño adecuado, ya que por motivos como estos se puede ver que se introduce un *single point of failure*. Depende de la estructura del gestor de contraseñas, si bien la cuenta del propio gestor o la aplicación de escritorio queda comprometida, puede desencadenar la pérdida del resto de cuentas. Por este motivo es especialmente crítica la robustez del sistema.

Estos factores hace que sea conveniente no tener como único medio de protección para cuentas importantes el gestor de contraseñas. Un uso adecuado de un gestor lo combinaría con un empleo de sistemas con múltiples pasos de autenticación, posiblemente combinando varios factores o modos de autenticación también.

Estos factores de autenticación corresponden a tres categorías: algo que se conoce (un secreto conocido, una contraseña), algo que se posee (un objeto físico que permita autenticación) o algo que se es (característica biométrica). Generalmente, como primer factor se realizaría la identificación habitual con contraseña, proceso que se refuerza empleando un segundo factor, como un sms o email de confirmación, o diferentes tipos de tokens, por ejemplo, los basados en algoritmos basados en generación de tokens en función del tiempo [26]. Este último sistema consiste en introducir un código obtenido a través de un dispositivo local, como un teléfono, que se haya sincronizado previamente con el servidor a través de un canal seguro. El dispositivo genera la clave de un solo uso, a través de un HMAC de la clave compartida junto a una marca temporal, que el usuario introduce como segundo factor de autenticación. Otros sistemas disponen de dispositivos hardware específicos para este propósito. Un ejemplo clásico de autenticación en dos pasos es el procedimiento que exige confirmación mediante a través de una tabla de coordenadas, después de una autenticación previa. Este modelo es un segundo factor de autenticación dinámico utilizado frecuentemente en banca electrónica.

El problema de este método es que no todos los servicios lo tienen disponible, y cada uno implementa sus propias versiones de autenticación en dos pasos [9], aceptando unos tipos pero otros no.. Esto resulta en un sistema difícil de unificar, pero sin embargo, muy útil para proteger las cuentas más importantes. Aunque se obtuviese la contraseña de esa cuenta, no se podría realizar el login sin la segunda fase de autenticación.

3 Análisis

En este apartado, se estudiará el problema propuesto, estableciendo los objetivos a alcanzar a partir de requisitos funcionales y no funcionales, además de los casos de uso.

3.1 Definición del proyecto

El objetivo principal de este proyecto es construir un sistema con estructura cliente-servidor, con el fin de utilizarlo como gestor de contraseñas, manteniendo toda la usabilidad posible así como facilitando un esquema seguro para que el usuario organice los datos de sus cuentas.

El modelo cliente-servidor corresponde al de un servidor web programado en Python con el framework Flask [28], donde se realizará el grueso de las operaciones de generación aleatoria de contraseñas, que el usuario obtendrá y podrá usar como clave de acceso para sus cuentas y el almacenamiento de las mismas. El servidor se comunicará con el cliente Android a través del cual el usuario podrá acceder a sus datos.

Este objetivo sirve como medio de un estudio sobre todo tipo de cuestiones de seguridad, profundizando sobre métodos de criptografía, especialmente sobre la manera de aplicarlas de una manera correcta en un sistema funcional. Es importante aplicar estos algoritmos de manera correcta, las debilidades de la criptografía, utilizando esquemas de cifrado o hash actualizados, como AES [29], RSA [30] o SHA-2 [31], no provienen de errores en estos algoritmos, por ejemplo, en el caso del RSA, los atacantes no disponen de un método para descomponer en primos un número del tamaño que se maneja, 1024-4096 bits [32]. Los problemas generalmente, provienen de malos usos de ellos.

En el ámbito de este proyecto se realizará el cliente que contacte con la base de datos para móviles Android, sin embargo, se proporcionará asimismo una interfaz con las rutas url, los parámetros y resultados del servidor web, de tal modo que se permitirá otras implementaciones para otros tipos de móviles o aplicaciones de escritorio.

El diseño de un cliente móvil como punto de acceso a la aplicación favorecerá el manejo de contraseñas, de tal manera que se puede tener acceso a ellas en cualquier punto. Además, el teléfono móvil consiste en un instrumento portátil de uso muy extendido actualmente, lo que lo convierte en una herramienta apropiada para este propósito, que ofrece además opciones de notificaciones, útiles para la aplicación.

3.2 Catálogo de requisitos

3.2.1 Requisitos Funcionales

En este punto se presentan los requisitos funcionales, esto es, los requisitos relacionados directamente con la funcionalidad que debe cumplir la aplicación.

RF 1 Para acceder a su cuenta, un usuario empleará su nombre de usuario junto con su contraseña maestra.

RF 2 El usuario podrá registrar los datos de una cuenta de determinada web, a la que se asignará una contraseña aleatoria.

RF 3 Se podrán consultar las contraseñas guardadas para permitir el acceso a las cuentas cuando el usuario lo requiera.

RF 4 El usuario puede solicitar a la aplicación que le asigne una nueva contraseña a una de sus cuentas.

RF 5 Para un conjunto de páginas de uso frecuente, se habilitará funcionalidad de cambio automático de contraseña.

RF 6 Generación automática de contraseñas aleatorias, según ciertos parámetros (longitud, minúsculas, mayúsculas, números, símbolos)

RF 7 Se notificará al usuario de las contraseñas que hayan expirado según el tiempo de vida estipulado, a través del sistema de notificaciones FCM.

RF 8 Se guardará un registro de las entradas de cada usuario a su cuenta, de tal manera que el usuario podrá comprobarlo para poder ver su actividad y detectar posibles intrusos.

3.2.2 Requisitos no funcionales

En este apartado se listan los requisitos no funcionales, todos los aspectos no relacionados directamente con la funcionalidad, si no con el resto de elementos deseados en la aplicación adicionales.

Seguridad

RNF 1 Toda la comunicación entre cliente-servidor se realizará exclusivamente con el protocolo SSL.

RNF 2 Las contraseñas de login de la aplicación de los usuarios se almacenarán tras un hash con el algoritmo SHA-256

RNF 3 Para acceder a las funcionalidades de la aplicación, se necesitará un login con los datos del usuario.

RNF 4 Los datos de las cuentas de cada usuario permanecerán encriptados con AES, donde la clave de cifrado depende de la contraseña maestra del usuario, de forma que no es posible el descifrado sólo desde el lado de servidor.

RNF 5 Todos los cifrados o hashes se combinarán previamente con una sal aleatoria con fin de reforzar la seguridad frente a varios tipos de ataques.

Disponibilidad

RNF 6 En la versión del sistema en producción, el servidor debe mantener una disponibilidad 24/7, es fundamental permitir a los usuarios acceso a sus datos en todo momento.

Documentación

RNF 7 El código será documentado adecuadamente en cuanto a comentarios para permitir la legibilidad adecuada y facilitar las ampliaciones.

Eficiencia y rendimiento

RNF 8 El cliente Android será lo suficientemente ligero como para conseguir una navegación entre menús y opciones de la aplicación rápida, no mayor de 2 segundos en estas transiciones.

RNF 9 El tiempo de proceso de datos en el servidor, incluyendo operaciones de cifrado/descifrado y acceso a base de datos se realizarán no serán superiores a 3 segundos, para evitar esperas en la comunicación con el cliente.

Usabilidad

RNF 10 Este sistema deberá ser una opción viable para que el usuario gestione sus cuentas adecuadamente a través de ella.

RNF 11 El cliente Android utilizará pautas y herramientas de material design

Interoperabilidad

RNF 12 El servidor web será realizado de tal manera que permita compatibilidad con cualquier tipo de cliente, mientras que este realice las peticiones al servidor de la manera adecuada.

Mantenibilidad

RNF 13 Ambas partes del sistema mantendrán una estructura modular, clara y adecuada a la ampliación con nuevas prestaciones si así se desea.

Portabilidad

RNF 14 El usuario podrá acceder al servicio web independientemente de la localización, mientras que disponga de su teléfono con la aplicación cliente instalada y conexión a internet.

3.3 Casos de uso

En esta sección se presenta el diagrama de casos de uso de la aplicación. Este muestra las principales funciones e interacciones que el usuario podrá realizar con la aplicación. Además, se detallan estos casos de uso según el actor y su descripción, además de precondiciones y postcondiciones.

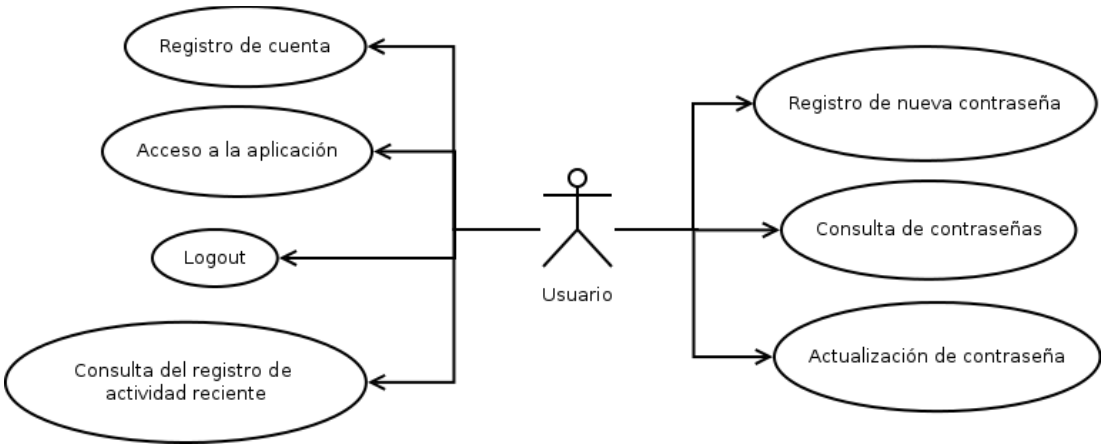

Figura 1: Diagrama de casos de uso

Caso de uso: Registro de cuenta

Actor: Usuario sin cuenta que quiere comenzar a usar el sistema.

Descripción: En usuario introduce el nombre de usuario que quiere tomar y su contraseña.

Precondiciones: El nombre de usuario deseado no está en uso.

Postcondiciones: El usuario queda registrado correctamente y logeado en el sistema.

Caso de uso: Acceso a la aplicación

Actor: Usuario registrado sin logearse.

Descripción: El usuario introduce sus credenciales y pulsa el botón de login.

Precondiciones: La cuenta del usuario existe.

Postcondiciones: El usuario queda logeado. Se continúa con la navegación de la aplicación.

Caso de uso: Registro de nueva contraseña.

Actor: Usuario logeado.

Descripción: El usuario navega a la pestaña correspondiente e introduce los datos de la nueva cuenta que sincronizar con el servidor.

Precondiciones: El usuario previamente ha iniciado sesión.

Postcondiciones: Se registra la cuenta nueva en el servidor. El servidor asigna una contraseña nueva aleatoria.

Caso de uso: Consulta de contraseñas.

Actor: Usuario logeado.

Descripción: El usuario navega a la pestaña de consulta de contraseñas y pulsa en la cuenta en particular que quiere consultar.

Precondiciones: El usuario ha iniciado sesión.

Postcondiciones: Se muestra la contraseña para que el usuario pueda introducirla y hacer login donde corresponda.

Caso de uso: Actualización de contraseña.

Actor: Usuario logeado.

Descripción: El usuario accede al menú de cambio de contraseñas. Seleccionando la cuenta a actualizar, y aceptando el dialogo de confirmación, el servidor proporciona una contraseña aleatoria nueva.

Precondiciones: El usuario ha iniciado sesión. El usuario dispone de cuentas a las que actualizar la contraseña.

Postcondiciones: El servidor devuelve una contraseña nueva para la cuenta del usuario. Se pide al usuario que actualice la contraseña en la página, a no ser que la cuenta sea una de las que dispone de cambio automático.

Caso de uso: Logout

Actor: Usuario logeado.

Descripción: El usuario decide salir de la aplicación y cerrar su sesión en el servidor.

Precondiciones: El usuario ha iniciado sesión.

Postcondiciones: Se cierra la sesión del usuario en el cliente y en el servidor. Se retorna a la pantalla de login.

Caso de uso: Consulta del registro de actividad

Actor: Usuario logeado.

Descripción: El usuario desde la opción correspondiente, comprueba el registro de entradas a su cuenta.

Precondiciones: El usuario ha iniciado sesión.

Postcondiciones: Se muestra el registro, incluyendo la fecha de acceso y la dirección ip asociada.

4 Diseño

En este apartado se detallará el diseño de tanto la aplicación cliente Android como el servidor web. Consideraremos la forma de estructurar este proyecto para alcanzar una solución adecuada, así como los factores y elementos de diseño que proporcionarán la mejor estructura y funcionalidad posibles.

4.1 Protocolos de comunicación y cifrado de datos

Como ya comentamos, el proyecto consiste en una aplicación Android como cliente que se comunica con un servidor web que realiza las operaciones correspondientes.

Esta comunicación será exclusivamente mediante el protocolo SSL, que permita una conexión segura entre ambos y protección de los datos transmitidos. Para habilitar esta comunicación, utilizaremos certificados X.509 [33], generados a través de OpenSSL [34].

Las contraseñas de login al servidor de los usuarios son la llave del usuario al resto de sus contraseñas. Cuando el usuario se registra se le asigna una sal para esa contraseña y se combinan y realiza el hash con el algoritmo SHA-256 (Secure Hash Algorithm), que se guarda junto a la sal en base de datos. Al realizar el login, se comparará contra este hash.

Por otro lado, el esquema de cifrado utilizado en el resto de contraseñas asociadas a la cuenta del usuario es el siguiente:

La contraseña nueva aleatoria se genera, que será la cadena de caracteres usada como llave de la cuenta correspondiente. Por defecto, serán 16 caracteres con números y símbolos incluidos. La contraseña maestra del usuario se combina con una sal aleatoria a través del algoritmo PBKDF2 [35]. Se trata de un algoritmo que itera una serie de procesos Hash/HMAC para conseguir una llave derivada, que se utilizará como clave del cifrado AES posterior. Este proceso aumenta la complejidad computacional que complica derivar la clave, además de atar la clave maestra como llave de las otras claves, de manera que exclusivamente desde el servidor no se puede desencriptar. Como apunte histórico, PBKDF2 es un algoritmo seguro, actualmente, aunque entre 2013-2015 se llevó a cabo un concurso, Password Hashing Competition [36] para buscar una alternativa. El algoritmo ganador ha sido el argon2 [37]. Este tipo de tecnología está constantemente en desarrollo, así que se podría considerar este algoritmo para propósitos similares, aunque PBKDF2 continúa siendo ampliamente utilizado.

Una vez obtenidos esos 2 elementos, clave de cifrado y contraseña, se cifra la contraseña con AES usando como clave los 16 bytes obtenidos del PBKDF2. Este texto cifrado resultante se almacena en la base de datos, junto con la sal aleatoria empleada. AES, Advanced Encryption Standard [29], es el algoritmo de cifrado simétrico utilizado, siendo el estándar para estos propósitos desde 2002. Se trata de un algoritmo de cifrado por bloques, originalmente conocido como Rijndael, posteriormente como AES tras ser aprobado por el NIST y establecido como estándar, sustituyendo al DES como algoritmo previo, quedando este retirado como método de cifrado aceptable [38], siendo el uso del DES desaconsejado.

4.2 Flujo de operación

La siguiente figura es un ejemplo del tipo de interacciones que el usuario realizará con el cliente móvil. Se muestra el procedimiento de almacenado/obtención de datos con el fin de gestionar la cuenta de un sitio web externo.

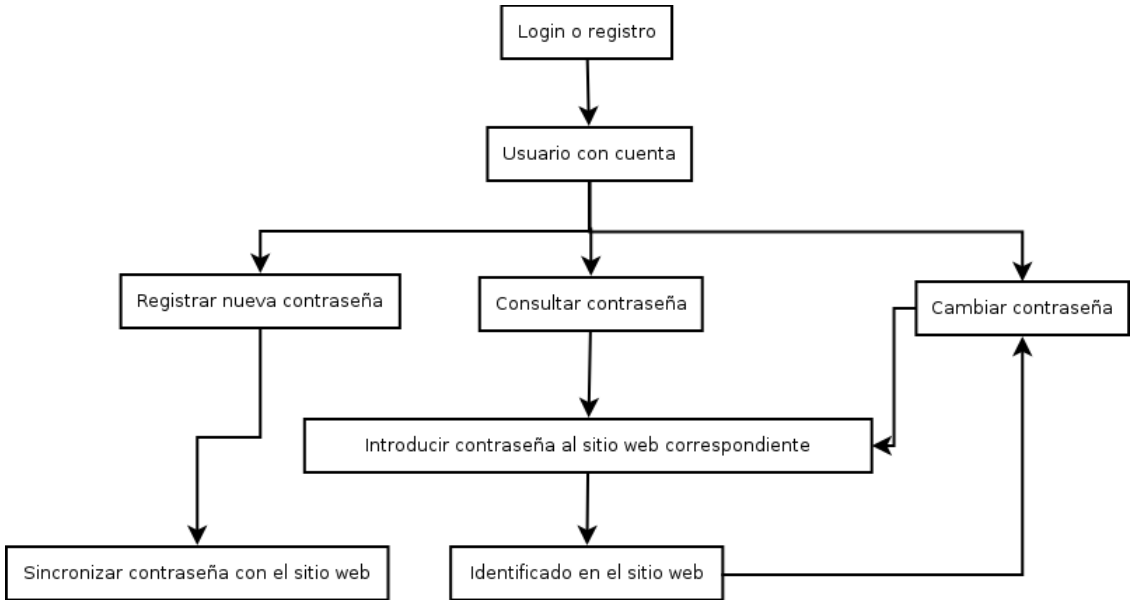

Figura 2: Diagrama del flujo de operación del cliente móvil

4.3 Estructura física de la aplicación

Teniendo en cuenta las características del proyecto, dividiremos el código en cliente y servidor. El cliente consta de la estructura de un proyecto habitual de Android, con los directorios propios de este tipo de proyectos: *gradle*, *resources*, etc. En cuanto al código propiamente, se sigue la estructura de paquetes habitual, con la notación de dominio inverso [39]. A continuación se detallan los paquetes con los archivos correspondientes y su propósito. En la figura 3 se puede ver esto mismo en el proyecto de Android Studio.

es.uam.eps.passwordmanager.activities. Este paquete contiene las clases que representan actividades. Hay algunas actividades propiamente dichas así como

es.uam.eps.passwordmanager.fragments. En este paquete residen las clases que extienden **Fragment.** Estas pertenecerán posteriormente a una actividad desde las que se ejecutan, en este caso, a MainActivity.

es.uam.eps.passwordmanager.networking. Aquí se hallan las clases relacionadas con todas las cuestiones de comunicación de datos. En particular la clase ServerMethods contiene la mayor parte de la comunicación con el servidor.

es.uam.eps.passwordmanager.view. Este paquete contiene las cuestiones relacionadas con la forma de expresar los datos complejos obtenidos desde el servidor. Se implementan los elementos necesarios para el funcionamiento de un RecyclerView, un elemento Android que permite mostrar información en forma de lista de manera flexible.

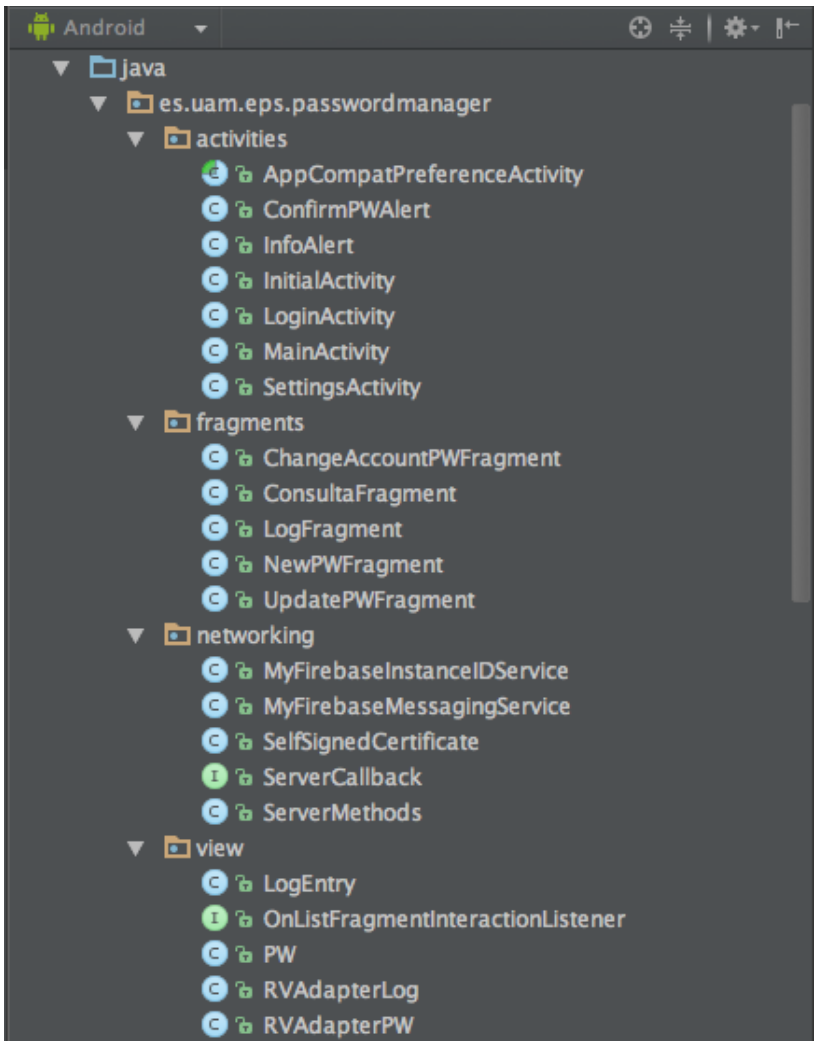

Figura 3: Estructura de paquetes de la aplicación Android

Por otro lado, el servidor es un proyecto Python, haciendo uso del framework Flask para realizar la función de un servidor web.

En primer lugar, en esta parte se encuentra la definición de la base de datos, en **schema.sql**, así como la base de datos propiamente dicha en **database.db**, archivo de base de datos de SQLite.

Seguidamente, consideraremos los ficheros **changepw.py** y **cypher.py**. Se tratan de módulos independientes para realizar las funciones correspondientes que se invocan desde el servidor.

El fichero **changepw.py** contiene toda la lógica diseñada para realizar cambios automáticos de contraseña en una serie de webs predeterminadas. Se hace uso del *driver* Selenium para hacer login automático en diferentes servicios, y completar las solicitudes de cambio de contraseña, para ahorrar tiempo al usuario.

cypher.py contiene la función de generación de contraseñas, la que se puede parametrizar según ciertos sets de caracteres que se quieran utilizar. Además se encuentra la clase **AESCipher**, una interfaz del algoritmo AES adaptada al uso que se hará de este cifrado en la aplicación. Se encarga de las cuestiones de *padding*, *unpadding*, gestión de la sal y de codificación/decodificación como *strings*. Estas operaciones hacen uso de los módulos de criptografía de Python **pycrypto**, así como de **/dev/urandom** para generación de cadenas de bits aleatorios.

4.4 Estructuras de datos

En este sistema existirá una única base de datos, que reside en el lado del servidor. Utilizaremos una base de datos SQLite, principalmente por su facilidad de uso. Los métodos de acceso residirán en el servidor, y estarán enmascarados por peticiones POST que parsearán los parámetros y luego realizarán las consultas correspondientes.

La ejecución de consultas se realiza con el equivalente de *prepared statements* de la API de SQLite para Python, para evitar ataques de inyección SQL que podrían ocurrir en el caso de manipulación de las consultas a realizar mediante operaciones habituales con *strings*.

El modelo relacional que utilizaremos será muy sencillo: una tabla de usuarios y una tabla de contraseñas. Además, se dispondrá de una tabla con el registro de accesos a cada cuenta. La definición de la base de datos se encuentra en el fichero schema.sql. Los campos y demás propiedades se pueden observar en la siguiente tabla:

Users							
Userid	Username	Userpw	Usersalt	phone_id			
Passwords							
Id	Ownerid	Website	Ws username	Pw	Salt	Created	Expires
Log							
Userid	Tstamp	Extrainfo					

Tabla 3: Esquema base de datos.

5 Integración, pruebas y resultados

En este apartado se considera el proceso de desarrollo para este proyecto, así como las herramientas utilizadas, pruebas y resultados obtenidos.

5.1 Equipo de desarrollo

El proyecto ha sido abarcado utilizando una serie de programas y técnicas para la realización del mismo. Para el desarrollo del cliente Android, se empleó el entorno de diseño Android Studio [40], la herramienta oficial para desarrollo de Android. Esto proviene con soporte para la compilación así como herramientas de *debug* o fragmentos de código o de ficheros de diseño disponibles para su uso.

Para las pruebas del cliente se utilizó tanto el emulador disponible desde Android Studio como un terminal físico.

Por otro lado, la codificación del servidor fue realizada en Python 3.4 [41], utilizando **pip** o **easy_install** como administrador de paquetes, para la instalación sencilla a través de línea de comandos de los paquetes necesarios. Se utiliza esta versión en lugar de 2.x ya que, pese a seguir habiendo gran cantidad de código funcionando escrito para Python 2, esta es una versión de legado, esencialmente [42]. Para un proyecto nuevo, el que no es necesario continuar un código existente, es más recomendable partir directamente desde la última versión. Uno de los pequeños inconvenientes sería que haya un ligeramente inferior soporte para bibliotecas o módulos pequeños, aunque generalmente no es un problema, ya que suele haber alternativas o maneras para portar código. Se ha codificado de la manera más estándar posible, aunque podrían existir problemas de compatibilidad con un intérprete de Python 2 [43], en particular con respecto al soporte de *encoding* en algunos puntos, aspecto que se mejoró para Python 3 [42]. En cualquier caso, ejecutando el código desde un intérprete de Python 3.4 con los paquetes necesarios instalados no debería suponer ningún problema.

Con el fin de probar los métodos del servidor web utilizaremos **cURL** [44], una herramienta de línea de comandos para realizar todo tipo de peticiones. En este caso, lo utilizaremos para realizar peticiones POST según protocolos http o https. cURL permite visualizar las cabeceras y otros campos de la petición, así como asignar parámetros POST fácilmente, ver códigos de respuesta, cookies, etc. Otro método para realizar lo mismo es utilizar las herramientas de desarrollador de un navegador, en este caso se empleó también los navegadores Firefox y Google Chrome.

En cuanto al control de versiones, la herramienta utilizada ha sido utilizado git, desde la terminal, alojando el contenido tanto en local como en un repositorio en **Bitbucket** [45].

Para la redacción del presente documento, se ha utilizado el procesador de texto de código abierto **LibreOffice**.

Para la gestión de las referencias consultadas se ha utilizado la herramienta **Mendeley** [46], una aplicación que permite la recolección y organización de referencias y fuentes. Además, dispone de un *plug-in* para LibreOffice para insertar las referencias directamente.

5.2 Herramientas y tecnologías utilizadas

Para este proyecto se han utilizado en combinación diferentes lenguajes y tipos de herramienta para poder llevar a cabo las diferentes funciones de manera adecuada. Procederemos en esta sección a detallar estas herramientas.

5.2.1 Flask

Flask es un micro web framework ligero escrito en Python. Actualmente, webs que utilizan Flask para provenir sus servicios son Pinterest o Linkedin.
Esta herramienta es una manera eficaz de establecer servicios web, con la ventaja de contar con todas las características y paquetes de Python.

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
      app.run()
```

El código anterior representa una aplicación sencilla, el *Hello World!* de Flask. Al ejecutarlo se despliega un servidor, por defecto en **localhost:5000**, con rutas url asignadas en el código utilizando el decorador **@app.route()**. La función decorada se invocará cuando se haga una petición al servidor a la url correspondiente. El retorno puede ser sencillamente una cadena, como en el ejemplo básico, o se puede generar páginas completas dinámicas utilizando *templates* con la sintaxis de Jinja2. Para la funcionalidad de servidor web que buscamos en este proyecto, manejaremos en el retorno cadenas estándar o con formato JSON, generalmente.

Además, se dispone de una interfaz sencilla para las peticiones GET/POST, manejando los parámetros de la petición de forma cómoda.

Finalmente, mencionar que Flask dispone de muchos paquetes y extensiones para mejorar o provenir con nueva funcionalidad que se pueden instalar fácilmente para mayor personalización. En el proyecto se usarán paquetes adicionales para tareas de login y manejo de sesiones.

5.2.2 Web Scraping

Para la parte de cambio automático de contraseñas fue preciso una labor de investigación y pruebas extensiva de diferentes técnicas de realizar este asunto. Se utilizaron diferentes herramientas en Python para este propósito. Las más genéricas son urllib y urllib2, de las que las demás parten para realizar peticiones. Herramientas más específicas son **Mechanize** (Python 2.7) o **RoboBrowser** (Python 3.4+). Como el proyecto ha sido realizado en Python 3.4, se utilizó RoboBrowser como herramienta para cambiar contraseñas, con éxito moderado.

La herramienta permite obtener formularios y cambiar el valor de algunos campos, para posteriormente hacer un *submit*. Esto, teniendo en cuenta el comportamiento tremendamente particular, la no consistencia de un sitio a otro, causa errores al enviar las peticiones, denegándolas en muchos casos. Varios de los formularios de páginas más complejas, tienen cantidad de campos ocultos, como tokens de sesión u otros valores desconocidos.

Teniendo en cuenta estos comportamientos, se dificulta la tarea enormemente por causa de esa falta de estandarización, lo que relega el *scraping* de este tipo a las páginas con formularios más sencillos, en el caso de estas pruebas, las páginas de periódicos varios, que no resultaban problemáticas en este sentido.

La solución más consistente a este problema, ha sido el empleo de una herramienta de más alto nivel, el framework de *testing* Selenium, con su API para Python.

Este enfoque resulta más exitoso, porque en lugar de cambiar valores de campos internamente en el html, esta herramienta interactúa de una forma similar a humano con la página, pulsando botones o introduciendo valores en campos.

Selenium ha sido la herramienta seleccionada para codificar gran parte esta sección del trabajo.Para la fase de *testeo*, hace uso del navegador Firefox directamente al ejecutar los cambios de contraseña. Sin embargo, si se busca una manera de realizarlo desacoplado de un navegador, de forma que no precisa abrir una ventana nueva, se puede utilizar como *driver* el *'Headless Browser'* **PhantomJS**, diseñado para automatizar interacción con la web, ya ofrece mejores prestaciones al no tener que mostrar una interfaz gráfica.

5.2.3 Base de datos

Para el servidor se hace uso de una base de datos SQLite. Diseñaremos una pequeña interfaz con una serie de funciones que enmascaren el trabajo de la base de datos, que se puedan invocar desde los métodos web para realizar las operaciones correspondientes. Cuando se realiza una petición, se inicia una conexión desde el servidor con la base de datos, en el *backend*.

A partir de ahí utilizaremos la API de SQLite3 para Python, para ejecutar las querys. Las ejecuciones tienen el siguiente formato:

```
g.db.execute("insert into tabla(col1, col2)
              values (?, ?)", [val1, val2])
```

Como comentado anteriormente, este es el equivalente de los *prepared statements* para Python, o *querys* parametrizadas, que utilizaremos para evitar ataques de inyección SQL [47].

5.2.4 Criptografía

Para las cuestiones de cifrado y hash, emplearemos como base los módulos de Python pycrypto y hashlib.

De pycrypto haremos uso de su implementación de AES, óptimizada en C [48], sobre la que crearemos la clase AESCipher para realizar el manejo de sales, padding, etc. Además, utilizaremos su implementación de random para la generación de contraseñas, una alternativa criptográficamente segura del módulo estándar.

De hashlib tomaremos el algoritmo SHA-256 para realizar el hashing de contraseñas.

Por otro lado, para establecer la conexión ssl entre cliente-servidor, se ha utilizado OpenSSL para la generación de los certificados correspondientes, a través de las utilidades disponibles mediante línea de comandos. Estos se tratan de certificados autofirmados. En caso de querer disponer de un servidor adecuadamente confiable, se precisa de certificados auténticos, firmados por una entidad certificadora como Cacert.org, autoridad de certificación gratuita.

5.2.5 Android

Para la parte de cliente se ha utilizado la biblioteca y recursos del SDK de Android, versión 23, teniendo en cuenta además los conceptos de diseño de Material Design para el desarrollo de la interfaz, y en general.

En cuanto a la conectividad, se ha utilizado directamente **HttpsURLConnection** como base de las comunicaciones, realizando peticiones POST al servidor a través de esa interfaz.

Para habilitar un módulo de notificaciones, se hará uso de la herramienta FCM [49], se trata de un sistema de notificaciones para Android que recientemente ha reemplazado a la versión anterior, GCM, Se utilizará el teléfono como receptor, a partir de su identificador único. El envío de las notificaciones push se realizará a través del servidor, en un hilo secundario que comprueba la caducidad de las contraseñas, y envía notificaciones al respecto, para recordar al usuario renovar sus contraseñas.

5.3 Proceso de desarrollo

El proyecto comenzó con una fase de investigación y pruebas sobre la manera más razonable de realizar cambios automáticos de contraseña en los servicios del usuario. El objetivo de esto es agilizar el cambio de contraseñas, para una serie de páginas web habituales. Para esta versión, se implementan interfaces para el cambio de contraseña de sitios como Gmail, Dropbox, Twitter, Facebook, ElPaís y 20minutos. Estos sitios proporcionan diferentes procedimientos de registro y actualización de contraseñas, lo que muestra la falta de uniformidad en este sentido. Uniformidad que simplificaría el trabajo para aplicaciones como esta u otras con el fin de llevar a cabo una mejor gestión de contraseñas [50].

Esta tarea es altamente particular para cada sitio, como se ha podido comprobar con esta investigación.

En primer lugar se utilizó *scraping* de páginas web en Python, con los paquetes **RoboBrowser** y **BeautifulSoup**, principalmente. El procedimiento es navegar la página correspondiente, obtener los formularios mediante el comando get_forms y identificar el formulario correcto de login, pudiendo existir más de uno. Una vez obtenido el formulario correcto, se examinan sus campos manualmente, obteniendo el identificador de login. Este identificador, así como el índice del formulario en la lista, es único de la página. En este punto, se rellenan los campos y se envía el formulario.

Con suerte, el servidor solo requiere los campos que se han cambiado relacionados a la información de la cuenta y acepta la petición.

No es trivial la identificación correcta de los campos con los que modificar y el uso de las páginas de JavaScript para insertar y/o gestionar los campos de login puede resultar en interacciones incorrectas con el software de *scraping*. Por ejemplo, el login en la versión web de Gmail se realiza en 2 pasos diferentes, separando usuario y contraseña en dos páginas consecutivas. Además, el formulario final de cambio de contraseña no era reconocido por el gestor de formularios de RoboBrowser, el paquete utilizado en un primer lugar.

Los problemas con JavaScript no acaban ahí, en ocasiones, se emplea este lenguaje para insertar valores ocultos en campos del formulario previo al envío. Soluciones basadas en *scraping* del html tendrían que ser compatibles con este comportamiento, sin conocer necesariamente cuál es el propósito de estos campos ocultos o ni siquiera su existencia.

Tras las primeras pruebas sobre los sitios elegidos, únicamente las páginas más sencillas, las páginas de periódicos, pueden realizar todos los pasos hasta llegar al cambio de contraseña, y en efecto, ejecutarlo correctamente. El resto de páginas lanzan problemas varios. Dropbox al realizar el login inicial contesta con un error 403: *"You tried to do something we can't verify"*, denegando la conexión. Twitter realiza el login correctamente, pero tras navegar a la página de cambio de contraseña y enviando el formulario correspondiente devuelve otro error 403: *"The server understood the request but is refusing to fulfill it"*.

Todo esto complica o imposibilita a este enfoque funcionar consistentemente. Considerando estos primeros resultados, se cambia la herramienta de trabajo al framework Selenium, principalmente un software enfocado al *testeo* y *debug* de páginas web, en este caso utilizaremos esta funcionalidad para manipular las páginas web correspondientes, realizar el login y cambio de contraseñas.

Esta estrategia es más exitosa, solventa los problemas resultados por la intervención de JavaScript, con los campos ocultos de los formularios, aunque no facilita la generalización de estos métodos particularmente, tan solo traslada el trabajo de obtener el índice del formulario a utilizar selectores de elementos, en este caso se ha utilizado selectores de XPath para localizar y introducir información a los campos. XPath es un lenguaje que permite construir expresiones para seleccionar elementos en un documento XML. Tendremos entonces que construir selectores XPath para localizar los campos del formulario. Esta tarea se realizó manualmente, utilizando la función *copy XPath selector* del inspector de Chrome, para agilizar el proceso.

Finalmente, tras este trabajo, se construyen las funciones de cambio de contraseña en **changepw.py.** Para cada una se especifican las urls de login y cambio de contraseña, así como los selectores para cada uno de los campos requeridos. Por la particularidad de cada página y diferencia en el proceso de login de cada una, se realiza una función por sitio web.

Estas funciones reciben como parámetro un objeto *data*, diccionario con llaves login_name, login_password y new_password con sus valores correspondientes asociados. Estas funciones serán utilizadas desde el servidor web cuando sea preciso para realizar los cambios de contraseña.

Por otro lado, en paralelo, se prepara un módulo que gestione la generación automática según los principios considerados anteriormente. El módulo **cypher.py** hace uso del paquete pycrypto para hacer una selección aleatoria de caracteres entre un set determinado por parámetros. Se puede seleccionar la longitud y la presencia de mayúsculas, minúsculas, números y/o símbolos.

Las solicitudes del servidor de nuevas contraseñas se remitirán a esta función. Por defecto se tomarán contraseñas aleatorias de 16 caracteres con mayúsculas, minúsculas, números y símbolos permitidos.

En este módulo también está presente la clase AESCipher. Es esencialmente una interfaz para el AES existente en pycrypto adaptado a los propósitos de esta aplicación. Permite encriptación y desencriptación de cualquier tamaño de cadena, a través de unas funciones lambda para realizar el *pad* y el *unpad*. Además, combina automáticamente la llave de cifrado con una sal aleatoria, que se puede obtener para su almacenamiento con la función de la clase getSalt(). La combinación se realiza a través del algoritmo PBKDF2 y se obtienen 16 bytes utilizados como llave del AES. El modo de operación del AES es *Electronic Code Book* [16]. Debido a que solo se cifrará un bloque único, este algoritmo es lo suficientemente eficaz para estos propósitos. Además, la funcionalidad modos de operación basados en vectores de inicialización ya está presente en el uso de una sal con la clave.

Flask

Con estos elementos listos, procedemos a la implementación del servidor. Como ya comentado, utilizaremos el framework Flask para este propósito.

El servidor se estructura grosso modo en dos partes: métodos de acceso a base de datos, y métodos con el decorador **app.route()** que permite el acceso desde el exterior.

Las funciones de base de datos proporcionan una interfaz simplificada a las querys que se quieren realizar para la gestión de usuarios y contraseñas. Aquí se encuentran las querys de SQLite parametrizadas, que se completan en los métodos individuales, con los parámetros correspondientes. Las consultas tienen el siguiente formato en el código:

```
INSERT = "insert into passwords(ownerid, website, wsusername, pw,
          salt) values (?, ?, ?, ?, ?)"
```

Por otro lado, están los métodos invocados directamente desde las peticiones al servidor. Estos métodos especifican el método de acceso disponible, generalmente, se tratan de peticiones POST. Además, algunos de ellos tienen el decorador @login_required, que deniegan el acceso a estos métodos. Este decorador forma parte del gestor de login utilizado en la aplicación, la extensión **flask_login**. Se pueden encontrar los métodos de acceso al servidor con sus características en el anexo A.

El funcionamiento de este login se basa en una clase usuario definida de forma personalizada para esta aplicación. Siguiendo el patrón indicado por **flask_login**, se implementan los métodos de clase necesarios, **is_authenticated**, **is_active**, **is_anonymous** y **get_id**. Además, se define un método load_user, con el decorador **@login_manager.user_loader**, tal que dado un nombre de usuario, se construye el objeto de la clase User correspondiente, consultando a la base de datos. Este método devolverá el objeto User.

Con estos requisitos de implementación cumplidos y adaptados a nuestros propósitos, se puede utilizar todas las funcionalidad de **flask_login**: *login_user, logout_user, current_user* y el tag *login_required.*

Esto permite alejar los métodos de obtención de información de la base de datos a usuarios no identificados. Al realizar el login se asigna una cookie firmada con una clave del servidor (24 bits aleatorios obtenidos de **/dev/urandom**), para que no sea modificable. Las cookies además, tienen un tiempo de vida de 30 min, especificado en parámetros de configuración del servidor. La gestión interna de estos procedimientos la realiza automáticamente Flask, tan sólo es preciso especificar los parámetros necesarios de configuración, e invocar a login/logout_user...

El siguiente punto sería establecer una conexión SSL con el servidor. Para este fin, utilizamos la aplicación de línea de comandos de OpenSSL para generar los certificados necesarios para autentificar a la contraparte con la que se está comunicando e intercambiar una llave simétrica. Estas operaciones se realizaron a través de la siguiente serie de comandos:

```
openssl genrsa -des3 -out server.key 2048
openssl req -new -key server.key -out server.csr
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
openssl x509 -req -days 365 -in server.csr -signkey server.key
-out server.crt
```

Finalmente, se implementará el sistema de notificaciones desde el servidor. En el método de registro, se guardará el identificador de teléfono del usuario, que será utilizado posteriormente como destinatario de notificaciones push mediante FCM. Las notificaciones se pueden enviar mediante una consola web con este propósito, o utilizando una api web. En nuestro caso, haremos uso de un pequeño módulo [51] de Python de código abierto que enmascara estas peticiones HTTPs, para mayor comodidad.

El modo de funcionamiento será a través de un hilo secundario trabajando en paralelo al funcionamiento principal del servidor. Su tarea será comprobar cada cierto tiempo la expiración de las contraseñas y mandar una notificación al respecto al cliente, para que proceda a su renovación.

Android

Con el servidor en marcha, el siguiente paso sería crear un cliente para realizar las peticiones de una manera cómoda y con interfaz usable. Durante la fase de pruebas, se utilizó la utilidad de terminal cURL, o navegador. La flexibilidad que aporta un servidor web de este tipo es que mientras que se respeten las peticiones a las funciones con los parámetros adecuados y se lleve a cabo una gestión correcta, permite interoperabilidad con otros sistemas. El cliente realizado está basado en Android, pero sería relativamente sencillo lanzar otras versiones para iOS u otras platarformas.

La aplicación comienza en una pantalla de bienvenida, tras la cual se encuentra la actividad de login/registro.

Aquí se comprueba que la contraseña es de una longitud aceptable y se envía la petición de registro o login al servidor. En caso de respuesta positiva, se accede a la actividad principal de la aplicación.

En ella se pueden consultar las contraseñas, la funcionalidad fundamental de la aplicación, al pulsar en un elemento del *RecyclerView* central. Este *RecyclerView* es un elemento flexible para mostrar información en forma de lista. Precisa de un objeto para el contenido a mostrar, en este caso la clase PW, *password*. Además, necesita de un adaptador cuyo objetivo es transcribir una lista del objeto a adaptar, PW, en contenido de una tabla. Aquí también se permite definir el comportamiento al interactuar con un objeto, con una fila de la tabla.

Estructuralmente, esto se trata de una actividad con un *NavigationDrawer*, una herramienta de material design que consiste en un menú desplegable en el lateral de la pantalla. Al deslizar hacia la derecha o pulsar el llamado "botón hamburguesa", en la esquina superior izquierda se despliega este menú (Figura 4).

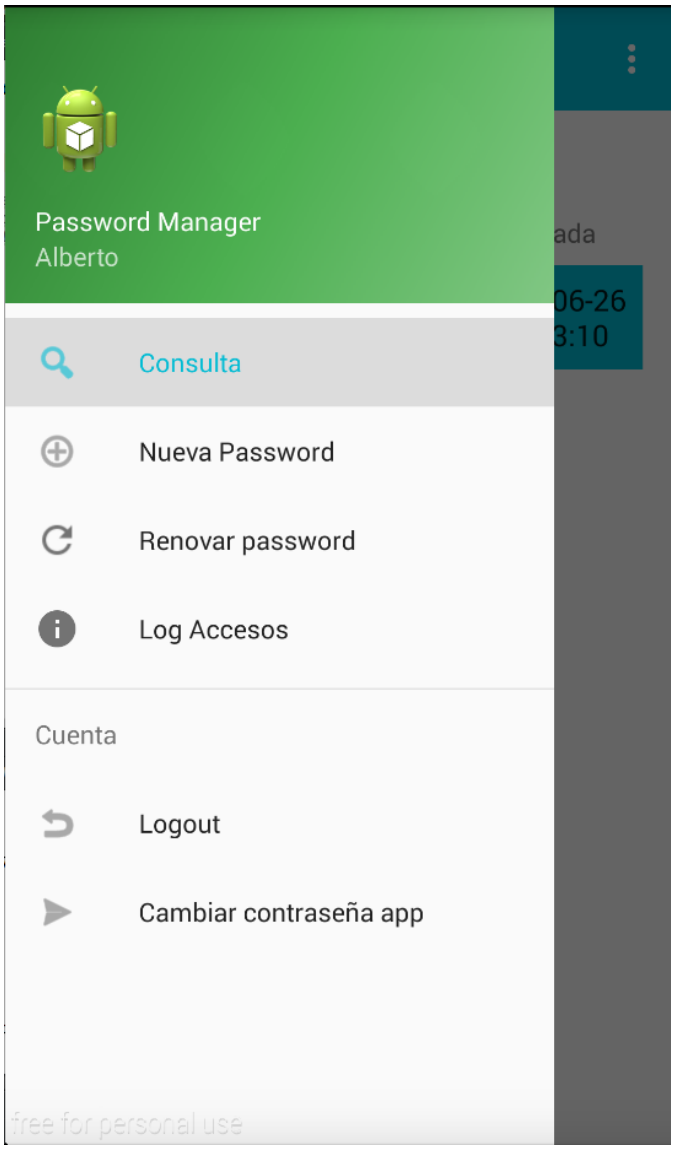

Figura 4: Menú desplegable del cliente

En este menú residen las opciones de navegación de la aplicación, que permiten acceder al resto de la funcionalidad. En primer lugar, está la opción para consultar tus

contraseñas, a modo de tabla. Seguidamente se incluye la opción de nueva *password*, que permite guardar información en el servidor información sobre una cuenta nueva. Este fragmento contiene un formulario en el que se introducen los datos. Al realizar la petición se guarda en el servidor junto a una nueva contraseña aleatoria asignada.

La siguiente opción es renovar *password*. Se observan las cuentas actuales y se puede seleccionar una para renovarla. Al hacer esto, el servidor realiza el cambio de contraseña, en caso de ser una que tenga la opción de cambio automático, se realizará eso mismo, en caso contrario, simplemente se mostrará un menú con información sobre el cambio. (Figuras 5 y 6)

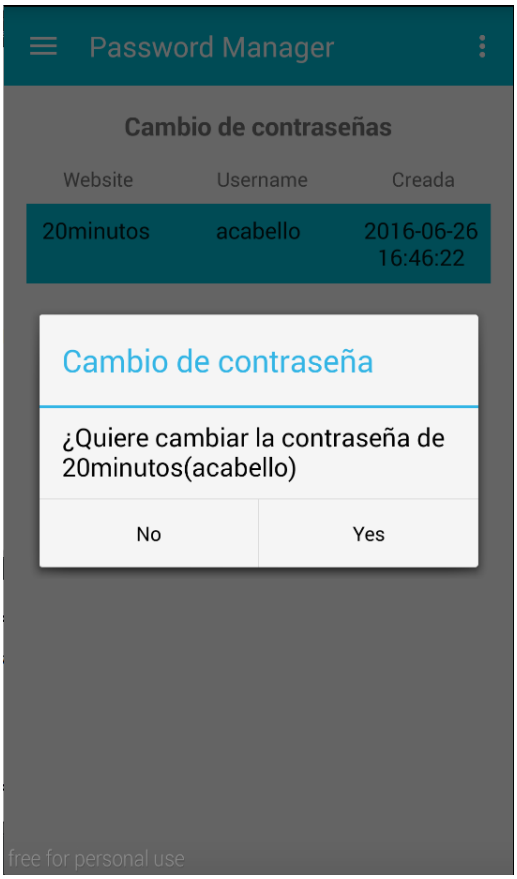

Figura 5: Cambio de contraseña

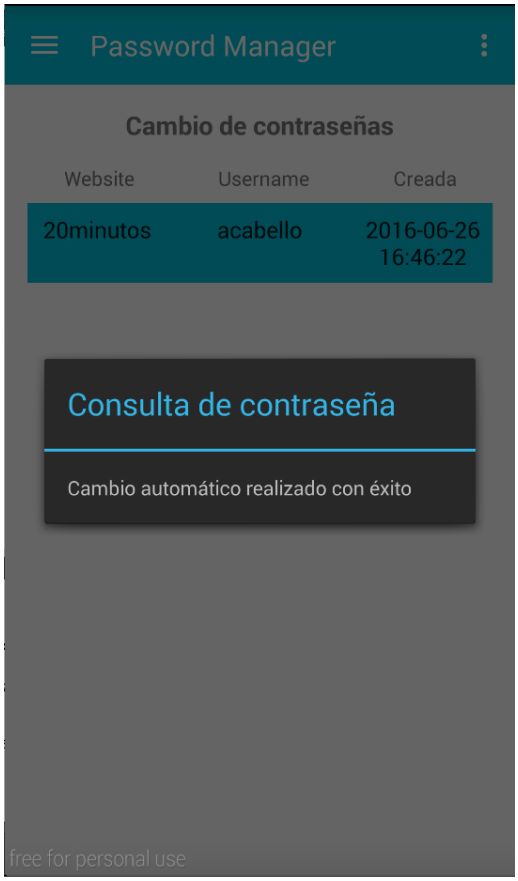

Figura 6: Cambio de contraseña automático realizado

A continuación se puede utilizar la opción 'Log Accesos'. Esta opción permite ver el registro que realiza el servidor de los accesos a la cuenta. De esta manera, se podrá llevar un control del tráfico sobre la cuenta del usuario. Este registro incluye fecha y ip de la localización de acceso (Figura 7)

Seguidamente está el botón del menú de logout, que realiza la operación homónima en el servidor, enviando la petición correspondiente, y devuelve al usuario a la entrada de la aplicación.

Finalmente se encuentra la opción de cambiar la contraseña de la aplicación, lo que cambia la contraseña de acceso al propio cliente, y desde el servidor, se vuelven a cifrar todas las cuentas con la nueva clave.

Por otro lado, se dispone de un servicio de notificaciones como parte de la funcionalidad del servidor web. Su propósito es analizar que contraseñas no se han

actualizado en el tiempo estipulado de vida, según las fechas de creación y expiración que residen en la base de datos. Cuando sobrepasan la fecha de expiración, se envía una notificación *push* al terminal, según el identificador de teléfono asociado en el registro de la cuenta. Este sistema permite recordar al usuario cuando debe cambiar las contraseñas, para incentivar a mantener el paradigma de seguridad adecuado. En la Figura 8 se puede ver unas notificaciones recibidas informando de la expiración de contraseñas. El resto de las figuras de las demás partes de la interfaz de pueden ver en el apéndice B.

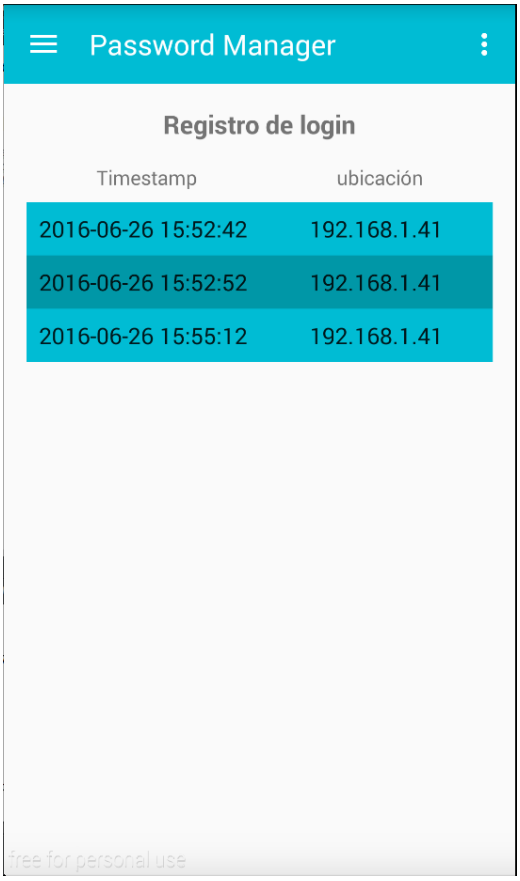

Figura 7: Registro de login

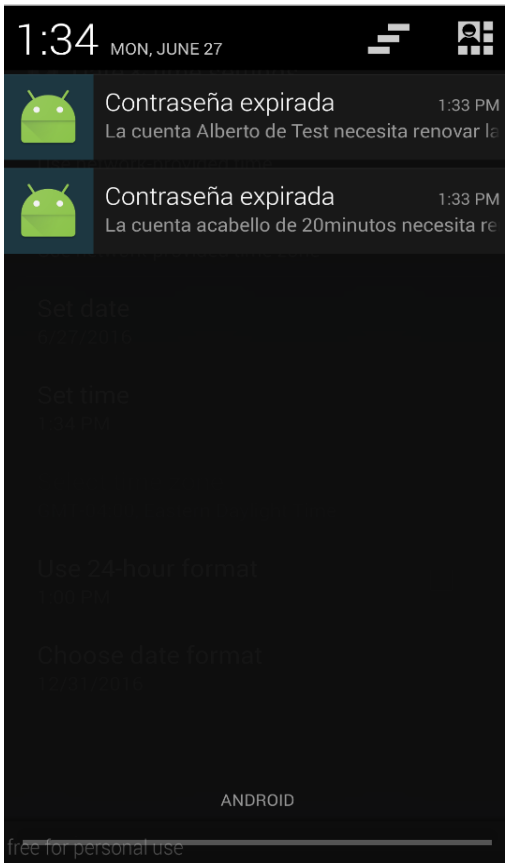

Figura 8: Recepción de notificaciones

En cuanto a la implementación en sí, la parte fundamental del proceso se realiza a través de la clase **ServerMethods**. Esta clase sigue el patrón *singleton*, para tener una sola instancia de la conexión con el servidor. La clase contiene la clase **ServerPostTask**, que extiende **AsyncTask** y realiza una petición genérica POST a una dirección URL determinada. En esta clase existen métodos intermedios como **loginTask** o **postNewPW** que encapsulan el código de llamada al servidor.

Previo a la petición, en el **onPreExecute**, este *task* despliega un **TrustManager** y crea un contexto SSL para aceptar los certificados X.509 del servidor.

El proceso de envío de peticiones al servidor es el siguiente: desde la actividad/fragmento correspondiente se obtiene la instancia de la clase **ServerMethods**. Si es el primer uso, se inicializan varios parámetros internos necesarios, entre ellos, el gestor de cookies. A través de esa instancia se llama a una de los métodos intermedios, con los parámetros adecuados. Desde este método se construyen los parámetros para el POST y se ejecuta el **AsyncTask** para realizar la petición.

Según el método utilizado, el procedimiento consiste en escribir en un **OutputStream** la petición POST, construida por un método auxiliar previo a la petición y leer de un InputStream la respuesta.

Estos métodos utilizan una serie de *callbacks* que se ejecutan al terminar la petición con éxito, o en error para interactuar con la interfaz y completar el contenido correspondiente. Por ejemplo, el método que realiza la petición de obtener las *passwords* contiene un **ServerCallback** que al tener éxito rellena los *RecyclerViews* correspondientes con los datos obtenidos en formato JSON del servidor.

Estos *callbacks* implementan la interfaz **ServerCallback**, que contienen métodos onSuccess y onError.

5.4 Pruebas

Durante la realización del proyecto, y en paralelo a la implementación en sí se realizaron diferentes tipos de pruebas para asegurar el funcionamiento correcto del código. Los tipos de pruebas han sido adaptados al tipo de problema en cuestión, teniendo en cuenta las particularidades de cada parte del proyecto, tanto para el cliente como para el servidor.

Consideraremos dos tipos de pruebas, las pruebas unitarias, cuya finalidad es comprobar que una función o módulo determinados cumplan su propósito de manera independiente. Por otro lado están las pruebas de integración. Su objetivo es verificar el trabajo adecuado y la coordinación entre diferentes módulos del proyecto en su conjunto.

5.4.1 Pruebas unitarias

Para el módulo de criptografía hallado en el servidor web, se dispone de una clase test que mediante la funcionalidad assert de Python se hace una serie de pruebas con diferentes parámetros. En estas pruebas se comprueba el cifrado y descifrado correcto, teniendo en cuenta cuestiones de longitud, *padding*, etc. Se comprueba además, el funcionamiento correcto del mecanismo de las sales, verificando que es necesaria la sal correcta para descifrar la contraseña, así como la clave, y se comprueba que con sales erróneas, el descifrado no es correcto.

Para el módulo de actualización automática de contraseñas, realizado con la interfaz de **Selenium** para Python, se definieron las funciones a utilizar. Por el tipo de contenido, que requiere de acceso a cuentas, no se dejó un plan de pruebas automático. En su lugar, se especificó en el fichero de configuración la variable *silent* que controla si la ejecución se realiza a través del navegador **Firefox** de manera visible o si se realiza a través de **PhantomJS**, como ocurriría en una versión final, desacoplado de un navegador con interfaz habitual. Al asignar esta configuración como False, se permite ver el proceso. De esta manera al ejecutar estas funciones a modo de prueba con datos de cuentas correctas, se puede ver como se insertan los datos automáticamente en los campos de los formularios, y se realiza la navegación por las páginas. Se puede observar de manera completamente visual este proceso, ya que al ejecutar las pruebas mediante Firefox con Selenium, se comporta de manera idéntica a la interacción de un usuario con el navegador.

Estas pruebas pueden resultar particularmente útiles con vista al mantenimiento de la aplicación. Ya que el proceso de cambio de contraseña no es estándar, el código está adaptado para cada página. Con este set de tests se puede comprobar si hay cambios en la página tal que el mecanismo deja de funcionar. Este fue el caso de Dropbox, que a mitad del desarrollo de este trabajo, introdujo un *Captcha* como parte de su proceso de identificación. Esto también serviría para detectar cambios en la estructura de la página, a nivel de HTML, en caso de que cambien los identificadores con los que se localizan los campos del formulario.

Las pruebas para la parte de Android se han realizado a través de tanto el emulador de dispositivos disponible desde Android Studio o el emulador **Genymotion** así como desde un terminal físico. Estas pruebas son principalmente manuales, en las que se realizan pruebas de la funcionalidad adecuada de la parte a comprobar en cada momento. Gran parte de la tarea del cliente móvil consiste en presentar una interfaz adecuada como puerta a la funcionalidad del servidor, así que pruebas unitarias pierden el sentido en este aspecto, teniendo más importancia y complejidad la interacción con el servidor.

5.4.2 Pruebas de integración

El objetivo principal de estas pruebas se encuentra en la comprobación del funcionamiento correcto del servidor al ser accedido mediante las **URLs** correspondientes. Para este punto se puede hacer uso de un navegador, o aún mejor, de cURL, una herramienta para realizar las peticiones HTTP y HTTPS que se necesiten con los parámetros GET/POST correspondientes. Este programa muestra las cabeceras y otros elementos que intervienen en este proceso, lo cual es muy útil. La parte referente a como se muestra la información en el terminal sí que se debe probar específicamente desde uno, para comprobar que en todo momento la interfaz responde correctamente a la información obtenida, en la forma de tablas, en este caso implementadas como RecyclerViews. Para este propósito se realizaron múltiples pruebas con diferentes cuentas y con diferente cantidad de contraseñas almacenadas.

Aquí es donde se comprueba la coordinación entre cliente y servidor, utilizando de forma manual los elementos del cliente, que resultarán en peticiones hacia el servidor, con su respuesta correspondiente. Se prueba de esta manera la creación de cuentas y proceso de login, mantenimiento de la sesión en el servidor, almacenamiento de información en este último… Otro punto a destacar es el funcionamiento de las notificaciones. Comprobaremos con diferentes cuentas los momentos en los que se envía la notificación mediante FCM, para avisar de cuentas expiradas, cada cierto tiempo.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

A través de la realización de este trabajo de fin de grado se han puesto en uso los conocimientos adquiridos a lo largo de los estudios en Ingeniería Informática. Debido a un interés personal en los campos de la seguridad o criptografía se seleccionó este trabajo, donde se pueden aplicar a un sistema práctico. Este campo de la seguridad es uno de gran importancia actualmente, así que esto supone una oportunidad para desarrollar los conocimientos sobre este aspecto.

Este proyecto ha sido un marco propicio para seguir el principio *security by design,* teniendo en cuenta desde las primeras etapas las consideraciones de seguridad necesarias para construir un modelo adecuadamente seguros. La capa de elementos de seguridad no ha sido un elemento adicional sobre una aplicación terminada, lo cual es a veces el caso, si no que todo el diseño está realizado con cuestiones de seguridad en mente. El diseño de la aplicación, entonces, sirve como un medio de aplicación de conocimientos y de investigación.

Además se trabajaron con aspectos no desarrollados en gran medida durante el grado, como es la programación en Python, con muchas opciones, desde el uso como lenguaje de *scripting* a aplicaciones en el lado de servidor a través de frameworks como Django o Flask. También Python es de uso extendido en cuestiones de *data science*, para el análisis de datos o aplicar técnicas estadísticas. Además, Python presenta una sintaxis asequible y un amplio soporte de módulos de terceras partes, de forma que se constituye como una potente herramienta para tareas de pruebas de seguridad [52], análisis forense, etc.

Por otro lado, el trabajo en el el desarrollo de aplicaciones de Android es un aspecto muy útil. Android es el sistema operativo móvil con la cuota de mercado más alta, con perspectiva creciente. Además se está haciendo un enfoque cada vez más fuerte al desarrollo de aplicaciones móviles en sí, debido a la alta disponibilidad y comodidad de las mismas. Por estos motivos, el trabajo en el desarrollo de aplicaciones móviles es otro aspecto interesante a practicar, habiendo solo una asignatura con este fin a lo largo del Grado.

El web *scraping* es otro punto interesante con el que no tenía experiencia ni había trabajado previamente, pero muy requerida en puntos de seguridad informática o sistemas de recomendación, permitiendo un análisis de los datos existentes en páginas web, y actuar en función a estos. Estos conocimientos se pueden aplicar en un marco de Big Data, muy popular actualmente, para desplegar herramientas de preservación de la privacidad y verificación de tal protección. Con este aspecto se relaciona el concepto OSINT [53], Open-source intelligence, consiste en la recolección de información de fuentes disponibles de forma pública, con objetivos de inteligencia militar [54], seguridad nacional, cuestiones legales, etc. Se podría de esta manera establecer mecanismos de detección y prevención de fugas de información sensible [55].

6.2 Trabajo futuro

En primer lugar, referente a los cambios automáticos de contraseña, un factor que facilitaría realmente el trabajo de login y actualización de contraseñas sería el de una estandarización, con anotaciones semánticas para identificar los campos que es preciso

completar. Los gestores de contraseñas en ocasiones son innecesariamente complejos en este aspecto, y aún así la compatibilidad no es perfecta. El trabajo de Frank Stajano [50], motivación e inspiración importante para este trabajo, propone un esquema de anotaciones semánticas para formularios HTML que permitan identificación no ambigua de los campos necesarios, para que programas actuando en nombre del usuario sean capaces de realizar su función adecuadamente.

Se trata de añadir pequeña cantidad de identificadores que marcan los campos HTML con clases con el prefijo pmf-, *password manager friendly,* con el fin de simplificar las heurísticas empleadas por gestores que pretenden funcionar de manera lo más genérica posible, y permitir la fácil adaptación a diferentes tipos de páginas.

```
<form action="/change" method="POST" class="pmf-change-password">
  <input type="password" name="current" class="pmf-password" />
  <input type="password" name="new" class="pmf-new-password" />
  <input type="password" name="confirm" class="pmf-new-password" />
</form>
```

Figura 9: Notación PMF propuesta por Stajano [50]

En cuanto a la aplicación propiamente, un punto interesante que se puede desarrollar es ampliar la funcionalidad del log de accesos existente. Aquí se pueden desplegar bastantes herramientas de detección y prevención de accesos indebidos. Por ejemplo, se puede llevar un seguimiento de direcciones ip habituales y pedir una doble verificación vía mail o notificación cuando se intenta acceder desde una red distinta.

También se puede ampliar la parte de las notificaciones, utilizándolas como procedimiento de autenticación con varios pasos. Al asociar una cuenta con un teléfono, se puede utilizar en el caso de Android FCM (u otras variantes) para impedir accesos desde otros puntos, ya que únicamente se enviará la notificación con un token de acceso, por ejemplo, al terminal correspondiente.

Otro camino por el que se podría potenciar la aplicación sería el de ofrecer servicios de incremento de la privacidad. Para evitar la trazabilidad no deseada, se podrían generar identidades de modo automático y aleatorio para acceder a ciertos servicios web, evitando rastros innecesarios. Para ello se podría hacer uso de servicios de mail anónimos [56] para generar perfiles de usuario de un solo uso. El objetivo sería poder obtener rápidamente cuentas para acceder a servicios de modo anónimo.

Una opción muy interesante como ampliación sería el uso de biométricas para la identificación y el login en la aplicación. De esta manera, todos los datos almacenados quedarían detrás de una huella dactilar, reconocimiento facial, retina, etc.

Este sistema ofrecería ventajas e inconvenientes particulares frente al uso de contraseñas corrientes.

En primer lugar, no todos los dispositivos móviles vienen equipados con el hardware necesario para realizar este tipo de identificación. Además, no es posible cambiar la "contraseña biométrica", no es muy frecuente cambiar de huellas dactilares, por ejemplo, así como otros inconvenientes [57].

Por otro lado, este concepto de identificación es bastante conveniente, no hay que recordar ningún dato adicional ni que escribir datos, simplemente es necesario utilizar la huella dactilar, o el método que se considere.

En definitiva, sería un método interesante que tener como factor de autenticación posible, no necesariamente el sustituto definitivo, pero una alternativa adicional.

En relación con este aspecto, en más de un artículo [58] se pretende augurar la caída de los sistemas de contraseñas, en favor de otras alternativas. Si bien, en efecto, estos sistemas suelen incrementar la seguridad (esperado, ya que suelen provenir de miembros de la comunidad de seguridad informática), algunos a costa de la usabiliad, todos actúan peor que las contraseñas en cuanto a implementación práctica de un sistema [59]. Ganancias que no suelen ser suficientes para sobreponer a los costes de transición a nuevos mecanismos. Por estos motivos, parece que la mejor opción es, dentro del marco de la autenticación vía contraseñas, incrementar las opciones de seguridad, de la mano de varios pasos de identificación, y, como en el caso de este trabajo, mediante gestores de contraseñas.

Referencias

[1] "Password fatigue," *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Password_fatigue. [Accessed: 12-Jun-2016].

[2] I. Paul, "Update: LinkedIn Confirms Account Passwords Hacked," *PcWorld*, 2016. [Online]. Available: http://www.pcworld.com/article/257045/security/6-5m-linkedin-passwords-posted-online-after-apparent-hack.html. [Accessed: 28-Jun-2016].

[3] S. Venken, "LinkedIn leaked hashes password statistics," 2012. [Online]. Available: http://pastebin.com/5pjjgbMt. [Accessed: 28-Jun-2016].

[4] R. Seacord, "Top 10 Secure Coding Practices," 2011. [Online]. Available: https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices. [Accessed: 27-Jun-2016].

[5] H. Sanchez and J. Murray, "Putting Your Passwords on Self-destruct Mode : Beating password fatigue," 2016. [Online]. Available: https://www.usenix.org/system/files/conference/soups2016/wsf16_paper_sanchez.pdf. [Accessed: 26-Jun-2016].

[6] "Fail2Ban Main Site," *MediaWiki*. [Online]. Available: http://www.fail2ban.org/wiki/index.php/Main_Page. [Accessed: 13-Jun-2016].

[7] J. Ellingwood, "How To Protect SSH with Fail2Ban on Ubuntu 14.04." [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-protect-ssh-with-fail2ban-on-ubuntu-14-04. [Accessed: 13-Jun-2016].

[8] H. Dunlop, "Understanding and generating the hash stored in /etc/shadow," 2012. [Online]. Available: http://www.aychedee.com/2012/03/14/etc_shadow-password-hash-formats/. [Accessed: 12-Jun-2016].

[9] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," *Adv. Cryptology-CRYPTO 2003*, vol. 2729, pp. 617–630, 2003.

[10] NIST, "Insider Attacks," 1994. [Online]. Available: http://csrc.nist.gov/publications/nistir/threats/subsection3_4_1.html. [Accessed: 12-Jun-2016].

[11] M. HOSENBALL, "Swiss spy agency warns U.S., Britain about huge data leak," *Reuters*, 2012. [Online]. Available: http://www.reuters.com/article/us-usa-switzerland-datatheft-idUSBRE8B30ID20121204. [Accessed: 12-Jun-2016].

[12] T. Wu, "The Secure Remote Password Protocol," *Proc. Symp. Netw. Distrib. Syst. Secur. NDSS 98*, vol. 4, pp. 97–111, 1998.

[13] D. Hardt, “The OAuth 2.0 Authorization Framework,” *RFC 6749*. 2012.

[14] M. Burnett, *Perfect password: Selection, protection, authentication.* Syngress Publishing, 2006.

[15] “Diceware,” 2015. [Online]. Available: http://www.dicewarepasswords.com/about/. [Accessed: 26-Jun-2016].

[16] W. Stallings, “Cryptology and Network Security,” in *Network Security*, vol. 4301, no. 5, 2006, pp. 66–74.

[17] D. R. Stinson, *Cryptography: Theory and Practice*, Third edit. Taylor & Francis, 2005.

[18] “Linux Programmer’s Manual - random, urandom - kernel random number source devices,” 2015. [Online]. Available: http://man7.org/linux/man-pages/man4/random.4.html. [Accessed: 28-Jun-2016].

[19] E. B. Barker and J. M. Kelsey, “NIST Special Publication 800-90A:Recommendation for random number generation using deterministic random bit generators (revised),” *Arch. NIST Tech. Ser. Publ.*, vol. 800–90A, no. January, 2012.

[20] B. Schneier, “Other Stream Ciphers and Real Random-Sequence Generators,” in *Applied cryptography: protocols, algorithms, and source code in C*, John Wiley & Sons, 1995.

[21] “eSTREAM: the ECRYPT Stream Cipher Project.” [Online]. Available: https://competitions.cr.yp.to/estream.html. [Accessed: 25-Jun-2016].

[22] S. Babbage, C. De Canni`ere, V. Rijmen, and A. Canteaut, “The eSTREAM Portfolio,” 2008. [Online]. Available: http://www.ecrypt.eu.org/stream/portfolio_revision1.pdf. [Accessed: 15-Jun-2016].

[23] N. J. RUBENKING, “The Best Password Managers for 2016,” *PcMag*, 2016. [Online]. Available: http://www.pcmag.com/article2/0,2817,2407168,00.asp. [Accessed: 15-Jun-2016].

[24] S. Yang, “Researchers recover typed text using audio recording of keystrokes,” *UC Berkeley News*, 2015. [Online]. Available: http://www.berkeley.edu/news/media/releases/2005/09/14_key.shtml. [Accessed: 11-Jun-2016].

[25] D. Silver, S. Jana, E. Chen, C. Jackson, and D. Boneh, “Password Managers: Attacks and Defenses,” *USENIX Secur. Symp. (USENIX Secur.*, pp. 449–464, 2014.

[26] D. M’Raihi, S. Machani, M. Pei, and J. Rydell, “TOTP: Time-Based One-Time Password Algorithm,” *RFC 6238*. 2011.

[27] J. Davis, “Two Factor Auth (2FA) List of websites and whether or not they support 2FA.” [Online]. Available: https://twofactorauth.org/. [Accessed: 12-Oct-2016].

[28] A. Ronacher, “Flask,” 2016. [Online]. Available: http://flask.pocoo.org/. [Accessed: 12-Jun-2016].

[29] J. Daemen and V. Rijmen, “197: Announcing the advanced encryption standard (AES).” 2001.

[30] R. Rivest, A. Shamir, and L. Adleman, “RSA Patent,” US 4405829 A, 1983.

[31] NIST, “Descriptions of SHA-256, SHA-384 and SHA-512,” 2001.

[32] E. Barker, “Recommendation for Key Management,” *NIST*, no. Revison 4, 2016.

[33] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, and R. Housley, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,” *RFC 5280*. 2008.

[34] “OpenSSL, Cryptography and SSL/TLS Toolkit,” *OpenSSL Software Foundation*, 2016. [Online]. Available: https://www.openssl.org/. [Accessed: 12-Jun-2016].

[35] B. Kaliski, “PKCS #5: Password-Based Cryptography Specification Version 2.0,” *RFC 2898*. 2000.

[36] “Password Hashing Competition,” 2013. [Online]. Available: https://password-hashing.net/. [Accessed: 12-Jun-2012].

[37] D. Dinu, D. Khovratovich, J.-P. Aumasson, and S. Neves, “Argon2,” *Github*. [Online]. Available: https://github.com/p-h-c/phc-winner-argon2. [Accessed: 12-Jun-2012].

[38] “NIST Withdraws Outdated Data Encryption Standard,” *NIST*, 2005. [Online]. Available: http://www.nist.gov/itl/fips/060205_des.cfm. [Accessed: 12-Jun-2016].

[39] Oracle, “Oracle Package Naming Convention.” [Online]. Available: https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html. [Accessed: 12-Jun-2016].

[40] “Android Studio.” [Online]. Available: https://developer.android.com/studio/index.html. [Accessed: 12-Jun-2016].

[41] Python Software Foundation, “Python Main Page,” 2016. [Online]. Available: https://www.python.org/. [Accessed: 13-Jun-2016].

[42] “Should I use Python 2 or Python 3 for my development activity?,” *Python wiki*. [Online]. Available: https://wiki.python.org/moin/Python2orPython3. [Accessed: 13-Jun-2016].

[43] E. Schofield, “Writing Python 2-3 compatible code,” 2015. [Online]. Available: http://python-future.org/compatible_idioms.html. [Accessed: 13-Jun-2016].

[44] “Curl Main Page.” [Online]. Available: https://curl.haxx.se/. [Accessed: 13-Jun-2016].

[45] “Bitbucket.” [Online]. Available: https://bitbucket.org/. [Accessed: 25-Jun-2016].

[46] “Mendeley.” [Online]. Available: https://www.mendeley.com/. [Accessed: 10-Jun-2016].

[47] “SQL Injection,” *OWASP*, 2016. [Online]. Available: https://www.owasp.org/index.php/SQL_Injection. [Accessed: 12-Jun-2016].

[48] V. Rijmen, A. Bosselaers, and P. Barreto, “PyCrypto’s AES implementation.” [Online]. Available: https://github.com/dlitz/pycrypto/blob/master/src/AES.c. [Accessed: 13-Jun-2016].

[49] “Set Up a Firebase Cloud Messaging Client App on Android.” [Online]. Available: https://firebase.google.com/docs/cloud-messaging/android/client. [Accessed: 26-Jun-2016].

[50] F. Stajano, M. Spencer, G. Jenkinson, and Q. Stafford-Fraser, “Password-manager friendly (PMF): Semantic annotations to improve the effectiveness of password managers,” in *Technology and Practice of Passwords*, Springer, 2015, pp. 61–73.

[51] E. Adegbite, “Python client for FCM - Firebase Cloud Messaging (Android & iOS),” *2016*. [Online]. Available: https://github.com/olucurious/pyfcm. [Accessed: 26-Jun-2016].

[52] TJ. O’Connor, *Violent Python A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*. 2010.

[53] “Open-source intelligence,” *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Open-source_intelligence. [Accessed: 14-Jun-2016].

[54] J. R. M. Martín, “COMO EXPLOTAR OSINT EFICAZMENTE.” [Online]. Available: http://www.defensa.gob.es/ceseden/Galerias/esfas/destacados/en_portada/COMOx20EXPLOTARx20OSINTx20EFICAZMENTE.pdf. [Accessed: 14-Jun-2016].

[55] “Avasecure: next-generation vulnerability scanner.” [Online]. Available: http://avasecure.com/. [Accessed: 26-Jun-2016].

[56] “Send Anonymous Emails: 20 Sites To Keep Your Identity Hidden.” [Online]. Available: http://www.hongkiat.com/blog/anonymous-email-providers/. [Accessed: 28-Jun-2016].

[57] T. Le Brass, “10 Reasons Why Biometrics Won’t Replace Passwords Anytime Soon,” 2015. [Online]. Available: http://blog.dashlane.com/10-reasons-biometrics-wont-replace-passwords-anytime-soon/. [Accessed: 28-Jun-2016].

[58] C. Caracciolo, “Eleven Paths Talks: ¿Las contraseñas desaparecerán?” [Online]. Available: http://blog.elevenpaths.com/2016/06/eleven-paths-talks-las-contrasenas.html. [Accessed: 26-Jun-2016].

[59] J. Bonneau, C. Herley, F. Stajano, P. C. Van Oorschot, P. C. Van Oorschot, and F. Stajano, “The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes,” *2012 IEEE Symp. Secur. Priv.*, no. 817, pp. 553–567, 2012.

Glosario

AES	Advanced Encryption Standard, originalmente Rijndael, es un algoritmo de cifrado simétrico por bloques de 128 bits, y diferentes tamaños de clave.
API	Application Programming Interface
Android	Sistema operativo para dispositivos móviles, diseñado por Google y basado en el núcleo linux. Está diseñado con el uso de patalla táctil como principal método de interacción.
Ataque de diccionario	Tipo de ataque que consiste en intentar encontrar una contraseña probando las palabras de un diccionario, teniendo en cuenta que muchos usuarios utilizarán una simple palabra como contraseña.
Ataque por fuerza bruta	Tipo de ataque contra texto cifrado que consiste en probar todas las combinaciones posibles de clave o contraseña hasta que se encuentra la correcta.
Ataque Man in the Middle	Tipo de ataque tal que un intruso intercepta, observa y modifica mensajes transmitidos entre dos partes.
Back-end	Capa de acceso a datos que reside en el servidor.
Captcha	Completely Automated Public Turing test to tell Computers and Humans Apart. Test, generalmente con la forma de un texto alterado con el fin de detectar presencia de un humano.
Cookie	Pequeña cantidad de datos que envía una página y se guarda en el navegador del usuario con el fin de mantener información del estado de la visita.
FCM	Firecloud Cloud Messaging, sistema de notificaciones de Google. Reemplaza a GCM, la versión anterior de este sistema.
FIPS	Federal Information Processing Standards. Estándares anunciados públicamente por el gobierno de los Estados Unidos referentes a
Flask	Framework minimalista escrito en Python y basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2.
Framework	Conjunto de estructuras, herramientas y módulos que sirven como base para el desarrollo de software.

Hash	Función de una sola vía que realiza una conversión entre una entrada de tamaño arbitrario y una salida de tamaño fijo. Se pueden utilizar con fines criptográficos, en este caso, para el almacenamiento de contraseñas.
HMAC	Hash Message Authentication Code, función criptográfica que combina el uso de un hash con una llave criptográfica secreta.
Ingeniería social	Práctica de obtener información confidencial, acceso o privilegios en sistemas a través de un usuario legítimo que es manipulado.
JSON	JavaScript Object Notation, formato estándar de transmisión de información mediante pares clave valor.
Keylogger	Dispositivo software o hardware diseñado para registrar las pulsaciones de la víctima en el teclado, tal que el usuario desconoce esta monitorización de sus acciones e introduce información como contraseñas que quedarán registradas.
Login	Identificarse para iniciar sesión.
Logout	Cerrar sesión.
NIST	National Institute of Standards and Technology. Agencia, de Estados Unidos cuyo objetivo es regular los estándares tecnológicos y promover su avance.
Padding	Serie de datos concatenados al final del mensaje con el fin de que la longitud del mensaje coincida con el tamaño requerido para un algoritmo de cifrado por bloques.
PBKDF2	Password-Based Key Derivation Function 2, función diseñada por los laboratorios RSA que genera una serie de bits para ser utilizados como clave criptográfica a través de una password maestra y una sal. Se aplica una función pseudoaleatoria como un hash una gran cantidad de veces (1000 - 100000 iteraciones) para obtener la salida.
Phishing	Tipo de ataque que consiste en impersonar a una entidad confiable para tratar que la víctima introduzca sus creedenciales, generalmente utilizando un correo electrónico para dirigir al objetivo a la página fraudulenta.
Python	Lenguaje de programación de alto nivel. Es interpretado y de propósito general, con una biblioteca estándar amplia, además de gran cantidad de módulos con múltiples própósitos.
RF	Requisito Funcional
RNF	Requisito No Funcional
Rainbow Table	Tabla precomputada de consulta para intentar deshacer funciones hash, normalmente para romper tablas de contraseñas hasheadas.

SDK	Software Development Kit, Conjunto de herramientas de desarrollador que permiten el diseño de software para una plataforma determinada.
Single point of failure	Punto de un sistema, referente a un sistema hardware, software, o seguridad, en este caso, que si falla, puede comprometer todo el resto, o dejarlo inoperable.
SQLite	Sistema relacional de base de datos, que a diferencia de muchos otros, no sigue un modelo cliente-servidor. En su lugar, encapsula todo el contenido de la base de datos en un único fichero.
Sal	Serie de bits aleatorios utilizados como entrada adicional a una función de una sola vía que hasheará unos datos. Su principal objetivo es defenderse contra ataques de diccionario, o frente a hashes precomputados.
Scraping	Técnica utilizada a través de cierto software para obtener información de un sitio web, simulando la navegación que realizaría un usuario.
SHA-256	Secure Hash Algorithm, función hash diseñada por la NSA, National Security Agency, en su versión con salida de 256 bits.
SSH	Secure Shell, protocolo criptográfico así como programa que lo implementa. Permite acceder a máquinas externas a través de la red, facilitando login remoto a las mismas.

Anexos

Anexo A: Interfaz servidor web

Se detalla en este anexo la lista de funciones del web server Flask que funcionan de interfaz con el mismo. Para hacer otra implementación de un cliente en otra plataforma, basta con seguir las indicaciones y respetar los parámetros de entrada y salida de los métodos correspondientes.

URL: /
Método: GET/POST
Descripción: Método de tipo echo para probar conectividad
Parámetros: Sin parámetros
Resultado: "Home Page"

URL: /login/
Método: POST
Descripción: Se proporcionan las credenciales del usuario y comprueba si son correctas. En caso afirmativo, se inicia la sesión para este cliente.
Parámetros: user: nombre de usuario
pw: contraseña
Resultado: "login ok" en caso correcto, "login err", en caso de login erróneo.

URL: /register/
Método: POST
Descripción: Se registra una nueva cuenta de usuario, en caso de que el nombre de usuario esté disponible. Tiene el efecto de login en caso de que el registro sea correcto.
Parámetros: user: nombre de usuario
pw: contraseña
Resultado: "register ok" en caso correcto, "register err", en caso de registro erróneo.

URL: /logout/
Método: POST/GET
Descripción: Se cierra la sesión del usuario, eliminando la contraseña del servidor.
Parámetros: Sin parámetros.
Resultado: Se elimina la sesión actual y devuelve "logout ok".

URL: /changeuserpw/
Método: POST
Descripción: Método para cambiar la contraseña maestra actual del usuario. Esto implica recifrar el resto de sus contraseñas con la llave nueva.
Parámetros: pw: Nueva contraseña a la que se quiere cambiar.
Resultado: “change ok”

URL: /selectalluser/
Método: GET/POST

Descripción: Obtiene la lista de contraseñas descifradas del usuario.
Parámetros: Sin parámetros
Resultado: Objeto JSON con el siguiente formato (como string):

```
{
  rows:[
    {
      website:"",
      wsusername:"",
      pw:"",
      created:"",
      expires:""
    },
    {...}
  ]
}
```

URL: /insertpost/
Método: POST
Descripción: Añade información sobre una nueva cuenta a la base de datos.
Parámetros: website: nombre de la web
user: nombre de usuario de esa web
Resultado: "insert ok"

URL: /updatepost/
Método: POST
Descripción: Actualiza la contraseña del sitio determinado
Parámetros: website: nombre de la web
wsusername: nombre de usuario de esa web
Resultado: 0 si se ha realizado cambio automático con éxito,
-1 si se ha realizado cambio automático fallido,
La password actualizada en otro caso.

URL: /getlog/
Método: GET
Descripción: Obtiene el registro de accesos a la cuenta
Parámetros: Sin parámetros
Resultado: Objeto JSON con el siguiente formato (como string):

```
{
  rows:[
    {
      userid:"",
      tstamp:"",
      extrainfo:""
    },
    {...}
  ]
}
```

Anexo B: Interfaz cliente móvil

En este punto se completa la muestra del contenido de la aplicación web con imágenes del resto de la interfaz no mostradas en la parte correspondiente de la memoria, por no ofuscar el contenido.

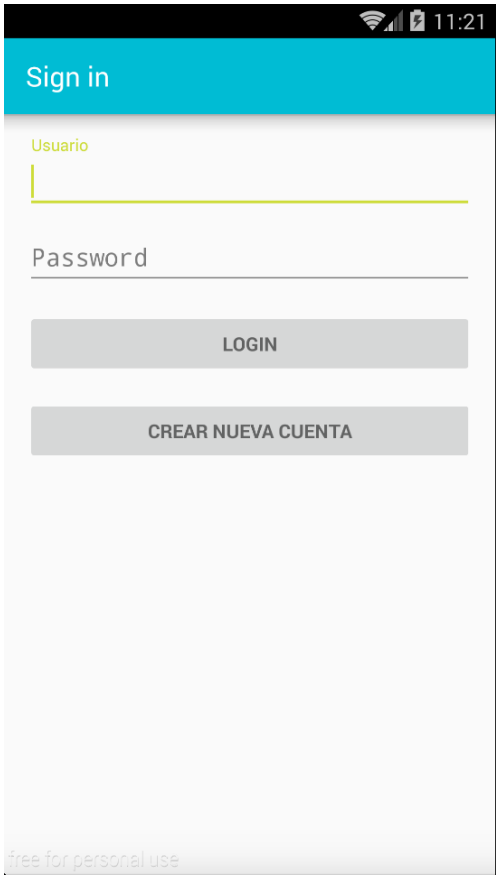

Figura 10: Pantalla de login y registro

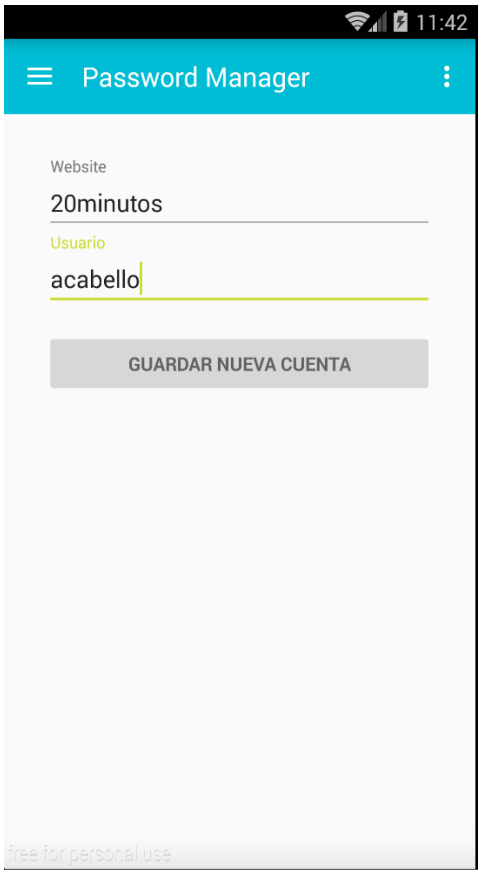

Figura 11: Guardado de nuevas cuentas

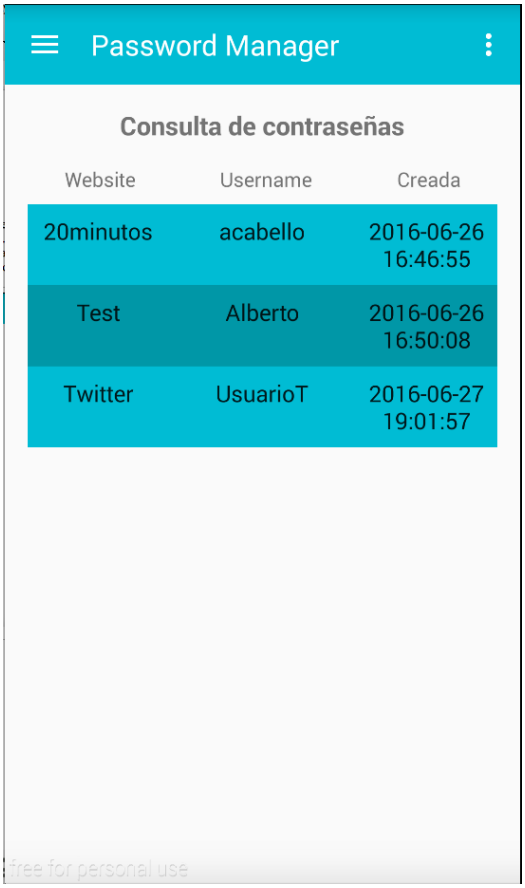

Figura 12: Consulta de contraseñas

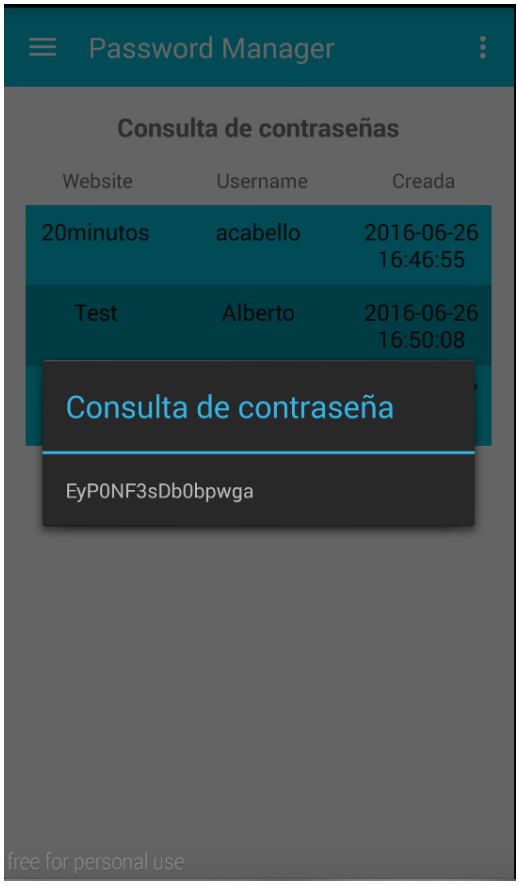

Figura 13: Consulta de contraseña tras pulsar la tabla

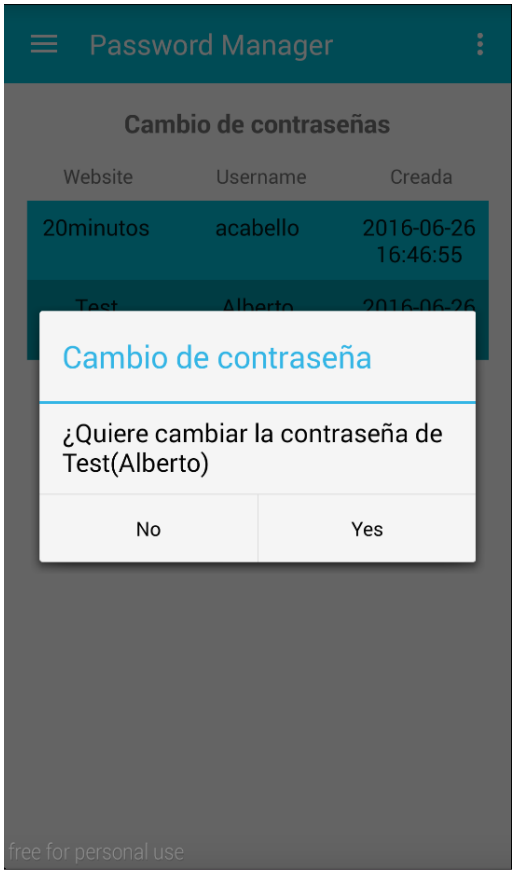

Figura 14: Cambio de contraseña II

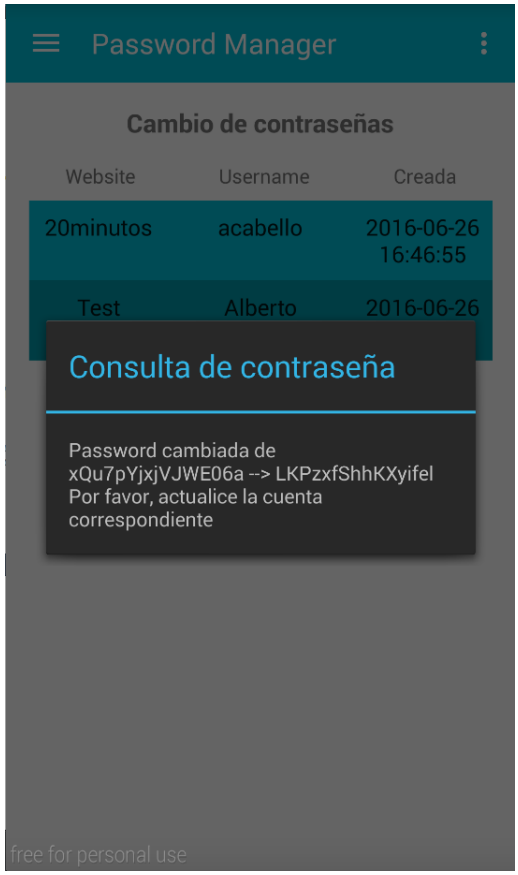

Figura 15: Cambio de contraseña no automático

Anexo C: Cronograma del proyecto

En este anexo consideraremos la planificación del proyecto, con fechas orientativas acerca del inicio de la realización de las tareas.

1. Inicio del proyecto. Estudio del estado del arte, lectura de documentación e investigación sobre herramientas de web scrapping. *1 Febrero 2016.*

2. Análisis y diseño. Incluye la definición del proyecto, marca de objetivos, análisis de requisitos y diseño. *8 Febrero 2016.*

3. Desarrollo del proyecto, codificación y pruebas.

    - Investigación sobre herramientas y opciones posibles para *web scrapping* en Python. *15 Febrero 2016.*

    - Desarrollo del módulo de actualización de contraseñas. *7 Marzo 2016.*

    - Familiarización con el framework Flask. *19 Marzo 2016.*

    - Desarrollo de la interfaz del servidor web. 21 *Marzo 2016.*

    - Codificación completa del servidor. *25 Marzo 2016.*

    - Pruebas unitarias del servidor. *2 Abril 2016.*

    - Desarrollo de la interfaz del cliente Android. *4 Abril 2016.*

    - Codificación completa del cliente Android. 8 *Abril 2016.*

    - Pruebas unitarias del cliente. 1 *Mayo 2016.*

    - Pruebas de integración cliente-servidor. 3 *Mayo 2016.*

4. Elaboración de la memoria.

    - Redacción de la memoria. *13 Mayo 2016.*

    - Revisiones de la memoria y cambios. *28 Mayo 2016.*

5. Finalización del proyecto. *28 Junio 2016*