

Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

PassGuard: Implementación de un gestor de contraseñas seguro.

Autor: Martín Iglesias Nalerio

Tutor: Javier Albert Seguí

2023

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo Fin de Grado

PassGuard: Implementación de un gestor de contraseñas seguro.

Autor: Martín Iglesias Nalerio

Tutor: Javier Albert Segui

Tribunal:

Presidente: Ana Castillo Martínez

Vocal 1º: José Amelio Medina Merodio

Vocal 2º: Javier Albert Segui

Fecha de depósito: 15 de Septiembre de 2023

Agradecimientos

Este proyecto de fin de carrera es la culminación de bastantes horas de trabajo dedicadas en solitario a la investigación sobre criptografía y la posterior implementación de esa criptografía en una aplicación software funcional. No obstante, ha habido más participación además de la mía en el proyecto.

En primer lugar, agradecer a mi familia más cercana, mis padres y mi hermana, principalmente por su apoyo incondicional y por saber comprenderme y ayudarme después de largas jornadas dedicadas a este proyecto, además de ayudarme con ciertas pruebas de la aplicación y aconsejarme sobre la misma desde una perspectiva alejada de la informática.

Agradecer a mis compañeros más cercanos de la carrera y, hoy por hoy, mis amigos, por animarme a realizar un proyecto personal de programación como este, el cual se ha convertido en mi Trabajo Fin de Grado. Mención especial a mis compañeros Óscar Pozo, Mohamed Hamen, Carlos Vicente, Víctor Lapeña y Alberto Estévez, por proveerme de puntos de vista de cara al diseño del proyecto y a resolver problemas que me encontré en la implementación del mismo.

Destaco también el trabajo de mi tutor para este trabajo, Javier Albert Seguí, él ha sido profesor mío de otras asignaturas durante los cuatro años de la carrera, y puedo decir que estoy muy contento con el trabajo que hace, de ahí que le presentara mi idea para este trabajo. Me ha dado buenas sugerencias para la aplicación, las cuales yo no me había planteado previamente, y se ha asegurado de que yo desarrolle una memoria del TFG con buena forma y contenido, además de que siempre ha estado disponible para resolver mis dudas y preocupaciones de manera rápida y eficaz, aspecto que agradezco y valoro mucho.

Por último, empecé este proyecto sin ni siquiera saber programar en el lenguaje de programación que elegí, ni tampoco sobre criptografía, he tenido que investigar e informarme sobre todos los conceptos y temas que se tratarán en este documento. Debo agradecer a muchos contribuyentes anónimos que, sin ánimo de lucro, dejaron en Internet sus soluciones a las dudas que gente como yo publicaron. Sin estas personas, este proyecto probablemente habría tardado mucho más tiempo en salir adelante, o directamente no habría sido posible llevarlo a cabo.

A todos los mencionados, nuevamente, ¡muchas gracias!

Resumen

En este documento se guiará al lector a la investigación realizada sobre diferentes algoritmos criptográficos según sus características, además de la posterior puesta en práctica de algunos de estos algoritmos en forma de una aplicación de escritorio de un gestor de contraseñas.

Se ha indagado acerca de múltiples funciones y algoritmos criptográficos, la mayoría de ellos utilizados hoy en día (Sept. 2023), además de su funcionamiento y sus ventajas y desventajas, y se han escogido algunos de estos para implementar una aplicación de escritorio funcional y de aspecto moderno.

Palabras clave: Gestor de contraseñas, Criptografía, Algoritmo, Cifrado, Descifrado.

Abstract

This document will guide the reader to the research done on different cryptographic algorithms according to their characteristics, in addition to the subsequent implementation of some of these algorithms in the form of a desktop application of a password manager.

Multiple cryptographic functions and algorithms, most of them used nowadays (Sept. 2023), in addition to their operation and their advantages and disadvantages, have been researched and some of these have been chosen to implement a functional and modern looking password manager desktop application.

Keywords: Password Manager, Cryptography, Algorithm, Encryption, Decryption.

Índice general

Agradecimientos	v
Resumen	vii
Abstract	ix
Índice general	xi
Índice de figuras	xiii
1 Introducción	1
1.1 Introducción	1
1.2 Estado del Arte	3
2 Estudio teórico	9
2.1 Introducción	9
2.2 Tipos de cifrado en criptografía según su clave	10
2.2.1 Cifrado Simétrico	10
2.2.2 Cifrado Asimétrico	12
2.2.3 Cifrado Híbrido	14
2.3 Tipos de cifrado en criptografía según los algoritmos	16
2.3.1 Cifrado por flujo	16
2.3.2 Cifrado por bloques	18
2.4 Otros algoritmos y funciones criptográficas	20
2.4.1 Funciones Criptográficas Hash	20
2.4.2 Funciones de derivación de claves KDF	23
2.4.3 Generadores pseudoaleatorios criptográficamente seguros	25
2.5 Ejemplos de los anteriores tipos de criptografía	26
2.5.1 3DES	26
2.5.2 AES	30
2.5.3 Diffie-Hellman	34
2.5.4 RSA	36
2.5.5 ECC	39
2.5.6 La familia de algoritmos SHA	43
2.5.7 SHA3	46
2.5.8 PBKDF2	49
2.5.9 Argon2	51
3 Desarrollo e Implementación	55
3.1 Introducción	55

3.2	Elecciones Iniciales	55
3.3	Diseño de la aplicación y listado de principales funcionalidades	57
3.3.1	Principales funcionalidades de la aplicación	58
3.3.2	Diagramas de diseño de la aplicación	59
3.3.2.1	Diagrama de casos de uso	59
3.3.2.2	Estructura de la tabla de contraseñas a nivel base de datos	69
3.3.3	Bocetos de vistas de la aplicación	70
3.4	Implementación de la aplicación	72
3.4.1	Descripción detallada de la implementación de las funcionalidades finales de la aplicación	72
3.4.2	Interfaz de usuario final de la aplicación	89
4	Conclusiones y líneas futuras	93
4.1	Sobre la base teórica	93
4.2	Sobre PassGuard	94
	Bibliografía	95
	Apéndice A Manual de usuario	101
A.1	Introducción	101
A.2	Vista principal de la aplicación	101
A.3	Crear un nuevo PassGuard Vault	102
A.4	Cargar un Passguard Vault	104
A.4.1	Vista principal con los contenidos de un Passguard Vault	104
A.5	Creación de contraseñas aleatorias rápidas	108
A.6	Menú de opciones en la vista principal	109
A.6.1	Crear una copia de seguridad de un Passguard Vault	110
A.6.2	Crear copias de seguridad automáticas de un Passguard Vault	110
A.6.3	Cambiar los colores de contorno de Passguard	111
	Apéndice B Manual de Despliegue e Instalación de Passguard	115
B.1	Introducción	115
B.2	Instalación de Passguard para el público general.	115
B.3	Despliegue de Passguard para desarrolladores.	116

Índice de figuras

2.1	Diagrama de flujo de cifrado simétrico.	10
2.2	Diagrama de flujo de cifrado asimétrico.	12
2.3	Diagrama de flujo de cifrado híbrido.	14
2.4	Diagrama de flujo de cifrado por flujo (de bits).	17
2.5	Diagrama de flujo de cifrado por bloques con modo de operación CBC.	19
2.6	Resultado de función SHA1 para dos cadenas de caracteres similares.	20
2.7	Diagrama general del funcionamiento de DES.	27
2.8	Esquema de red o Cifrado Feistel.	29
2.9	Diagrama de funcionamiento general de AES.	33
2.10	Ejemplo de intercambio por Diffie-Hellman.	36
2.11	Diagrama de flujo de RSA simplificado.	39
2.12	Aspecto de la curva elíptica con fórmula $y^2 = x^3 - (7 \cdot x) - 10 \bmod 19$	41
2.13	Diagrama simplificado de proceso para criptografía de clave elíptica.	42
2.14	Diagrama simplificado de flujo general de SHA256.	46
2.15	Composición de la matriz multidimensional estado.	48
2.16	Diagrama general de SHA3.	49
2.17	Diagrama de funcionamiento de PBKDF2.	51
2.18	Diagrama general de Argon2.	53
3.1	Logo de la aplicación PassGuard.	55
3.2	Logos de tecnologías utilizadas: Parte 1.	56
3.3	Logos de tecnologías utilizadas: Parte 2.	57
3.4	Diagrama de casos de uso simplificado: Parte 1.	60
3.5	Diagrama de casos de uso simplificado: Parte 2.	61
3.6	Diagrama de casos de uso simplificado: Parte 3.	62
3.7	Diagrama de base de datos.	69
3.8	Boceto inicial de vista principal de la aplicación.	70
3.9	Boceto inicial de la vista una vez se inició sesión en la Passguard Vault.	71
3.10	Boceto inicial de la vista para iniciar sesión en un Passguard Vault.	71
3.11	Boceto inicial de la vista para generar contraseñas aleatorias.	72
3.12	Diagrama de derivación de claves y descryptado de contenido de un Passguard Vault.	73
3.13	Tabla de contraseñas en formato PDF.	84
3.14	Vista principal de la aplicación.	90
3.15	Vista de creación de contraseñas.	90
3.16	Vista con la tabla de contraseñas.	91
3.17	Vista de generación de algunas estadísticas.	91
3.18	Vista del gestor de configuraciones de colores de contorno.	92
3.19	Listado de todos los archivos generables y descargables en Passguard.	92

A.1	Vista principal de la aplicación con cuatro partes importantes.	101
A.2	Vista de creación de un nuevo Passguard Vault.	102
A.3	Ventana emergente una vez se ha creado correctamente un Passguard Vault.	103
A.4	Mensaje emergente en caso de errores en la creación de un Passguard Vault.	103
A.5	Vista de inicio de sesión o carga de una Passguard Vault.	104
A.6	Vista de la tabla de contraseñas.	105
A.7	Lista de opciones de ordenado.	105
A.8	Mensaje emergente de confirmación de borrado de una contraseña.	106
A.9	Ventana emergente de borrado de contraseñas en un Passguard Vault.	107
A.10	Mensaje emergente del exportado correcto de una Passguard Vault como PDF.	107
A.11	Ventana de generación de estadísticas.	108
A.12	Vista de creación de contraseñas aleatorias rápidas con cinco partes importantes.	108
A.13	Menú de opciones en la vista principal.	109
A.14	Ventana emergente de creación de una copia de seguridad de una Passguard Vault.	110
A.15	Ventana emergente de configuración de copias de seguridad automáticas de un Passguard Vault.	111
A.16	Ventana emergente del gestor de configuraciones de colores de contorno.	112
A.17	Ventana emergente para añadir una configuración de colores.	113
A.18	Ventana emergente de ayuda al funcionamiento de la vista con tabla de configuraciones de colores de entorno.	114
B.1	Listado de dependencias para el instalador de Passguard.	115
B.2	Archivos con los que se trabajará la instalación de Passguard.	116
B.3	Lista de programas y dependencias para desarrolladores en Passguard.	116
B.4	Archivo solución de PassGuard, debe ser abierto con Visual Studio 2022.	117
B.5	Configuración de Visual Studio 2022 para ejecutar el proyecto Passguard.	117

Capítulo 1

Introducción

1.1 Introducción

Estamos en una época en la que los documentos, interacciones con otras personas, transacciones y otros servicios tienden a inclinarse más hacia el mundo digital que hacia el mundo real. Por poner ejemplos: en vez de tener una biblioteca en casa con múltiples libros podemos tener un dispositivo Kindle donde tener cientos de libros disponibles al momento, en vez de realizar reuniones presenciales en la oficina podemos realizarlos a través de aplicaciones como Microsoft Teams, Google Meet o Skype desde la comodidad de nuestro hogar, si queremos enviar dinero a una cuenta bancaria podemos hacerlo con suma facilidad desde la aplicación móvil del banco, más rápido y sencillo que ir a la sucursal más cercana. En definitiva, hoy en día el mundo digital es muy importante para nosotros, puede que incluso más que el mundo real.

Prácticamente cualquier servicio que queramos imaginarnos puede ser satisfecho a través de Internet, el cuál ha facilitado de sobremano la globalización. Si queremos ver una película, nos crearemos una cuenta en un servicio de streaming mediante suscripción como Netflix, Disney o HBO, pagaremos dicha suscripción y listo. Si queremos comprar ropa en una tienda online, nos crearemos una cuenta en la página web de ese local y realizaremos nuestras compras fácil y rápidamente. La mayoría de los servicios en Internet requieren de crear una cuenta para interactuar con ellos, principalmente para almacenar todos los datos relacionados contigo, como por ejemplo tu dirección de entrega o un método para comunicarse contigo directamente, además de proveer una experiencia de usuario personalizada y obtener datos de la actividad para así mejorar el servicio.

Las cuentas de usuario suelen consistir principalmente en un correo electrónico o nombre de usuario y una contraseña, la cual sólo debes conocer tú para que así únicamente tú seas la persona que pueda entrar a esa cuenta. Con tantos servicios en Internet, lo más probable es que una persona tenga muchas cuentas de usuario en diferentes servicios, sin embargo, es complicado realizar una estimación debido a que hay muchos tipos de usuarios con diferentes necesidades, sin tener en cuenta que se pueden tener múltiples contraseñas en un mismo sitio web.

Las personas solemos tener un puñado de correos electrónicos, sería incómodo además de poco práctico tener un correo electrónico o nombre de usuario nuevo por cuenta, por tanto, lo que debe ir cambiando en cada cuenta es la contraseña que utilicemos. El principal motivo por el cual no es seguro utilizar la misma contraseña en todas nuestras cuentas es que hay cibercriminales que pueden utilizar técnicas para robar o descifrar tu correo o nombre de usuario y tu contraseña, y si lo consigue y utilizas esa combinación en otros sitios webs, podemos concluir que esas cuentas también han sido comprometidas, ya que el cibercriminal puede comprobar esa combinación en diferentes sitios webs que utilizamos comúnmente y

podrá desbloquearlos. Sin embargo, es ciertamente complicado que una persona recuerde más de 100 contraseñas distintas y seguras por cada sitio web.

Existen múltiples aproximaciones para este problema. La primera de ellas es seguir utilizando la misma combinación de correo o nombre de usuario y contraseña para todas las cuentas, se han descrito antes los potenciales problemas de esta decisión. La segunda aproximación es usar unas pocas contraseñas o una contraseña y las posibles modificaciones de la misma que se le ocurran al usuario para todas sus cuentas. Esta aproximación es mejor que la anterior ya que se usan diferentes contraseñas, pero sigue teniendo el mismo problema, si descubro una combinación correo electrónico o nombre de usuario y contraseña puedo desbloquear más de una cuenta. La tercera opción, y la que debería ser usada por todos, es tener una contraseña diferente por cada cuenta, pero seguimos teniendo el dilema de dónde y cómo guardamos tantas combinaciones de correo electrónico o nombre de usuario y contraseña, ya que asumimos que si eres capaz de memorizar muchas contraseñas distintas lo más probable es que sea debido a que partes de combinaciones sencillas de contraseñas con variaciones del mismo tipo, y este tipo de contraseñas sencillas por lo general no son seguras.

Hay varias principales maneras de abordar este problema. La primera es apuntarse cada combinación correo electrónico o nombre de usuario y contraseña en un papel o en un archivo de texto plano, en esta situación debemos tener en cuenta que guardar las contraseñas en texto plano no es una buena idea, ya que basta con hacerse con el archivo para tener todas las cuentas desbloqueadas. La segunda opción es que nuestro navegador web recuerde los inicios de sesión (y, por tanto, las combinaciones correo electrónico o nombre de usuario y contraseña) para cada sitio web, este caso tampoco es conveniente debido a que, al igual que guardar las contraseñas en texto plano, podemos ser víctimas de diferentes ataques, por ejemplo, de malware en caso de los archivos locales o de envenenamiento de sesión/cookies en el caso del navegador, con los cuales el cibercriminal puede obtener nuestras credenciales descriptadas o acceder a nuestras cuentas y cambiar los datos de inicio de sesión.

La tercera opción, y la que se recomienda, es guardarlas en un gestor de contraseñas dedicado. Un gestor de contraseñas por lo general encripta tus contraseñas con una contraseña maestra (la cual nunca se debe utilizar para otros inicios de sesión) que introduce el usuario, de tal manera que éste sólo tiene que acordarse de su contraseña maestra para así desbloquear el resto de sus contraseñas. Es muy importante informarnos de si el gestor de contraseñas guarda nuestras contraseñas en texto plano o encriptadas, y descartar los que la guarden en texto plano ya que caeríamos en el mismo problema que en anteriores aproximaciones al problema. Como estadística, se observó [1] que tanto en 2021 como en 2022 la mayoría de los estadounidenses que sufrieron un robo de credenciales utilizaban las mismas pocas contraseñas en todas sus cuentas o las guardaban en un archivo de texto plano o en el navegador, la minoría de ellos guardaba sus contraseñas apuntadas a papel o en un gestor de contraseñas dedicado. Por tanto, y como conclusión de estos datos, se recomienda utilizar una combinación correo electrónico o nombre de usuario y contraseña distinta por cada cuenta, y utilizar un gestor de contraseñas que encripte tus contraseñas para así manejarlas de manera segura.

Como idea general, para crear un gestor de contraseñas necesitaremos utilizar algoritmos de encriptación de datos para cifrar las contraseñas del usuario, de tal manera que en caso de sufrir un robo de las contraseñas el atacante no vea conveniente o no le salga a cuenta tratar de descriptar las contraseñas, así que a lo largo de este documento se revisarán diferentes algoritmos criptográficos, la mayoría de ellos usados en la actualidad, de cara a luego seleccionar los necesarios para diseñar e implementar un gestor de contraseñas seguro. A continuación, se informará acerca del Estado del Arte de los gestores de contraseñas en la actualidad, su estado actual y sus ventajas e inconvenientes entre ellos, de cara a situar al lector en el contexto actual sobre este tema. Como nota: el Estado del Arte de los algoritmos criptográficos se cubrirá en la Base Teórica de este documento, debido a que ahí se estudiarán los algoritmos y funciones

criptográficas principales de este momento, lo cual ya provee al lector de un contexto acerca de este apartado.

1.2 Estado del Arte

En la actualidad existen múltiples gestores de contraseñas, esto es debido a que, pese a que la funcionalidad es la misma, cada uno cubre diferentes necesidades y ofrece diferentes funcionalidades o características al usuario. A continuación, se exponen algunos de ellos.

LastPass [2] es uno de los servicios de gestor de contraseñas de pago por suscripción más utilizados globalmente, con alrededor de 33 millones de usuarios alrededor del mundo. Tiene una interfaz sencilla y de aspecto moderno, y sus principales funciones [3] son el almacenamiento y generación segura de contraseñas fuertes, además del autocompletado de contraseñas, multinivel de factor de autenticación, compatibilidad multiplataforma (tanto navegadores webs accesibles desde un ordenador de escritorio como una aplicación móvil) y auditoría segura de tus contraseñas.

Cuando creamos una cuenta en LastPass, deberemos establecer una contraseña maestra, y en base a esa contraseña maestra se crea una caja fuerte donde se guardan tus contraseñas. La contraseña maestra [4] debe tener 12 caracteres (aunque se recomiendan 16 o más caracteres) y mezclar letras mayúsculas, minúsculas, números, símbolos y caracteres especiales, y en la medida de lo posible que no sea algo predecible, es decir, que, por ejemplo, sabiendo la mitad de la contraseña no pueda adivinar la otra parte con rapidez. Un ejemplo de esto sería que, si yo pongo “Barack”, la gran mayoría de gente sabría decirme que lo que sigue es “Obama”.

Una vez establecida esta contraseña maestra, LastPass utiliza [5] funciones de derivación de claves (en concreto, PBKDF2-SHA-256) para derivar una clave segura (a partir de la unión de tu correo electrónico y tu contraseña maestra) para tu caja fuerte con tus contraseñas. Esta clave se utiliza para descryptar tus contraseñas, las cuales han sido encriptadas con el algoritmo AES-256. Según LastPass, todos los procesos de encriptado y descryptado se producen en el lado del cliente (el usuario final) y no en el lado del servidor (LastPass), y además el usuario final es el único capaz de descryptar su caja fuerte, ya que LastPass no guarda la contraseña maestra en sus servidores (guarda un hash generado de manera muy específica, y es inviable obtener a partir de un hash el texto plano que lo generó), por tanto, el usuario es el único que puede generar a partir de su contraseña maestra la clave de seguridad que descrypta su caja fuerte de contraseñas.

Entre las principales ventajas de LastPass se encuentran el ser un gestor de contraseñas reconocido en el mercado con alrededor de 33 millones de usuarios, su compatibilidad multiplataforma y con factor de autenticación, su sencillez y facilidad de uso, el uso de funciones y algoritmos criptográficos robustos y con eficacia probada, sin dejar de contar su modelo de seguridad cero-conocimiento por el cual, entre otros aspectos, no guardan tu contraseña maestra en sus servidores, de tal forma que solo la conozca el usuario.

LastPass tiene ciertas desventajas, entre la que destaca una bastante importante para un gestor de contraseñas. Entre estas desventajas se encuentra la dificultad o imposibilidad de operar con LastPass de manera offline, el hecho de que algunas características como el acceso a través de varios dispositivos estén solo disponibles en la versión de pago, y la preocupación de todos los servicios de gestión de contraseñas, que es que tus contraseñas están guardadas en una caja fuerte en un servidor en la nube, e inmediatamente saltan preguntas del tipo ¿quién tiene acceso a ese servidor?, ¿cómo está ese servidor protegido?, ¿si elimino mi cuenta de LastPass, cómo me cercioro de que la caja fuerte se borra completamente del servidor en la nube y que nadie se ha hecho una copia? Hay una desventaja importante que se relaciona

con LastPass y es su seguridad. Pese a que los algoritmos y funciones criptográficas utilizados son seguros, LastPass ha tenido importantes brechas de seguridad en donde las cajas fuertes con contraseñas encriptadas de múltiples usuarios se han visto comprometidas, han tenido brechas de seguridad importantes tanto en 2015 con actividad sospechosa sobre sus sistemas, en 2021 con el compromiso de la seguridad de las contraseñas maestras y recientemente en finales de 2022 con el robo de datos de usuario y cajas fuerte parcialmente encriptadas con contraseñas. Esto ha mermado considerablemente la confianza sobre este gestor de contraseñas, y pese a que LastPass ha puesto más medios y colaboración para identificar y solucionar la brecha, no se sabe si esto recuperará la confianza de los usuarios.

1Password [6] es otro servicio de gestión de contraseñas multiplataforma de pago por suscripción con alrededor de 15 millones de usuarios globales [7] en la cual se permite guardar contraseñas, así como otros tipos de información privada en una caja fuerte virtual descifrable únicamente a través de una contraseña maestra. Tiene una interfaz simple y amigable, y sus principales funciones son la generación de contraseñas seguras, el almacenamiento de contraseñas y su autocompletado, además de un apartado de notas seguras, auditoría de contraseñas y un buen sistema de sincronización a través de múltiples plataformas.

La forma de operar con 1Password [8] es ciertamente similar a con LastPass, basta con crearse una cuenta y pagar la suscripción, y se creará una caja fuerte vacía para guardar nuestras contraseñas. Cabe destacar que 1Password provee una manera de importar nuestras contraseñas [9] tanto desde un archivo local como puede ser un archivo CSV o de otros gestores de contraseñas populares en el mercado, como puede ser el caso de LastPass. La contraseña maestra de 1Password no necesita requisitos específicos [10] de qué tipos de caracteres utilizar en tu contraseña maestra, pero sí que requiere que tenga al menos 10 caracteres [11].

Hacen falta dos elementos para acceder y leer los datos de la caja fuerte, el primero es tu contraseña maestra, la cual la define el usuario y es la única que necesita recordar, y 1Password no la almacena. El segundo elemento es una clave secreta, es única por cuenta y se trata de una clave de 128 bits que contiene 34 letras y números separados por guiones, unos 340.2 sextillones de combinaciones, inviable de descifrar por fuerza bruta. Esta clave se genera en el dispositivo del cliente y se guarda en el mismo de manera segura, de tal forma que [12] la combinación de tu contraseña maestra y la clave secreta genera la clave que descifra los contenidos de tu caja fuerte con contraseñas. 1Password permite crear un kit de emergencia [13], el cual básicamente es un PDF con los detalles de tu cuenta para que puedas acceder a ella.

1Password [14] utiliza AES-GCM-256 para encriptar tu información y PBKDF2-HMAC-SHA256 para generar claves seguras además de SQLite como herramienta para la estructura de la caja fuerte. 1Password guarda únicamente datos encriptados, el encriptado y descifrado se realiza del lado del cliente, y no guarda ni tu contraseña maestra (únicamente la sabe el cliente) ni la clave secreta (se guarda en el dispositivo del cliente de manera segura). Como vemos, en cuanto a algoritmos de criptografía es muy similar a LastPass, además de que 1Password también tiene una política de cero-conocimiento.

Algunas ventajas de 1Password son los algoritmos criptográficos seguros que utiliza, la transparencia en cuanto a los formatos y algoritmos utilizados, su sincronización multiplataforma a través de aplicaciones móviles, de escritorio y en navegadores web, además de su facilidad de uso debido a su interfaz de usuario sencilla.

Pocas desventajas tiene 1Password, algunas de ellas podrían ser el hecho de ser un servicio de pago, o por ejemplo no poder operar con tus contraseñas de manera local sin usar la nube. 1Password no ha tenido grandes brechas de seguridad reportadas, sí que es cierto que hay herramientas que han mejorado

la velocidad con la que descifran contraseñas maestras de 1Password, pero no se considera una brecha de seguridad.

Bitwarden es un servicio de gestión de contraseñas por suscripción con entre 15 y 20 millones de usuarios globales [15], permite crear contraseñas seguras, almacenar contraseñas y otros elementos como notas o tarjetas bancarias y poder compartirlas con usuarios específicos que el usuario desee, autocompletado de contraseñas, así como su auditoria de contraseñas y de la caja fuerte y la posibilidad de utilizar tu caja fuerte en modo lectura alejada de los servidores de Bitwarden en la nube creando un servidor de Bitwarden en tu red local. Bitwarden destaca entre las anteriores opciones por ser una aplicación open-source, es decir, de código abierto [16].

En Bitwarden hace falta crearse una cuenta y establecer una contraseña maestra de al menos 12 caracteres que utilice caracteres especiales y símbolos y que no sea predecible. Una vez hecho este paso, podrás establecer doble factor de autenticación y crear una nueva caja fuerte de contraseñas a la que podrás importar contraseñas que tengas guardadas en otros servicios como LastPass, 1Password, Dashlane o en tu navegador web. Una vez creada tu caja fuerte, añadir una contraseña será tan simple como agregar el nombre de la misma, usuario y contraseña y URL de la página web a la que corresponde la contraseña, además de una carpeta donde quieras guardar la contraseña dentro de tu caja fuerte.

Bitwarden funciona de manera similar a LastPass en el caso de las cajas fuerte. Con tu contraseña maestra y tu correo electrónico se genera una clave secreta utilizando la función PBKDF2 SHA-256 por defecto, y esa clave es la que descripta la caja fuerte de contraseñas encriptada con el algoritmo AES-CBC 256. Todo el encriptado o el descriptado se realiza del lado del cliente y Bitwarden no guarda ni tu contraseña maestra ni tus claves criptográficas [17]. Además, Bitwarden permite cambiar el algoritmo de derivación de claves y su número de iteraciones, es decir, podemos cambiar los valores por defecto de PBKDF2 SHA256 y 600000 iteraciones a Argon2id con 3 iteraciones, 64MB de memoria y un grado de paralelismo de 4, estas configuraciones de Argon2id son equivalentes en la protección que proveen, pero hay diferencia entre uso de CPU y de RAM según los valores que utilicemos [18], por tanto, es conveniente informarse antes de modificar estos valores ya que podemos o desproteger nuestra caja fuerte de contraseñas, o podemos protegerla tanto que, sabiendo la contraseña maestra, tardemos muchos minutos en descriptar nuestra caja fuerte. No podemos modificar el algoritmo de encriptación ya que se considera que con una buena implementación del algoritmo AES y una buena contraseña maestra, el algoritmo es inviable de descifrar.

Como hemos visto, Bitwarden tiene, entre otras, las siguientes ventajas: la sencillez de la interfaz de usuario, la seguridad de los algoritmos y las funciones criptográficas utilizadas, el hecho de que sea una herramienta de código abierto y con una comunidad activa (más adelante se discutirá por qué un gestor de contraseñas sería de código abierto, ya que parece contraintuitivo) y la posibilidad de configurar ciertos parámetros criptográficos, además de poder operar con tus contraseñas de manera ajena a los servidores en la nube de Bitwarden, el hecho de que tenga un plan gratuito con una caja fuerte de contraseñas y la mayoría de funcionalidades principales de Bitwarden y el uso de su política cero-conocimiento, en la que no guardan tu contraseña maestra ni claves criptográficas y la encriptación o descriptación se hace del lado del cliente, y su sincronización multiplataforma (aplicación escritorio, web y móvil). Añadir como ventaja que Bitwarden no tiene brechas de seguridad conocidas [19].

Las desventajas principales de Bitwarden son el hecho de que funciones como el doble factor de autenticación esté disponible sólo en los planes de pago, el hecho de que desplegar un servidor local de Bitwarden en tu red local puede ser complejo para el usuario medio, el autocompletado de contraseñas puede no funcionar tan bien como los de sus competidores y que algunas opciones que sí se ofrecen en otros gestores como puede ser espacio de almacenamiento seguro, factor de autenticación biométrico o monitorización activa de brechas de seguridad en la Dark Web no se proveen en Bitwarden.

KeePass es un gestor de contraseñas gratuito y de escritorio, desde 2019 su versión KeePass 2.x se ha descargado más de 45 millones de veces según fuentes oficiales [20]. Este gestor de contraseñas se caracteriza por ser completamente offline, su capacidad para generar contraseñas seguras, almacenar contraseñas, notas y otra información sensible de manera segura, y por ser un gestor de contraseñas de código abierto. Con algunos plugins se pueden conseguir funcionalidades adicionales como doble factor de autenticación o la integración del gestor de contraseñas con el navegador web.

La manera de operar con KeePass es diferente al resto de gestores de contraseñas, en primer lugar, tendremos que descargar e instalar la aplicación como cualquier otra y, una vez iniciada, tendremos que crear una nueva base de datos de contraseñas. Para esta base de datos deberemos establecer la clave maestra y una vez realizado este paso podremos añadir mayor seguridad a nuestra base de datos añadiendo un archivo de claves de KeePass con el cual encriptar tu base de datos, con esto conseguimos que, aunque alguien consiguiese tener tu archivo de bases de datos de contraseñas y tu contraseña maestra, no sea capaz de descryptar tu base de datos de contraseñas ya que le falta el archivo de claves que se utilizó para encriptar la base de datos de contraseñas, es un estilo de segundo factor de autenticación, solo que en vez de un código numérico común tenemos un archivo de claves KeePass.

Una vez establecida la contraseña maestra y, opcionalmente, el archivo de claves, KeePass nos permitirá modificar el algoritmo de encriptación y de derivación de claves: en cuanto a la encriptación podremos elegir entre AES256 o ChaCha20, y en cuanto a la función de derivación de claves podremos elegir AES-KDF (es una función de derivación de claves derivada de AES, en la cual básicamente se aplica AES tantas veces como iteraciones se especifiquen, hasta derivar una clave de encriptación segura; se trata de un método seguro aunque costoso a nivel de CPU, pero es más vulnerable a ataques con GPU de lo que pueden ser algoritmos como Argon2, bcrypt o scrypt) o Argon2. Una vez elegidas las configuraciones, creará la base de datos de contraseñas y nos recomendará descargar el kit de emergencia, el cual es un archivo que contendrá todos los datos necesarios para abrir la base de datos de contraseñas, en ese archivo KeePass rellenará algunos datos, pero otros como tu clave maestra deberán ser rellenados por el usuario.

Entre las ventajas de utilizar KeePass se encuentran el hecho de utilizar un gestor completamente offline y tener la seguridad de que solo el usuario tiene la base de datos con contraseñas, los algoritmos y funciones criptográficas robustas utilizadas además de su personalización, el hecho de que sea una herramienta de código abierto con una comunidad activa, la tenencia de un buen generador de contraseñas y el hecho de poder añadir un archivo como parte de la clave que encripta la base de datos de contraseñas.

KeePass tiene ciertas desventajas, las principales de ellas son que su interfaz de usuario está claramente por detrás de sus competidores, la sincronización entre dispositivos debe ser manual ya que KeePass está disponible para Windows, MacOS o Linux (por tanto, no tiene integración con móvil o web), el hecho de que para obtener funcionalidades existentes de base en otros gestores de contraseñas en KeePass debas añadirle plugins a la aplicación y que las copias de seguridad de la base de datos deben ser realizadas manualmente por el usuario.

Por otra parte, KeePass es una aplicación de escritorio que es activamente analizada por la comunidad criptográfica y, por ello, hay personas u organizaciones que tratan de romper la aplicación (ya que la herramienta es de código abierto) en busca de vulnerabilidades. Desde su creación, a KeePass se le han descubierto ciertas vulnerabilidades, la mayoría de ellas, según los desarrolladores de KeePass [21], no ponen en peligro las bases de datos de contraseñas, pero ha habido exploits o scripts que ayudan a obtener la contraseña maestra de una base de datos de contraseñas de KeePass. Todas las vulnerabilidades tienen una explicación clara del equipo de desarrollo de KeePass y en cuestión de días la gran mayoría de estas suelen ser parcheadas. Pese a estas vulnerabilidades que aparecen de vez en cuando, KeePass es considerado un gestor de contraseñas seguro [22].

Algunos gigantes tecnológicos como Google o Apple tienen su gestor de contraseñas integrado, en el caso de Google lo tienen a través de su navegador Google Chrome y en el caso de Apple lo tiene integrado en iOS. En el caso de Google basta con tener una cuenta de Google y ahí se guardarán tus contraseñas, con Apple probablemente se guarden en iCloud. No hay mucha información sobre cómo se manejan las contraseñas a nivel de criptografía, se conoce que Google utiliza AES-256 [23] en sus servicios en la nube, por tanto, presumiblemente lo utilicen con las contraseñas, y de Apple no se conocen los algoritmos utilizados, pero muy probablemente sean similares a los ya mencionados.

Vistos ya los principales gestores de contraseñas del mercado actual, aprovecho este apartado para dar algunos puntos de vista acerca de un par de preguntas que posiblemente el usuario medio se podría estar haciendo en estos momentos.

La primera duda que podría tener un usuario medio es por qué utilizaría un gestor de contraseñas y, básicamente, poner uno de los elementos más preciados que tiene un ser humano en el mundo digital como son todas sus contraseñas de sus cuentas en un contenedor o base de datos encriptada de una empresa de la cual sí que tenemos información de cómo trabaja con nuestras contraseñas, pero nunca lo podremos comprobar con certeza. Para esto debemos pensar que los investigadores de seguridad recomiendan utilizar gestores de contraseñas [24] debido a que las bondades del uso del mismo pesan más que las desventajas de que pueda haber una brecha de seguridad. La idea de un gestor de contraseñas es buena y es segura si la teoría de cómo deben funcionar y los algoritmos son implementados correctamente, de lo contrario tendrían brechas de seguridad. Cabe resaltar que los gestores de contraseñas no son inmunes a ataques de keyloggers (software que registra las teclas que se pulsan en un teclado para luego mandarlas en un fichero por Internet) por ejemplo. El hecho de confiar o no en las empresas de gestión de contraseñas va en cada usuario, estas empresas tienen un modelo de negocio en el que se tienen que ganar la confianza del usuario a través de un historial sin brechas de seguridad, muchas auditorías y demás para que el usuario confíe y guarde sus contraseñas ahí. Las posibilidades a priori que tienen estas empresas son o almacenar de manera segura las contraseñas para que los clientes sigan pagando la suscripción y que sean rentables, o robar todas las contraseñas de los usuarios, y para esta última hay formas menos costosas y más rápidas de conseguirlo. En definitiva, la última decisión la tiene el usuario.

Otra duda sería por qué un gestor de contraseñas sería una herramienta de código abierto. Si alguien quiere guardar contraseñas como si fuesen secretos, pero al mismo tiempo se publica en otro lugar cómo guardas esos secretos, parece fácil deducir que pueden descifrar tus contraseñas. Esto tiene cierta parte de razón, sin embargo, la respuesta a esta cuestión es un balance de aspectos positivos y negativos. Por una parte, es cierto que publicas cómo guardas los datos, pero los gestores de contraseñas a nivel teórico son viables y seguros, de forma que si el gestor de contraseñas está bien implementado no hay por qué temer ya que sería inviable ponerse a descifrar los datos que guarda. Por otra parte, el hecho de tener publicado cómo manejas la información permite al usuario ver que tus datos no se venden a terceros, que no hay puertas traseras por donde obtener tus datos descifrados y desaparecer, permite a los usuarios auditar el código en busca de vulnerabilidades y, en definitiva, permite a la comunidad buscar vulnerabilidades y reportarlas o incluso arreglarlas. Las empresas de gestores de contraseñas de código abierto suelen tener programas en los que invitan a gente a buscar vulnerabilidades y si las encuentran pueden recibir recompensas y crédito por haberlas encontrado. También puedes crear versiones personalizadas de la aplicación, ya que tienes acceso al código de la misma, por tanto, alguien con conocimientos puede llevar a cabo modificaciones, publicarlas en caso de tener permiso para ello, y que eso haga que la empresa implemente esas nuevas características, mejorando así la aplicación. Como vemos, hay bastantes ventajas de tener la aplicación como código abierto frente a la posibilidad de revelar cómo guardas las contraseñas en este caso específico.

Como nota final para este apartado, es posible que, sobre todo para los gestores de contraseñas por suscripción, las descripciones de los algoritmos y funciones criptográficas sean ciertamente generales. Por ejemplo, para gestores de contraseñas como LastPass o 1Password, debemos tener en cuenta que el encriptado y desencriptado se hace en el lado del cliente, así que tenemos que autenticarnos con el servidor del gestor de contraseñas de turno para que nos envíe nuestra caja fuerte de contraseñas y que podamos desencriptarla y leer nuestras contraseñas. Esta autenticación se hace también con nuestra contraseña maestra y lo que se hace es derivar de la misma una clave de autenticación con el servidor, de tal manera que este sepa que en realidad sí somos quien decimos ser y nos pueda enviar la caja fuerte de contraseñas que tiene guardada en la nube. El método para derivar esta clave de autenticación varía según el gestor de contraseñas, LastPass vuelve a utilizar la clave generada al iterar con el correo y la contraseña maestra utilizando PBKDF2, esta clave la junta con la contraseña maestra e itera con la función PBKDF2 múltiples veces hasta derivar esa clave de autenticación. 1Password por su parte utiliza criptografía de clave pública para, junto con la contraseña maestra y la clave secreta, desencriptar la clave privada que te autentica con el servidor y te permite traerte tu caja fuerte de contraseñas al lado del cliente. También es cierto que las iteraciones realizadas para derivar las claves criptográficas no se realizan completamente en el lado del cliente, sino que unas pocas iteraciones se realizan en el lado del cliente (ya que no se sabe qué potencia tendrá el lado del cliente y, por tanto, se ponen un número de iteraciones que cualquier dispositivo podría aguantar en un tiempo razonable) y el resto de iteraciones se realizan en el lado del servidor, que tiene la potencia suficiente para manejar múltiples iteraciones de múltiples cajas fuertes al mismo tiempo. Como se ve, estos detalles de implementación varían según el gestor de contraseñas, y nos tomaría mucho más tiempo explicar cómo funcionan estos gestores de manera detallada, por eso se ha realizado una descripción general de los mismos. El objetivo de este apartado es explicar cómo está la situación de gestores de contraseñas en la actualidad, más que entrar en detalles específicos de cómo funciona cada gestor de contraseñas del mercado.

Capítulo 2

Estudio teórico

2.1 Introducción

En este apartado se redactará e informará al lector acerca de la base teórica investigada y obtenida de cara a la implementación del caso práctico de un gestor de contraseñas seguro, PassGuard. Se explicarán algunos aspectos básicos de la criptografía de cara este proyecto y se continuará explicando los diferentes algoritmos y funciones criptográficas, principalmente destacando su descripción, cómo funciona el algoritmo y sus principales ventajas y desventajas.

Para empezar este apartado y de cara a los lectores con menor experiencia en el apartado criptográfico, procederemos a definir y exponer ciertas bases de la criptografía.

La criptografía [25][26] son un conjunto de técnicas matemáticas que permiten transformar datos de tal manera que nadie no autorizado pueda leerlos o modificarlos. Esto se consigue modificando los datos en texto plano (es decir, legibles para cualquiera) a través de ciertos procesos en un texto cifrado, consiguiendo así la integridad y confidencialidad de los datos, cabe resaltar que dependiendo de los algoritmos utilizados ese texto cifrado puede volver a ser texto plano o no. La criptografía tiene como inicio los antiguos jeroglíficos egipcios, es una técnica muy antigua ya que desde antaño se ha querido enviar información de tal manera que solo las personas para las que el mensaje está destinado puedan leerlo y ninguna otra persona más, y hoy en día sigue presente debido a que han cambiado los medios de comunicación, pero seguimos teniendo esa necesidad. Pensando en el caso de un gestor de contraseñas, tenemos tres formas distintas de procesar la información: la primera es tenerla en texto plano y, por tanto, visible para todo el mundo. Esto es, por razones obvias, una mala manera de almacenar las contraseñas. La segunda forma es utilizar funciones criptográficas hash, esto se trata de una función matemática que toma como entrada un mensaje y produce una salida cifrada de tamaño fijo, llamado hash. Estas funciones cambian el hash resultante radicalmente en cuanto se produce el mínimo cambio en la entrada, y son funciones diseñadas para ser rápidas e irreversibles, es inviable descifrar el texto plano que produjo un hash. Normalmente nos referimos a las funciones criptográficas hash como funciones criptográficas de un solo sentido, esto es debido a que el texto plano puede ser convertido a texto cifrado, pero el texto cifrado no se puede revertir a texto plano. La tercera forma es utilizar funciones de cifrado, se tratan de funciones que contienen claves, ya sean públicas o privadas, con las cuales se encripta el texto plano y se convierte en texto cifrado. Dependiendo del tipo de algoritmo, la misma clave o una distinta es utilizada para pasar del texto cifrado a texto plano. Por tanto, los algoritmos de cifrado son también llamados algoritmos de doble sentido, ya que se puede pasar tanto de texto plano a texto cifrado como de texto cifrado a texto plano utilizando la clave secreta. Existen múltiples tipos de algoritmos criptográficos dependiendo del número

de sentidos (uno o dos) y de la complejidad y seguridad que manejan. Algunos de estos algoritmos o funciones criptográficas son la criptografía de clave secreta o cifrado simétrico, la criptografía de clave pública o cifrado asimétrico, las funciones criptográficas hash, funciones de derivación de claves KDF y luego otro tipo de técnicas como la criptografía de clave elíptica ECC o los generadores de números aleatorios RNG, entre otros. En los siguientes apartados definiremos y ampliaremos el conocimiento acerca de estos algoritmos que posiblemente podrían ser utilizados en el caso práctico de implementación de un gestor de contraseñas.

2.2 Tipos de cifrado en criptografía según su clave

En este apartado se describirán los dos tipos de criptografía según cómo se manejan las claves [27], se trata de la criptografía de cifrado simétrico o criptografía de claves secreta, y la criptografía de cifrado asimétrico o criptografía de clave pública. Recordemos que el proceso de cifrado o encriptación es la transformación de datos legibles por cualquiera (texto plano) en datos cifrados, con el propósito principal de proteger la información de personas a las que no esté dirigida.

2.2.1 Cifrado Simétrico

La criptografía de cifrado simétrico o criptografía de clave secreta [28], es una técnica criptográfica muy utilizada en la actualidad para cifrar o encriptar información. La idea básica y fundamental de este tipo de cifrado es que, tanto para el proceso de encriptado como desencriptado de la información, se utiliza la misma clave secreta, de ahí el nombre del cifrado. Mantener la clave de manera secreta es la parte fundamental de los algoritmos de encriptación, de hecho, la seguridad de la información encriptada depende de la clave y mantenerla oculta de manera segura, más que del algoritmo utilizado.



Figura 2.1: Diagrama de flujo de cifrado simétrico.

La clave secreta debe ser conocida únicamente por el par emisor-receptor. En el caso de tener únicamente un emisor y un receptor, la clave secreta se comparte entre los dos y no hay mayor problema, empieza a haber un problema cuando hay un emisor y n receptores. El problema radica en cómo mantener la comunicación con los múltiples receptores de manera segura sin añadir la complejidad de manejar múltiples claves secretas, una para cada receptor, y que pueda haber fallos de seguridad en diferentes partes. El uso de múltiples claves secretas añade un problema de escalabilidad, y puede resultar poco práctico si tenemos cientos o miles de receptores. Normalmente la solución a este problema es gestionar la misma clave para todos los receptores. Ahora nos surge la duda de cómo transmitir la clave de manera segura (sin que pueda ser interceptada por terceros) a cada receptor. Para esto hay diferentes aproximaciones, como puede ser que un centro de distribución de claves KDC que genere y distribuya claves de sesión a través de un canal seguro, o el intercambio de claves por el protocolo criptográfico Diffie-Hellman, el cuál es un algoritmo matemático por el cual los miembros del grupo colaboran para crear una clave secreta compartida sin transmitirla directamente, asegurando así que solo los miembros del grupo puedan

tener la clave secreta. Cuando tenemos múltiples receptores y la clave secreta es la misma para todos los miembros del grupo, podemos tener problemas a la hora de hacer que un miembro del grupo deje de poder acceder a los datos, esto se suele resolver actualizando la clave y volviendo a encriptar el contenido, expandiendo o actualizando la clave, o creando claves que tengan un cierto tiempo de vida y que cuando pase ese tiempo ya no puedan descifrar aquellos datos.

El proceso por el cual se generan claves es básicamente crear bits aleatorios del tamaño necesario. Es muy importante que las claves secretas sean únicas, lo más aleatorias posibles (se sabe que los equipos tecnológicos no pueden generar valores completamente aleatorios, son pseudoaleatorios) y que este proceso de generación sea criptográficamente seguro. Cuanto mayor sea el tamaño de la clave más difícil será de descifrar por fuerza bruta. La complejidad de la clave es importante, así como del algoritmo utilizado, los algoritmos de cifrado suelen utilizar matemáticas avanzadas para realizar operaciones junto con la clave, consiguiendo así encriptar los datos. Las matemáticas utilizadas para encriptar datos utilizando la clave secreta son ciertas operaciones que tienen su matemática inversa, es decir, son operaciones reversibles, de tal manera que con la misma clave puedes encriptar los datos y luego realizar las operaciones inversas para, con la misma clave, descifrar los datos.

Las principales ventajas de la criptografía de clave simétrica son la eficiencia y simplicidad de la aproximación. Este tipo de criptografía es más rápida y requiere un menor esfuerzo computacional que su contraparte, la criptografía de clave asimétrica o criptografía de clave pública. El hecho de que sea más rápido y eficiente que su contraparte hace que este tipo de criptografía se utilice para grandes cantidades de datos, de hecho, los conocidos ransomwares (o malware de rescate) suelen utilizar criptografía híbrida [29], por una parte, utilizan criptografía de clave simétrica para encriptar múltiples archivos de manera rápida, y utilizan criptografía de clave asimétrica para guardar las claves con las que se encriptan los datos de la víctima (se expandirá más sobre esto en el siguiente apartado). Puesto que los algoritmos criptográficos de clave simétrica utilizan operaciones matemáticas avanzadas junto con la clave para derivar datos cifrados, estos pasos de los algoritmos pueden ser añadidos a los microprocesadores para que se realicen de manera rápida, esto es otra ventaja de este tipo de criptografía. Hoy en día, los microprocesadores de marcas como Intel o AMD (los nuevos procesadores M1 y M2 de Apple parecen tenerlo implementado a su manera, lo que hace que a veces funcione más lento que como funcionaría en Intel o AMD) tienen instrucciones a nivel de ensamblador [30] para realizar una ronda del algoritmo (por ejemplo, del cifrado por el algoritmo AES), realizar la ronda final, entre otros, tanto en la parte de encriptado como de descifrado. Este tipo de algoritmos han sido estudiados y, en la teoría, son seguros, aunque pueden volverse inseguros si hacemos una mala implementación del mismo.

Los inconvenientes y problemas más notorios de la criptografía de clave simétrica son el problema de cómo distribuir la clave secreta para que les llegue a los intermediarios destinados independientemente de si la enviamos por un canal de comunicación seguro o no, el hecho de que si la clave la consigue un tercero este podrá descifrar datos pasados y presentes, lo que da lugar a que con este tipo de autenticación el receptor no puede saber a ciencia cierta que el emisor que le envía el mensaje es verdadero o es alguien que se ha hecho con la clave secreta, además de que el tipo de cifrado funciona muy bien cuando hay un emisor y un receptor, pero en cuanto tenemos múltiples receptores el trabajo de gestión de claves y canales de comunicación seguros hace que sea una técnica con poca escalabilidad.

Las principales aplicaciones hoy en día de la criptografía de clave simétrica son para encriptar grandes cantidades de datos en diferentes áreas, desde una VPN para que los datos transmitidos desde el lado del cliente al servidor VPN estén seguros.

Los principales algoritmos de cifrado simétrico en el mercado actual son DES, 3DES, AES, Blowfish, IDEA o RC5. Se estudiarán algunos de estos algoritmos en detalle en siguientes apartados.

2.2.2 Cifrado Asimétrico

La criptografía de cifrado asimétrico o criptografía de clave pública [31], es un método de criptografía también utilizada en la actualidad. En este caso, el concepto principal de esta aproximación se fundamenta en que en vez de una tenemos dos claves, una clave pública y una clave privada. La clave pública sirve para encriptar los datos, y la clave privada sirve para descryptarlos, y este par de claves están relacionadas, de tal forma que una clave deshace lo que hace la otra.

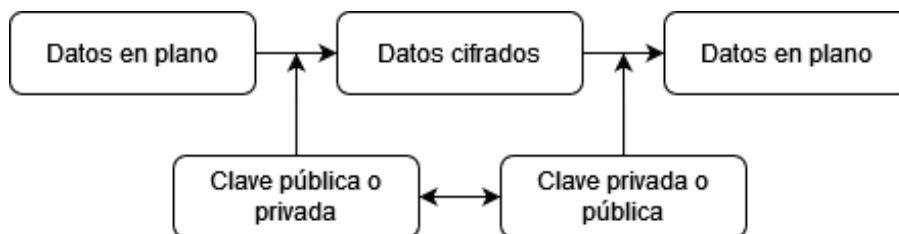


Figura 2.2: Diagrama de flujo de cifrado asimétrico.

La generación de las dos claves sigue un proceso diferente a la generación de la clave secreta en el caso de cifrado simétrico, ya que tenemos que crear dos claves, pero tienen que estar relacionadas de alguna manera, de tal forma que la clave pública sirva para encriptar datos, y la clave privada para descryptarlos. Esta relación entre claves suele depender del algoritmo, pero lo común es que estén relacionadas a través de algoritmos matemáticos complejos, y normalmente estos algoritmos implican operaciones con números primos (comúnmente multiplicación y factorización de los mismos, debido a las propiedades de estos tipos de números) y aritmética modular. Con esto conseguimos que una clave encripte o descrypte los datos, y únicamente la otra clave sea capaz de encriptar o descryptar esos datos, y las matemáticas de estas dos claves hacen que sea inviable que alguien obtenga la misma combinación de claves privada-pública o que a partir de una clave se pueda obtener la otra.

Una vez se han generado las dos claves, si el emisor simplemente quiere enviar un mensaje de manera segura al receptor, entonces puede generar un par de claves privada-pública, escoger cualquiera de ellas y encriptar los datos, esa será su clave pública y esta clave puede ser vista por cualquiera, ya que sólo se utiliza para encriptar los datos y no se puede hacer nada relacionado a descryptar los datos con ella. Cuando se terminen de encriptar los datos, se envían al receptor, el cual deberá haber escogido la otra clave del par que generaron él y el emisor, su clave privada, la cual deberá mantenerse en absoluto secreto y se asegurará de ser el único capaz de descryptar los datos.

Hay un punto que me gustaría resaltar de cara a establecer una comunicación segura entre emisor y receptor (y no simplemente como el caso anterior donde el emisor se comunicaba con el receptor, pero no era recíproco), imaginemos que el receptor encripta con su clave privada un mensaje y lo envía. Parece contraproducente ya que cualquiera tiene la clave pública y puede descryptar esos datos, lo cual tiene sentido, pero arroja un hecho importante, el hecho de que el receptor haya encriptado con su clave privada el mensaje y se haya descryptado con la clave pública implica que el mensaje es auténtico y el receptor es el que lo envía, ya que solo el receptor es el que tiene la clave privada. Con esto, podemos establecer un sistema en el que el emisor tiene un par de claves privada-pública (llamémoslo ePriv para la clave privada del emisor y ePub para la clave pública del receptor), el receptor tiene otro par de claves privada-pública (llamémoslo rPriv para la clave privada del receptor y rPub para la clave pública del receptor), y el emisor quiere enviar de manera segura el mensaje (llamémoslo msg) al receptor, de tal forma que sólo lo pueda leer el mismo. Como sabemos de la teoría, las claves públicas pueden ser conocidas por cualquiera, entonces el emisor conoce la clave pública del receptor, y el receptor conoce la clave pública del emisor. Lo que hará el emisor en caso de querer mandar un mensaje de forma segura al receptor es,

primero encriptar el mensaje con su clave privada (sólo la conoce el emisor) y luego, al resultado de ese cifrado, volver a encriptar esos datos cifrados con la clave pública del receptor (la cual es conocida por cualquiera). El receptor por su parte recibe estos datos cifrados dos veces, y el proceso para desencriptarlo sería desencriptar el mensaje primero con su clave privada (sólo la conoce el receptor) y luego desencriptar el resultado de esa desencriptación con la clave pública del emisor (la cual es conocida por cualquiera).

Con la exposición del anterior párrafo conseguimos múltiples aspectos [32], el primero de ellos es una comunicación segura entre emisor y receptor, debido a que ambos saben que los datos se encriptan o se desencriptan con las claves públicas (las cuales tiene cualquiera y no importa que la tengan, por eso son públicas) y las claves privadas (las cuales cada uno tiene la suya). El segundo aspecto es que la otra parte sabe que el mensaje recibido es auténtico ya que se ha encriptado con la clave privada de la primera parte, la cual sólo la tiene esa parte. También y, por último, se sabe que el mensaje no se puede modificar durante el camino ya que cualquier forma de interactuar con el mensaje requiere de las claves con las cuales se cifró y se descifra. Con esto tenemos un gran método por el cual tener una comunicación segura y sabiendo que hay autenticidad en los mensajes que se envían.

Las ventajas clave de esta aproximación de cifrado asimétrico son, como se acaba de explicar, el poder establecer una comunicación entre dos partes de manera segura y auténtica sin que terceros puedan fisgonear en la información. Este aspecto soluciona el problema que tiene la criptografía de clave simétrica cuando hay uno o más emisores y más de un receptor. La criptografía de clave asimétrica también soluciona el problema de escalabilidad de la criptografía de clave simétrica, si tenemos un grupo de personas a los que hay que enviar un archivo, basta con que tengan su par de claves privada-pública, no requiriendo así compartir claves privadas.

Los inconvenientes encontrados a la criptografía de clave asimétrica son el hecho de que es un tipo de criptografía computacionalmente exigente, las matemáticas que hay por detrás de este tipo de criptografía hace que sea bastante más lento el encriptado y desencriptado de archivos frente a criptografía de clave simétrica, por eso este tipo de criptografía se suele utilizar cuando la cantidad o tamaño de los archivos a encriptar es pequeña. Otro problema es que el tamaño de las claves de la criptografía asimétrica es mucho más grande que el tamaño de claves de la criptografía simétrica para proveer el mismo nivel de seguridad, por ejemplo, mientras que el algoritmo de cifrado simétrico AES es seguro con 128 bits de clave (aunque permite hasta 256 bits), el algoritmo de cifrado asimétrico RSA necesita claves con un tamaño mínimo de 1024 bits, y se suelen utilizar claves de 2048 bits. Un posible inconveniente es que el resultado de encriptar datos con criptografía asimétrica suele derivar en datos cifrados de mayor tamaño que si encriptases esos mismos datos utilizando criptografía de clave simétrica.

Las principales aplicaciones en la actualidad de la criptografía de clave asimétrica son el intercambio seguro de claves, las firmas digitales y la autenticación del usuario en diferentes áreas como correos electrónicos, transacciones bancarias, blockchain...entre otras. El intercambio seguro de claves permite que podamos utilizar un híbrido (lo veremos en el siguiente apartado) entre criptografía de clave asimétrica y clave simétrica, para obtener las ventajas de ambas partes. Con el ejemplo que se expuso anteriormente en este apartado podemos asegurar la autenticidad de los mensajes enviados y tener certeza de que quien nos envía mensajes es quien dice ser.

Los algoritmos más populares en el día de hoy que utilizan criptografía de clave asimétrica son Diffie-Hellman, DSA, RSA o ECC.

Con lo básico que nos quedamos de este y del anterior apartado es que tenemos por una parte un tipo de criptografía de clave simétrica que encripta y desencripta de manera rápida y segura, pero nos es complicado establecer un canal seguro por la cual el emisor pueda enviar de manera segura la clave secreta al receptor, y por otra parte la criptografía de clave asimétrica que encripta y desencripta de

manera más lenta que criptografía simétrica, pero permite establecer un canal seguro de comunicación y asegurar la autenticidad del mensaje y la identidad del emisor al receptor y viceversa. Si pudiéramos tener las bondades de ambos tipos de criptografía sería fantástico. Por suerte, podemos tener esto con la criptografía de tipo híbrido.

2.2.3 Cifrado Híbrido

La criptografía de tipo híbrida [33] es un tipo de criptografía que trata de combinar los mejores aspectos de la criptografía de clave simétrica y de la criptografía de clave asimétrica. Como se ha mencionado anteriormente, la criptografía de clave simétrica es muy eficiente al cifrar o descifrar datos, pero tiene el problema de necesitar establecer un canal seguro (encriptado, y para encriptar un canal necesitaremos otra clave) para que el emisor y los receptores puedan compartirse la clave; la criptografía de clave asimétrica es ineficiente para cifrar o descifrar archivos, pero es capaz de garantizar al emisor y al receptor un canal de comunicación seguro en donde ambos tienen certeza de la identidad del otro. La criptografía híbrida mezcla estos dos aspectos para obtener un tipo de criptografía segura y eficiente.

Para este caso trabajaremos con un archivo con gran tamaño (que no compense cifrarlo con criptografía asimétrica), de forma que tenga sentido utilizar criptografía híbrida, ya que ésta se suele utilizar o con archivos muy pesados o con muchos archivos. Tenemos un emisor que quiere enviarle un archivo de gran tamaño (pongamos 10GB, por poner un ejemplo) a un receptor. Lo primero que hará tanto el emisor como el receptor es generarse un par de claves privada-pública para cada uno. Como notación les pondremos ePriv y ePub al par de claves privada-pública del emisor, y rPriv y rPub al par de claves privada-pública del receptor. Luego de esto, el emisor se generará una clave para el cifrado simétrico, llamémosla cSim, con la cual encriptará su archivo de gran tamaño. Una vez hecho esto, cada parte tiene su par de claves privada-pública y, además, el emisor ha encriptado su archivo de gran tamaño con cSim. El orden de los siguientes dos pasos es un poco irrelevante, ya que es lo mismo que el emisor envíe la clave simétrica al receptor y luego le envíe los datos cifrados, a que lo haga a la inversa. El emisor procederá a encriptar la clave simétrica cSim primero con su clave privada ePriv, y el resultado de ese cifrado lo encriptará con la clave pública del receptor, rPub. Esto se lo mandará al receptor, y posteriormente enviará también el archivo cifrado con cSim al receptor. Con esto, el receptor tendría que descifrar el cifrado de la clave con su clave privada rPriv primero, luego descifrar ese resultado con la clave pública del emisor ePub, y con eso accedería a la clave simétrica cSim con la que podría descifrar el archivo cifrado.



Figura 2.3: Diagrama de flujo de cifrado híbrido.

Las ventajas de este tipo de criptografía son la eficiencia que provee al encriptar de manera rápida y la seguridad que ofrece al compartir los datos de manera segura, sin tener problemas a la hora de compartir claves y asegurando a ambas partes de que sus identidades son comprobables, ya que solamente el emisor o el receptor pueden encriptar utilizando sus claves privadas, solo la tienen ellos.

Los inconvenientes de la criptografía híbrida son que puede llegar a ser un proceso computacionalmente exigente, además de que tiene el mismo inconveniente que los anteriores dos tipos de criptografía, depende de que, en este caso, las claves privadas se mantengan privadas y únicamente conocidas por los dueños, de lo contrario puede haber suplantaciones de identidad y robos de información.

Las principales aplicaciones que utilizan este tipo de criptografía son HTTPS, SSL o TLS mensajería instantánea o correos electrónicos, transacciones bancarias, entre otras.

Aprovecho este apartado para hablar de una aplicación de la criptografía híbrida que aprovecha sus ventajas a la perfección, se trata de los malwares de rescate, también conocidos como ransomware. Para esta explicación me quería centrar en el ransomware Wannacryptor, también conocido como Wannacry [34], aunque los ransomwares en general tienden a actuar en este aspecto de la misma manera.

Un ransomware es un software malicioso que suele introducirse en el dispositivo objetivo a través de archivos o páginas web infectadas, o de versiones gratuitas de software de pago que contienen algún programa detrás para validar la instalación. El objetivo de este software malicioso es encriptar todos los datos que la víctima pueda considerar importantes (principalmente deduce esto a partir de las extensiones de los archivos), de tal manera que el atacante pueda pedir un rescate por los archivos (habitualmente en forma de criptomonedas, y concretamente en Bitcoin) por el cual si no se paga ese rescate la víctima no obtendrá la clave que desencripta sus archivos (tampoco está asegurado que si envías las criptomonedas se te provea de la clave). Estos ataques suelen estar dirigidos más hacia empresas, ya que son las que más posibilidades y urgencia tienen de pagar el rescate. En el caso de Wannacry, se distribuía a través de la red utilizando la vulnerabilidad Eternal Blue de Windows, que en su momento era una vulnerabilidad zero-day, lo cual quiere decir que Windows no sabía de ella (aunque la NSA sí que sabía de ella) y no estaba tratando de solucionarla, y pedía 300 dólares americanos en Bitcoin para descifrar los datos.

Debido a que Wannacry necesita encriptar una gran cantidad de archivos de manera rápida (para que no se entere el usuario o el antivirus básicamente) pero al mismo tiempo no puede dejar que se descubra la clave secreta, se utiliza criptografía híbrida, y la aproximación que tomó Wannacry para encriptar los datos es ciertamente inteligente e ingeniosa [35]. Los creadores del ransomware Wannacry crearon múltiples pares de claves privada-pública (estas claves y futuras que introduciremos suelen ser claves RSA, aunque hay otras opciones) al inicio y guardaron estas claves (al menos la parte privada de las mismas) en un servidor de comando y control que en la práctica no es localizable. Un servidor de comando y control es un servidor que controla los malware, por ejemplo, en una botnet (conjunto de dispositivos infectados y controlados remotamente por un atacante) este servidor sería el encargado de decirle a todos los dispositivos infectados que lancen peticiones a una dirección IP concreta. Entonces, este servidor tiene una clave pública y una clave privada (llamémoslas sPriv para la parte privada y sPub para la parte pública). Una vez se extrae el ransomware en el dispositivo víctima, genera un par de claves privado-públicas (llamémoslas vPriv para la parte privada y vPub para la parte pública) para esa víctima en concreto (ya que, si fuese el mismo par de claves para todas las víctimas, con que una de ellas pagase el rescate y se le diese la clave se podrían desencriptar todos los archivos de todas las víctimas), y por cada archivo que considere importante para la víctima (según las extensiones de los archivos) generará una clave simétrica (del algoritmo AES principalmente, hay pocas opciones más) para cada archivo que tenga que encriptar. El proceso sería, para cada archivo a encriptar, generar una clave de AES (llamémosla cSim_n para la clave simétrica que encripta el archivo de orden n) y sustituir el archivo en plano por el archivo cifrado, añadiéndole la clave cSim_n encriptada, de tal manera que luego se pudiese desencriptar si se ha pagado el rescate. Esa clave cSim_n se encripta con la clave pública de la víctima vPub, y la clave privada de la víctima vPriv ha sido encriptada con la clave pública sPub del servidor de comando y control. De esta manera, tenemos encriptados muy rápidos y con la criptografía asimétrica podemos garantizar que solo el atacante puede desencriptar los archivos (salvo que haya fallas en la implementación

de todo este proceso). El problema para la víctima es que todos sus archivos han sido encriptados con múltiples claves simétricas AES, claves simétricas las cuales han sido encriptadas por la clave pública de la víctima, clave pública cuya parte privada ha sido encriptada por la clave pública del servidor, cuya parte privada no es localizable y está oculta a nosotros. La víctima necesitaría la clave privada de la víctima vPriv para así desencriptar el cifrado realizado por la clave pública de la víctima vPub y así obtener la clave simétrica cSim_n con la cual desencriptar su archivo n, el problema de esto es que esa clave privada de la víctima vPriv ha sido encriptada con la clave pública del servidor sPub, por lo cual necesitamos la clave privada del servidor sPriv para desencriptarla, clave privada que es prácticamente imposible de resolver. En el caso de que paguemos el rescate, en la práctica se enviaría la clave privada de la víctima vPriv al servidor de comando y control, y este nos retornaría la clave privada de la víctima vPriv en plano, de tal manera que con eso ya podemos desencriptar las claves simétricas cSim_n que descifran nuestros archivos.

Wannacry al final fue contrarrestado con los parches de Windows sobre la vulnerabilidad Eternal Blue, y con el hecho de que Wannacry tenía la forma de no expandirse implementada en su código, así que el investigador Marcus Hutchins consiguió parar la propagación del virus [36], dando tiempo a otros investigadores para que descubriesen que las librerías que utilizaban los creadores de Wannacry (librerías de encriptado en Windows), dado ciertas condiciones como que el proceso de Wannacry siguiese vivo y que el ordenador no se hubiese reiniciado desde su infección, podían no eliminar los números que generaban las claves privadas y públicas de la víctima. Por tanto, teniendo esto en cuenta se publicaron herramientas como WannaKey, las cuales recuperaban esos números, componían las claves de vuelta y desencriptaban los archivos. Como vemos, un fallo en la implementación del algoritmo mitigó bastante los efectos de Wannacry.

Considero que este caso es un ejemplo interesante e instructivo para entender la utilidad y conveniencia de la criptografía híbrida, aprovechando las ventajas de la criptografía simétrica y criptografía asimétrica, y espero que haya servido para tal propósito.

2.3 Tipos de cifrado en criptografía según los algoritmos

En este apartado se describirán los dos tipos de criptografía según la forma en la que operan los algoritmos de cifrado o descifrado, se trata de la criptografía de cifrado en flujo, y la criptografía de cifrado en bloque.

2.3.1 Cifrado por flujo

El cifrado por flujo [37] es un tipo de cifrado para criptografías de claves simétricas. La característica fundamental de este cifrado es que el encriptado y desencriptado de archivos sucede de 1 bit o byte en 1 bit o byte. Este tipo de cifrado trabaja con los datos planos de manera continua para obtener datos cifrados, así que para aspectos que funcionen en tiempo real o de ejecución puede ser de gran utilidad.

Para realizar un cifrado en flujo partimos de una secuencia de entrada de bits de tamaño n que queremos encriptar, la llamaremos input. Este input se combinará con un flujo de datos pseudoaleatorio, que también debe tener tamaño n , llamado flujo de claves. El resultado de combinar los bits uno a uno del input con el flujo de claves es el dato cifrado. Este tipo de cifrado tiene ciertas similitudes y es una aproximación a la libreta de un solo uso (OTP) [38], la cual es un cifrado que, dándose ciertas condiciones, está probado que es indescifrable. La libreta de un solo uso es un tipo de cifrado por el cual, teniendo unos datos en plano que queremos cifrar, los combinamos con una clave o libreta aleatoria de igual tamaño y de únicamente un uso. Se ha demostrado (en 1949 [39] que el método, de tener una

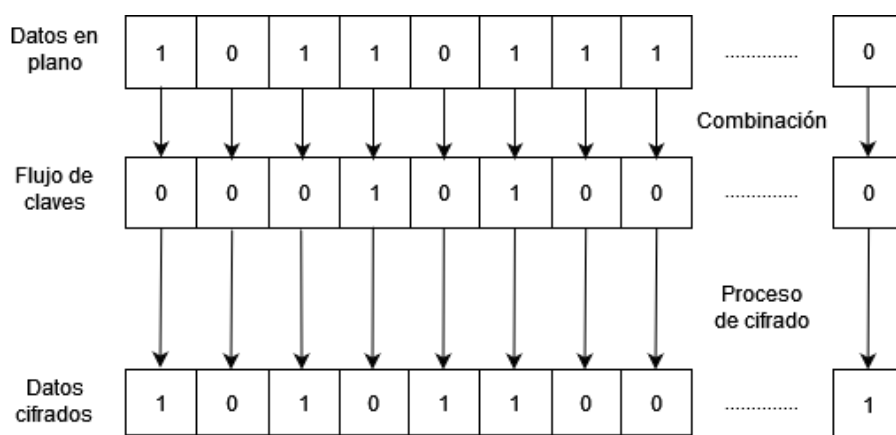


Figura 2.4: Diagrama de flujo de cifrado por flujo (de bits).

clave verdaderamente aleatoria (no funciona con claves pseudoaleatorias, ya que se podría descubrir el algoritmo de generación y la semilla y saber la combinación utilizada) y de un solo uso, el algoritmo no se puede romper. A este algoritmo se le denomina libreta ya que en sus inicios los elementos que componían la clave se distribuían en una libreta, pudiéndose quemar después de usarse. Un algoritmo de este tipo es el cifrado de Vernam-Mauborgne, el cual tomaba esta aproximación, aunque la clave no era del todo aleatoria.

Las preguntas que nos surgen con este tipo de cifrado en flujo son principalmente dos. La primera de ellas es cómo se combinan los bits de los datos de entrada junto con los bits del flujo de claves, y la segunda es cómo se generan los bits del flujo de claves. La forma en la que se combinan el input con el flujo de claves suele ser, para cada bit, realizar la operación XOR. Esta operación a nivel de bits es bastante sencilla y muy rápida de implementar en hardware, la operación retornará uno o positivo si los dos bits a comparar son diferentes entre sí, y devolverá cero o negativo en caso contrario, en el caso de que los bits a comparar tengan el mismo valor.

La forma en la que se generan los bits del flujo de claves depende del algoritmo utilizado, deben ser de manera aleatoria para que sea seguro, sin embargo, los dispositivos electrónicos no pueden crear valores aleatorios, se hace a través de un algoritmo (por lo cual ya no es aleatorio, ya que un algoritmo es una manera determinista de reproducir algo, y lo aleatorio no es determinista) y un valor que indica el estado inicial por el cual empezar, ese valor se denomina semilla. Al no poder utilizar claves verdaderamente aleatorias, debemos intentar que nuestros algoritmos de generación de valores pseudoaleatorios tengan un periodo lo más grande posible. Si el periodo es pequeño, el algoritmo empezará a repetir los valores iniciales demasiado pronto, lo cual hace que el tipo de cifrado no sea seguro.

Las ventajas de este tipo de cifrado son su alto grado de paralelización, lo cual ayuda a que sean implementados en hardware (la combinación por XOR es implementable de manera muy sencilla), dando lugar a encriptados y desencriptados de muy alta velocidad. Este tipo de cifrados necesita poco tamaño de memoria para ser implementados, y son eficientes para encriptar datos en tiempo real.

Las desventajas de este tipo de cifrado son el problema de cómo transmitir la clave del cifrado al receptor de manera segura y la complejidad de implementar este tipo de cifrados de manera completamente segura, debido a que se necesitan claves completamente aleatorias para asegurar que son cifrados indescifrables.

Las principales aplicaciones de este tipo de algoritmos son las conexiones inalámbricas, dispositivos del Internet de las Cosas IoT, VPNs y aplicaciones que utilicen aspectos en tiempo real.

Los algoritmos más utilizados relacionados con este tipo de cifrado son Salsa20 y ChaCha20, o RC4, aunque a este último se le han encontrado vulnerabilidades que han hecho que su utilización disminuya, pero no desaparezca por temas de compatibilidad.

2.3.2 Cifrado por bloques

El cifrado por bloques [40] es un tipo de cifrado de clave simétrica muy utilizado en la actualidad. Para este caso, la idea fundamental es que tanto el encriptado como desencriptado se realiza en bloques de bits, a diferencia del cifrado en flujo que se realiza de un bit en un bit. Esta forma de cifrado obtiene iterativamente bloques de un tamaño fijo y los va encriptando o desencriptando, la complejidad de esto es menor que la complejidad del cifrado por flujo, pero el cifrado en bloque suele ser más lento que el cifrado en flujo.

El tamaño de estos bloques es fijo, y al procesarlos para su encriptado, el resultado son datos cifrados del mismo tamaño que el bloque utilizado. La mayoría de los algoritmos de cifrado en bloque son iterativos, esto quiere decir que se transforman bloques de tamaño fijo de datos en plano en bloques de tamaño fijo de datos cifrados mediante la repetición de una función reversible conocida como la función de rondas (round function), siendo cada repetición de la función llamada una ronda. Normalmente en cada ronda se utilizan diferentes claves, y esas claves se derivan de la clave original. También se suele modificar los datos con funciones sencillas como XOR y partes de la clave, tanto al principio como al final de la encriptación. En los algoritmos actuales los bloques iterados pasan por un proceso de sustitución y permutación de los bits cifrados, añadiendo las propiedades de confusión y difusión (a nivel de bit, para que se pueda realizar el proceso de desencriptado) que Claude Shannon identificó [41] como necesarias en un proceso de cifrado para que métodos como el criptoanálisis no funcionen.

Hay múltiples maneras de operar con los bloques, primero hay que trocear los datos en bloques del tamaño fijado para los mismos. En el caso más sencillo, el modo ECB, trocearíamos el mensaje en bloques y los encriptaríamos de manera separada e independiente, sin embargo, esta aproximación no es buena ya que patrones iguales en los datos en plano, al encriptarlos con la misma clave, se verían idénticos en los datos cifrados. Para resolver este problema tenemos definidos varios modos de operación para el cifrado en bloques que siguen recomendaciones de entidades como NIST o estándares ISO. La aproximación general consiste en añadir un componente aleatorio a los datos en plano, normalmente se realiza a través de una entrada adicional llamada vector de inicialización, el cual debe ser aleatorio, aunque también sirve que sea pseudoaleatorio y único. Con este vector de inicialización conseguimos que, si encriptamos los mismos datos con la misma clave, la gran mayoría de veces obtendremos diferentes datos cifrados.

Teniendo ya claro el concepto de vector de inicialización podemos empezar a describir ciertos modos de operación. En el modo CBC, el cual es muy popular, se pasa un vector de inicialización aleatorio (a veces puede ser simplemente un nonce, es decir, un número utilizado una única vez) junto con el primer bloque de datos en plano a través de una combinación XOR, y esta combinación es la que se procede a encriptar. El resultado de esta encriptación es un bloque de datos cifrado que después se utiliza como vector de inicialización para el siguiente bloque de datos en plano, emulando así una operación en cadena. En el modo CFB, se encripta inicialmente el vector de inicialización y se añade al bloque de datos planos a encriptar, y ese resultado se pasa como vector de inicialización al siguiente bloque de datos en plano. En el modo OFB, se encripta inicialmente el vector de inicialización, este vector encriptado se pasa al siguiente paso con un nuevo bloque de datos en plano y se vuelve a encriptar, al volverse a encriptar se pasa al siguiente paso donde se vuelve a encriptar, y así sucesivamente. Por último, en el modo CTR se generan vectores de inicialización que tienen que cumplir que sean únicos (no hace falta que además de únicos sean pseudo o aleatorios de verdad), y este vector de inicialización se combina con un contador

para encriptarse, obteniendo la aleatoriedad necesaria. Todos estos modos de operación (excepto el modo ECB) aseguran la seguridad semántica, es decir, que si recibes datos cifrados sin conocer la clave con la que se cifraron, la única información que puedes derivar es el tamaño del mensaje (y esto es debido a la naturaleza del cifrado en bloque y sus características).

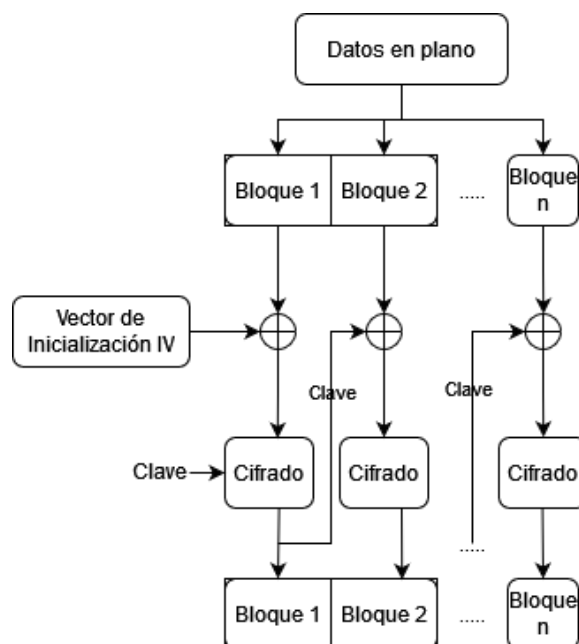


Figura 2.5: Diagrama de flujo de cifrado por bloques con modo de operación CBC.

Uno de los aspectos importantes del cifrado por bloques es qué sucede si el tamaño de mi entrada no llega al tamaño fijo definido para el bloque. El tamaño de bloques en este tipo de cifrado es un múltiplo de 32 bits, pero es posible que, por ejemplo, después de haber procesado múltiples bloques, el tamaño del bloque final sea de (un ejemplo) 27 bits. En este caso lo que se hace es rellenar el último bloque con bits para que llegue al tamaño fijado del bloque, pero hay múltiples maneras de rellenar esos bits faltantes. La manera utilizada es añadir un bit con valor uno y luego tantos bits con valor cero como sean necesarios para rellenar el tamaño requerido. Este método se conoce como Padding Method 2 y está definido en el ISO/IEC 9797-1, y es utilizado debido a que se ha demostrado seguro frente a ataques de tipo “padding oracle attacks”, los cuales son ataques que se aprovechan de que el relleno o padding es añadido a los datos en plano antes de la encriptación para hacer preguntas (como si se tratase de un oráculo) al sistema a la hora de descryptar datos y así ir deduciendo el tamaño del relleno, desde donde se aplicó, ganando así y poco a poco información sobre los datos en plano.

Las ventajas del cifrado por bloques es que la seguridad que ofrecen frente a diferentes ataques y la eficiencia que nos brinda al encriptar datos de gran tamaño. Algunos tipos de este cifrado permiten la paralelización de la encriptación de los diferentes bloques de los datos en plano, lo cual aumenta también la velocidad de encriptado y descryptado. Algunos algoritmos que implementan este tipo de cifrado son ampliamente utilizados hoy en día por agencias de seguridad de grandes países, este hecho nos provee cierta tranquilidad de que utilizamos cifrados que, si se implementan correctamente en la práctica, son eficientes y seguros.

Las principales desventajas de este tipo de cifrado son el hecho de que haya que añadir datos aleatorios (como el vector de inicialización) y su consecuente gestión para que se mantengan lo más aleatorios posibles, de cara a evitar que una misma entrada de datos en plano, utilizando la misma clave, provea la misma salida de datos cifrados, dando pistas a los atacantes. Además, el hecho de que el tamaño

de bloques sea fijo supone problemas, ya que esto deriva en casos en los que no se llega al tamaño de bloques, teniendo así que utilizar relleno o padding para ese bloque, pero con ese relleno podemos introducir vulnerabilidades que revelen información sobre los datos en plano.

Las principales aplicaciones de este tipo de algoritmos son los encriptados o desencriptados de grandes cantidades de datos, por ejemplo, vimos el caso del ransomware Wannacry, el cual encriptada múltiples archivos de un equipo.

Los algoritmos más utilizados relacionados con este tipo de cifrado son DES, 3DES/TDES, AES, el cual es ampliamente utilizado, además de RC5, Blowfish o IDEA.

2.4 Otros algoritmos y funciones criptográficas

En este apartado se evaluarán otro tipo de funciones criptográficas que serán de utilidad de cara a implementar el caso práctico del gestor de contraseñas. Se trata de las funciones criptográficas hash, las funciones de derivación de claves KDF, y los algoritmos criptográficamente seguros de generación de números aleatorios.

2.4.1 Funciones Criptográficas Hash

Una función criptográfica hash [42] es un algoritmo de cifrado muy utilizado en la actualidad. Funciona tal que, dada una entrada de datos en texto plano se obtiene una salida de datos cifrados de tamaño fijo. Dependiendo del tipo de función hash, el tamaño del resultante hash será distinto, y se puede decir que el hash utilizado para largas cadenas de caracteres o archivos grandes es como un “resumen” de los datos de entrada.

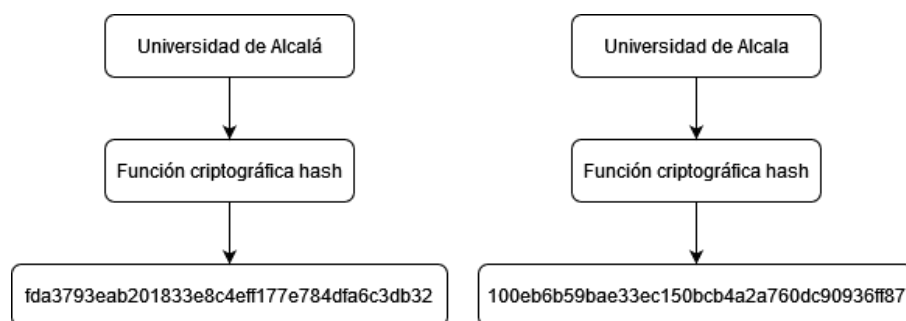


Figura 2.6: Resultado de función SHA1 para dos cadenas de caracteres similares.

Hay ciertas propiedades o requisitos que deben tener estas funciones hash para que nos sean útiles criptográficamente hablando. La primera de ellas es que sea sencillo calcular el valor hash de una entrada y que, por el contrario, sea (idealmente) inviable calcular la entrada dado un resultado hash. Esto tiene sentido debido a que las funciones criptográficas hash están pensadas para que sean de un único sentido, es decir, puedas obtener el valor hash de una entrada, pero no puedas obtener la entrada dado el hash. Las funciones criptográficas hash tienen como requisito ser lo suficientemente rápidas como para obtener el valor hash de una entrada de pequeño (una cadena de caracteres) o gran (un vídeo de dos horas) tamaño de manera muy rápida, pero al mismo tiempo no tiene que ser tan rápido, ya que si lo es entonces problemas como las colisiones se pueden crear de manera más rápida, esto se verá más adelante. Las funciones criptográficas hash también tienen que cumplir que, si cambio lo más mínimo el valor de la entrada, aunque cambie solo un bit, el resultado hash debe ser completamente diferente. Este efecto se llama “el efecto avalancha”, está tanto para funciones criptográficas hash como para cifrados por bloques,

y las funciones hash deberían mostrar este efecto de manera notable debido a que así se demuestra un alto grado de aleatoriedad en los resultados de la función, asegurando así que un criptoanalista no pueda hacer pruebas y predecir la forma en la que funciona el algoritmo. Otro aspecto que deben tener los algoritmos hash es que tienen que ser deterministas, es decir, para una misma entrada de datos deberemos devolver la misma salida de datos. En función de si queremos que la función criptográfica hash esté diseñada para comprobar la integridad de un mensaje o la autenticación del origen del mismo, podremos definir otro tipo de requisitos adicionales.

Un aspecto de las funciones criptográficas hash es que deberían ser resistentes a las colisiones. Las colisiones hash suceden cuando dos entradas dan como salida el mismo resultado hash. Las funciones hash deben ser lo más resistentes a las colisiones que sea posible, sería muy difícil crear una función hash completamente resistente a colisiones, esto se explica por el Principio del Palomar [43]. El Principio del Palomar indica que si hay n palomas que quieren posarse en m palomares, en el caso de que n sea estrictamente superior a m (es decir, si hay más palomas que palomares disponibles) entonces con un 100 % de certeza habrá al menos un palomar que tenga más de una paloma. Esto se relaciona con nuestras funciones hash de la siguiente manera [44], la cantidad de cadenas de caracteres y archivos que podemos crear es inmensa, debido a que las funciones hash devuelven siempre un hash de un tamaño fijo, debemos asegurarnos de que el tamaño de ese hash resultante permita tantas combinaciones distintas como posibles combinaciones podamos crear de cadenas de caracteres o archivos para que no haya colisiones. Esto, como se presupone, es ciertamente complicado, y, de hecho, hay funciones hash muy utilizadas hace años que hoy en día se considera que están rotas, ya que se le han encontrado múltiples colisiones, y el hecho de encontrar colisiones permite analizar esos casos y conseguir que nosotros creemos colisiones artificiales, es decir, dado un hash de un archivo de nombre a , podemos crear un archivo b (posiblemente malicioso o, simplemente, modificado) que devuelva el mismo hash que el archivo a . Esto llevaría cierto tiempo, ya que hay que analizar los casos y ver cómo debe ser la secuencia de bits del archivo b para conseguir el mismo hash que el archivo a , pero teóricamente sería posible. En definitiva, una función criptográfica hash que tenga más valores de entrada posibles que valores de salida inevitablemente tendrá colisiones, cuanta más dificultad haya para encontrar o generar esas colisiones, más seguridad provee la función criptográfica hash.

Una pregunta importante a estas alturas y que ayuda a explicar cómo funcionan las funciones criptográficas hash es cómo es posible que una entrada que podría ser un carácter o un número, al pasarlo por una función criptográfica hash, retorna un valor hash del mismo tamaño que si hubiese tenido como entrada de datos un archivo de gran tamaño. Las funciones hash actuales funcionan por bloques, como lo hacen los algoritmos de cifrado por bloques, e internamente se componen básicamente de un bucle que iterará sobre cada bloque obtenido de la entrada y realizará operaciones sobre algunas variables internas que tiene la función hash. El resultado hash será el valor que tengan estas variables internas, de ahí que no importe el tamaño de la entrada ya que el tamaño de la salida siempre será el mismo, y es dependiente del algoritmo utilizado. El hecho de que las funciones criptográficas hash funcionen con bloques nos trae de vuelta a un problema que tienen ese tipo de cifrados, y es qué pasa cuando el tamaño de la entrada no es exactamente igual al tamaño del bloque, lo cual pasa casi siempre. En el cifrado por bloques poníamos como solución añadir un relleno o padding al bloque que no llegase al tamaño fijado por el bloque, en este caso haremos eso también, ya que en caso de tener un valor de entrada que sobrepase el tamaño del bloque, simplemente dividiremos esa entrada en bloques, operaremos con ellos y para el bloque final tendremos (o no) el caso de que no llegue al tamaño fijo del bloque. Dependiendo de la función criptográfica hash con la que estemos trabajando se adoptará una u otra manera de rellenar el último bloque.

Las funciones hash más utilizadas hoy en día parten de un mensaje que dividen en bloques de tamaño fijo y de unas variables internas que representarán en conjunto a nuestro resultado hash una vez hayamos

procesado todo el mensaje. Dentro de la función hash tendremos una función de compresión, la cual es el núcleo del algoritmo y donde radica toda su eficiencia y su seguridad, y dependiendo del algoritmo se implementa de una u otra forma. En esta función de compresión se tomarán los valores de las variables internas y se mezclarán con un trozo de mensaje para producir unos nuevos valores de las variables internas. En el caso de que ese fuese el último bloque a procesar, nuestras variables internas contienen el resultante hash que deberemos devolver como retorno, si por el contrario queda más mensaje que analizar, se volverá a realizar el mismo ciclo con el nuevo valor de las variables internas y el nuevo trozo del mensaje. Cabe destacar que esta estructura de funcionamiento responde a la construcción Merkle–Damgård, la cual es un método de construcción de funciones criptográficas hash resistentes a colisiones a través de la construcción de funciones de compresión de un único sentido y resistentes a colisiones. Este tipo de construcción asegura que si el método por el cual rellenar bloques cuyo tamaño no llega al tamaño de bloque fijado es seguro y la función de compresión es resistente a colisiones, entonces la función criptográfica hash es resistente a colisiones.

Algo importante que se debe resaltar es que, viendo las propiedades de las funciones criptográficas hash podríamos pensar que son un gran método para guardar contraseñas. En vez de que una empresa guarde las contraseñas en una base de datos en texto plano (teniendo el riesgo de que un cibercriminal pueda colarse en la base de datos y exportar todos los datos fuera del control de la empresa) o cifrar las contraseñas con una clave (con el riesgo de que un cibercriminal pueda averiguar dónde se utiliza la clave secreta y haciéndose con ella y con una copia de las contraseñas), podemos guardar en la base de datos el resultado hash de las contraseñas que utilice el usuario. Esta aproximación no basta para asegurar las contraseñas debido a que cae en un error sencillo, y es que el resultado hash de dos entradas iguales es el mismo, así que se vería quiénes en la base de datos utilizan la misma contraseña, de tal forma que si un cibercriminal consigue descifrar el texto plano detrás de un hash puede haber comprometido múltiples cuentas en vez de solo una.

Este problema anteriormente mencionado se soluciona con el uso de sal criptográfica [45]. La sal criptográfica son simplemente valores aleatorios que se juntan con la contraseña antes de obtener su resultado hash. Por cada contraseña generamos una sal lo suficientemente larga [46] como para que haya múltiples combinaciones posibles, y calculamos el valor hash de contraseña+sal. Con esto, si dos personas utilizan la misma contraseña (cosa bastante probable), debido a que la sal se utilizó es diferente (ya que el usuario no tiene por qué enterarse que se está utilizando una sal junto con su contraseña), los resultados hash serán radicalmente diferentes. La sal se suele guardar en texto plano [47] junto con el hash que genera la combinación contraseña+sal, de tal manera que cuando el usuario inserte su contraseña se pueda saber qué sal aleatoria se utilizó y autenticar al usuario, parece contraintuitivo añadir un elemento de aleatoriedad a las contraseñas y luego guardarlo en texto plano, ciertamente el único problema del atacante será saber dónde se pone la sal en relación a la contraseña para seguir realizando ataques de diccionario o de fuerza bruta, sin embargo los ataques por tabla arcoiris (enseguida veremos qué es esto) serán inútiles, ya que aunque el atacante sepa la sal utilizada para cada contraseña, eso solo le servirá para saber una contraseña, y no podrá descifrar múltiples contraseñas de una vez. En resumen, con la sal prevenimos que dos contraseñas iguales tengan el mismo resultado hash, y conseguimos que un cibercriminal no descubra todas las contraseñas que hay detrás de sus respectivos hashes pasándolas por una tabla arcoiris.

Existe un tercer posible elemento además del texto del que queremos obtener su hash y la sal criptográfica, esta es la pimienta criptográfica [48]. Las agencias de seguridad suelen llamar a este concepto “sal secreta” en vez de pimienta, enseguida veremos el por qué, en este documento se le llamará pimienta criptográfica por simplicidad. La pimienta criptográfica es otro elemento que le añadimos a la entrada la cual queremos obtener su hash, es similar a la sal criptográfica, pero guarda una diferencia importante,

mientras que la sal criptográfica puede guardarse, la pimienta criptográfica no se guarda, es secreta. Esto hace que cuando un usuario introduce su contraseña y procedemos a añadirle la sal criptográfica para averiguar su hash, tendremos que además comprobar todos los valores posibles de la pimienta criptográfica para ver si la credencial introducida es correcta. Un ejemplo de posible pimienta criptográfica podrían ser dos números juntos (desde 00 hasta 99). Esto nos da 100 posibilidades de pimienta criptográfica y, por tanto, para verificar las credenciales del usuario deberíamos calcular en el peor de los casos 100 resultados hash, nos tomaría 100 veces más tiempo para realizar esa verificación. Puede parecer un inconveniente, pero calcular un hash es rápido, es uno de sus requisitos, suelen calcularse en menos de un segundo, entonces el impacto es aceptable, y añade seguridad debido a que al atacante también se le multiplica el tiempo para descubrir la contraseña. Es importante que dado uno o pocos valores de la pimienta criptográfica no se pueda derivar el resto de posibles valores de esta, ya que entonces pierde su efecto.

La principal ventaja de los algoritmos criptográficos hash es que sirven como una manera rápida de calcular un equivalente ofuscado de nuestro mensaje, lo cual nos puede venir bien para ciertas aplicaciones que veremos.

Las desventajas de este tipo de algoritmo son sus vulnerabilidades debido a las posibles colisiones y los ataques de fuerza bruta, los cuales son posibles debido a la rapidez del algoritmo. Existe un aspecto a tener en cuenta (como “vulnerabilidad”) también respecto a estas funciones criptográficas hash, son conocidas como tablas arcoiris o Rainbow Tables. En principio, la única forma en la que debería poder averiguar el texto plano que generó un hash es probar todas las combinaciones para obtener su resultado hash y comprobar si es igual al hash que tenemos. Las Rainbow Tables son tablas, generalmente de bases de datos, donde se guardan el texto plano y el valor hash resultante de un algoritmo de millones y millones de combinaciones de caracteres. De esta forma, por ejemplo, si un atacante ha conseguido el valor hash de una contraseña, puede comprobar en esta tabla si ese hash ha sido calculado previamente, lo cual es puede ser más rápido que ponerse a calcular valores hash para descifrar el texto plano asociado.

Las principales aplicaciones de este tipo de funciones hash son para comprobar la integridad de los datos, o para compartir un valor o resultado con el receptor y que el receptor al calcular ese resultado o ese valor sepa que es el mismo que el del emisor. Las funciones hash se utilizan para guardar contraseñas, pero necesitan de un aspecto que se comentará más adelante para que el guardado sea seguro.

Las funciones criptográficas hash más utilizadas en la actualidad son MD5 (aunque ya se considera roto), la familia de algoritmos SHA (compuesta por SHA1, SHA2 y SHA3), BLAKE2 o Whirlpool.

2.4.2 Funciones de derivación de claves KDF

Las funciones de derivación de claves (o key derivation functions, KDF) son funciones ciertamente utilizadas en la actualidad. Estos algoritmos [49] consisten en derivar una o más claves secretas a partir de una entrada de datos, la cual podría tratarse de una clave maestra o simplemente una contraseña, utilizando una función pseudoaleatoria. Esta función pseudoaleatoria suele requerir una función criptográfica hash o un algoritmo de cifrado por bloques. Estas funciones de derivación de claves se suelen utilizar para, dado una entrada de texto sencilla, obtener una clave secreta criptográficamente fuerte para utilizarla normalmente en un algoritmo de cifrado simétrico.

Al igual que las funciones criptográficas hash, estas funciones de derivación de claves deben tener ciertas propiedades. La mayoría de las propiedades requeridas por estas funciones de derivación de claves son compartidas con las funciones hash. Necesitamos que nuestras funciones de derivación de claves sean rápidas de calcular para una entrada de datos, pero sea inviable calcular el texto plano que generó esa

clave secreta, también necesitamos que las funciones sean rápidas de calcular, pero lo suficientemente lentas como para que sean criptográficamente seguras, por el mismo problema que tenían las funciones criptográficas hash, si son demasiado rápidas entonces son susceptibles de ataques de fuerza bruta con el objetivo de crear colisiones (múltiples entradas de datos que produzcan la misma salida de datos) artificiales, examinarlas y poder deducir datos del algoritmo. Se necesita, por tanto, que las funciones de derivación de claves sean resistentes a colisiones en la mayor medida de lo posible. Necesitamos también que las funciones de derivación de claves tengan el “efecto avalancha”, por el cual si cambiamos lo más mínimo la entrada de datos entonces tendremos una salida de datos de aspecto totalmente diferente.

Las funciones de derivación de claves suelen tener como parámetros la clave original de la que se quiere derivar una clave segura, una sal criptográfica que vimos en el anterior apartado de qué se trataba, un tamaño (normalmente en bytes) de clave segura que queremos derivar, y un parámetro llamado iteraciones o coste. Todos los parámetros excepto el último están claros, este último parámetro se refiere al número de iteraciones (o de veces) que se va a ejecutar este algoritmo, que, en esencia, es un bucle. Si elevamos este coste o este número de iteraciones, nos costará más tiempo derivar una clave dado un valor de entrada, pero al atacante que quiera revertir esto y obtener el valor de entrada dada la clave secreta le costará mucho más tiempo, ciertamente estaremos haciendo más lento el algoritmo de manera totalmente deliberada, pero ese tiempo extra es algo que podemos asumir y que al atacante al menos le hará pensárselo dos veces antes de intentar el proceso de inversión. Sin duda la forma de derivar claves depende del algoritmo del que estemos hablando, en general lo que tienen estas funciones de derivación de claves por dentro es una función pseudoaleatoria del tipo PRF. Una familia de funciones pseudoaleatorias PRF [50] son una serie de algoritmos eficientes que tratan de emular salidas aleatorias para las entradas dadas, es decir, esta familia se caracteriza por que no se pueda distinguir entre una función de esta familia escogida al azar y una función en la que la salida se fija aleatoriamente para cualquier valor. Todas las salidas de las funciones parecen aleatorias, es diferente a los generadores pseudoaleatorios (veremos algo similar más adelante) en los que una única salida parece aleatoria si la entrada fue aleatoria. Lo que se hace es poner la contraseña junto con la sal criptográfica en una de estas funciones pseudoaleatorias y repetir este proceso tantas veces como iteraciones se hayan especificado. Con esto conseguimos una clave criptográficamente segura del tamaño especificado. La función pseudoaleatoria que se utilizan en múltiples funciones de derivación de claves suele ser alguna de la familia de funciones hash SHA.

Las ventajas de utilizar funciones de derivación de claves son que las claves utilizadas, por ejemplo, en cifrados de clave simétrica son más seguras por el hecho de que con estas funciones podemos derivar claves seguras de contraseñas no tan seguras. Recordemos que la diferencia entre contraseña y clave es que la clave sirve para cifrar o descifrar algoritmos criptográficos, y la contraseña está más orientada al usuario para que se verifique o con ella generar claves criptográficas. Además, estas funciones permiten parametrizarse y configurarse para añadir más o menos coste o iteraciones a la creación de claves, así que viene bien para ir aumentando el coste a medida que la tecnología aumenta su velocidad de cómputo. Estas funciones ofrecen resistencia ante diferentes tipos de ataques, y con el uso de sal criptográfica permite contrarrestar los ataques por tabla de arcoiris o Rainbow Tables.

La principal desventaja de este tipo de funciones es el hecho de tener que hacer más lento un proceso de cómputo deliberadamente, ya que esto añade coste computacional a los cálculos realizados y el rendimiento se ve afectado. Sin embargo, lo que hacemos es compensar, aumentamos la seguridad al utilizar una función de derivación de claves y poder crear claves criptográficas más seguras y resistentes a diferentes ataques, pero disminuimos la velocidad de cálculo, aumentando el tiempo para obtener resultados. Como podemos ver, en este caso es una cuestión de equilibrio entre los dos aspectos para calcular claves seguras en poco tiempo.

Las principales aplicaciones de este tipo de algoritmos son en gestores de contraseñas, en donde dada la contraseña maestra del usuario necesitamos derivar una clave criptográficamente segura para encriptar sus datos. En general se suele utilizar en toda parte de la criptografía que necesite derivar claves seguras o de un tamaño específico a partir de una entrada de datos.

Las funciones de derivación de claves más populares a día de hoy (centradas en el ámbito de las contraseñas) son PBKDF2, Argon2, scrypt o bcrypt, entre otros.

2.4.3 Generadores pseudoaleatorios criptográficamente seguros

Un generador pseudoaleatorio criptográficamente seguro [51] es un generador pseudoaleatorio con propiedades que hacen que sea utilizable en criptografía. Estos algoritmos se utilizan principalmente para generar números aleatorios, esto es normalmente requerido en criptografía, pero la definición de aleatorio es diferente para cada caso. Los dispositivos no pueden generar valores aleatorios como tal, generan valores a partir de un algoritmo, lo cual lo hace determinista ya que si conseguimos ese algoritmo generaremos los mismos valores, por tanto, se trata más bien de valores pseudoaleatorios. En caso de ser pseudoaleatorios, deberemos tratar de asegurarnos de que el periodo del generador, es decir, el número de valores aleatorios que puede generar antes de que comience a repetir esos valores, sea lo más largo posible (como dato, el periodo de una secuencia aleatoria es infinito).

El concepto de aleatoriedad y qué tan aleatorio es un valor depende lo que necesitemos, en caso de crear un número que utilizaremos una única vez (en inglés se denomina nonce), dependiendo de para qué lo utilicemos quizá nos basta con que sea único. Sin embargo, para generación de claves criptográficamente seguras necesitaremos alta aleatoriedad y que los valores no comiencen a repetirse pronto. Esto se relaciona con el concepto criptográfico de entropía [52], es la medida de la cantidad de aleatoriedad o desorden de una variable, o la cantidad de incertidumbre a la que se enfrenta un atacante cuando intenta saber un secreto criptográfico, se mide habitualmente en bits. Debemos tener en cuenta que hay algunos algoritmos criptográficos que confían en ciertos aspectos o propiedades sean aleatorios, en caso de no ser aleatorios no garantizan su seguridad.

La idea de este tipo de algoritmos es obtener una fuente de gran entropía y buena calidad, con esa entropía podríamos generar buenos datos aleatorios, el problema viene de dónde obtenemos esa fuente de entropía. Normalmente los sistemas operativos tienen métodos para obtener esta entropía. En el caso del sistema operativo Windows OS, puede obtener entropía a través el chip TPM, el cual es un estándar internacional para procesadores criptográficos, también puede obtener entropía a través del reloj hardware del sistema, a través de los tiempos en los que hubo interrupciones de procesos, o a través de los tiempos en los que se pulsan teclas del teclado o se mueve el ratón. Este último caso por ejemplo lo utiliza el gestor de contraseñas KeePass. Recordemos que en KeePass además de nuestra contraseña maestra podíamos designar un archivo de claves con el cual cifrar nuestra base de datos de contraseñas, una opción que nos ofrece KeePass para crear este archivo es mover el ratón de manera lo más aleatoria posible dentro de un cuadrado en la aplicación, con el objetivo de generar bits aleatorios que se añadan a tu contraseña maestra y así cifrar tu base de datos de contraseñas.

Es necesario que este tipo de algoritmos es necesario que pasen ciertas pruebas estadísticas de aleatoriedad y que se mantengan resistentes a ataques. Cada generador de números aleatorios criptográficamente seguros debe cumplir que, dado los primeros n bits de una secuencia aleatoria generada con el generador, no pueda crearse un algoritmo que en tiempo polinomial (es decir, que el tiempo durante el cual este el algoritmo funcionando para una entrada de datos de tamaño n está relacionado con una potencia o polinomio de la complejidad de la entrada de datos) pueda predecir el siguiente bit de la secuencia aleatoria con una probabilidad de éxito de más del cincuenta por ciento. Por otra parte, este tipo de algoritmos

deberían resistir técnicas por las cuales inferir los bits pasados o futuros dados unos bits comprometidos, es decir, si el estado ha sido revelado debe ser imposible reconstruir los bits aleatorios anteriores o posteriores a ese estado. Este suele ser el caso por los que los generadores de números pseudoaleatorios no son criptográficamente seguros, cuando su estado se descubre podemos saber los bits anteriores al estado revelado.

La ventaja clave de utilizar generadores de números pseudoaleatorios criptográficamente seguros es tener esa seguridad o certeza de que son valores con un alto grado de aleatoriedad y que son resistentes a ataques en los que se descubriese la secuencia o forma utilizada para generar estos valores.

Una desventaja de este tipo de funciones es que son más lentas, por el hecho de que tienen que ser criptográficamente seguras, que un generador de números aleatorios convencional. Como tal, para este apartado no hay ventajas o desventajas, es simplemente decidir si necesitas añadir seguridad o no a los valores aleatorios que necesitas generar.

La principal aplicación de estos generadores es la de producir valores aleatorios con seguridad en la manera de hacerlos, esto es aplicable principalmente a campos criptográficos de generación de sal criptográfica, números de un uso o nonces...entre otros.

Algunos algoritmos de generación de números pseudoaleatorios criptográficamente seguros son Fortuna, el algoritmo de Yarrow o la API de Windows CryptGenRandom.

2.5 Ejemplos de los anteriores tipos de criptografía

En este apartado se procederá a redactar acerca de ejemplos actuales de algoritmos o de funciones que utilizan las técnicas o características que se han expuesto anteriormente, como cifrado simétrico o asimétrico, procesamiento en flujo o en bloques, u otro tipo de funciones criptográficas. Se tratará de introducir el algoritmo o función, su funcionamiento, ventajas e inconvenientes de su utilización frente a otros algoritmos similares, y campos o sectores en donde el algoritmo es utilizado.

2.5.1 3DES

El algoritmo DES [53] es un algoritmo de cifrado de clave simétrica el cual fue ampliamente utilizado por todo el mundo desde que fuese escogido como un estándar FIPS (Federal Information Processing Standard, estándares publicados por el gobierno estadounidense, en el caso de DES es el FIPS 46). IBM inició la creación de este algoritmo en la década de 1970, y tras hacer múltiples consultas con la NSA y el NIST, se procedió a escoger una versión reforzada de la propuesta para evitar ataques de criptoanálisis, la cual se publicó como FIPS en 1977. El hecho de que el algoritmo tuviese algunas características dudosas de justificar (enseguida veremos las características del algoritmo) y que la NSA estuviese detrás hizo que se sospechase de algún mecanismo para romper este cifrado que solo la NSA supiese.

Este algoritmo tiene ciertas características, su clave en total tiene 64 bits, 56 bits de esa clave es la clave en sí misma utilizada en el algoritmo y los 8 bits restantes son utilizados para comprobar la paridad (es una medida sencilla de comprobación de posibles errores) y luego no se usan más. DES es un algoritmo de cifrado por bloques, y el tamaño de bloques especificado es de 64 bits, lo cual en la época de creación de DES podría ser un buen tamaño, pero ahora se ha quedado pequeño. El proceso de DES es iterativo para cada bloque, y se explicará a continuación. DES por sí mismo no es un medio seguro de encriptación, sino que, como vimos anteriormente para los cifrados por bloques, deberemos utilizar algún modo de operación para prevenir los problemas que tienen este tipo de cifrados.

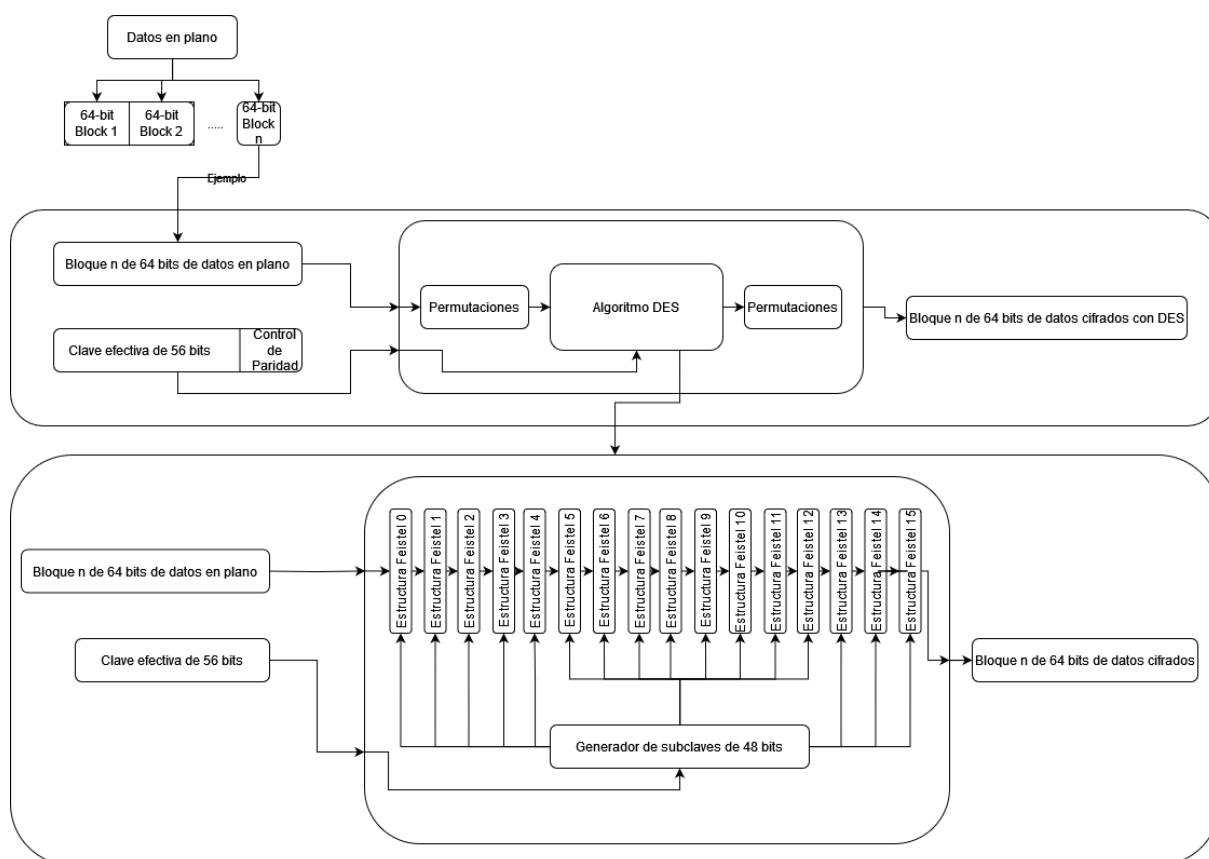


Figura 2.7: Diagrama general del funcionamiento de DES.

Al inicio del proceso tenemos bloques de datos en plano de 64 bits y la clave efectiva de 56 bits, ya hemos realizado comprobaciones con los últimos 8 bits de la misma. Lo primero que haremos será introducir el bloque de 64 bits de datos en plano a través de un módulo de permutaciones que, a nivel criptográfico, no tienen mayor relevancia debido a que lo que se hace es cambiar los bits de orden según unas directrices definidas en el algoritmo. Cabe destacar que en este paso la clave todavía no ha intervenido para nada. A continuación, lo que haremos será dividir el bloque en dos partes iguales a ser posible, de no ser así aproximaremos los tamaños y trabajaremos con mitades desiguales. Lo que tenemos delante son 16 rondas donde se realizará el mismo proceso, el motivo de que sean 16 rondas parece no haber trascendido, pero puede ser debido a que a partir de 16 rondas a los criptoanalistas se les complica significativamente realizar sus ataques. Utilizaremos la siguiente nomenclatura, si queremos referirnos a la mitad izquierda del bloque en la primera ronda lo llamaremos R1L, para la mitad derecha se llamará R1D, en el caso de la segunda ronda serían R2L y R2D respectivamente. Cabe resaltar que hay una parte del algoritmo en donde dada la clave de 56 bits, derivamos 16 claves, una por cada ronda, de 48 bits cada una, la clave utilizada en la primera ronda se llamará K1, y así sucesivamente. El algoritmo para derivar las claves se llama Key Schedule, y consiste básicamente en dividir la clave de 56 bits en dos mitades de 28 bits cada una y realizarle a cada mitad operaciones como rotaciones de bits, de tal manera que para DES derivamos el número de subclaves de 48 bits necesarias. En cada ronda lo que se hace es, por ejemplo, para la primera ronda y teniendo ambas mitades R1L y R1D, la mitad derecha R1D la pasaremos por el bloque F, el cual es un bloque que se compondrá de una función pseudoaleatoria para obtener una salida ofuscada de esa mitad derecha $F(R1D)$ utilizando la clave asociada K1 para esa ronda, y ese resultado $F(R1D)$ lo haremos una combinación XOR con la mitad izquierda R1L. Estos cálculos se pasan como entrada a la siguiente ronda, es decir, para la siguiente ronda tendremos como valor de la mitad izquierda la anterior mitad derecha, es decir, $R2L = R1D$, y la mitad derecha de la segunda ronda será esa combinación XOR

realizada, es decir, $R2D = R1L \text{ XOR } F(R1D, K1)$. Con estos nuevos valores se vuelve a realizar el mismo proceso utilizando una nueva clave $K2$, de esta manera, los valores iniciales para la tercera ronda serán $R3L = R1L \text{ XOR } F(R1D, K1)$ para la mitad izquierda, y $R3D = R2L \text{ XOR } F(R1L \text{ XOR } F(R1D, K1)), K2)$, o lo que es lo mismo, $R3L = R2D$ para la mitad izquierda, y $R3D = R2L \text{ XOR } F(R2D, K2)$. Este proceso se repite por 16 rondas y luego las mitades se intercambian de lado, y cuando tengamos los 64 bits resultantes de juntar las dos mitades de la última ronda tendremos ya nuestros datos cifrados.

Una pregunta que se nos viene a la cabeza cuando vemos esto es qué es F y que realiza exactamente. La función F en el caso de DES es una función que opera con la mitad que le mandemos y la clave de 48 bits derivada de la clave principal de 56 bits, y lo hace a través de cuatro pasos, los cuales son expansión, mezclado con la clave, sustitución y permutación. En la fase de expansión se expande el bloque de 32 bits a 48 bits, se realiza duplicando la mitad de los bits según defina el algoritmo. La salida de esto son 8 piezas de 6 bits, además de una copia del bit adyacente de cada pieza de entrada a cada lado. En la fase de mezclado de claves, el resultado anterior se combina con la subclave de 48 bits utilizando XOR. En la fase de sustitución, cada una de las 8 piezas de 6 bits se pasan por cajas S , las cuales son cajas de sustitución que indican cómo sustituir unos valores de bits por otros que parecen aleatorios, pero son equivalentes en la práctica, ya que para esta sustitución se utiliza una tabla de catálogo o tabla lookup. Estas sustituciones son importantes ya que de lo contrario el algoritmo sería rompible de manera más sencilla. De cada 6 bits de la pieza que entran en alguna de las 8 cajas de sustitución que hay salen 4 bits, de tal manera que tenemos una salida del tamaño de la entrada inicial, de 32 bits. En la fase final de permutación, los 32 bits de salida se reorganizan de acuerdo a una caja P , es decir, una caja de permutación. Con estas últimas fases conseguimos la confusión y difusión que Claude Shannon en la década de 1940 estableció como condición necesaria para que un cifrado fuese seguro y utilizable en la realidad.

Esta forma de operar consistente en dividir la entrada en dos mitades y realizar operaciones sobre cada mitad de manera independiente para luego juntarse en un siguiente paso y que se vuelva a realizar el mismo proceso, forma de operar utilizada para derivar subclaves y en general en el algoritmo DES, corresponde con la estructura o esquema de Feistel, realizado por el físico alemán Horst Feistel. El cifrado de Feistel [54] o esquema de Feistel, es un método de cifrado en bloques con una estructura particular, lo que hace que multitud algoritmos de cifrado por bloques lo utilicen. Se compone de rondas, y básicamente tendremos un bloque de texto en plano que se dividirá en mitades, tendremos la función de Feistel F y una clave K diferente para cada ronda realizada, y se llevará a cabo la función F con una de las mitades y la clave K de esa ronda, luego esa cadena obtenida se intercambia de posición con la mitad que no ha sufrido operaciones para la siguiente ronda y se sigue haciendo esa operación. La función de Feistel F puede ser una función pseudoaleatoria, una función criptográfica hash, pero es requerido que sea fuerte criptográficamente ya que de esa función es donde radica la fuerza y resistencia frente a ataques de los algoritmos que implementen esta estructura, por esto es que el cifrado de Feistel es más una estructura o una plantilla donde se elige la función de Feistel que un cifrado por bloques en sí. El motivo por el cual se utiliza esta plantilla en el caso de DES es que es fácil, o al menos no muy complejo, implementar esta estructura Feistel utilizada en DES en hardware. Esto es debido a que este tipo de estructuras utilizan el mismo algoritmo para encriptar que para desencriptar, lo cual hace que en hardware sea rápido de implementar. En el caso de DES, para desencriptar este algoritmo tenemos que generar las mismas subclaves de 48 bits que generamos anteriormente y utilizarlas en orden inverso (desde $K16$ hasta $K1$), junto con la función de Feistel F para DES, la cual se ha descrito anteriormente, y el texto cifrado, el cual se dividirá en dos partes y comenzarán las rondas.

Las ventajas de DES es que es ciertamente rápido a la hora de encriptar o desencriptar datos, fue un estándar durante unos cuantos años hasta que fue sustituido por el algoritmo que veremos en el siguiente

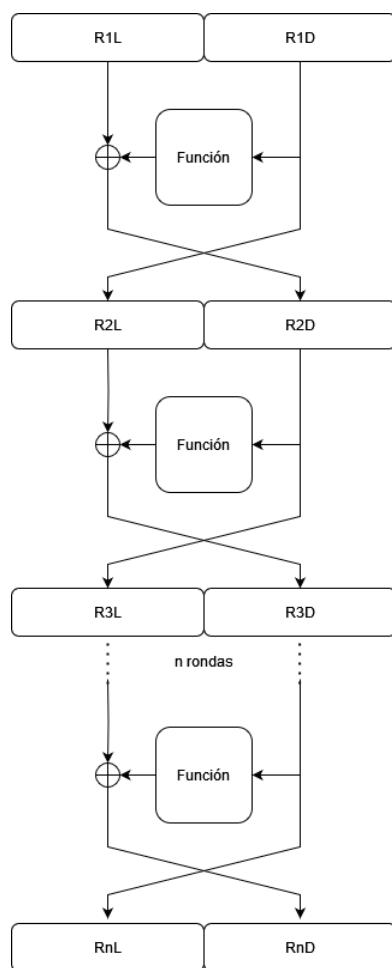


Figura 2.8: Esquema de red o Cifrado Feistel.

apartado, y dio lugar en la época a múltiples estudios acerca de algoritmos de cifrado por bloques y simétricos (ya que la comunidad sospechaba de la NSA y de que pudiese haber alguna puerta trasera en DES), mejorando así el conocimiento y obteniendo algoritmos más avanzados.

Los inconvenientes de DES es que utiliza una clave de poco tamaño, 56 bits, lo cual dan 2^{56} combinaciones de claves. DES es susceptible a diferentes ataques de criptoanálisis, y el hecho de que opere con claves pequeñas hace que a día de hoy sea un algoritmo que está roto. Ya en Junio de 1977 Whitfield Diffie y Martin Hellman argumentaron que la clave era pequeña y que es susceptible de ataques de fuerza bruta, y efectivamente, en menos de 24 horas se puede averiguar el mensaje oculto tras un texto cifrado con una clave de 56 bits y DES, lo cual lo hace inservible. DES fue quitado como estándar por el NIST y se recomienda utilizar el algoritmo que veremos en el siguiente apartado.

Los campos en los que se utiliza DES son en cualquier campo que necesite encriptado de datos, ya sea de gran tamaño o tamaño normal. Si se sigue utilizando debe ser por dificultades para moverse a otro algoritmo o problemas de compatibilidad, debido a que se ha demostrado que es inseguro y que no protege los datos de forma duradera.

Si el lector ha sido atento, se habrá dado cuenta de que únicamente he hablado de DES, pero este apartado está dedicado a 3DES. 3DES o TDES hace referencia a Triple DES [55], es decir, un algoritmo de cifrado por bloques y de clave simétrica que aplica el algoritmo de cifrado DES que hemos visto con detalle anteriormente tres veces a cada bloque de datos. Cuando se empezó a hablar de que DES no era seguro ya que podía ser susceptible de ataques de fuerza bruta, una de las aproximaciones que salieron fue

esta. Básicamente 3DES lo que hace es realizar para cada bloque de datos tres rondas de encriptado por DES, la clave por tanto es alargada es de tres veces 56 bits, es decir 168 bits. La contraparte de encriptar múltiples veces datos con DES es que, efectivamente, el algoritmo es múltiples veces más lento, aunque este intento de balancear tiempo con seguridad se ha explicado previamente. 3DES es ciertamente más seguro que DES, sin embargo, se recomienda pasar al algoritmo que veremos en el siguiente apartado, el cual es más rápido que 3DES y no tiene la vulnerabilidad que tiene DES.

2.5.2 AES

AES [56] es un estándar sobre un algoritmo de cifrado por bloques y de clave simétrica llamado Rijndael, el cual fue desarrollado por dos criptógrafos belgas, Vincent Rijmen and Joan Daemen. Este algoritmo es adoptado en la actualidad como estándar por el NIST y el FIPS, y este algoritmo se utiliza prácticamente para todo lo que necesite de un cifrado de clave simétrica. En Noviembre de 2001 se anunció AES después de un proceso de estandarización del algoritmo de un lustro, en donde diferentes personas trataron de romper el algoritmo, se vio que era seguro, y se establecieron diferentes normas o estándares para el algoritmo.

A finales de la década de los 90 el NIST veía que se avecinaban problemas con DES, ya hubo varios rumores de que la clave de DES no tenían suficiente tamaño y que son susceptibles de ataques de fuerza bruta con un esfuerzo asumible. DES fue desarrollado por IBM y publicado como estándar por la NSA y diferentes agencias, por ejemplo, el NIST. Para contrarrestar estos problemas, además de la aproximación de utilizar 3DES, el NIST decidió crear un concurso abierto, literalmente cualquiera podía entrar en el concurso y proponer un algoritmo, en el cual escoger un nuevo algoritmo que sería el nuevo estándar de encriptación AES. Había ciertas condiciones para el algoritmo propuesto, básicamente el NIST quería un algoritmo de código abierto, es decir, disponible para todo el mundo, ser un algoritmo de cifrado por bloques y de cifrado simétrico, y cuyo tamaño de bloques sea de 128 bits. El NIST también quería que las claves soportadas por el algoritmo sean de múltiples tamaños y comenzando por 128 bits hasta 256 bits, que el algoritmo fuese implementable de manera rápida tanto en software como en hardware y que, en definitiva, tuviese como mínimo la seguridad que proveía 3DES pero que fuese significativamente más rápido que este. Un año después se seleccionaron 15 algoritmos en la primera convocatoria, y a estos se le harían ciertos análisis. Estos análisis tendrían múltiples aspectos en cuenta, no únicamente que el algoritmo fuese indescifrable, también se tenía en cuenta que se pudiese implementar en múltiples plataformas de manera sencilla y que fuese suficientemente rápido en plataformas con no muchos recursos, que fuese un algoritmo eficiente en términos de CPU y de memoria y no dejase un rastro que pudiese ser analizado...entre otros aspectos. Pasó otro año, ya nos encontramos en mediados de 1999, y en segunda convocatoria quedaron los 5 finalistas, los cuales fueron MARS (desarrollado por IBM), RC6 (desarrollado por RSA Labs), Rijndael (desarrollado por los dos criptógrafos belgas anteriormente mencionados), Serpent (desarrollado por Ross Anderson, entre otros) y Twofish (desarrollado por Bruce Schneier, entre otros). Estos algoritmos se podían analizar de manera abierta, y a mediados del año 2000 se produjo la tercera conferencia donde se discutieron los resultados de los análisis realizados y, con esos análisis, el NIST realizó una votación para elegir al ganador. De 212 votos totales, la medalla de bronce fue para Twofish con 31 votos, la medalla de plata fue para Serpent con 59 votos y el ganador y medalla de oro fue para Rijndael con unos 86 votos. Este algoritmo sería el nuevo estándar AES y a finales del año 2001 se publicó en el FIPS y de manera oficial como nuevo estándar.

Debemos remarcar que AES no es tal cual el algoritmo de Rijndael, AES es un estándar, es decir, son ciertas normas que se aplican sobre el algoritmo de Rijndael. AES define que el tamaño de bloque es de 128 bits, el algoritmo de Rijndael permite tamaños de bloque mayores, normalmente múltiplos de 32

bits comenzando desde 128 bits. Las claves que soporta AES son de 128, 192 y 256 bits, mientras que el algoritmo de Rijndael permite tamaños de clave intermedios y de mayor tamaño, como por ejemplo 160 o 512 bits. El algoritmo de Rijndael es iterativo, debido básicamente a que se trata de un cifrado por bloques, el número de rondas en AES está definido en 10 para claves de 128 bits, 12 rondas para claves de 192 bits y 14 rondas para claves de 256 bits. Mientras, el número de rondas para el algoritmo de Rijndael varía según el tamaño de clave utilizado, que puede ser distinto al estándar que define AES. A pesar de estas diferencias, los nombres se siguen intercambiando para referirse al algoritmo de encriptación, en este documento se llamará AES al algoritmo Rijndael que reúne las condiciones especificadas por el AES.

El funcionamiento de AES es conocido, se trata de un algoritmo de cifrado por bloques (por tanto, es iterativo) y clave simétrica, es decir, se utiliza la misma clave tanto para encriptar datos como para desencriptarlos. El tamaño de bloque es fijo y son 128 bits, es decir, 16 bytes, un byte es un grupo de 8 bits. La estructura de AES no sigue la estructura de su predecesor, DES. Recordemos que DES seguía una estructura Feistel en donde el bloque se dividía en dos mitades, se hacían operaciones con la mitad derecha y luego se combinaba ese resultado con la mitad izquierda y se cambiaban de lado de cara a la siguiente ronda. La estructura de AES y, en verdad, del algoritmo de Rijndael, es una estructura de sustitución-permutación, o estructura SP. Esta estructura básicamente hace uso de operaciones matemáticas relacionadas para realizar sustituciones y permutaciones alternativas entre los bits y, dado bits de entrada en plano, obtener bits de datos cifrados. En estas rondas se utilizan subclaves derivadas de la clave original para que el cifrado dependa de la clave utilizada. Teniendo un bloque de 128 bits de texto en plano y una clave de 128 bits para este ejemplo, sabemos que tendremos que realizar 10 rondas. Lo primero que hacemos es derivar una subclave K_0 de también 128 bits utilizando el algoritmo de Key Schedule que vimos en el apartado sobre 3DES, después de esto realizaremos una combinación XOR entre el bloque de 128 bits de datos en plano y los 128 bits de la subclave derivada K_0 . El resultado de esto pasará por 10 rondas de AES junto con otra clave (para la primera ronda será K_1 hasta K_n , siendo n el número de rondas a realizar), imaginémonos esas 10 rondas como 10 bloques que hacen lo mismo. Dentro de cada bloque lo que hay son cuatro fases, que también se pueden ver como bloques que realizan operaciones. La primera fase es de sustitución, y para esto los 16 bytes resultantes de la anterior operación se colocan en forma de matriz de 4 filas y 4 columnas, en donde la primera columna empieza con los bytes 0 desde arriba hasta el byte 3, la segunda columna contiene los bytes 4 a 7, la tercera columna contiene los bytes 8 a 11 y la última columna tiene los bytes 12 a 15. Cabe mencionar que las versiones del algoritmo de Rijndael con mayor tamaño de bloque añaden más columnas. Una vez tenemos esta matriz de bytes, en la fase de sustitución sustituiremos cada byte por otro equivalente según una tabla. Esta función de sustitución hace que ningún valor sea sustituido por sí mismo y que ningún valor se sustituya por ese mismo valor cambiando los unos por ceros y ceros por unos, además de que debe ser una función que devuelva resultados no lineales. Estas cajas S de sustitución son un punto fuerte por el cual el algoritmo es fuerte, y además es rápido y fácil este paso debido a que es mirar en una tabla de catálogo o tabla de lookup. En la siguiente fase tenemos nuestra matriz con los datos sustituidos y lo que haremos será mover las filas de la matriz según su orden, es decir, la primera fila de la matriz no se desplaza, la segunda fila de la matriz se desplaza una posición a la izquierda (es decir, b_1 pasa a estar en la posición de b_{13} , b_5 en la posición en la que estaba b_1 ...cada byte se desplaza a la columna anterior, y el de la primera columna se desplaza a la última columna, manteniéndose todos en la misma fila), la tercera fila de la matriz se desplaza dos posiciones a la izquierda y la cuarta fila se desplaza tres posiciones a la izquierda. En el siguiente paso tomaremos el resultado anterior y mezclaremos en este caso las columnas, utilizando una matriz fija para multiplicarlo. Multiplicaremos una matriz de 4 filas por 4 columnas por cada columna y obtendremos una matriz de 4 filas por 1 columna. Debemos tener en cuenta un aspecto, y es que la mayoría de las operaciones de AES se realizan en un cuerpo

finito [57], también llamado campo de Galois (por el joven matemático Évariste Galois). Este aspecto entra de lleno en matemáticas y en concreto en teoría de grupos, AES utiliza un campo de Galois de 2^8 elementos, es decir $\text{GF}(2^8)$, el motivo es que 2^8 elementos son 256 elementos, que son los valores binarios posibles de un byte. La sustitución realizada en fases anteriores también utiliza este campo o cuerpo de Galois para cumplir las condiciones que le hemos puesto también previamente. En este cuerpo de Galois que utilizaremos tenemos un cierto número de elementos (256), y ciertas operaciones que podemos realizarle a estos elementos como sumar, restar, multiplicar o dividir. Una característica de este cuerpo es que cualquier operación de las nombradas que le hagamos a elementos de ese cuerpo de Galois nos resultará en otro elemento que estará dentro de ese cuerpo de Galois, es decir y para nuestro caso, que no hay desbordamiento ni pasamos a números negativos por ejemplo, lo cual nos viene bien para que las operaciones que hagamos no aumenten el número de bits con el que trabajamos. Normalmente a la hora de multiplicar columnas hacemos multiplicaciones y sumas de los valores, en nuestro caso haremos la multiplicación por una matriz fija de tamaño 4 filas por 4 columnas, solo que al estar operando sobre este cuerpo de Galois tendremos que las sumas son operaciones XOR y que las multiplicaciones se harán en modulo un polinomio irreducible concreto de grado 8 (concretamente $x^8 + x^4 + x^3 + x + 1$) en el campo de Galois $\text{GF}(2^8)$. En el campo de Galois $\text{GF}(2^8)$ tenemos que un número como puede ser 1000 0011 en verdad es un polinomio $x^7 + x^1 + 1$, de tal manera que al sumar sumamos esos polinomios (modulo dos ya que estamos en binario) y la multiplicación es el producto modulo ese polinomio. Estas operaciones son confusas, tiene sentido ya que con esto hacemos que los atacantes no puedan saber lo que hemos encriptado, pero son operaciones que tienen ciertas formas de que en hardware se implementen de manera sencilla (puerta lógica XOR y desplazamientos de bits), haciendo estas operaciones muy rápidas. Después de haber realizado la fase de mezcla de columnas, añadiremos (haremos XOR) de la subclave que estemos utilizando en esta ronda con el resultado anterior, y el resultado de esta adición es el resultado de la primera ronda de AES, nos quedarían nueve rondas con estos mismos pasos (en la última ronda no realizamos la fase de mezclado de columnas ya que no nos sirve de nada). Con la ronda de sustitución y las dos rondas de mezclado tenemos la confusión y difusión que Claude Shannon estableció necesarias para que un algoritmo de cifrado sea seguro y práctico, y todas las operaciones realizadas tienen una operación inversa tan sencilla de implementar como la operación que invierten, de tal manera que lo único que necesitas para descifrar es la clave. La teoría de este algoritmo es sólida, se puede fallar en la implementación del mismo en la práctica, dando lugar a brechas de partes de la clave, pero en caso de estar implementado correctamente el atacante lo tiene sumamente difícil para romper este algoritmo.

Como hemos podido ver en la implementación de este algoritmo, las principales de este algoritmo son su seguridad, debido a que AES es un estándar desde 2001 y ha sido probado múltiples veces en busca de fisuras, y su eficiencia, debido a que es posible implementarlo y que funcione en múltiples dispositivos de distintas características computacionales y que funcione de manera rápida. De hecho, los procesadores Intel y AMD del mercado (y también los procesadores Apple M1 y M2, aunque hay algunas diferencias) implementan instrucciones para ejecutar una ronda o la ronda final de AES, esto hace que sea muy rápido, se dice que encripta un archivo de 1 gigabyte en alrededor de medio segundo, y que encripta con una velocidad del orden de gigabits por segundo. Las versiones más pequeñas en cuanto a tamaños de AES son seguras a día de hoy, por tanto, AES nos permite tener cierto margen de que si empiezan a no ser seguras por la creciente velocidad de los dispositivos tecnológicos podemos aumentar esos tamaños y ganar seguridad en ese aspecto sin perder mucho tiempo de cómputo. AES no tiene ataques conocidos hasta 2005, y se considera que una implementación correcta de la teoría relativa a AES no tiene ataques prácticos, los ataques a AES están más centrados en ataques laterales, es decir, atacar más la implementación del mismo que la teoría detrás del algoritmo.

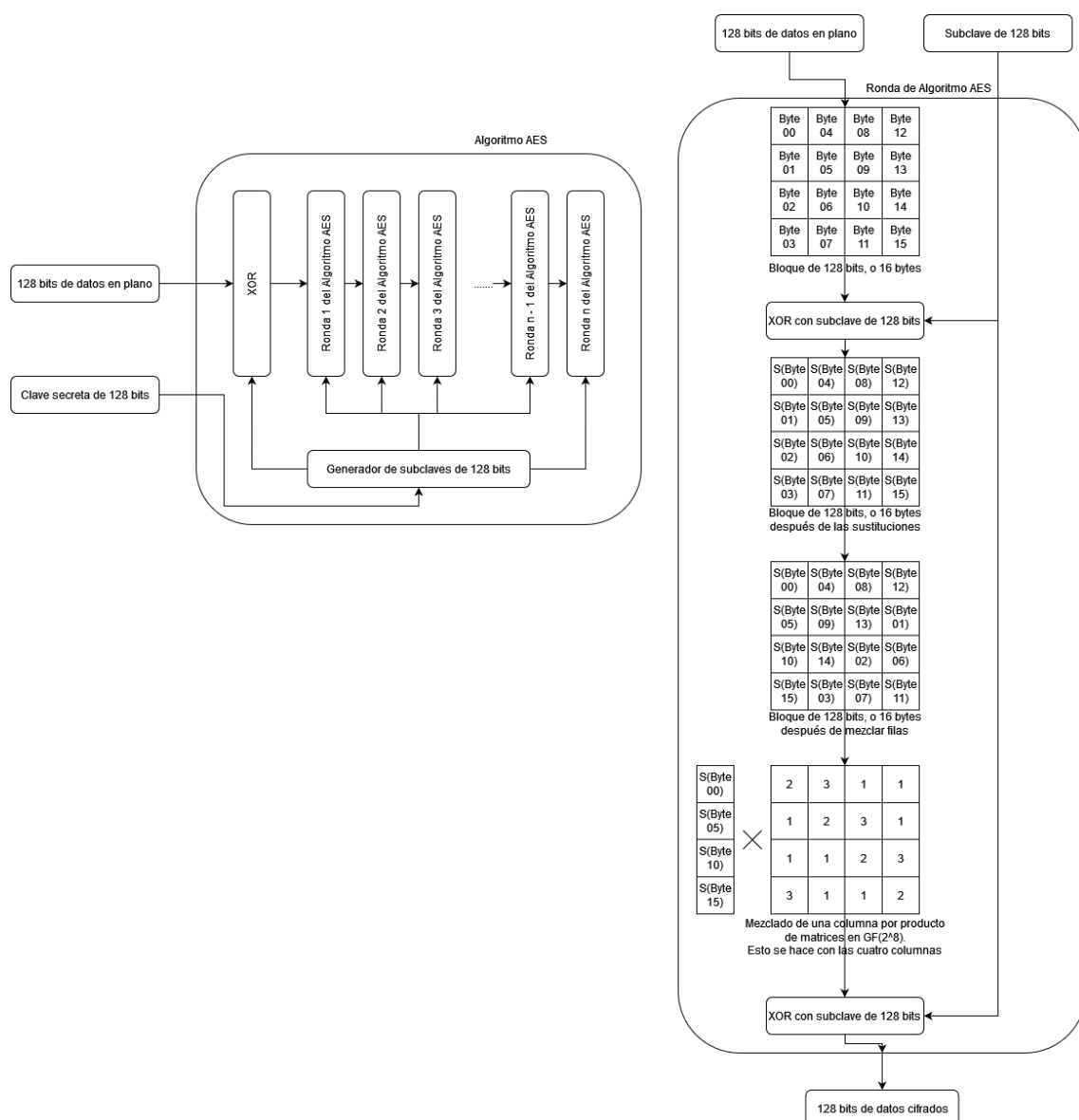


Figura 2.9: Diagrama de funcionamiento general de AES.

Una cuestión inquietante en estos días son los ordenadores cuánticos y si nuestras contraseñas y métodos de cifrados serán vulnerables a ataques con estos ordenadores. Un ordenador cuántico [58] es un ordenador que explota las propiedades de la mecánica cuántica para realizar sus operaciones, su unidad básica es el qubit (a diferencia del clásico bit de un supercomputador) y puede tener valores de uno, cero, o ambos a la vez, aprovechando la dualidad partícula-onda. En algunas operaciones un supercomputador cuántico puede llegar a ser cientos de millones de veces más rápido que el supercomputador más potente. En el caso de AES, no es que en un supercomputador cuántico se encripte millones de veces más rápido que en otro computador, simplemente es que ciertas operaciones las hace de manera mucho más rápida aprovechando ventajas de la mecánica cuántica. En el caso de AES, hay un algoritmo cuántico de búsqueda que facilita los ataques de fuerza bruta, es el caso del algoritmo de Grover [59]. Este algoritmo, teniendo en cuenta claves de un tamaño de 128 bits y de manera hipotética, podría reducir las 2^{128} combinaciones de claves a 2^{64} combinaciones. 2^{128} combinaciones está fuera del alcance de los supercomputadores durante unos cuantos años, sin embargo, 2^{64} combinaciones sí que está al alcance. Por tanto, podríamos tener un problema en este aspecto, sin embargo, este algoritmo reduce significativamente las combinaciones.

pero no tenemos muchos problemas en aumentar el tamaño de las claves y seguir funcionando. Podríamos utilizar claves más grandes y tener 2^{256} combinaciones, que con el algoritmo de Grover se reducirían a 2^{128} combinaciones, lo cual se mantiene fuera del alcance actual de cómputo. Lo más probable que ocurra es que conforme vayan avanzando los supercomputadores cuánticos y sus algoritmos, los algoritmos de cifrado tengan versiones resistentes a estos superordenadores cuánticos y que nos quedemos en una situación similar a la actual.

Hay pocos inconvenientes de utilizar AES como método de cifrado, por eso es un estándar muy utilizado en la actualidad. Tiene los problemas que puede tener un cifrado por bloques, es decir, cómo transmitir la clave simétrica de manera segura, pero no hay mucho más y por eso se recomienda utilizarlo.

El algoritmo AES se utiliza en muchos campos debido a que es el estándar a utilizar, por ejemplo, VPNs, mensajería instantánea o correos electrónicos, transacciones bancarias, protocolos de transporte como SSL o TLS, así como cualquier campo en el que se necesite encriptar o desencriptar datos de cualquier tamaño.

2.5.3 Diffie-Hellman

Diffie-Hellman [60] hace referencia a un método matemático que permite intercambiar de manera segura y rápida claves criptográficas a través de un canal público donde cualquiera tiene acceso. Fue de los primeros protocolos de criptografía de clave pública, y este método fue concebido por Ralph Merkle aunque se nombró a partir de Whitfield Diffie y Martin Hellman en su publicación en el año 1976. De hecho, Martin Hellman propuso que el algoritmo se llamase Diffie-Hellman-Merkle para reconocer su contribución al inicio de la criptografía de clave pública o asimétrica. Antiguamente para transmitir una clave entre dos partes y utilizarla para comunicarse había que hacerlo de manera física, ya sea reuniéndose las dos personas por la noche en un parque para comunicar la clave, o utilizando un mensajero. Este proceso podía tardar días o directamente no ser posible. Con este método matemático e Internet no hace falta que las personas se reúnan o se conozcan acaso, ya que ambos a partir de unos pasos a seguir y aprovechando propiedades matemáticas podrán llegar a componer la misma clave.

De entre las propiedades que debe haber en Diffie-Hellman, necesitaremos ciertos parámetros de entrada con los que el emisor y el receptor trabajarán para obtener las claves. Estos parámetros son dos, un generador g y un número n . El número n es un número primo (recordemos que los números primos son aquellos números que únicamente son divisibles por 1 y por sí mismos) muy grande, que garantiza la seguridad del algoritmo, se utilizan números n de alrededor de 2000 bits de tamaño, aunque se empiezan a utilizar números n de alrededor de 4100 bits de tamaño en algunas áreas, de todas formas, hay que evaluar el balance de tiempo-seguridad, ya que si aumentamos la clave no ganamos tanta seguridad en comparación con la eficiencia o el tiempo que perdemos, tenemos que tener un balance en ese aspecto. Es muy importante que n sea de ese tamaño grande y que sea primo, ya que para romper este método habría que utilizar factorización de enteros, y si el número no es primo podría reducirse el número de posibilidades de n , haciendo que sea más vulnerable a ataques de fuerza bruta. El generador g por su parte es un número primo también, en anteriores años se ponían números primos pequeños, pero, en teoría, puede ser un número primo de cualquier tamaño, ya que la capacidad computacional de la actualidad lo permite y no afecta a la seguridad. Un aspecto clave de este generador g es que g es una raíz primitiva en módulo n . Las operaciones que realizaremos en este apartado utilizan aritmética modular y, concretamente, el grupo multiplicativo de enteros módulo n . El hecho de que g sea una raíz primitiva en módulo n significa que yo puedo elevar g a todos los posibles valores desde 1 hasta $(n - 1)$ ambos inclusive en modulo n , y obtendré todas las clases de congruencia del módulo n , es decir, todos los valores

del resto de la división entre n . Esto se utiliza principalmente para asegurarnos de que la resultante clave compartida tome valores desde 1 hasta $(n - 1)$.

El funcionamiento de este protocolo es sencillo, en pocos pasos el emisor y el receptor tendrán su clave con la que se compartirán información de manera privada. Se basa en un aspecto clave de las matemáticas y es que se puede obtener un resultado matemático (un número) de múltiples formas distintas. El primer paso es que tanto el emisor como el receptor escojan un generador g y un número n para ambos. Para el ejemplo, el emisor y el receptor han acordado utilizar el número primo $n = 11$ y el generador $g = 2$, el cual es una raíz primitiva módulo $n = 11$. Estos dos valores pueden ser públicos y no hay problema en que lo sepa cualquiera. A continuación, tanto el emisor como el receptor escogerán sus claves o números secretos, los cuales deberán estar entre 1 y n , deberán ser grandes y aleatorios, y no deberán compartirse con nadie. En este caso, escogeremos números pequeños para que el ejemplo sea sencillo de entender, pero al aplicar este protocolo estos números deberán tener esas características mencionadas. Para el caso del emisor, su clave secreta la llamará a y tendrá un valor de $a = 7$. El receptor por su parte llamará a su clave secreta b , y le establecerá un valor de $b = 4$. El siguiente paso es que el emisor calculará el valor de $g^a \bmod n$, y el receptor por su parte calculará el valor de $g^b \bmod n$. El hecho de que cada uno calcule su respectivo valor hace que estés elevando un número grande a un número todavía más grande, es una cuenta de gran tamaño, sin embargo, la mantenemos en la órbita de n realizando la operación en módulo n . El valor de $g^a \bmod n$ para el emisor es $2^7 \bmod 11 = 128 \bmod 11 = 7$. Por su parte, el valor de $g^b \bmod n$ para el receptor es de $2^4 \bmod 11 = 16 \bmod 11 = 5$. Una clave de este protocolo es que se sabe g y se sabe n , el cual es un número primo enorme, pero dada la operación que hemos realizado para el emisor y el receptor, es muy complicado saber los valores de a y b , tendríamos que probar todas las combinaciones (desde 1 hasta $(n - 1)$), lo cual es inviable. A continuación, se comparten los valores, es decir, el receptor recibe el valor de $g^a \bmod n$, y por su parte el emisor recibe el valor de $g^b \bmod n$. Estos dos valores son ahora públicos ya que se han compartido, pero las claves privadas del emisor y del receptor siguen siendo privadas, no son extraíbles de las dos fórmulas calculadas. El siguiente paso es que el emisor va a realizar la operación $(g^b \bmod n)^a \bmod n$, es decir, el resultado recibido del receptor lo elevará al valor de su clave secreta a , todo ello en módulo n . Por su parte, el receptor realizará la operación $(g^a \bmod n)^b \bmod n$, es decir, el resultado recibido del emisor lo elevará al valor de su clave secreta b , todo ello en módulo n .

Un aspecto de operar con aritmética modular es que ciertas operaciones mantienen sus propiedades en esa aritmética, una de ellas es la exponenciación, y una propiedad de esta es que si elevas un valor elevado a otro, el resultado es el valor elevado a la multiplicación de exponentes. Un ejemplo de eso que si tienes $(2^2)^3$ el valor es $2^{2 \cdot 3}$, el cual da como resultado $2^6 = 64$. El valor de la operación para el emisor es $(g^b \bmod n)^a \bmod n$, es decir, $5^7 \bmod 11 = 78125 \bmod 11 = 3$. Por su parte, el valor de la operación para el receptor es $(g^a \bmod n)^b \bmod n$, es decir, $7^4 \bmod 11 = 2401 \bmod 11 = 3$. Como hemos visto, el valor de la clave compartida a la que han llegado tanto el emisor como el receptor es 3, y han llegado a las claves de forma distinta, a través de sus claves secretas que eligieron como quisieron. En esencia, para romper el algoritmo necesitamos saber el valor de la clave secreta del emisor o del receptor, y tendríamos que resolver un logaritmo discreto, lo cual es una operación sencilla, pero es ciertamente difícil incluso para los supercomputadores actuales, principalmente debido a que la operación de exponente es muy fácil, pero la operación de logaritmo discreto es varias veces más difícil que exponente.

La principal ventaja de este protocolo es el hecho de que poder establecer una clave compartida secreta y segura (por ejemplo, una clave para un algoritmo de cifrado simétrico como AES) entre un grupo de personas (ya que este protocolo también funciona con más de dos partes) a través de un canal público. Este tipo de algoritmo y la aproximación que utiliza se ve en algoritmos populares de clave pública.

Uno de los inconvenientes importantes de Diffie-Hellman es que no provee autenticación de los usuarios, el emisor no es capaz de saber que quien envía el mensaje es efectivamente esa persona, además que el

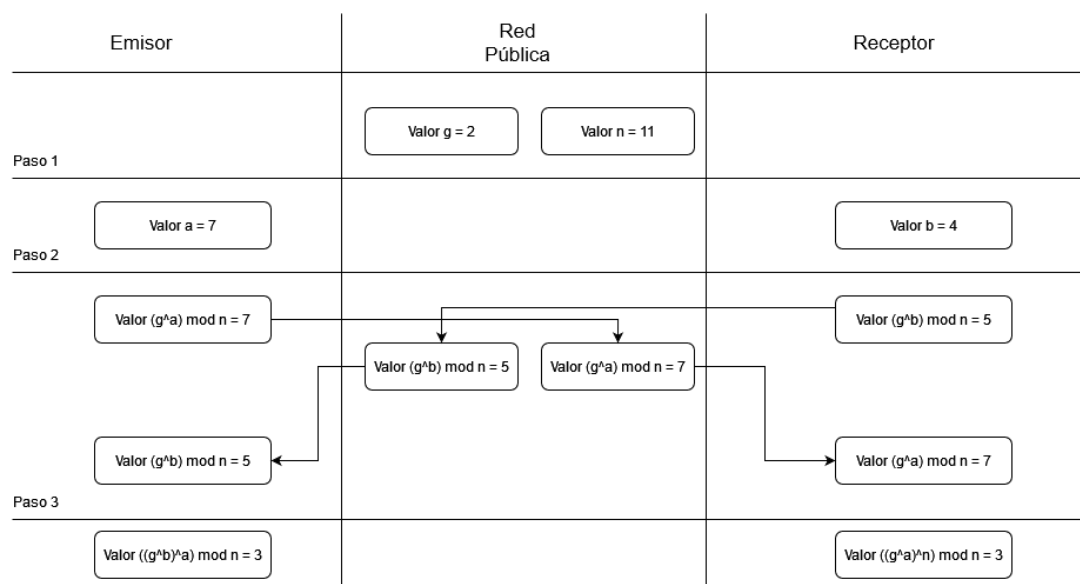


Figura 2.10: Ejemplo de intercambio por Diffie-Hellman.

hecho de manejar números tan grandes puede ser computacionalmente costoso. También, Diffie-Hellman es susceptible de ataques de intermediario, en donde un atacante intercepta los mensajes de una parte y de otra, crea sus claves secretas para comunicarse con cada uno y actúa como un intermediario que intercepta los mensajes entre el emisor y el receptor y los modifica a su gusto con sus claves, pasando a ser una parte activa de la comunicación y sin que el emisor o el receptor lo sepa. Por este tipo de ataques que dejan a Diffie-Hellman susceptible es que en la actualidad se utiliza una aproximación similar a Diffie-Hellman pero que previene este tipo de ataques.

La aproximación de Diffie-Hellman es muy utilizada (el algoritmo en sí no tanto ya que no provee autenticación), desde el protocolo TLS o VPNs hasta intercambio seguro de archivos, esta aproximación es fundamental debido a que todo el rato necesitas establecer una comunicación segura sobre un canal que puede ser accedido por cualquiera para transmitir información, y en estos casos que suceden tanto en la actualidad es necesario esa forma de establecer una comunicación segura.

2.5.4 RSA

RSA [61] son las siglas de un sistema criptográfico de clave asimétrica creado por tres personas, las cuales sus iniciales de sus apellidos forman las siglas, estos son Ron Rivest, Adi Shamir y Leonard Adleman. Este algoritmo llevó cierto tiempo de creación desde la creación y publicación de Diffie-Hellman en 1976. Rivest y Shamir, informáticos ambos, buscaban crear el algoritmo utilizando funciones matemáticas invertibles y Adleman, matemático, le trataba de encontrar debilidades a las funciones de sus compañeros, hasta que el algoritmo fue publicado por primera vez en 1977, y es un algoritmo de clave pública bastante utilizado y estandarizado en la actualidad. RSA es un algoritmo que confía su seguridad en matemáticas ciertamente avanzadas, haciendo uso de operaciones sobre números primos y aritmética modular.

La fundación matemática detrás de RSA es amplia, y es conveniente explicarla para comprenderlo mejor. RSA parte de la base de que es sencillo encontrar tres números enteros e (normalmente utilizado para encriptado), d (normalmente utilizado para desencriptado) y n (normalmente utilizado para representar el mensaje) con los cuales se cumpla la fórmula $(m^e)^d \equiv m \bmod n$ para todo entero m entre 0 y n , utilizando exponenciación modular, y sobre todo sabiendo que es muy difícil encontrar el valor de d incluso sabiendo el resto de los valores de la fórmula. Utilizaremos también la función ϕ de Euler [62],

desarrollada por el gran matemático Leonhard Euler, la cual es muy utilizada en teoría de números y nos indica que, si n es un número entero positivo, entonces $\varphi(n)$ se define como la cantidad de números enteros positivos menores a n y coprimos con n , es decir, es la cantidad de números enteros positivos menores que n tales que el máximo común divisor con respecto a n es 1. Esta función φ de Euler tiene algunas propiedades en aritmética modular que nos vendrán bien para nuestro propósito, y además nos ayuda con un teorema que también utilizaremos, es el Teorema de Euler-Fermat en teoría de números [63]. Este teorema es una generalización del pequeño teorema de Fermat e indica que, si a y n son coprimos, entonces $a^{\varphi(n)} \equiv 1 \pmod{n}$. Cabe resaltar que dos números a y b son coprimos [64] si su máximo común divisor es 1, es decir, no tienen otro común divisor entre ellos que no sea 1, para calcular si dos números son coprimos utilizamos el Algoritmo de Euclides [65]. Es importante conocer el concepto de número semiprimo o biprimo, es un número fruto de la multiplicación de dos números primos. Otro concepto importante a conocer en aritmética modulares el inverso multiplicativo modular [66]. El inverso multiplicativo modular de un número entero a en módulo m es un número entero b que satisfaga la ecuación $a \cdot b \equiv 1 \pmod{m}$. Este inverso existe solo si a y m son coprimos, su demostración es algo extensa para este apartado, pero para calcular este inverso se hace uso del algoritmo de Euclides Extendido y de ecuaciones diofánticas lineales. Estas teorías y propiedades matemáticas nos ayudan a encontrar pares de claves para encriptar y desencriptar de tal forma que sea muy fácil lo primero pero muy difícil lo segundo.

Algunas fórmulas y propiedades a tener en cuenta para RSA son:

Si a y n son coprimos, entonces se cumple que: $a^{\phi(n)} \equiv 1 \pmod{n}$

Inverso multiplicativo modular: $a \cdot b \equiv 1 \pmod{n}$

Si p es primo, entonces: $\phi(p) = p - 1$

Si m y n son enteros positivos coprimos, entonces: $\phi(m \cdot n) = \phi(m) \cdot \phi(n)$

Y algunas fórmulas para cifrado y descifrado en RSA son las siguientes:

$$e \cdot d \equiv 1 \pmod{\phi(n)}$$

$$(msg^e \pmod{n}) = msg_{\text{Encriptado}}$$

$$(msg_{\text{Encriptado}}^d \pmod{n}) = msg$$

$$(m^e)^d \equiv m \pmod{n}$$

Comenzaremos con un ejemplo de RSA en donde el emisor quiere mandarle el mensaje 33 al receptor. Hay dos pasos principales para este algoritmo RSA, el primero es generar las claves necesarias, y el segundo es utilizar esas claves para encriptar y desencriptar el mensaje. Dentro del primer paso, lo primero que se hace es seleccionar dos números primos, los llamaremos p y q , y los utilizaremos para obtener el valor n , el cual es equivalente al producto de p y q y se utilizará como el valor del módulo para las claves privada y pública que calcularemos más adelante. Estos números p y q deberían ser aleatorios y secretos, ser suficientemente grandes y tener una gran diferencia entre ellos (veremos más adelante que si no se escogen estos números de manera correcta podemos romper RSA). En nuestro caso, escogeremos el valor de $p = 5$, el valor de $q = 71$ y, por tanto, el valor de n (el cual es un número semiprimo) es el producto de 5 y 71, es decir, $n = 355$. El segundo paso es calcular el valor de la función φ de Euler de n , es decir, $\varphi(355)$. Para esto aprovecharemos ciertas propiedades conocidas de la función φ de Euler, concretamente de dos propiedades. La primera de ellas es que $\varphi(p) = p - 1$ si p es primo. La segunda es que, si m y n son números enteros positivos coprimos, entonces $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$. Mientras que la demostración de la segunda propiedad no es muy intuitiva, la demostración de la primera propiedad sí que lo es. Recordemos que un número primo tiene como divisores el uno y sí mismo, y dos números son coprimos si su máximo

común divisor es uno, por tanto, es fácil deducir que para un valor n primo, el número de coprimos con n y menores que n (que es lo que calcula la función φ de Euler) son todos los números anteriores a n y positivos, es decir, $n - 1$, ya que los únicos divisores de n son el uno (el cual es un divisor de todos los números) y sí mismo, por el hecho de ser primo. Teniendo en cuenta estas propiedades y que dos números primos a y b diferentes son coprimos, podemos establecer que $\varphi(355) = (p - 1) \cdot (q - 1) = (5 - 1) \cdot (71 - 1) = 280$. Ahora que hemos realizado estos cálculos podemos computar los valores de la clave pública y privada. La clave pública e debe tener tres características para ser apta, debe ser un número primo, debe ser menor que el valor de $\varphi(n)$ y no debe ser un factor de $\varphi(n)$, es decir, no debe ser un divisor de $\varphi(n)$. Además, debería ser no muy pequeño, debido a que es susceptible de más ataques por fuerza bruta. En mi caso escogeremos el valor de clave pública $e = 277$, como vemos es un valor menor que $\varphi(355) = 280$, y además es coprimo con 280, así que no divide a 280. La clave pública es pública y, por tanto, ese valor de 277 puede ser conocido por cualquiera, pero no se puede saber cómo has llegado al mismo. Recordemos que el proceso para hallar números coprimos es un algoritmo llamado algoritmo de Euclides. Para la clave privada d , deberemos escoger un valor d que sea inverso multiplicativo de e en módulo $\varphi(n)$, es decir, debe satisfacer la ecuación $(e \cdot d) \equiv 1 \pmod{\varphi(n)}$. Esto se ha mencionado cómo se calcula, puede ser un proceso lento y por tanto se ha escogido para este ejemplo el valor $d = 93$. El valor $d = 93$ cumple con la ecuación estipulada, ya que tenemos $(e \cdot d) \equiv 1 \pmod{\varphi(n)}$, es decir $(277 \cdot 93) \equiv 1 \pmod{\varphi(355)}$, es decir $(277 \cdot 93) \equiv 1 \pmod{280}$, es decir $(25761) \equiv 1 \pmod{280}$. Si dividimos 25761 entre 280 nos da cociente 92 y resto 1, cumpliendo la ecuación. Una vez realizado esto ya tenemos todos los valores necesarios para poder realizar el encriptado y desencriptado, lo cual realizaremos en el segundo paso. Para comenzar este segundo paso encriptaremos nuestro mensaje msg siguiendo la fórmula $msg^e \pmod{n} = msg_encriptado$. En este caso nuestro mensaje recordemos que tenía un valor de 33, por tanto, el valor de nuestro mensaje encriptado es $msg_encriptado = 33^{277} \pmod{355} = 13$. El valor $msg_encriptado = 13$ es el que el emisor le enviará al receptor, el cuál realizará la operación $msg = msg_encriptado^d \pmod{n}$. Esa operación es equivalente a $13^{93} \pmod{355} = 33$, este valor resultante es el mismo valor del mensaje msg que encriptamos inicialmente. Este proceso del algoritmo RSA es conmutativo, es decir, puedo encriptar con la clave pública o con la clave privada, y puedo desencriptar con la clave privada o la clave pública respectivamente. Para comprobar esto, volvemos a realizar el proceso (simplificado) con el mensaje con valor $msg = 33$. Al encriptarlo con el valor de la clave privada $d = 93$ obtenemos un valor de $msg_encriptado = 33^{93} \pmod{355} = 123$. Al desencriptar este valor cifrado con nuestra clave pública $e = 277$, tenemos que $msg = 123^{277} \pmod{355} = 33$, el valor que habíamos encriptado al principio. Cabe destacar que en este ejemplo utilizamos números pequeños por conveniencia, pero en la realidad RSA utiliza números de alrededor de 300 cifras, haciendo que los cálculos no sean tan sencillos como quizá parecen.

Las principales ventajas es que provee una gran seguridad debido a las matemáticas complejas que tiene por detrás y que sea fácil de encriptar datos y desencriptar los datos en caso de tener la clave pública y clave privada, pero de no tenerlas es muy complicado descifrar los valores que crean la clave privada, ya que habría que factorizar n para obtener los valores p y q que lo conformaron y poder calcular la función φ de Euler y así calcula la clave privada d . El problema de factorizar n es que RSA utiliza valores de n enormes, el estándar es de 2048 bits y hay campos en los que se utiliza 4096 bits, para poner en contexto son números de alrededor de 600 y 1200 cifras respectivamente, por tanto, no es factible hoy en día. Para poner aún más en contexto, en 1991 RSA Labs propuso una competición en la que daban varios números de diferentes cifras (100, 200, hasta 600) y quien consiguiese factorizarlos recibía un premio económico. Fueron alrededor de 50 números y se han factorizado alrededor de la mitad desde entonces (desde 1991 han pasado más de 30 años), de hecho, el número de 1024 bits aún no ha sido factorizado (el más grande factorizado fue hace unos años, y tiene 829 bits), y el estándar es de utilizar 2048 bits desde el año

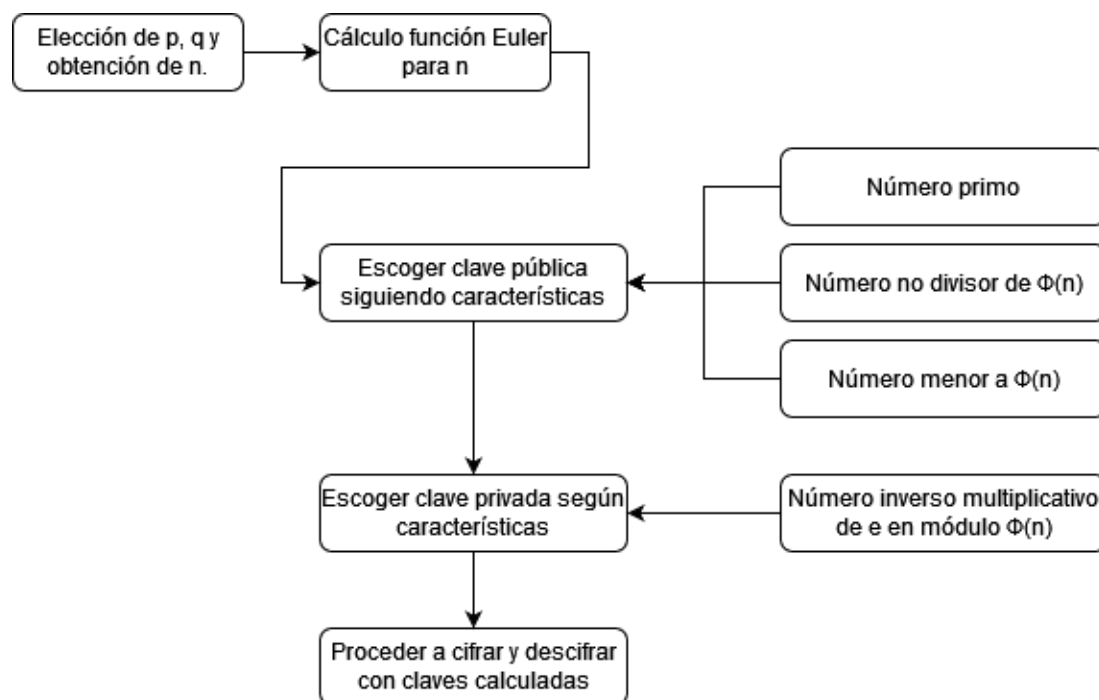


Figura 2.11: Diagrama de flujo de RSA simplificado.

2015. Además, RSA permite transmitir datos de manera segura y con autenticación a través de canales públicos, y es un algoritmo muy utilizado en la actualidad.

Los principales inconvenientes de RSA es que para archivos de gran tamaño es inconveniente el encriptado y desencriptado debido a que con claves grandes es exigente computacionalmente, es ciertamente lento en comparación con claves y algoritmos de cifrado simétrico, por eso se suele utilizar para establecer comunicaciones seguras sobre canales públicos y autenticación, y para cifrados se utiliza AES que es más rápido. Un problema de RSA es que utiliza tamaños de claves muy grandes en comparación con otros competidores, el cifrado de clave simétrica y el algoritmo que veremos en el siguiente apartado usan tamaños de clave más pequeños y proveen una seguridad similar. Otro problema, aunque este es general de todos los algoritmos de cifrado, es que malas implementaciones de RSA lo hacen ciertamente vulnerable. Hay ciertas malas decisiones [67] a la hora de escoger valores [68] de p y q (por ejemplo, escoger valores tal que $p == q$, o que p y q sean cercanos y estén alrededor de la órbita del valor \sqrt{n}) que hacen que el algoritmo sea descifrable de manera trivial utilizando el método de factorización de Fermat [69] y otras propiedades. Un problema que puede afectar a RSA en el futuro es la computación cuántica, ya que se podrían utilizar computadores cuánticos junto con el algoritmo de Shor [70] para descomponer N en factores, pero se necesitan ordenadores con unos cuantos qubits para descomponer el número.

El algoritmo RSA se utiliza bastante en la actualidad, está muy establecido y se utiliza en firmas digitales y para establecer comunicaciones seguras sobre canales no tan seguros, de hecho, si miras el certificado utilizado en diferentes páginas web, por ejemplo, la página web de la Universidad de Alcalá, verás que se utilizan AES para ciertas partes, y RSA o el algoritmo que veremos en el siguiente apartado.

2.5.5 ECC

ECC [71] son las siglas en inglés que hacen referencia a la criptografía de curva elíptica (Elliptic Curve Cryptography). Este tipo de criptografía es una aproximación del tipo criptografía de clave pública que se aprovecha de las propiedades de un tipo específico de funciones matemáticas, estas son las curvas

elípticas [72], las cuales utilizaremos sobre un campo finito o campo de Galois. El uso de estas curvas en criptografía fue propuesto por los matemáticos Neal Koblitz y Victor Saul Miller en 1985, y entraron y se propagó su uso alrededor del año 2005. En una conferencia en el año 2005, la NSA anunció un paquete de algoritmos criptográficos para tener una base para información clasificada y no clasificada, y en ese paquete la criptografía ECC estaba recomendada por el NIST, concretamente una criptografía de curvas elípticas asociadas al protocolo Diffie-Hellman (ECDH) [73] con un tamaño de clave de 384 bits, veremos más adelante en este apartado si es mucho o no.

La criptografía de clave elíptica basa su seguridad en ciertas propiedades de las curvas elípticas. Para obtener una visión general de cómo funcionan y cómo se utilizan en criptografía, debemos saber que hay diferentes tipos de curvas elípticas, nos centraremos en las que tienen la fórmula $y^2 = x^3 + (a \cdot x) + b$. Esta ecuación cumple que el polinomio de la parte derecha del símbolo igual no tiene ninguna raíz doble (no hay dos soluciones para un mismo valor x o y), y eso es debido a que la característica K del campo o cuerpo sobre el que está definido las curvas elípticas que cumplen esta ecuación no tiene valor 2 ni valor 3 (en caso de ser 2 o 3, necesitaríamos más polinomios en la ecuación de la curva para poder definirla de manera general), esta característica [74] $\text{char}(K)$ es un concepto matemático ciertamente avanzado y que se desvía del objetivo de este apartado, por tanto, no se explicará en detalle en este documento, sin embargo, hay bastante documentación en Internet acerca de esto para quien quiera entender la matemática avanzada detrás de las curvas elípticas. Una característica importante de estas curvas elípticas es que son simétricas en el eje x , es decir, sabemos que si existe un punto (x, y) en la curva, también existe un punto $(x, -y)$ en la misma. Además, sabemos por la fórmula que estas curvas tienen dos dimensiones y que las variables de la curva son a y b de cara a criptografía. Sabemos también como característica de las curvas elípticas que una recta que cruce a la curva la intersecará en tres puntos. Por ejemplo, en caso de que la recta interseque a la curva en un solo punto, entonces ese punto cuenta dos veces y el tercer punto es infinito. El hecho de cortar la curva en un solo punto suele producirse en casos donde la recta es paralela a alguno de los dos ejes de coordenadas x o y . En caso de que definamos dos puntos pertenecientes a la curva elíptica y dibujemos la recta que los une, esa recta generará un tercer punto de corte en la curva. Puede suceder algún caso en que la recta definida corte a la curva en únicamente dos puntos, no generándose ese tercer punto, esto suele suceder cuando uno de los dos puntos de corte es tangente a la curva, en ese caso ese punto tangente se cuenta dos veces. Cabe destacar también que en ECC solemos trabajar con aritmética modular, todas las operaciones se realizan en un módulo n primo y mayor que 3. Con esto, tenemos lo que necesitamos para proveer un ejemplo de uso de este tipo de criptografía, este ejemplo de uso será utilizando la aproximación de curvas elípticas con Diffie-Hellman (ECDH).

Para este ejemplo de curvas elípticas con Diffie-Hellman (ECDH) primero introduciremos la situación junto con las fórmulas que se utilizarán, luego realizaremos el proceso. Tenemos un emisor que quiere enviar un mensaje secreto de valor $\text{msg} = 44$ al receptor. Para ello se ponen de acuerdo en que la curva elíptica de ecuación $y^2 = x^3 - (7 \cdot x) - 10 \bmod 19$. Esta curva tiene 24 puntos en ese módulo (contando punto en el infinito). El emisor y receptor también se ponen de acuerdo en utilizar como generador o base un punto en la curva, llamado G , con valor $x = 1$ y valor $y = 2$, el generador genera un subgrupo con 8 puntos en ese módulo. Para sumar dos puntos $P = (x_1, y_1)$ y $Q = (x_2, y_2)$, primero tener en cuenta que si $x_2 = x_1$ e $y_2 = -y_1$ entonces la suma es el punto en el infinito O . De lo contrario, $P + Q$ darán como resultado un punto R con valores (x_3, y_3) en donde x_3 se calculará como $\lambda^2 - x_1 - x_2 \bmod n$, y por su parte y_3 se calculará como $\lambda \cdot (x_1 - x_3) - y_1 \bmod n$, siendo n el valor del número primo que hayan acordado, en este caso 19. Lambda λ es la pendiente de la recta que pasa por los puntos P y Q , y en caso de que P y Q sean distintos entonces la fórmula es conocida como $\lambda = (y_2 - y_1) / (x_2 - x_1) \bmod n$, sin embargo, en caso de que P y Q sean iguales entonces $\lambda = (3 \cdot (x_1)^2 + a) / (2 \cdot y_1) \bmod n$, el valor de a se

obtiene de la ecuación de la recta. Estas fórmulas nos servirán para calcular los valores en la recta que necesitan el emisor y receptor para comunicarse de manera segura.

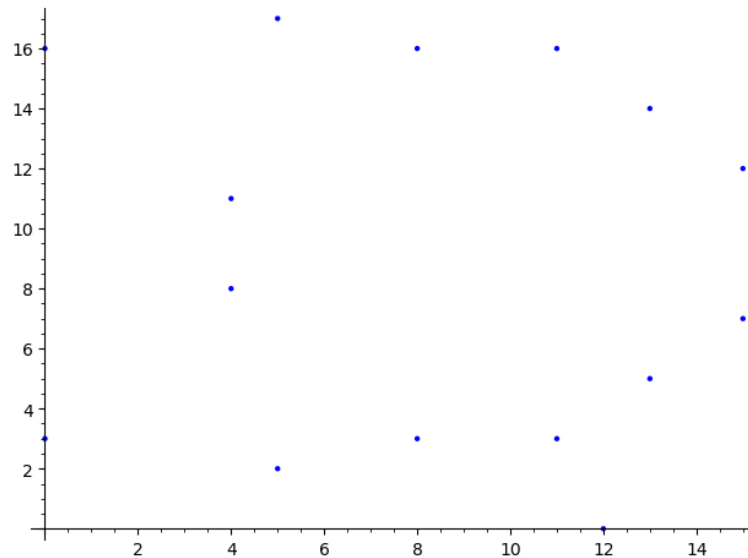


Figura 2.12: Aspecto de la curva elíptica con fórmula $y^2 = x^3 - (7 \cdot x) - 10 \bmod 19$.

Comenzando ya con el ejemplo práctico, tanto el emisor como el receptor escogen un número para que sea su clave privada, en este caso el emisor escoge un valor $a = 4$ y el receptor escoge un valor $b = 5$. Como ambos tienen acceso al generador, el emisor calculará el valor $A = a \cdot G$, el receptor por su parte calculará el valor $B = b \cdot G$. Estos valores A y B son valores que pertenecen a la curva elíptica, y para mayor comodidad en caso de que se calculen a mano se utilizarán propiedades de las curvas elípticas para obtener el doble de un punto o para multiplicarlo, el emisor calculará el valor de $4G$ (sumar G cuatro veces, o cualquier equivalente) y el receptor calculará el valor de $5G$. Realizaremos estos cálculos doblando los puntos, para ambos casos necesitamos saber $2G$, es decir $G + G$, así que comenzamos a calcular. En este caso, utilizando la fórmula explicada anteriormente, P y Q son iguales (son G y G), así que calcularemos la pendiente λ como $\lambda = (3 \cdot (x_1)^2 + a) / (2 \cdot y_1) \bmod n$, es decir $(3 \cdot 1^2 + (-7)) / (2 \cdot 2) = -1 \bmod 19$. Como nos ha salido un número negativo mod 19, utilizaremos el inverso multiplicativo modular para obtener el valor, esto se realiza utilizando el Algoritmo de Euclides Extendido y no se entrará en detalles para no alargar en demasía este ejemplo. Sabemos que el inverso multiplicativo modular de -1 en módulo 19 es 18, así que $\lambda = 18$. Procederemos a calcular el valor de x_3 como $\lambda^2 - x_1 - x_2 \bmod n$, es decir $(-1)^2 - 1 - 1 \bmod 19$, vuelve a dar -1 así que sabemos que x_3 toma el valor de 18. Calcularemos el valor de y_3 como $\lambda \cdot (x_1 - x_3) - y_1 \bmod n$, es decir $(-1) \cdot (1 - 18) - 2 \bmod 19 = 15$. Por tanto, el punto $2G$ tiene como coordenadas los valores 18 y 15. Si sumamos $2G$ con $2G$ obtendremos $4G$, que es el valor que necesita el emisor. Es un caso similar al anterior, así se procederá de manera resumida. El valor de la pendiente es $(3 \cdot 18^2 + (-7)) / (2 \cdot 15) = (193/6) \bmod 19$. En este caso tenemos una fracción con módulo 19, sabemos que la fracción $(193/6)$ la podemos expresar como un producto de 193 y 6^{-1} , el inverso multiplicativo modular de 6 en modulo 19 es 16, así que la operación es equivalente a $193 \cdot 16 \bmod 19$, por tanto el valor de la pendiente es $3088 \bmod 19 = 10$. El valor de x_3 se puede calcular como $(10)^2 - 18 - 18 \bmod 19 = 64 \bmod 19 = 7$. El valor de y_3 es en este caso $(10) \cdot (18 - 7) - 15 \bmod 19 = 95 \bmod 19 = 0$. Por tanto, concluimos que el punto $4G$ tiene los valores $x = 7$ e $y = 0$. Teniendo el valor que necesita el emisor, el valor de $4G$, procedemos a realizar la suma de G y $4G$ para obtener el valor que necesita el receptor, $5G$. Para este caso, los valores de los puntos son distintos, así que la pendiente se calculará como $\lambda = (0 - 2) / (7 - 1) \bmod 19 = (-1/3) \bmod 19$. Este resultado se puede expresar como un producto de (-1) y 3^{-1} , sabemos que el valor del inverso multiplicativo modular de 3 en módulo 19 es

13, y que -1 en módulo 19 equivale a 18, por tanto es equivalente al producto de 18 y 13 mod 19, es decir $234 \bmod 19 = 6$. El valor de x_3 se define como $(6)^2 - 1 - 7 \bmod 19 = 28 \bmod 19 = 9$. El valor de y_3 se puede calcular como $(6) \cdot (1 - 9) - 2 \bmod 19 = -50 \bmod 19 = -12 \bmod 19 = 7$. Por tanto, obtenemos que el punto 5G tiene los valores $x = 9$ e $y = 7$. Teniendo estos valores, el siguiente paso es que el emisor y el receptor se intercambien sus valores, de tal manera que el emisor recibe el valor computado de B ($x = 9, y = 7$) y el receptor recibe el valor computado de A ($x = 7, y = 0$). El siguiente paso es que el emisor calcule el valor $a \cdot B$ y que el receptor calcule el valor $b \cdot A$, es decir, multipliquen sus claves privadas por los valores que han recibido del otro. El valor de $a \cdot B$ es equivalente a $4 \cdot B$, que es equivalente a $4 \cdot 5G$, no se desarrollará el cálculo de este valor ya que alargaría demasiado el ejemplo, pero el emisor calcula este valor llamado P y obtiene sus coordenadas con valor $x = 7$ y valor $y = 0$. Por su parte, el receptor tiene que calcular el valor de $5 \cdot A$, que es equivalente a $5 \cdot 4G$, tampoco se desarrollará este cálculo por el motivo expuesto anteriormente, así que lo dejamos en que el receptor realiza este cálculo y obtiene el valor de las coordenadas $x = 7$ y valor $y = 0$, son las coordenadas del punto P, ya que el emisor y el receptor han calculado el mismo número, $4 \cdot 5G$ es equivalente a $5 \cdot 4G = 20G$. Como vemos, el emisor y el receptor han llegado al mismo punto de la curva elíptica de dos maneras distintas, y lo que han encontrado es P, el valor de la clave pública con la que cifrarán los mensajes. El atacante puede conocer los valores de G, la ecuación de la curva elíptica y los valores de A y B, pero no tiene las claves privadas a y b, por tanto, no puede saber cuántos desplazamientos ha habido desde el generador G hasta llegar a P, de hecho, si dibujásemos la curva elíptica y dibujásemos un punto sobre la curva donde el valor es $20G$, si no se tienen los valores necesarios no se sabe decir si ese punto corresponde a $20G, 33G, 33000G$ o, en definitiva, el valor que multiplica a G. La forma de saberlo sería resolver un problema del logaritmo discreto para deshacer ese número, sin embargo, ese problema tiene complejidad exponencial para ciertos valores de gran tamaño, en el ejemplo se han utilizado valores pequeños y una curva elíptica preparada y sencilla para mostrar el funcionamiento, pero en la realidad se usan curvas elípticas recomendadas por el NIST de gran tamaño, con mínimo 192 bits [75].

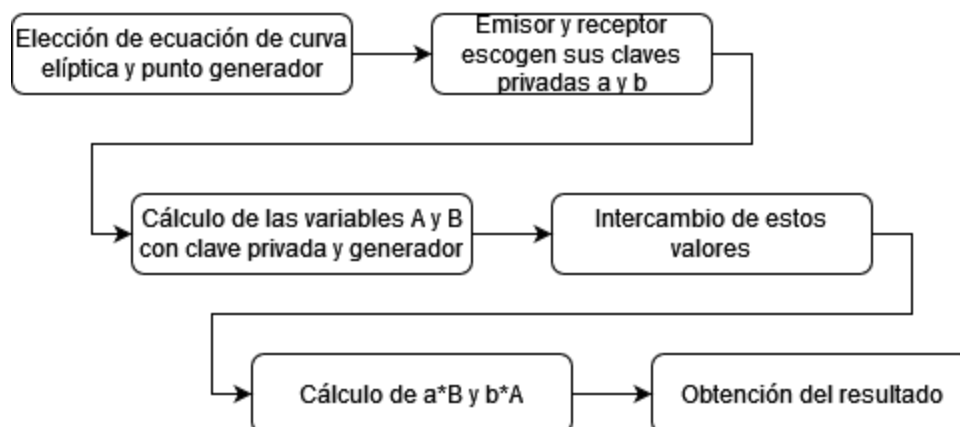


Figura 2.13: Diagrama simplificado de proceso para criptografía de clave elíptica.

Las principales ventajas de esta criptografía de curva elíptica es que ofrece encriptados y desencriptados seguros en caso de utilizar curvas elípticas y parámetros de dominio adecuados, esto se puede hacer siguiendo las recomendaciones del NIST. También es un tipo de cifrado eficiente y personalizable, ya que hay multitud de tamaños de claves diferentes. En diferentes protocolos o por ejemplo en Bitcoin [76], el tamaño de clave es de 256 bits, pero se pueden tener claves de 192 hasta 571 bits que ofrecen buena seguridad. Debemos destacar que si tenemos fallos de implementaciones entonces el atacante no tendrá que hacer el peor caso posible, que es comprobar todas las combinaciones de claves, es decir, el ataque de fuerza bruta, para obtener el valor de la misma. Para tener comparaciones y un poco de contexto,

las curvas elípticas de 256 bits ofrecen una seguridad similar a las claves de 3072 bits de RSA o del protocolo Diffie-Hellman, lo cual es más o menos similar a la seguridad que ofrece una clave simétrica de 128 bits de AES [77]. Uno de los motivos del uso de estas criptografías de curvas elípticas es que reducen el número de bits, lo cual puede reducir el coste computacional, manteniendo la misma seguridad que otros protocolos como RSA que usan claves de mucho más tamaño. Los inconvenientes de este tipo de criptografía es que es muy exacto, y, por tanto, necesitamos estudiar las curvas a utilizar de manera precisa para saber que son seguras, y deberemos revisar con detalle la implementación. Como hay que estudiar las curvas para ver si son seguras y válidas o no, el hecho de no estudiarlas de manera apropiada puede dar como consecuencia una curva elíptica insegura, o una curva elíptica con una puerta trasera. Esto era el caso de un generador de números aleatorios [78] que utilizaba una curva elíptica para generar los bits aleatorios con unos parámetros, y que la NSA estableció como estándar NIST. Los criptógrafos empezaron a estudiarla, el generador de números aleatorios utilizaba diferentes múltiplos de un generador para actualizar la semilla secreta que genera nuevos números aleatorios (múltiplo P) y para derivar esos números aleatorios (múltiplo Q), los criptógrafos empezaron a investigarla y se dieron cuenta de que en caso de que P fuese igual a x veces Q, es decir, si hubiese una relación entre P y Q, se podría acceder a la semilla secreta. Esto eran especulaciones que podían ser factibles, especulaciones de que la NSA sabía ese número x , pero no se puede calcular por la matemática relacionada con las curvas elípticas. Este generador de números aleatorios basados en curvas elípticas se estableció como un estándar NIST en la publicación NIST SP 800-90A en 2006, sin embargo, fue retirado en el año 2014. Los campos de utilización de esta criptografía son muchos, ciertos servicios web de Google utilizan una combinación de RSA y ECC para los certificados. Se ha hablado de que Bitcoin utiliza este tipo de criptografía para diferentes aspectos, por ejemplo, verificar transacciones, en tarjetas bancarias y en firmas digitales para la autenticación, así como diferentes protocolos y métodos de comunicación.

2.5.6 La familia de algoritmos SHA

La familia de algoritmos hash seguros SHA [79], son una familia de funciones criptográficas hash publicadas por el NIST y estandarizadas FIPS. Incluyen los algoritmos SHA0, SHA1, SHA2 y SHA3, pero este último lo dejaremos para el siguiente apartado debido a que funciona ligeramente diferente que sus predecesores. Estos algoritmos, dada una entrada de datos de cualquier tamaño, computan un valor hash de diferente tamaño según el algoritmo, por ejemplo, SHA1 devuelve un hash de 160 bits. Una característica importante que deben tener es que, si cambio el valor de la entrada un solo bit, el valor hash de la salida es completamente distinto, de tal forma que no se pueden hacer criptoanálisis de estos para saber cómo se calculan las funciones.

En este apartado hablaremos de la familia SHA, todos los algoritmos funcionan de manera similar excepto SHA3. El caso de SHA0 es curioso debido a que, poco después de su publicación en 1993 fue retirado debido a que encontraron una falla de seguridad importante, y más adelante se encontró otra vulnerabilidad. Este algoritmo SHA0 fue reemplazado por la versión SHA1. SHA1 es una función hash publicada en 1995 que devuelve valores de 160 bits, unos 40 caracteres hexadecimales. Esta función a día de hoy es ciertamente utilizada, sobre todo para HMAC (código de autenticación de mensajes basado en hash) ya que aún sirve para eso, aunque este algoritmo se considera roto, esto es debido a que se encontraron múltiples colisiones que han hecho al algoritmo inseguro. El algoritmo fue considerado no seguro desde 2005 y el NIST oficialmente anunció su desuso en 2011. Con esto pasamos a SHA2, el cual es el algoritmo estandarizado y más utilizado de la familia SHA actualmente, y que también fue publicado por la NSA en 2001. Tanto SHA1 como SHA2, y otros algoritmos hash bastante utilizados en el pasado como MD5 están diseñados según una estructura Merkle-Damgård que se comentó previamente en este documento, además de una función de compresión para obtener el tamaño fijo de salida independientemente de la

entrada. Los tamaños de SHA2 son de 256 y 512 bits, pero existen versiones truncadas con tamaños de 224 y 384 bits. Si nos preguntamos cómo puede ser que, por ejemplo, un archivo grande de 5GB pueda pasarse por una función hash y devolvernos el mismo tamaño fijo de salida que si a esa función hash le pasamos la cadena abc, esto es debido a que el funcionamiento de SHA es en bloques de 512 bits (en caso de que nuestro mensaje no llegue a ese tamaño se le añade relleno según unas normas, en caso de que se sobrepase de ese tamaño se divide el mensaje en bloques de 512 bits y si al último bloque le falta relleno se le añade) e iterando sobre cada uno de ellos. Partimos de un estado interno (variables internas con ciertos valores) que define el estándar para el algoritmo hash, y conforme vamos procesando bloques del mensaje, vamos modificando los valores de esas variables internas mediante la función de compresión y diferentes operaciones matemáticas hasta que nos quedamos sin bloques que procesar, entonces miramos los valores de nuestras variables internas y ese será el valor hash que retornemos.

Para comprender el funcionamiento de SHA necesitaremos saber ciertos conceptos matemáticos, se va haciendo costumbre con esto de la criptografía :-). SHA trabaja con aritmética modular, trabajaremos tanto en distintos módulos como mod 2 para los números binarios y mod el tamaño de la clave para que al hacer productos o sumas nuestro valor no crezca de tamaño de bits. También utilizaremos operadores lógicos como puede ser XOR, además de operaciones para rotar bits hacia derecha o izquierda y para desplazar bits hacia derecha o izquierda. Además, SHA utiliza algunas funciones internas que, cuando vayan saliendo en el ejemplo se explicarán convenientemente.

A continuación, se explicará cómo se realiza el cálculo del valor hash con la función SHA256, para el caso de SHA1 o SHA512, por ejemplo, es muy similar sólo que cambian los tamaños de bloques, el número de variables internas y los valores de inicialización del algoritmo. Aunque pongamos una entrada de datos de poco tamaño, SHA256 opera y devuelve 256 bits, lo cual se nos hace un poco grande y tedioso de trabajar para el ejemplo, así que se explicarán los pasos sin poner ningún ejemplo concreto. Las directrices de SHA256 están detalladas y estandarizadas en NIST [80]. Si tenemos una entrada de cualquier tipo de datos que no sea binario, lo primero que deberemos hacer es convertir esa entrada de datos a binario y contar cuántos bits tenemos de entrada. Una vez hecho esto, podremos pasar a la fase de preprocesamiento, en donde pondremos la entrada de datos de tal forma que su tamaño en bits sea un múltiplo de 512 bits. Imaginemos que nuestra entrada de texto es Universidad de Alcalá, utilizando ASCII podemos pasar esa entrada de datos a binario, nos da un número de 168 bits, 21 bytes. En este caso trabajaremos con un bloque, por tanto, para nuestro tamaño de 168 bits, el múltiplo más cercano de 512 bits es 512, y en este bloque de 512 bits podremos meter 448 bits de mensaje (veremos en un momento el motivo). Si nuestro mensaje tuviese 500 bits de tamaño, tendríamos que utilizar dos bloques, en total tendríamos en tamaño de bloques 512 bits + 512 bits, pero de esos 1024 bits solo podríamos utilizar 960 bits de mensaje (como antes, se verá en un momento el motivo). Como vemos, no llegamos al mínimo de 512 bits, así que necesitamos añadir el relleno o padding. En este caso lo que se hace es añadir un bit a 1 al final de nuestros bits (tenemos ahora 169 bits) y añadir ceros hasta llegar al tamaño de 448 bits. Los últimos 64 bits hasta llegar a 512 bits son para saber el tamaño del mensaje, 168 en binario es 1010 1000, así que nuestros 64 bits serán 56 ceros seguidos de 1010 1000. Una vez hemos hecho esto, nuestro bloque de 512 bits con relleno tiene 168 bits con el mensaje, 1 bit con valor 1, 279 bits con valor 0, 56 bits más con valor 0 y después 8 bits con valor 1010 1000. El siguiente paso es dividir el mensaje con relleno en bloques de 512 bits, nosotros en nuestro caso solo tenemos un bloque así que N es 1. A continuación, dividiremos el bloque de 512 bits en 16 palabras de 32 bits de tamaño, desde M_0 hasta M_{15} . El siguiente paso es inicializar los valores iniciales Hash, estos vienen definidos en el algoritmo, pero básicamente son 8 valores desde $H_0^{(0)}$ hasta $H_7^{(0)}$, estos valores corresponden con los primeros 32 bits de la parte decimal de las raíces cuadradas de los primeros 8 números primos. Es decir, escogemos los primeros 8 números primos (2 para $H_0^{(0)}$, 3, 5, 7, 11, 13, 17 y finalizamos con 19 para $H_7^{(0)}$), computamos

su raíz cuadrada y nos quedamos con la parte decimal, y de esa parte decimal la pasamos a binario y nos quedamos con los 32 primeros bits. Con esto, el algoritmo estándar asegura que son bits aleatorios. Otras constantes que utiliza el algoritmo van desde $K_0^{(256)}$ hasta $K_{63}^{(256)}$, en este caso también representan 64 palabras de 32 bits que corresponden a los primeros 32 bits de la parte decimal de las raíces cúbicas de los primeros 64 números primos. Con esto ya definido, podemos pasar a la parte de computación, en la que lo primero que tenemos es un bucle for que itera por todos los bloques, en nuestro caso solo tenemos un bloque así que realizaremos esto solo una vez. Este bucle for (for i = 1 to N:) tiene cuatro pasos que comenzaremos a describir. El primero de los cuatro pasos consiste en calcular el message schedule W_t , en este caso son 64 valores desde W_0 hasta W_{63} , y la fórmula que tiene este message schedule es, para los valores de t entre cero y quince ambos inclusive, el valor de W_t será el valor de $M_t^{(i)}$, para el resto de los valores tendremos que $W_t = \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16}$. De toda esta fórmula, lo que no sabemos son las funciones σ , la función σ_0 corresponde a $(ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x))$, y la función σ_1 corresponde a $(ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x))$. Volvemos a tener dudas al no saber qué significan las funciones ROTR y SHR. ROTR significa rotar los bits hacia la derecha, es decir, $ROTR^7(x)$ nos indica que rotemos los bits de x siete posiciones a la derecha (los bits más a la derecha rotan y pasan a estar a la izquierda). Del mismo modo, $SHR^3(x)$ nos dice que desplazemos los bits de x 3 posiciones a la derecha, de tal manera que a la izquierda entran tres ceros y a la derecha se desechan 3 bits. Las adiciones se realizan con XOR en mod 2^{32} . El segundo de los cuatro pasos consiste en asignar valores iniciales a las variables con las que trabajaremos en este bucle, las cuales son desde a hasta h. El valor de a será $H_0^{(i-1)}$, el valor de h será $H_7^{(i-1)}$. El tercero de los cuatro pasos es otro bucle for (for t = 0 to 63:) en donde se asignarán valores a las variables a hasta h según dicta el algoritmo, y también a dos variables T1 y T2. La variable T1 toma el valor de $(h + \varepsilon_1(e) + \text{Ch}(e, f, g) + K_t^{256} + W_t)$. Por su parte el valor de T2 es $\varepsilon_0(a) + \text{Maj}(a, b, c)$. Nos saltan dudas acerca de qué son las funciones $\varepsilon_1(e)$, $\varepsilon_0(a)$, $\text{Ch}(e, f, g)$ y $\text{Maj}(a, b, c)$. Las primeras dos funciones son similares a la función σ , sus valores respectivos son $(ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x))$ y $(ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x))$ respectivamente. Por su parte, la función Ch hace referencia a Choose, es decir elegir, tenemos tres cadenas de bits de igual tamaño e, f y g y comprobaremos cada bit en orden, si el bit en e es 0 escogeremos el bit de g, si es 1 escogeremos el bit de f, y así hasta evaluar toda la entrada. La función $\text{Maj}(a, b, c)$ comprueba en orden el valor de los bits de a, b y c. Si la mayoría de los bits son 0 entonces se pone un cero, si la mayoría de los bits son 1 entonces se pone un 1. En el cuarto de los cuatro pasos actualizamos el valor de las $H_0^{(0)}$ hasta $H_7^{(0)}$ sumándoles a lo que tengan respectivamente el valor de a hasta h, por ejemplo, $H_0^{(i)}$ tendrá como valor $a + H_0^{(i-1)}$. Cuando hayamos procesado todas las iteraciones, construiremos el hash uniendo los 32 bits de cada $H_0^{(n)}$. De esta forma, habremos obtenido el valor hash de SHA256(Universidad de Alcalá).

Las principales ventajas de la familia de algoritmos SHA es su amplia adopción y su seguridad, además de que ofrece variedad de tamaños de claves y se pueden calcular de manera eficiente. Estas ventajas aplican actualmente a los algoritmos SHA2 y SHA3, ya que SHA0 y SHA1 están en desuso y no recomendados debidos a sus vulnerabilidades.

Los inconvenientes de esta familia de algoritmos son las colisiones (es decir, dos valores que producen el mismo resultado hash, por tanto, se pueden evaluar esos casos para romper el algoritmo), aunque esto es un problema para todos los algoritmos hash. Como podemos ver, SHA0 es defectuoso, a SHA1 le encontraron colisiones, y SHA2 es muy utilizado actualmente, pero con el avance tecnológico los dispositivos se vuelven más rápidos y se pueden realizar más pruebas en menos tiempo para encontrar colisiones, por tanto, un inconveniente de las funciones criptográficas hash es que parecen tener una fecha de caducidad, los algoritmos hash parecen tener duración hasta que se encuentran colisiones y se explotan estas.

La familia de algoritmos SHA, y más concretamente SHA2, es muy utilizado en la actualidad y está muy estandarizado, más de hecho que su sucesor, del cual hablaremos a continuación. Este algoritmo se

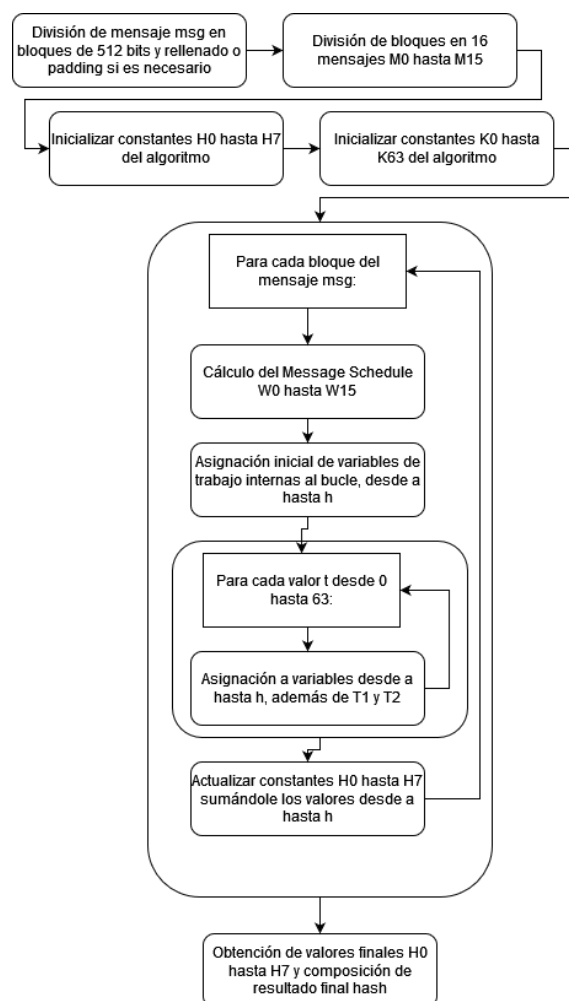


Figura 2.14: Diagrama simplificado de flujo general de SHA256.

utiliza en procesos con criptomonedas como Bitcoin, firmas digitales, comprobar la integridad o el valor de cualquier dato que se envía por Internet, como puede ser una contraseña o un archivo.

2.5.7 SHA3

SHA3 [81] es el último miembro de la familia de algoritmos SHA, este algoritmo fue publicado por el NIST en agosto de 2015 y se basa en la familia de algoritmos Keccak. Este algoritmo fue creado por Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche, nótese que Joan Daemen es cofundador del previamente explicado AES. Este algoritmo tiene su estándar FIPS, pero no está pensado para que sustituya en estos momentos a SHA2, sino para ser un complemento del mismo, de hecho, el NIST todavía no ha retirado SHA2 como estándar. Este algoritmo destaca con respecto a sus predecesores porque no sigue la estructura que tienen ellos, sino que adopta una aproximación de construcción de esponja.

En el año 2006 el NIST organizó una edición de su competición de funciones hash para crear un nuevo estándar hash llamado SHA3. Normalmente esto se realiza cuando empiezan a haber ataques peligrosos sobre el actual estándar hash que se utiliza, pero en este caso SHA2 llevaba publicado desde 2001 y no se habían demostrado esos ataques peligrosos sobre el mismo. El motivo real de esta competición por parte del NIST era tener un algoritmo listo para cuando SHA2 fuese inseguro cambiar de manera inmediata, tanto a MD5 como SHA1 les estaban encontrando colisiones y se estaban volviendo ciertamente inseguros, así que el NIST puso el parche antes de la herida y permitió 2008 se podían mandar propuestas. En 2009

se eligieron 14 algoritmos de los 51 presentados, entre ellos el algoritmo Keccak. A finales de 2010 Keccak y otros 4 algoritmos avanzaron a la fase final, estos algoritmos eran BLAKE, Grøstl, JH y Skein. NIST quería que los algoritmos funcionasen de manera rápida tanto en software como en hardware y que sean ciertamente seguros (al menos tan seguros como SHA2, pero obviamente es mejor si es más seguro que SHA2), por ejemplo, una salida de n bits debía tener una resistencia de $n/2$ bits a ataques de colisiones (encontrar dos valores que produzcan el mismo resultado hash) y una resistencia de n bits a ataques de preimagen (es decir, dado un hash, encontrar valores de entrada que lo generen), lo cual es la máxima resistencia posible. Además, se pedía que los algoritmos fuesen probados y analizados criptográficamente de múltiples maneras para que no hubiese puertas traseras ni nada extraño, y que operase en al menos cuatro modos, es decir, que devolviese al menos cuatro tamaños distintos de hash. En octubre de 2012 se anunció a Keccak como ganador de la competición, y en 2015 el NIST anunció que Keccak se había publicado como el estándar criptográfico SHA3.

Una de las diferencias notorias entre el resto de los algoritmos SHA y SHA3 es su estructura interna. Mientras las anteriores versiones de SHA seguían una estructura interna de Merkle–Damgård, SHA3 toma una aproximación de construcción en esponja. En este tipo de construcciones se tiene un estado interno finito que toma una entrada de bits de cualquier tamaño y devuelve una salida de bits de la longitud requerida. En este caso, SHA3 se podría dividir en dos grandes partes, la parte de absorción de la esponja y la parte de compresión de la misma, primero la estructura absorbe los datos del mensaje y los consume por bloques, para luego comprimir la esponja y sacar los bits que se han especificado. Este algoritmo funciona por bloques, por tanto, al igual que SHA2, el primer paso de SHA3 será añadir relleno o padding en caso de que se necesite (caso sumamente común) y establecer los valores de unos parámetros iniciales. Definiremos el valor de la variable l como un entero desde cero hasta seis ambos inclusive, la implementación principal de SHA3 utiliza el valor $l = 6$ y cuanto más alto es este valor el algoritmo es más seguro. Una vez definido el valor de l , definiremos el valor de b como el tamaño del estado, este tamaño tomará un valor de $b = 25 \cdot 2^l$, en este caso b sería $25 \cdot 64 = 1600$ bits. El estado es una matriz multidimensional de $5 \cdot 5 \cdot 5$ que serán los datos con los que trabajaremos, b puede tener tamaños de 25, 50, 100, 200, 400, 800 o 1600 bits. El número de rondas lo definiremos por $12 + 2 \cdot l = 24$ rondas. Dependiendo de la longitud de la salida (SHA3 con 224, 256, 384 o 512 bits) definiremos el valor de r o ratio = 1152 y valor de c o capacidad = 448, valor de r o ratio = 1088 y valor de c o capacidad = 512, valor de r o ratio = 832 y valor de c o capacidad = 768, o valor de r o ratio = 576 y valor de c o capacidad = 1024, respectivamente al número de bits de SHA3. La variable r o ratio define los bits de los bloques de datos que se procesan por la función en cada paso del algoritmo, la variable c o capacidad representa la capacidad restante del estado interno una vez se absorbieron los datos de entrada, y es la variable que determina en gran parte la seguridad del algoritmo, debido a que así el algoritmo mantiene un estado pseudoaleatorio en cuanto a su estado interno. Los bits de capacidad no se utilizan para la salida del algoritmo. El tamaño del mensaje tiene que ser un múltiplo del ratio, para añadir relleno simplemente se añade un uno al final del mensaje y después se añade otro uno, y entre esos dos unos añadidos añadimos tantos ceros como necesitemos hasta que el mensaje tenga el tamaño requerido. Hemos visto que el estado es una matriz multidimensional de $5 \cdot 5 \cdot 5$, y tiene distintas partes, las cuales vemos en la siguiente imagen [82] obtenida del FIPS 202 que define el estándar:

A continuación, se realizan ciertas transformaciones sobre el bloque, en concreto se llevan a cabo las funciones θ (theta), ρ (rho), π (pi), χ (chi) y ι (iota) en ese orden tantas veces como rondas tengamos, cada vez que acabemos la función ι tendremos un nuevo ratio y capacidad y aumentaremos el contador de rondas, esto se ejecuta para cada bloque de datos que tengamos. Lo que se hace en estas funciones es básicamente tomar los datos en forma de la matriz multidimensional estado y mezclarlos de manera que la salida sea totalmente distinta a la entrada y parezca aleatorio. La función θ tiene dos subfunciones,

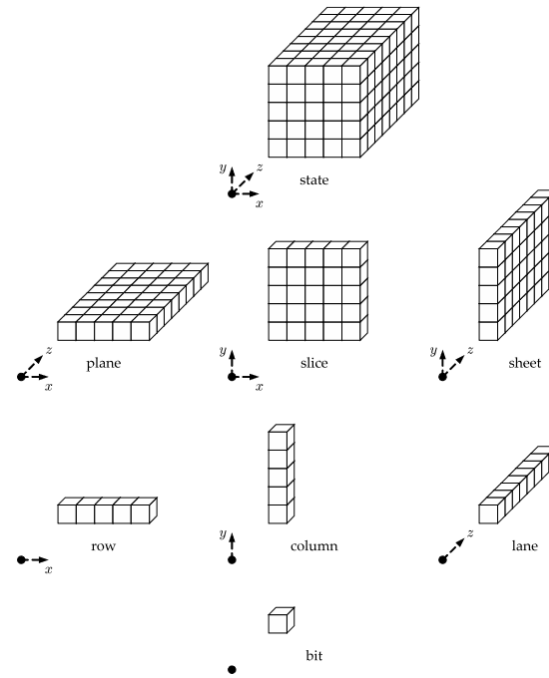


Figura 2.15: Composición de la matriz multidimensional estado.

la función C y la función D . Cada función actúa sobre algunos valores específicos, y al final lo que se logra con la función θ es realizar un XOR de cada bit en el estado con las paridades de dos columnas según su posición en la matriz estado. Con esta función obtenemos la parte de difusión necesaria para los algoritmos hash. La función ρ toma el resultado de la función θ anterior y básicamente desplazamos y modificamos los bits de su posición dependiendo de sus coordenadas en la matriz estado, estos bits los desplazamos tantas posiciones como dice una tabla que está definida en el estándar, y que los creadores han calculado teniendo en cuenta propiedades de matrices y aritmética modular, y modificamos los bits con un offset que depende de las coordenadas de la columna en el eje Z o lane en el que se encuentre el bit. La función π toma el resultado anterior y opera sobre los bits en posiciones diagonales de un slice o segmento, y lo que hace es reorganizar las posiciones de las columnas en el eje z (o lanes) en la matriz estado, según fórmulas que definen en el estándar. La función χ toma el resultado anterior y realiza operaciones lógicas con los operadores XOR, NOT y AND según el estándar con los bits en el eje X , y se aplica a cada plane o plano de la matriz, se modifica el valor del bit con una función con estos operadores que utiliza otros dos bits en su misma fila. La función ι finalmente recoge el estado anterior y realiza una operación XOR sobre la columna en el eje Z o lane en las coordenadas $0, 0$. Es la única columna que se ve afectada por esta función de las 25 columnas totales que hay y se hace con el objetivo de hacer depender esa columna de la ronda en la que estemos. Todo esto se realiza tantas veces como rondas haya, hasta obtener unos nuevos valores de ratio y capacidad que componen el nuevo estado. De este nuevo estado, el ratio se combina con otro bloque a procesar con un XOR y se vuelve a insertar junto con la capacidad de vuelta a la función. Una vez hayamos terminado con todos los bloques del mensaje llega la parte de compresión, dependiendo de los tamaños que hayamos utilizado en el algoritmo obtendremos los bits necesarios del último ratio que hayamos generado, sabemos que si queremos una salida de 256 bits tendremos un ratio de 1088 bits y una capacidad de 512 bits, lo que haremos será obtener los primeros 256 bits de esos 1088 bits del ratio y ese será nuestro valor hash resultado.

Las principales ventajas de SHA3 es que es más rápido cuando se implementa en hardware que SHA2 en el mismo escenario, aunque en software rinden de manera similar, es un algoritmo ciertamente analizado

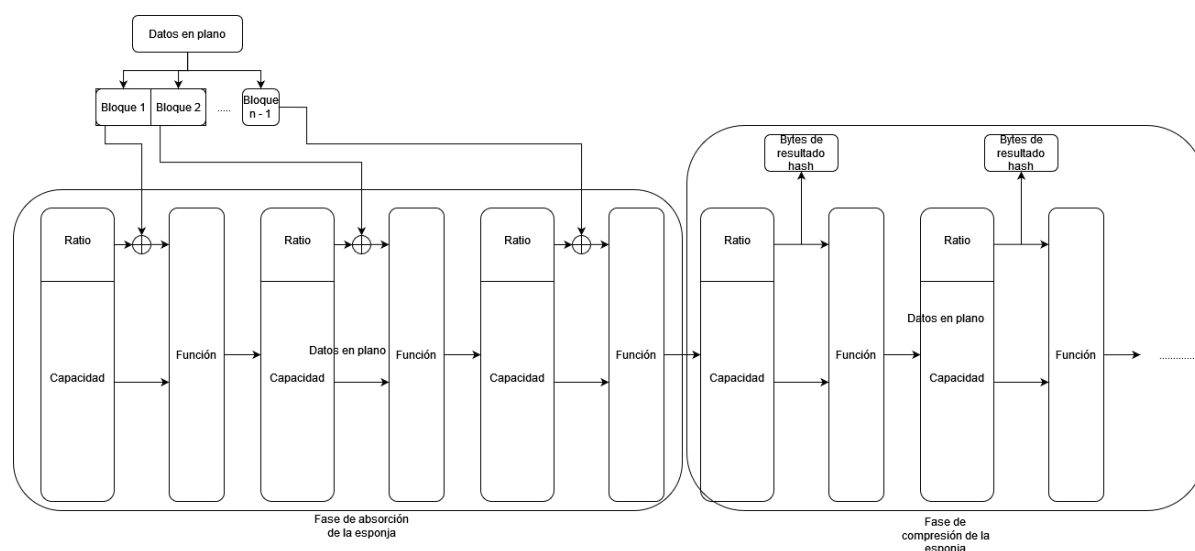


Figura 2.16: Diagrama general de SHA3.

y estudiado, ofrece diferentes tamaños de salida al igual que sus predecesores y ofrece al menos tanta seguridad y eficiencia como su inmediato predecesor SHA2.

Los inconvenientes de SHA3 es que dependiendo del tamaño del mensaje y en el caso de que sea un mensaje pequeño, la complejidad puede hacer que sea algo lento en comparación con sus predecesores. Además, al ser una estructura nueva la construcción de esponja con respecto a sus predecesores es posible que sea difícil de comprender al principio. SHA1 se publicó en 1995, y SHA2 todavía sigue vigente, siguen la misma estructura desde hace muchos años y está muy estudiada y explicada, SHA3 innova en este aspecto.

Las principales aplicaciones de SHA3 es en comprobar integridad de los datos y puede ser utilizado en funciones de derivación de claves. La criptomoneda Ethereum utiliza este método, frente a Bitcoin que utiliza SHA2, y también se puede utilizar en firmas digitales.

2.5.8 PBKDF2

PBKDF2 [83] son las siglas de Password-Based Key Derivation Function 2, es decir, segunda función de derivación de claves basada en contraseñas. Este algoritmo es parte del estándar de criptografía de clave pública de RSA Laboratories (PKCS), la empresa privada de criptografía y seguridad de los creadores del algoritmo RSA: Ronald Rivest, Adi Shamir y Leonard Adleman; y también está publicado en el RFC2898 (original) y RFC8018 (versión actual). Es un algoritmo publicado en 2017, sustituye a su predecesor PBKDF1 el cual tenía el problema [84] de estar limitado a salidas de datos de 160 bits debido a que utilizaba funciones hash como MD2, MD5 o SHA1, consideradas actualmente rotas, y está recomendado para hashing de contraseñas, ya que deja de utilizar las funciones hash de su predecesor para implementar SHA2.

Se trata de un algoritmo de derivación de claves con un coste variable, estos algoritmos se utilizan para obtener una mayor resistencia frente a los ataques de fuerza bruta. Podemos verlo como un algoritmo hash que ha sido diseñado para que pueda volverse más lento de manera intencionada. Con esto, pese a que se puedan realizar mayores ataques de fuerza bruta con el paso del tiempo debido a la rapidez añadida de la nueva tecnología, los datos se mantienen seguros en ese aspecto ya que se puede modificar un parámetro que afecta al tiempo de computación de la función. El objetivo de este tipo de funciones de derivación de claves basadas en contraseñas es, dada una contraseña, utilizar una función pseudoaleatoria junto

con la contraseña introducida y una sal criptográfica repetidas veces para derivar una clave criptográfica segura, de tal manera que esa clave segura y fuerte se pueda utilizar en otros algoritmos criptográficos. Este tipo de algoritmos tienen un funcionamiento similar, principalmente debido a que suelen tener los mismos parámetros troncales. Para este caso, la función de derivación de claves PBKDF2 toma cinco parámetros principales, de tal manera que la clave derivada es el resultado de la función PBKDF2 con los parámetros PRF, contraseña, sal criptográfica (NIST recomienda 128 bits), iteraciones y longitud deseada. Describiendo los cinco parámetros, PRF es la función pseudoaleatoria, contraseña es la entrada que le damos a la función PBKDF2, la sal es una secuencia de bits aleatoria y diferente por cada contraseña, las iteraciones son el número de veces que se repetirá el proceso y longitud deseada son el número de bits que se desean para la clave derivada. La función pseudoaleatoria que utiliza PBKDF2 es un código de autenticación de mensajes basado en hash con claves, el cual utiliza la función criptográfica hash SHA2, tenemos que explicar qué es esto.

Recordemos que existen dos tipos de cifrado según los algoritmos, estos son el cifrado por bloques y el cifrado por flujo. El primero dividía la entrada en bloques de cierto tamaño para que el algoritmo los procese, el segundo opera con los datos de entrada bit a bit. Cuando nos comunicamos por Internet mandamos los datos encriptados para que nadie los pueda ver, esto tiene sentido, pero surge un problema y es que el hecho de que los mensajes entre un emisor y un receptor no se puedan ver no significa que alguien que esté espiando la comunicación no pueda manipular los mensajes encriptados sin desencriptarlos, y esto podría potencialmente tener consecuencias en los datos en plano de los mensajes. En cifrados por bloque esto es improbable que pase ya que si cambias un bit de los datos cifrados, aunque tengas la clave criptográfica que se utilizó, ya no podrás desencriptar los datos. Sin embargo, este sí que es un problema de los cifrados en flujo, ya que estos van bit a bit. Una aproximación que se nos ocurriría para enviar mensajes y saber si se han modificado por el camino o no es enviar el mensaje y añadirle el hash del mismo, de tal manera que el receptor puede recalcular el hash del mensaje recibido y puede así ver si pasó algo durante la comunicación, sin embargo, esto no impide que alguien modifique el mensaje y recalculé el hash. Por tanto, otra aproximación es que el emisor y el receptor compartirán una clave secreta y el emisor enviará el mensaje junto con el hash de la clave secreta y el mensaje. Con esto, no se podría recalcular el hash como antes debido a que no se posee la clave secreta (teóricamente). Sin embargo, hay ciertos tipos de ataques [85] que con saber la longitud de la clave utilizada podrían comprometer esto, además de que las funciones hash utilizadas tienen un estado interno que se podría averiguar para el momento en el que se obtuvo el hash de la clave. Para esto lo que tenemos son (K)HMACs [86], o códigos de autenticación de mensajes basado en hash con claves. Lo que se hace es derivar dos claves k_1 y k_2 a partir de la clave original k . Estas dos claves se derivan teniendo las constantes hexadecimales `inner_pad = 0x36` y `outer_pad = 0x5C`, las cuales se eligen así debido a son valores que se consideran que tienen una distancia de Hamming [87] suficientemente grande, es decir, es el número de bits que tienen que cambiarse para obtener de una palabra válida otra palabra válida, si dos palabras tienen una distancia Hamming de 4, se necesitan 4 errores para convertir una en la otra. Lo que haremos será obtener el hash del mensaje con k_1 , y ese resultado lo juntaremos con k_2 y obtendremos el valor hash. Este valor resultante lo enviaremos junto con el mensaje, y el punto de esto es que el atacante no puede realizar el mismo ataque que en la anterior aproximación debido a que no se sabe el estado interno de la función hash cuando terminó de realizar el hash del mensaje junto con k_1 . Con este sistema sabremos que el mensaje no ha sido modificado durante la comunicación entre el emisor y el receptor. HMAC pueden utilizar múltiples funciones criptográficas hash, por ejemplo, HMAC-SHA256 denota la utilización de la función criptográfica hash SHA256.

Una vez visto el concepto de (K)HMAC, el funcionamiento de PBKDF2 es más sencillo de entender. La clave derivada CD se compone de la unión de múltiples bloques generados por la función pseudoaleatoria,

los llamaremos T_n . Cada bloque T_n lo calcularemos con la función F a la que le daremos como entrada la contraseña, la sal criptográfica, el número de iteraciones c y el orden i . La función F es la combinación XOR de c iteraciones de la función HMAC-SHA256 encadenada, es decir, el resultado de F con esos parámetros lo denotaremos como la combinación XOR de tantos resultados parciales U como iteraciones c tengamos. Los resultados parciales U son la función pseudoaleatoria a la que le pasamos la contraseña y el resultado anterior U . Por ejemplo, U_1 es la función pseudoaleatoria a la que le pasamos como parámetros la contraseña y la sal criptográfica unida con el valor entero de 32 bits en orden Big Endian del orden en el que estemos (i tiene valores desde 1 hasta las iteraciones c), U_2 por su parte es la función pseudoaleatoria a la que le pasamos la contraseña y el resultado U_1 , y así sucesivamente.

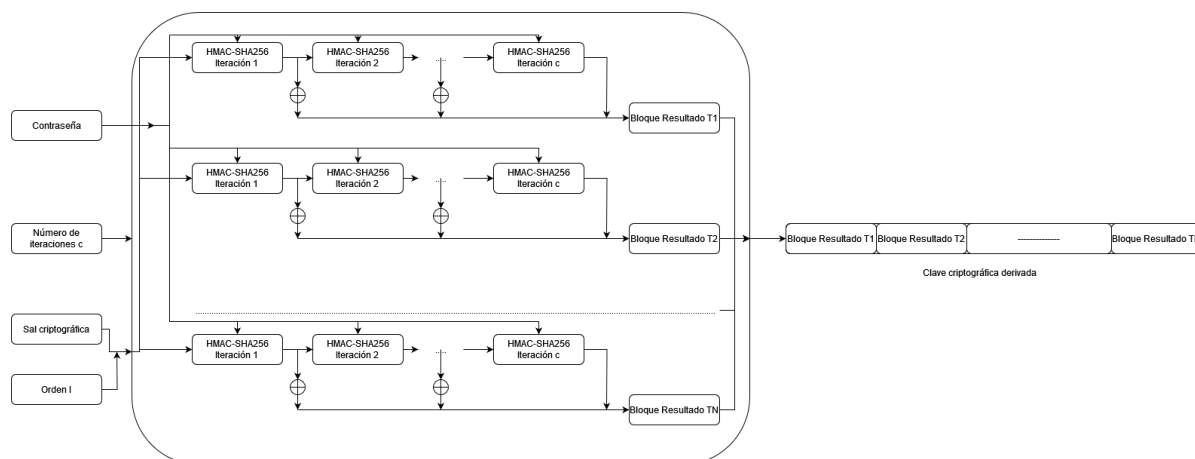


Figura 2.17: Diagrama de funcionamiento de PBKDF2.

Las ventajas de PBKDF2 son sus características principales, poder derivar claves criptográficamente seguras a partir de contraseñas, su eficiencia debido a que es sencillo implementar esta función tanto en software como en hardware y su seguridad actual.

Algunos inconvenientes de PBKDF2 es que es una función computacionalmente exigente, por tanto, hay que regular los parámetros de entrada con cuidado para que no afecte a los usuarios válidos, pero si afecte a los atacantes. Además, un problema de PBKDF2 es que su seguridad depende completamente de la función criptográfica hash que utilice, ahora mismo utiliza la función SHA256, la cual no tiene ataques relevantes o peligrosos, pero en cuanto se encuentren colisiones importantes en SHA256 y este algoritmo se rompa al igual que pasó con SHA1, en ese momento PBKDF2 dejaría de ser seguro y utilizable. Un inconveniente importante es que, al ser una función fácilmente implementable en hardware, se pueden implementar en ASIC (son chips para un uso concreto, se dedican a una tarea) de manera sencilla y barata que traten de romper la función criptográfica hash que sostiene la seguridad de PBKDF2.

Las aplicaciones principales de PBKDF2 son en derivación de claves para usar en otras aplicaciones como gestores de contraseñas, además de en VPNs o en diferentes certificados de comunicaciones inalámbricas como WiFi, programas de compresión de archivos como WinZip, entre otros [88].

2.5.9 Argon2

Argon2 [89] es una función de derivación de claves KDF diseñada por Alex Biryukow, Daniel Dinu y Dmitry Khovratovich, criptógrafos de la Universidad de Luxemburgo. En 2013 se anunció una competición abierta de hashing de contraseñas [90], similar a las competiciones para determinar el nuevo AES o SHA, pero estaba dirigida por criptógrafos directamente, aunque contaba con el apoyo del NIST. El objetivo de esta competición es dar visibilidad a la necesidad de utilizar algoritmos para hashear contraseñas y así

evitar las brechas de contraseñas que empresas que guardan las contraseñas en texto plano o encriptado. En 2015 se anunció a Argon2 como ganador de esta competición por delante de otros cuatro algoritmos que obtuvieron reconocimiento por los organizadores, estos fueron: Catena, Lyra2, yescrypt y Makwa.

El objetivo de esta función de derivación de claves es proveer una solución al problema que tienen en mayor o menor medida otras funciones de derivación de claves como PBKDF2, bcrypt o scrypt, el cual es que estas funciones se pueden implementar en chips integrados ASIC, circuitos FPGA o utilizar muchas GPUs (tienen una velocidad de cómputo muy superior a las CPUs para este tipo de cálculos) y calcular muchas veces un hash en menor tiempo. El hecho de que se pueda realizar esto no implica directamente que no se deban utilizar las mencionadas funciones de derivación de claves, simplemente indica que se puede acelerar el proceso de ataque por fuerza bruta sobre estas funciones. El aspecto clave de que se puedan implementar esas funciones en diferentes chips es que esa aproximación funciona en caso de que el cómputo realizado no necesite mucha memoria, pero en cuanto la necesitan entonces no se vuelve tan fácil. La respuesta a esto son funciones memory-hard, llamadas así del inglés, y que se refieren a funciones que necesitan gran cantidad de memoria para ser llevadas a cabo y que en caso de que se hagan con poca memoria reciben penalizaciones computacionales. Estas funciones no son fáciles de implementar, pero Argon2 toma una aproximación sencilla a estas funciones.

La explicación del funcionamiento de Argon2 es compleja, se intentará simplificar de cara a tener una idea general de su funcionamiento. Para empezar, hay varias versiones de Argon2, ya hemos visto el objetivo de Argon2, por tanto, podemos deducir que es una función que utiliza cierto tamaño de memoria para así contrarrestar las múltiples pruebas que se pueden realizar en ASICs o GPUs, entonces dependiendo de si el bloque de memoria que utilizaremos lo rellenamos de manera dependiente a los datos de entrada, de manera independiente a los datos de entrada, o tomamos una aproximación híbrida dependiendo de la fase del algoritmo tendremos diferentes versiones. Para el primer caso tenemos Argon2d, el cual resiste bien los ataques de GPU, pero es susceptible de ataques de canal lateral. Para el segundo caso tenemos Argon2i, que resiste los ataques de canal lateral, y para el tercer caso tenemos Argon2id, en este la primera mitad de la memoria se accede de manera independiente y el resto se accede de manera dependiente. Las entradas o parámetros de Argon2 son varios y se clasifican en principales y secundarios, los principales son el mensaje P y el nonce (número utilizado una única vez) S, en el caso de hashing de contraseñas serán la contraseña y la sal criptográfica respectivamente. P puede ser de tamaño 0 hasta $2^{32} - 1$ bytes, S por su parte comienza desde 8 bytes hasta el mismo número, aunque se recomiendan 16 bytes para hashing de contraseñas. Los parámetros secundarios son el grado de paralelismo p, el número de bytes de salida r, el tamaño de la memoria m, el número de iteraciones t, el número de la versión v, un valor secreto pero cambiante K, posibles datos asociados X, y finalmente el tipo de Argon2 utilizado y. El grado de paralelismo p determina cuantas partes computacionales independientes pero sincronizadas pueden ser ejecutadas, en la teoría puede tener valores desde 1 hasta $2^{24} - 1$. El tamaño de memoria m puede tener desde $8 \cdot p$ hasta $2^{32} - 1$ kilobytes. El valor actual de bloques se denota como m', el cual es m redondeado hacia abajo al múltiplo más cercano de $4 \cdot p$. El número de iteraciones puede ser desde 1 hasta $2^{32} - 1$. El número de versión es 0x13. El valor secreto puede ser desde 0 hasta $2^{32} - 1$ bytes, al igual que los datos asociados X. El tipo y es 0 para Argon2d, 1 para Argon2i y 2 para Argon2id. Lo primero que realiza Argon2 es obtener entropía [91] a través de obtener el valor hash H de todos los parámetros mencionados, algunos de ellos pueden tener una longitud variable y se les antepone antes de su valor. Argon2 después procede a rellenar la memoria con tantos bloques de 1024 bytes como m' y organizada en una matriz $B[i][j]$ de bloques con p filas o lanes y q columnas, siendo $q = m'/p$. Esa memoria es una tabla del tamaño definido y esa tabla se rellena de manera secuencial por las columnas en base al valor hash H que se ha calculado, y cada columna se rellena teniendo en cuenta lo que tenga la anterior columna. Esa tabla, como se ha dicho antes, son bloques de memoria, de tal manera que los datos de una columna

dependen de los datos de otro bloque de memoria, si estamos tratando de Argon2d ese bloque dependerá de la contraseña y por tanto utilizará diferentes posiciones de memoria para diferentes contraseñas, pero con Argon2i la posición de memoria de ese bloque será independiente de la contraseña. Ese bloque se rellenará con las funciones H y G, y se rellenan de manera distinta según la versión del algoritmo, la función H es la función hash que utiliza Argon2, en este caso se trata de Blake2b, y la función G es la función de compresión del algoritmo que toma como entrada dos bloques de 1024 bytes y devuelve uno del mismo tamaño, y que realiza permutaciones y combinaciones XOR a nivel de fila para adaptar la salida al tamaño adecuado. La tabla es rellena múltiples veces (el número de iteraciones) según el resultado de la función criptográfica hash y las funciones de compresión, y cuando se ha realizado la última iteración se extraen los bits necesarios para la clave criptográfica segura.

Resumiendo y aclarando el anterior párrafo, Argon2 genera un bloque grande de memoria (puede ser de gran tamaño, del orden de gigabytes) y lo rellena con una tabla de dos dimensiones, la cual se compone de sub-bloques de 1024 bits de tamaño. La tabla se rellena por sus columnas de manera secuencial con la función hash (H) Blake2b, la cual utiliza su correspondiente función de compresión G, de tal manera que los datos de cada columna dependan de los datos de las columnas anteriores y, por tanto, de otros bloques de 1024 bits de memoria anteriores. Los primeros dos bloques de 1024 bits tienen un contenido dependiente del valor del hash H calculado para la contraseña, la sal criptográfica y los demás parámetros de Argon2, el resto de los bloques dependen del bloque anterior y de otro bloque de 1024 bits que lo llamaremos b. Este bloque b, dependiendo del modo de Argon2, si estamos con Argon2d dependerá de la contraseña, y para diferentes contraseñas tendremos diferentes valores del bloque y estará localizado en diferentes posiciones de la memoria, en el caso de estar con Argon2i entonces el bloque es independiente de la contraseña y, por tanto, se localizará en la misma posición de memoria. El motivo por el cual se recomienda Argon2i para hashing de contraseñas es que un atacante no podrá obtener información del hash de la contraseña mirando la posición de memoria del bloque b utilizado, para otras aplicaciones como minado de criptomonedas es más apropiado utilizar Argon2d. El relleno de datos en la tabla es un proceso que puede resultar lento según el caso, la tabla se rellena (se rellena una vez y después se sustituyen los valores según corresponda) tantas veces como iteraciones se hayan especificado, y luego de eso extraeremos los bits que nos han indicado y habremos derivado una clave criptográfica segura.

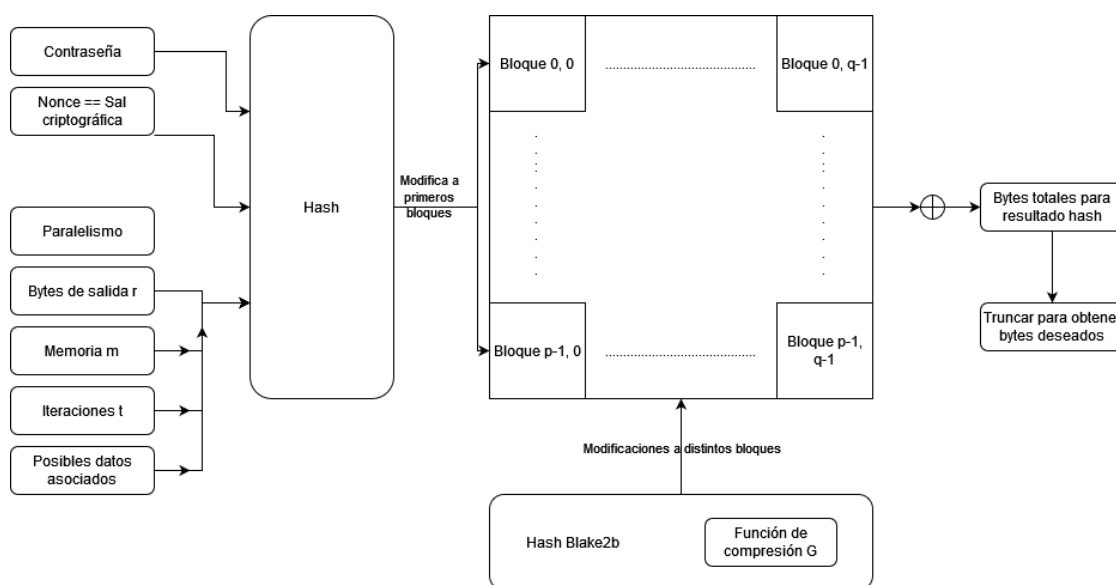


Figura 2.18: Diagrama general de Argon2.

Las principales ventajas de Argon2 son que es un proceso resistente a diferentes ataques, tanto de fuerza bruta utilizando GPUs, ASICs o FPGAs, como diferentes ataques de canal lateral o ataques de compromiso espacio-tiempo (TMTO), se puede ajustar según el dispositivo donde se utilice debido a sus múltiples parámetros (existen guías con los valores adecuados de los parámetros para tener un buen punto de partida) y en general es un algoritmo seguro y más o menos eficiente dependiendo de los parámetros que le pongamos.

Los inconvenientes de Argon2 es que es computacionalmente exigente, más que otros algoritmos de su estilo como PBKDF2 o bcrypt, además de que tiene múltiples parámetros y puede ser fácilmente configurado con errores, desprotegiendo así la contraseña de la que queremos obtener claves.

Las principales aplicaciones de Argon2 pasan por derivación de claves, guardado de contraseñas de manera segura o sistemas de prueba de trabajo (proof of work) que utilizan criptomonedas como Bitcoin o Ethereum, aunque esta última dejará de ser así para migrar a un sistema de prueba de participación.

Con este subapartado ya termina la parte de investigación y obtención de una base teórica con la que trabajar, ya tenemos las herramientas necesarias para comprender e implementar el caso práctico que se desvelará en el siguiente apartado. ¡Vamos a ello!

Capítulo 3

Desarrollo e Implementación

3.1 Introducción

En este apartado, se guiará al lector durante el proceso de diseño e implementación de un gestor de contraseñas seguro, basado en los conocimientos adquiridos con la base teórica explicada anteriormente. En primer lugar, se tomarán unas elecciones iniciales básicas y se explicará el motivo de las mismas, de tal forma que luego se especifiquen las funcionalidades principales que se desea que tenga la aplicación. Una vez tengamos este listado de funcionalidades básicas, se crearán diferentes diagramas de cara a estructurar de manera correcta la aplicación además de orientar al lector sobre esta misma estructura, y también se crearán bocetos de la interfaz gráfica de la aplicación, para tener una idea de cómo se verá. En cuanto se haya realizado y explicado todos estos diagramas y especificaciones ya se implementará la aplicación, y se mostrará cómo han sido implementadas las principales funcionalidades de la misma en el código, además de mostrar pantallazos de la interfaz de usuario final de la aplicación.

Sin más dilación comencemos con el diseño e implementación de PassGuard, un gestor de contraseñas seguro.

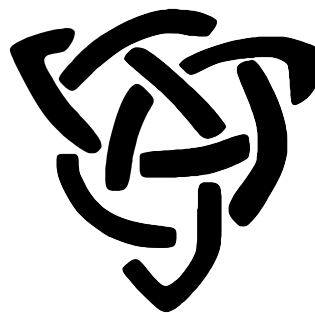


Figura 3.1: Logo de la aplicación PassGuard.

3.2 Elecciones Iniciales

A la hora de empezar un proyecto de programación, sea del tipo que sea, tenemos que tener la idea presente de qué queremos implementar y qué herramientas necesitamos para implementarlo.

Se quiere implementar un gestor de contraseñas, es decir, una aplicación en donde el usuario tendrá que introducir unos datos mínimos para autenticarse, se autenticará de cara a acceder básicamente a una zona donde tendrá una lista de las contraseñas guardadas. El usuario deberá poder realizar acciones sencillas con respecto a esa lista de contraseñas, como añadir contraseñas, poder editar los datos de una contraseña guardada y borrar una contraseña. Además, para que al usuario le entre por los ojos la aplicación y quiera utilizarla, debe tener un diseño elegante y sencillo, además de ser fácil de utilizar, ya que se puede desarrollar la mejor aplicación posible, pero si es difícil de entender o no es mínimamente decente visualmente ningún usuario querrá utilizarla. La base de la aplicación sería la descrita, y a partir de ahí se le pueden añadir múltiples funcionalidades y comodidades más, por ejemplo, poder pasar de tema claro a tema oscuro, o poder generar contraseñas aleatorias según diferentes parámetros.

Teniendo en cuenta la funcionalidad básica y principal de la aplicación, podemos empezar a ver las herramientas que utilizaremos para implementarla. La mayoría de los servicios de gestores de contraseñas vistos anteriormente, como pueden ser LastPass o 1Password, se basaban en un modelo de suscripción, en el que te creabas una cuenta y los datos de la cuenta y otros parámetros se utilizaban para generar una caja fuerte de contraseñas o Vault, que los proveedores guardaban en sus servidores en la nube. De cara a un proyecto de TFG puede ser una buena idea, sin embargo, no lo he considerado del todo conveniente. En su lugar he optado por una aproximación similar a KeePass, es decir, una aplicación de escritorio y completamente offline en lo que a vaults se refiere. Se ha preferido optar por esta aproximación debido a que ya tengo experiencia previa en el desarrollo de este tipo de aplicaciones, y además la aplicación podrá funcionar de manera completa todo el tiempo, a diferencia de una aplicación web basada en la nube en la que se tiene que pagar por los servicios que se utilizan.



(a) Logo de C#.

(b) logo de Visual Studio.

Figura 3.2: Logos de tecnologías utilizadas: Parte 1.

Para realizar una aplicación de escritorio hay múltiples lenguajes de programación disponibles con multitud de opciones. Pese a que tengo experiencia realizando aplicaciones de escritorio con Java y utilizando diferentes entornos de desarrollo integrado IDE como IntelliJ IDEA o Netbeans, se ha optado por utilizar un lenguaje nuevo para mí como es C#. Tengo cierta experiencia básica de desarrollo en lenguajes como C o C++, pero no he desarrollado absolutamente nada en C#, así que me ha parecido buena idea utilizar un nuevo lenguaje para este proyecto. Cabe destacar que hay múltiples frameworks o tecnologías para desarrollar aplicaciones en C#, me han llamado la atención dos de ellas concretamente, las cuales son WPF o Windows Presentation Foundation, y Winforms o Windows Forms. De estas dos, la principal diferencia es que el desarrollo y creación de la interfaz de usuario en WPF se realiza con un lenguaje de marcado llamado XAML, mientras que este aspecto en Winforms se maneja con C# y se asemeja más al desarrollo de aplicaciones de escritorio con Java en IDEs como Netbeans. Por este motivo, Winforms me ha convencido y será la tecnología que utilice con C# y .NET7.0 para desarrollar la aplicación. Hay múltiples entornos de desarrollo integrado en los que poder desarrollar aplicaciones en

C#, se ha elegido Visual Studio 2022 debido a que es intuitivo a la hora de diseñar la interfaz gráfica de usuario y tiene múltiples funcionalidades que facilitan de sobremanera la creación de código.

Ya tenemos la funcionalidad base de la aplicación y la gran mayoría de herramientas para empezar a programar, sin embargo, nos falta un detalle importante. A la hora de pensar en un gestor de contraseñas, pensamos en una lista de todas las contraseñas guardadas por el usuario y ciertos botones u opciones para interactuar con esa lista. Nos tenemos que preguntar cómo guardamos esa lista con los datos de las contraseñas. Esos datos tienen que ser persistentes por razones obvias, y aunque los expertos recomienden a los negocios y a particulares no guardar contraseñas y, en la medida de lo posible, subcontratar o delegar esa tarea en otra empresa especializada (de hecho, es por eso que muchos sitios web al crearte una cuenta te dejan iniciar sesión con Google u otros servicios, de tal forma que la responsabilidad de guardar la contraseña de la cuenta recae en Google o en otra empresa en vez del sitio web), en este caso estamos diseñando un gestor de contraseñas, por tanto debemos guardar los datos de las contraseñas de manera estructurada, de tal manera que podamos consultar y operar sobre ellos. Al leer estas anteriores palabras nos viene a la cabeza la idea de una base de datos. Algunos de los gestores de bases de datos relacionales más populares son PostgreSQL, MySQL o MariaDB, Microsoft SQL Server o SQLite. He decidido utilizar este último gestor ya que lo he visto muy conveniente, se trata de un gestor de bases de datos contenido. Es decir, es un archivo con extensión .sqlite, .sqlite3 o .db3 (como si fuese un archivo de texto) que representa a toda la base de datos, y a medida que añades datos a la base de datos ese archivo crece de tamaño. Esto me ha parecido muy conveniente ya que es tener todos los datos en un solo archivo sencillo y poder interactuar con ese mismo archivo de manera muy intuitiva. Este tipo de bases de datos contenidas son muy utilizadas en aplicaciones pequeñas, como pueden ser aplicaciones móviles, en nuestro caso también nos será de mucha utilidad.



(a) Logo de SQLite.



(b) Logo de DotNet.

Figura 3.3: Logos de tecnologías utilizadas: Parte 2.

En resumen, programaremos el gestor de contraseñas como una aplicación de escritorio en C# utilizando la tecnología Winforms. Programaremos la aplicación utilizando el entorno de desarrollo integrado de Visual Studio 2022 y la plataforma de desarrollo .NET7.0. Además, los datos de las contraseñas del usuario se guardarán en una base de datos de tipo SQLite3, veremos más adelante cómo protegemos este archivo ya que es fundamental para la seguridad de la aplicación.

3.3 Diseño de la aplicación y listado de principales funcionalidades

En este apartado se crearán diferentes diagramas que ayudarán a comprender la estructura y funcionamiento de la aplicación, además de listar de manera precisa las funcionalidades que tendrá la misma.

3.3.1 Principales funcionalidades de la aplicación

Procedemos a redactar de manera precisa las funcionalidades principales que tendrá la aplicación. Empezaremos por las funcionalidades básicas:

1. La aplicación permite crear una nueva caja fuerte de contraseñas. Esta caja fuerte de contraseñas por detrás será la base de datos SQLite3 con las contraseñas, al crear la nueva caja fuerte deberá crearse una tabla en SQLite3 donde se guarden los datos de manera encriptada y ordenada. Los gestores de contraseñas suelen tener una contraseña maestra, esta contraseña maestra desbloquea la caja fuerte de contraseñas y la persona puede acceder a sus contraseñas guardadas. En este caso le pediremos al usuario una ruta en donde guardar el archivo de la caja fuerte de contraseñas, el nombre que desea ponerle a su caja fuerte, y además deberemos pedirle ciertos datos o credenciales que nos ayuden a derivar una clave criptográfica con la que encriptar y desencriptar la caja fuerte de contraseñas. Es importante que la aplicación no guarde la contraseña maestra, ya que en caso de ataques se desencriptarían todas las cajas fuertes del usuario, si no se guardan las contraseñas maestras el atacante deberá adivinar la contraseña maestra, y por nuestra parte haremos nuestro mejor intento para ponérselo tan difícil que no le salga rentable.
2. La aplicación permite insertar las credenciales que se utilizaron para crear una caja fuerte de contraseñas junto con la ruta a la misma con el objetivo de desencriptar esa caja fuerte y acceder a las contraseñas guardadas. Una vez se inserten las credenciales, se realizará algún proceso con la contraseña maestra para verificar que es correcta (ya que la aplicación no debe guardarla) y en caso de que desencripte la caja fuerte entonces se le mostrarán los datos al usuario, de lo contrario, se informará al usuario de que sus credenciales no son válidas para esa caja fuerte.
3. En cuanto el usuario haya introducido correctamente las credenciales, se accederá a una vista en la que se muestran todos los datos de las contraseñas guardadas. En este punto, la aplicación proveerá vistas y herramientas para interactuar con la caja fuerte de contraseñas, ya sea para añadir una contraseña nueva, editar una contraseña existente o borrar una contraseña existente en la caja fuerte.

Una vez hemos definido lo básico de la aplicación, podemos empezar a definir más funcionalidades que no son fundamentales, pero que pueden estar muy bien de cara a aumentar la experiencia de usuario y su funcionalidad:

1. Ya que estamos tratando con contraseñas y guardarlas de manera segura, se proveerá una funcionalidad para crear contraseñas aleatorias y seguras. En esta parte el usuario podrá elegir los caracteres que desea que tenga su contraseña (letras minúsculas, letras mayúsculas, números y símbolos) además de la longitud de la contraseña y cuántas contraseñas quiere generar.
2. Prácticamente todas las aplicaciones y sitios webs lo tienen, y hoy en día parece indispensable, se trata de que el usuario pueda elegir el tema claro u oscuro para la aplicación.
3. Un aspecto importante de cara a la comodidad de uso de la aplicación es, dentro de la lista de contraseñas, poder hacer clic sobre la fila con la contraseña que deseas y que se te copie al portapapeles, de tal manera que el usuario la pueda pegarla en el sitio web al que intenta acceder. Es un aditivo que ciertamente podría mejorar la experiencia de usuario.
4. Otro aspecto que quizá pueda querer el usuario es exportar sus contraseñas en PDF. Parece contraproducente querer guardar tus contraseñas en texto plano en un PDF, ya que es como si las se

guardasen en un archivo de texto plano, si alguien accede al dispositivo puede copiarse ese archivo desprotegido y tendrá acceso a todas las cuentas. Sin embargo, considero que usuarios que están menos familiarizados con la tecnología en general pueden sentir cierta seguridad al poder exportar sus contraseñas con PDF, y si pierden su contraseña maestra no pierden todas sus contraseñas, ya que tienen una copia en PDF. Por este motivo, por lo menos parece una opción que se podría dar y que los usuarios decidan si utilizarla o no.

5. Si se puede exportar una caja fuerte de contraseñas como PDF, tiene sentido dar la opción de crear un back-up o copia de seguridad de la caja fuerte, de tal manera que si un usuario tiene su caja fuerte en el escritorio y accidentalmente la borra, que pueda tener una copia guardada en otro lugar.
6. Ya que se da la opción de realizar un back-up o copia de seguridad de una caja fuerte de contraseñas, tiene sentido pensar en que se automatice este proceso. Se provee una forma de establecer para una caja fuerte de contraseñas en una ruta concreta, la opción de, periódicamente, realizar una copia de seguridad de esa caja fuerte de contraseñas en una ruta que especifique el usuario.
7. Una utilidad importante puede ser la de, dada una lista larga de contraseñas, tener un buscador para buscar una contraseña por alguna característica, sin embargo, esta característica pierde importancia o utilidad cuantas menos contraseñas se guarden en el gestor de contraseñas. Se implementará esta utilidad ya que la cantidad de cuentas que tienen los usuarios de media hace que pueda ser ciertamente utilizada.
8. Se generarán estadísticas generales sobre las contraseñas guardadas del usuario. Esto ayudaría a un usuario con muchas contraseñas guardadas a ver una distribución sobre aspectos como la longitud o composición de sus contraseñas. Como pasa con la anterior característica, pierde utilidad cuando tenemos pocas contraseñas guardadas.
9. Dependiendo de cómo se realice la interfaz de usuario, se intentará dar al usuario la posibilidad de que cambie los colores de contorno (los colores que se utilizan en botones o en diferentes componentes) a su gusto, sin embargo y repito, es dependiente de cómo se implemente la interfaz gráfica de usuario.

Con todas las funcionalidades descritas, se considera importante recalcar que, de cara a la implementación de la aplicación, mientras se estén realizando operaciones sobre la caja fuerte de contraseñas debe estar descriptada el mínimo tiempo posible, y que en cuanto se terminen las operaciones sobre la misma esta se vuelva a encriptar para guardar la seguridad de las contraseñas.

3.3.2 Diagramas de diseño de la aplicación

A continuación, se mostrarán diferentes tipos de diagramas de forma que se pueda comprender principalmente la estructura y el funcionamiento de la aplicación.

3.3.2.1 Diagrama de casos de uso

El diagrama de casos de uso facilita visualizar las funcionalidades de la aplicación y qué puede realizar el usuario. A continuación, se mostrará un diagrama de casos de uso simplificado, es decir, se mostrarán las principales funcionalidades que puede realizar el usuario, pero se omitirán en el diagrama ciertos casos de uso secundarios, aunque estos se nombrarán y comentarán en la explicación de los casos de uso. Debido a que el diagrama es ciertamente grande, se ha dividido en tres subdiagramas para su mejor visualización. A continuación, se muestran los subdiagramas de casos de uso:



Figura 3.4: Diagrama de casos de uso simplificado: Parte 1.



Figura 3.5: Diagrama de casos de uso simplificado: Parte 2.



Figura 3.6: Diagrama de casos de uso simplificado: Parte 3.

A continuación, se explicarán de forma numerada cada caso de uso del diagrama expuesto anteriormente:

1. Crear una contraseña aleatoria: Este caso de uso incluye el caso de uso número dos, y representa la acción del usuario de presionar el botón de crear una nueva contraseña en la vista principal, se le dirigirá a una vista donde podrá elegir los parámetros y generar las contraseñas que desee.
2. Elegir caracteres para generar contraseña: Representa la acción de marcar o desmarcar las casillas de letras en minúscula, letras en mayúscula, números y símbolos, además de la longitud y cantidad de contraseñas a generar y una característica de las mismas que se explicará en la parte de implementación.
3. Obtener ayuda de cómo funciona la vista de crear una contraseña aleatoria: Representa las acciones de pulsar dos botones en la vista de creación de contraseñas aleatorias y que salgan sus respectivas ventanas con información en formato texto sobre la característica de las contraseñas que veremos más adelante, o de cómo funcionan los símbolos en las contraseñas generadas.
4. Copiar contraseñas generadas al portapapeles: Las contraseñas generadas saldrán en orden en un cuadro de texto, este caso de uso representa la acción de copiar el contenido de ese cuadro de texto al portapapeles del usuario.
5. Crear un nuevo Passguard Vault: Representa la acción del usuario pulsando el botón de crear una nueva Passguard Vault en la vista principal de la aplicación, se le aparecerá una vista en donde tendrá que poner, entre otros datos, su contraseña maestra y la ruta donde desee guardar la Passguard Vault, al final del todo le dará a crear y en caso de que haya algún dato inválido se le notificará, de lo contrario se le notificará con una ventana los datos que debe guardar obligatoriamente para poder acceder a su nueva Vault, además de alguna otra información importante.
6. Guardar correo electrónico: Este caso de uso es una extensión del caso de uso número 5, y representa la acción del usuario pulsando el botón de guardar correo electrónico en la vista activa en el momento. Esto lo que hará es que guardará persistentemente, es decir, para futuras ejecuciones el valor del correo electrónico que haya en el campo de texto del correo electrónico.
7. Cargar correo electrónico guardado: Representa la acción de cargar el correo electrónico guardado en el campo de texto asociado al correo electrónico.
8. Cargar Security Key guardado: Representa la acción de cargar el Security Key guardado en el campo de texto asociado al Security Key. El Security Key es un campo que veremos qué es exactamente y cómo funciona en la parte de implementación de la aplicación.
9. Guardar Security Key: Representa la acción de guardar el Security Key escrito en el campo de texto asociado a eso de manera persistente, es decir, para futuras ejecuciones de la aplicación.
10. Cargar un Passguard Vault existente: Representa la acción del usuario pulsando el botón de cargar Passguard Vault en la vista principal de la aplicación, esto le mostrará otra vista en la que tendrá que introducir la ruta de la Passguard Vault que desea cargar junto con las credenciales de acceso a la misma, que en la parte de la implementación veremos qué parámetros son necesarios.
11. Ver todas las contraseñas guardadas en el Vault: Una vez iniciado sesión en nuestra Passguard Vault, inmediatamente se nos mostrará una tabla con todas las contraseñas guardadas en la misma.

12. Añadir una contraseña al Vault: Representa la acción de pulsar en el botón de añadir contraseña cuando hemos iniciado sesión en nuestra Passguard Vault. Se nos mostrará una ventana en la que tendremos que rellenar los datos de una nueva contraseña, y en caso de que le demos al botón de enviar y haya datos inválidos o ya haya una contraseña en la Passguard Vault con esas características se le notificará al usuario, de lo contrario se añadirá la contraseña y se volverá a la anterior vista de la tabla con todas las contraseñas guardadas.
13. Borrar todas las contraseñas del Vault: Representa la acción de, una vez estamos en la ventana de borrar una contraseña, hacer clic en la casilla de borrar todas las contraseñas. Al hacer esto, se nos deshabilitarán el resto de los controles y nos dejará únicamente activo el botón para borrar todas las contraseñas. Al darle a ese botón nos saltará un aviso de que es una acción no reversible y nos pedirá si queremos continuar, y en caso de querer se borrarán todas las contraseñas y volveremos a la vista de tabla de contraseñas, de lo contrario nos quedaremos en la ventana de borrar una contraseña.
14. Exportar tus contraseñas en formato PDF: Este caso de uso incluye al caso de uso número 10, y representa el hecho de, en caso de no haber iniciado sesión en la Passguard Vault, en el menú de opciones darle clic a esta opción, rellenar las credenciales y la ruta de la Passguard Vault y exportar los datos; en caso de haber iniciado sesión en la misma es la representación de hacer clic en el botón de exportar. Este PDF se nos guardará en la carpeta Documentos del sistema operativo Windows OS.
15. Obtener ayuda de cómo interactuar con la vista de contraseñas: Representa la opción de hacer clic en el botón de ayuda en la vista de la tabla de contraseñas. Al hacer esto nos saltará una nueva ventana con información en formato texto describiendo la vista una vez hemos iniciado sesión en nuestra Passguard Vault y explicando cómo interactuar con ella.
16. Copiar el valor de una celda al portapapeles: La vista con la tabla de contraseñas guardadas es una tabla compuesta por botones, el texto de esos botones son los datos de las contraseñas guardadas, si hacemos clic en algunos de esos botones se nos copiará el texto de esos botones al portapapeles, algunos botones son para eliminar contraseñas de manera rápida y no copian nada en el portapapeles.
17. Resetear el cuadro de texto de búsqueda de contraseñas: En la vista con la tabla de contraseñas guardadas tenemos un campo de texto para el buscador, este campo tiene asociado dos botones, uno de ellos para realizar la acción de búsqueda y el otro es para resetear el contenido del campo de texto. En caso de darle el campo de texto pasará a no tener texto y se cargarán todos los datos guardados en la Passguard Vault con la que estemos trabajando en ese momento.
18. Marcar o desmarcar una contraseña como importante: En la vista con la tabla de contraseñas guardadas tenemos una tabla con los datos de las contraseñas, no todas las celdas de esas tablas son botones, sino que hay casillas para marcar o desmarcar contraseñas como importantes o no de manera rápida, este caso de uso representa esa acción. En cuanto se le dé a esa casilla saltará una ventana de confirmación, y dependiendo de la respuesta del usuario se marcará o se desmarcará la contraseña como importante.
19. Ordenar la lista de contraseñas por una columna: En la vista con la tabla de contraseñas guardadas tenemos una tabla con cabeceras para indicar qué significa cada columna, si hacemos clic en algunos de esos campos nos saltará una lista de opciones para ordenar ese campo por un orden que queramos, ya sea orden normal (orden de introducción en la Passguard Vault), u orden ascendente o descendente alfabético.

20. Seleccionar la columna y el orden por el cual ordenar: Este caso de uso incluye al caso de uso número veinte, y representa, una vez hemos hecho clic en la cabecera de la tabla y nos han salido una lista de opciones de orden para esa columna, la selección del orden normal, ascendente o descendente para ese ordenado.
21. Editar una contraseña existente en el Vault: Este caso de uso incluye al caso de uso número veintitrés, y representa la acción de hacer clic en el botón de editar una contraseña en la vista con la tabla de contraseñas guardadas. Este botón abrirá una nueva ventana donde podremos seleccionar la contraseña a editar, cambiarle los parámetros y guardarla correctamente. En caso de que al guardarla algún dato sea inválido se le notificará al usuario, de lo contrario se cerrará la ventana y se volverá a la vista de la tabla con contraseñas guardadas.
22. Borrar una contraseña del Vault: Este caso de uso incluye al caso de uso número veintitrés, y representa la acción de hacer clic en el botón de borrar una contraseña en la vista con la tabla de contraseñas guardadas. Este botón abrirá una nueva vista donde se podrá elegir una contraseña y eliminarla (antes de eliminarla se pedirá una confirmación previa) o da la posibilidad de borrar todas las contraseñas existentes en la Passguard Vault.
23. Seleccionar el dato con el que trabajar: Este caso de uso representa la acción de escoger en la lista desplegable de la vista en la que nos encontremos una contraseña, ya sea para editarla o para borrarla de la Passguard Vault.
24. Buscar una contraseña por su nombre: Este caso de uso incluye al caso de uso número veinticuatro, y representa la acción de introducir en el campo de texto de búsqueda el nombre de una contraseña y darle al botón de buscar. Esto lo que hará es, en la misma tabla en donde se muestran todas las contraseñas, resetearla y mostrar únicamente los resultados de la búsqueda.
25. Introducir nombre de contraseña: Este caso de uso representa la acción de escribir un nombre en el campo de texto de búsqueda de la vista con la tabla de contraseñas guardadas, y, por tanto, habilitar el botón para buscar esa contraseña.
26. Mostrar estadísticas de la lista de contraseñas: Este caso de uso incluye al caso de uso número veintiocho, y representa la acción de darle clic al botón de mostrar estadísticas en la vista con la tabla de contraseñas guardadas. Al realizar esto nos llevará a una nueva ventana donde podremos elegir las estadísticas a generar.
27. Resetear estadísticas mostradas: Este caso de uso es una extensión del caso de uso número veintiséis, y representa el hecho de resetear la lista de estadísticas que podemos generar y los gráficos que hemos generado, dejando la ventana como si hubiésemos entrado a ella por primera vez.
28. Seleccionar y mostrar estadísticas: Este caso de uso incluye al caso de uso número veintinueve, y representa la acción de seleccionar un tipo de estadísticas de una lista desplegable y después darle clic a generar, lo cual nos generará inmediatamente debajo de esos componentes unos gráficos con las estadísticas generadas y nos dará la opción de hacer clic en unos botones y descargar datos más detallados de esas estadísticas.
29. Descargar datos detallados de estadísticas generadas: Representa la acción de, una vez hemos generado los gráficos con las estadísticas seleccionadas, hacer clic a alguno de los botones debajo de cada gráfico en los que nos indica que podemos descargar más detalles de esa estadística generada. Esto lo que hará es descargarnos un archivo JSON con detalles de esa estadística generada.

30. Abrir el menú de opciones: Este caso de uso representa la acción del usuario de hacer clic en el botón con una imagen de un engranaje en la vista principal de la aplicación. Al usuario le aparecerá una lista de opciones, las cuales son crear back-ups de una Passguard Vault, exportar una Passguard Vault en diferentes formatos, cambiar el tema y colores de contorno, diferentes opciones de ejecución de Passguard o cerrar la aplicación.
31. Crear un back-up de un Passguard Vault: Representa la acción del usuario dando clic a la opción de crear back-up en la lista de opciones del menú. Esto lo que hará es aparecer una nueva ventana en donde el usuario podrá elegir la ruta de la Passguard Vault a la que le realizará una copia de seguridad y la ruta donde desea guardar dicha copia de seguridad. Una vez hecho y dado al botón de crear, se realizará una copia de seguridad (salvo que se haya realizado la misma copia en el mismo instante, entonces se avisará al usuario de ese suceso).
32. Configurar y establecer Autobackup de un Vault: Representa la acción de dar clic a la opción de configurar un auto back-up de una Passguard Vault en el menú de opciones. Al realizar esto nos aparecerá una nueva ventana en la que nos saldrá el estado actual de auto back-up, y podremos cambiar o escoger la ruta de la Passguard Vault a la que se lo activaremos, la ruta donde guardaremos los archivos de copia de seguridad y una lista desplegable para elegir la frecuencia con la que realizaremos estas copias de seguridad automáticas. También tendremos una nota en donde se leerá que estas copias de seguridad automáticas sólo se generan mientras la aplicación está en funcionamiento.
33. Exportar tus configuraciones de colores en formato PDF: Representa la acción de dar clic a la opción de exportar configuraciones de colores de contorno en el menú de opciones. Con esto nos aparecerá una ventana en la que nos pedirá en qué formato queremos exportar esos colores, si en archivo PDF o JSON. En cuanto lo elijamos se nos exportará en la carpeta Documentos del sistema operativo Windows OS.
34. Configurar Passguard para que se ejecute al iniciar el dispositivo: Representa la acción de dar clic a la opción de configurar Passguard para que se ejecute (o no) al iniciar el dispositivo en el menú de opciones. En cuanto lo hagamos, configurará la aplicación y el sistema operativo Windows para que cuando se inicie el dispositivo se inicie o no la aplicación.
35. Configurar Passguard para que se abra y se cierre minimizado: Representa la acción de dar clic a la opción de configurar Passguard para que se abra y se cierre minimizado en el menú de opciones. En cuanto lo hagamos, configurará la aplicación y el sistema operativo Windows para que cuando se cierre la aplicación, se mantenga activa en segundo plano, de tal forma que no se cierre del todo. En caso de tener la opción de que Passguard se ejecute cuando se inicie el dispositivo activada junto con esta opción, en cuanto se inicie el dispositivo se iniciará y se abrirá de manera minimizada.
36. Cerrar Passguard: Representa la acción de hacer clic en la opción de guardar los datos y cerrar Passguard en el menú de opciones. En caso de realizarlo, dependiendo de cómo esté configurada la aplicación se quedará ejecutándose en segundo plano o se cerrará completamente.
37. Cambiar el tema de fondo: Este caso de uso incluye al caso de uso número 38, y representa la acción de hacer clic en la opción de cambiar tema en el menú de opciones.
38. Elegir y guardar tema claro u oscuro: Al hacer clic en la opción de cambiar tema en el menú de opciones se nos desplegará una nueva lista de opciones en la que se nos dará a escoger entre tema claro y tema oscuro, en caso de hacer clic en cualquiera de ellas nos preguntará si el cambio de

tema queremos que se mantenga solo en esta ejecución de la aplicación o en futuras ejecuciones, el usuario elegirá según necesite y se guardarán sus preferencias.

39. Abrir el gestor de configuraciones de colores: Representa la acción de hacer clic en la opción de configuración de colores de contorno en el menú de opciones. Al hacer esto se nos abrirá una ventana con un aspecto similar a la vista con las contraseñas guardadas, pero en este caso aplicado a las configuraciones de colores guardadas por el usuario.
40. Ver lista de los colores guardados: Al abrir la vista con la tabla de colores, se podrá ver una tabla con botones en cuyo texto se muestran los datos de la configuración guardada.
41. Añadir un nuevo color: Representa la acción de añadir un color cuando hemos accedido al menú de configuración de colores. Al hacer clic nos saldrá una nueva ventana en donde se nos pedirá la información del nuevo color y podremos darle en un botón a guardar. En caso de que no haya datos inválidos se guardará, de lo contrario se notificará al usuario.
42. Borrar todos los colores guardados: Representa la acción de, una vez estamos en la ventana para borrar un color, activar la casilla para borrar todos los colores guardados. Lo que hará eso es inhabilitar el resto de los controles y habilitar el botón para eliminar todos los colores guardados, el cual si se hace clic pedirá una confirmación previa. En caso de borrar todos los colores se borrarán todos y se añadirá el color Default automáticamente.
43. Importar colores de un archivo JSON: Representa la acción de darle clic al botón de importar en la vista de configuración de colores de contorno. Este botón al hacerle clic abrirá una notificación en la que se informará al usuario de que, si importa colores de un archivo, esas configuraciones importadas se guardarán para siguientes ejecuciones y se descartará lo que haya en ese momento guardado, en caso de aceptarlo el usuario seleccionará el archivo JSON y si tiene la forma y contenido adecuado se importará correctamente a Passguard.
44. Exportar tus configuraciones de colores en formato JSON: Representa la acción de, estando en la vista de configuración de colores, darle clic al botón de exportar y escoger la opción de exportar datos como archivo JSON. Esto lo que hará es exportarnos los datos guardados de los colores de contorno a un archivo JSON que se guardará en la carpeta Documentos de Windows OS.
45. Obtener ayuda de cómo interactuar con la vista de configuración de colores: Representa la opción de hacer clic en el botón de ayuda en la vista de configuraciones de colores. Al hacer esto nos saltará una nueva ventana con información en formato texto describiendo la vista y explicando cómo interactuar con ella.
46. Resetear el cuadro de texto de búsqueda de colores: En la vista de configuración de colores hay un cuadro de texto para buscar una configuración de color por su nombre. Si se hace clic en el botón asociado para resetear el campo de texto para esta búsqueda se vaciará el campo de texto y se mostrará en la tabla de configuración de colores las configuraciones guardadas hasta el momento.
47. Marcar o desmarcar una configuración de color como favorita: En la vista con la tabla de configuración de colores no todo son botones con texto, también hay columnas con casillas para marcar o desmarcar una configuración de color específica como favorita o no. Al hacer clic en alguna de esas casillas se preguntará al usuario por confirmación y según responda se guardarán sus preferencias.
48. Obtener ayuda de cómo funciona el sistema RGB para elegir un color: Representa la acción de hacer clic en un botón en donde se nos abrirá nuestro navegador preferido y nos llevará a un sitio

web donde podremos elegir un color y saber sus valores RGB, que es el sistema de color con el que funciona Passguard para guardar las configuraciones de color.

49. Ordenar lista de colores por una columna: Este caso de uso incluye al caso de uso número veinte, y representa la acción de hacer clic a algunas de las cabeceras de la vista con la tabla de configuraciones de colores. Esto nos mostrará un desplegable para elegir el orden según sea ascendente, descendente o normal.
50. Editar un color existente: Este caso de uso incluye al caso de uso número veintitrés, y representa la acción de darle al botón de editar en la vista con la tabla de configuraciones de colores. Esto nos abrirá una nueva ventana en la que seleccionaremos la configuración a editar, podremos modificar sus valores y, al darle a guardar, si los datos son válidos volveremos a la vista anterior, de lo contrario se le notificará al usuario de los errores encontrados.
51. Borrar un color existente: Este caso de uso incluye al caso de uso número veintitrés, y representa la acción de darle al botón de borrar en la vista de configuración de colores. Al realizar esta acción se nos mostrará una nueva ventana en donde podremos elegir una configuración y borrarla, o borrar todas las configuraciones guardadas. Al darle a borrar se pedirá confirmación, y si borramos todas las configuraciones quedará activa de todas maneras la configuración de color por defecto.
52. Seleccionar y guardar una configuración de color para su uso: Este caso de uso incluye al caso de uso número cincuenta y tres, y representa la acción de darle a guardar cambios en la vista con la tabla de configuraciones. Al hacer eso se aplicará la configuración marcada para esta ejecución (se pueden ver los valores RGB en esta vista) y volveremos a la ventana anterior.
53. Seleccionar duración de uso de la configuración: Al seleccionar una configuración marcando la casilla de esa configuración en la tabla, se nos preguntará si queremos utilizar esa configuración. En caso de querer utilizarla se nos preguntará si queremos utilizarla para esta ejecución (es decir, en cuanto reinicies el programa el color cambiará al que tenías guardada para futuras ejecuciones) o para el resto de las ejecuciones. Se tomará la respuesta del usuario y se guardarán los datos según sus preferencias.
54. Buscar un color por su nombre: Este caso de uso incluye al caso de uso número cincuenta y cinco, y representa la acción de escribir un nombre de una configuración en el campo de texto habilitado para ello y después dar clic al botón de buscar. Al realizar esas acciones la tabla de configuraciones de colores se limpiará y mostrará el resultado de esa búsqueda.
55. Introducir nombre de la configuración de color guardada: Representa la acción de escribir el nombre de la configuración de color a buscar en el campo de texto para ello, habilitando así el botón para poder hacerle clic y buscar esa configuración en nuestra tabla.
56. Volver a la pantalla principal: Representa la acción de hacer clic en el logo de Passguard localizado en la esquina superior izquierda, estando en la ventana principal de la aplicación. Esto hará que volvamos a la pantalla principal de la misma, en donde aparecerá el día y hora actual y los botones habituales.
57. Obtener más información sobre el creador de la app: Representa la acción de hacer clic sobre el texto localizado en la esquina inferior izquierda de la vista principal de la aplicación, que pone Designed by martilux2580. Esto nos abrirá nuestro navegador web favorito con el perfil GitHub del creador de esta aplicación.

Con esto hemos revisado si no todas, la gran mayoría de las acciones que puede realizar el usuario en la aplicación y qué pasos debe seguir para llegar a ellas. Cabe destacar que [92] cuando un caso de uso A incluye a otro caso de uso B, nos referimos a que sin el caso de uso B el caso de uso A no podría funcionar bien, el caso de uso B es parte esencial del caso de uso A. En el caso de cuando un caso de uso A es una extensión de otro caso de uso B, nos referimos a que tanto se puede llevar a cabo el caso de uso A sin el caso de uso B como se puede llevar a cabo el caso de uso A con el caso de uso B, es decir, el caso de uso B no se tiene que cumplir para que pueda realizarse el caso de uso A, y el caso de uso B añade otras funcionalidades o permite realizar la actividad o acción que representa cuando se realiza el caso de uso A.

3.3.2.2 Estructura de la tabla de contraseñas a nivel base de datos

A continuación, se mostrará y se explicará la estructura de la única tabla que tendrá la base de datos, la cuál contendrá la información de cada contraseña del usuario:

<i>Vault</i>				
Key	Column Name	Datatype	Characteristics	Example
	URL	TEXT		
PK	Name	TEXT	NOT NULL UNIQUE	Gmail
	Username	TEXT	NOT NULL	YourEmail@gmail.com
	SitePassword	TEXT	NOT NULL	8@02cbvG29uNokCu
	Category	TEXT		
	Notes	TEXT		Email personal
	Important	TEXT		1

Figura 3.7: Diagrama de base de datos.

Para esta aplicación no necesitaremos muchas entidades y relaciones entre ellas, únicamente necesitaremos una tabla para almacenar los datos de las contraseñas del usuario. Esta tabla, como vemos en el diagrama, tiene las siguientes columnas:

- URL: El usuario guardará aquí la URL del sitio web de donde es su combinación usuario-contraseña. Este campo puede ser nulo.
- Name: El usuario almacenará aquí un nombre distintivo de su combinación usuario-contraseña. Se trata de un campo que es la clave primaria de la tabla, es decir, es un campo que debe tener un valor único para cada combinación usuario-contraseña, y que no puede ser nulo.
- Username: El usuario guardará aquí la parte del usuario de su combinación usuario-contraseña. Algunos sitios web en vez de un nombre de usuario utilizan directamente el correo electrónico, el usuario también podrá guardar un correo electrónico en este campo. Se trata de un campo que no puede ser nulo.

- SitePassword: El usuario almacenará en este campo la parte de la contraseña de su combinación usuario-contraseña. Es un campo que no puede ser nulo.
- Category: El usuario podrá asignarle una categoría a cada contraseña, con el objetivo de clasificarlas. Es un campo que puede ser nulo.
- Notes: El usuario podrá añadir unas breves notas a la contraseña, por si los datos del resto de campos no le son suficientes. Se trata de un campo que puede ser nulo.
- Important: El usuario puede marcar contraseñas como importantes con este campo, el cual puede tomar valor de 0 para contraseñas no importantes o 1 para contraseñas marcadas como importantes.

Todos los campos tienen un máximo de 2100 caracteres en su forma plana, es decir, sin encriptar. Este límite es debido a que había que limitar el tamaño de los datos para asegurar el buen funcionamiento de la aplicación y resulta que los navegadores más antiguos [93] tienen como tamaño máximo de URL alrededor de 2100 caracteres, de ahí sale el número. También cabe destacar que todos los campos son de tipo texto por un aspecto que se comenta a continuación.

En el diagrama se ve una columna con ejemplos de los posibles valores que podría tomar cada columna. Esos posibles valores son en texto plano para que se entienda correctamente la estructura, ya que en la realidad los datos de cada contraseña estarán encriptados. Si un atacante accediese a la base de datos se encontraría con una tabla de contraseñas completamente encriptadas, de ahí que el tipo de datos sea texto para la tabla de la base de datos.

3.3.3 Bocetos de vistas de la aplicación

En este apartado se mostrarán unos bocetos de una aproximación inicial de cómo se vería gráficamente la aplicación. Estos diagramas son la primera aproximación, es decir, se dibujaron como una idea sin haber redactado ni programado nada. A continuación, se muestran los bocetos:

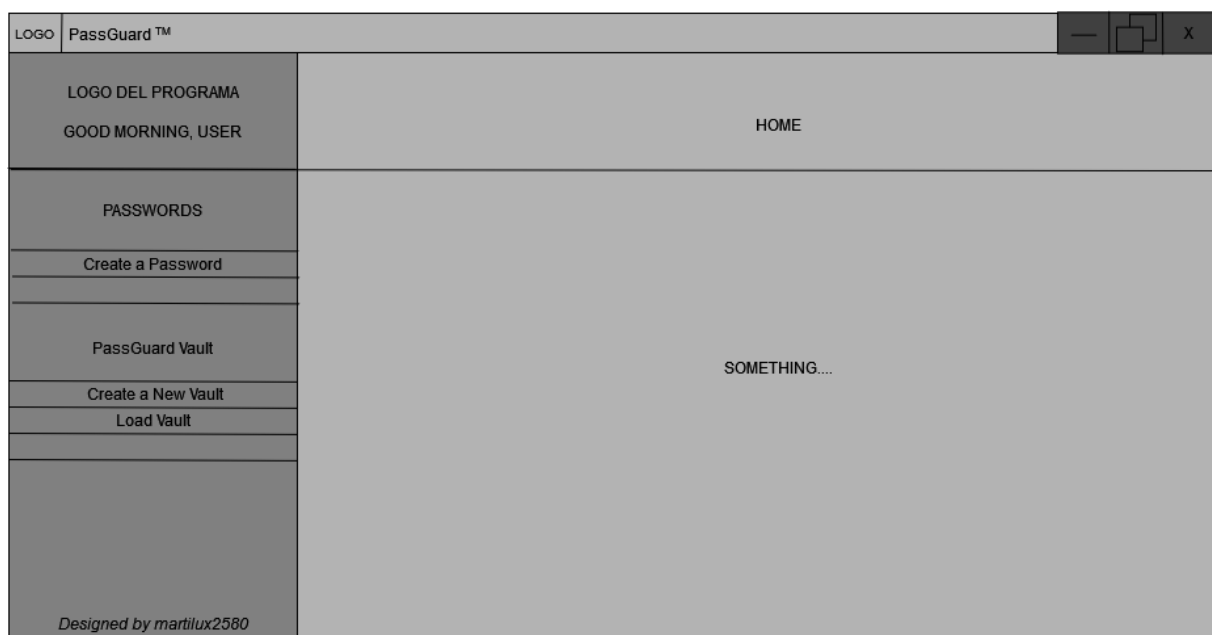


Figura 3.8: Boceto inicial de vista principal de la aplicación.

The image shows a web application interface for 'PassGuard™'. The top bar contains the logo and 'PassGuard™'. Below this, a sidebar on the left lists navigation options: 'LOGO DEL PROGRAMA', 'GOOD MORNING, USER', 'PASSWORDS', 'Create a Password', 'PassGuard Vault', 'Create a New Vault', and 'Load Vault'. The main content area is titled 'HOME' and displays a table titled 'MyVault' with columns: URL, Name, Username/Email, Password, Category, Notes, and Importance. The table contains two entries: one for 'netflix.es' with a blue star icon, and another for 'uah.blackboard.com'. Below the table are buttons for 'Reseteo', 'Buscar', 'Añadir', 'Editar', 'Borrar', 'Exportar', and 'Ayuda'. The footer of the sidebar says 'Designed by martilux2580'.

MyVault						
URL	Name	Username/Email	Password	Category	Notes	Importance
netflix.es	Netflix	netflix@gmail.com	33abudhabi	Ocio		★
uah.blackboard.com	UAH	alumno1@uah.es	alumno1	Diario		

Figura 3.9: Boceto inicial de la vista una vez se inició sesión en la Passguard Vault.

The image shows the login screen of the 'PassGuard™' application. The top bar contains the logo and 'PassGuard™'. The sidebar on the left is identical to the previous figure. The main content area is titled 'HOME' and contains three input fields: 'Email:', 'Master Password:', and 'Vault Path:'. Each field has a corresponding input box. Below the 'Vault Path' field is a 'Select Path' button. At the bottom of the main area is a 'Load Passguard Vault' button. The footer of the sidebar says 'Designed by martilux2580'.

Figura 3.10: Boceto inicial de la vista para iniciar sesión en un Passguard Vault.

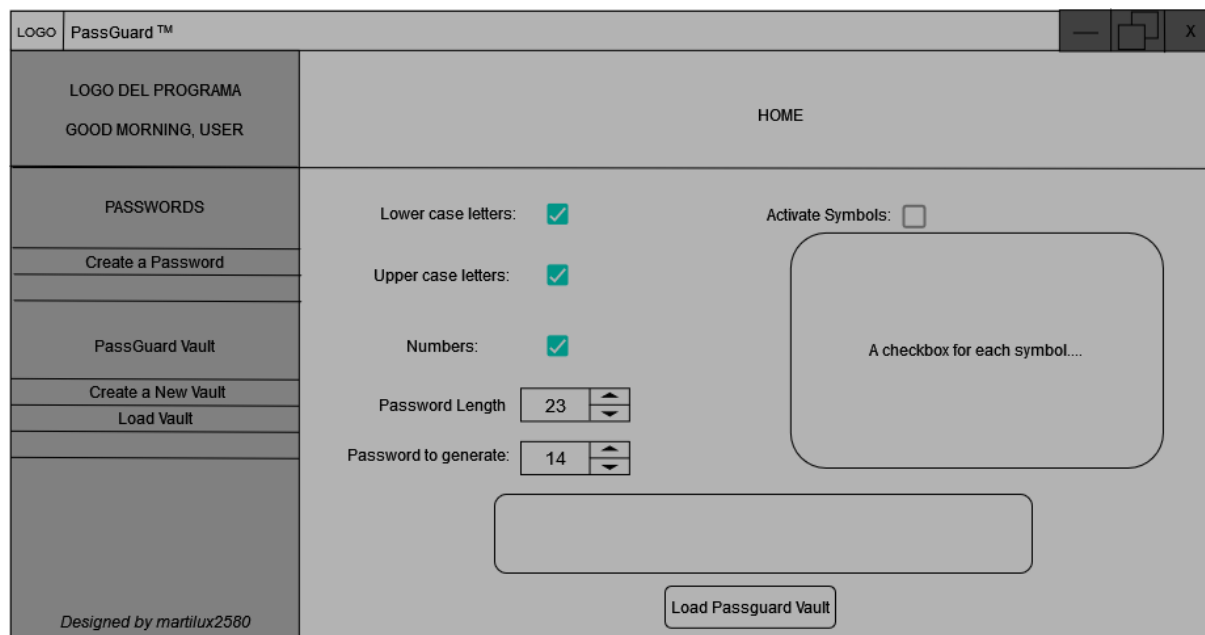


Figura 3.11: Boceto inicial de la vista para generar contraseñas aleatorias.

Esas han sido algunos bocetos iniciales de vistas de la aplicación, en un siguiente apartado se mostrará cómo han quedado las vistas finales con un mayor nivel de detalle.

3.4 Implementación de la aplicación

En este apartado se verá y se redactará con detalle acerca de cómo se han ido implementando distintas partes importantes y funcionalidades que se han ido mencionando a lo largo del documento. En anteriores apartados se mencionaron tres funcionalidades principales y nueve funcionalidades secundarias, se redactará en orden sobre la aproximación tomada para cada funcionalidad y su implementación.

3.4.1 Descripción detallada de la implementación de las funcionalidades finales de la aplicación

La primera funcionalidad principal de un gestor de contraseñas es almacenar contraseñas de manera segura, y para almacenar algo necesitaremos un lugar en donde almacenarlo. En este caso y como se dijo previamente, se utilizará una base de datos SQLite3 con una tabla donde se guardarán los datos. Uno de los aspectos importantes es proteger esta base de datos y su contenido frente a posibles atacantes, de cara a que, aunque puedan obtener el archivo con las contraseñas no sean capaces de descifrarlas. Para ello, al usuario se le piden cuatro parámetros de entrada para crear una nueva Passguard Vault, estos parámetros son un correo electrónico, un nombre para su Passguard Vault, una contraseña maestra y una ruta de guardado. El correo electrónico tiene que tener la forma de un correo electrónico válido, pero no se le mandarán correos electrónicos al usuario. El nombre debe ser un nombre válido de archivo, y la contraseña debe ser una cadena de al menos 16 caracteres compuesta de letras minúsculas, letras mayúsculas, números y símbolos, de tal forma que nos aseguramos de que es una contraseña suficientemente fuerte. Una vez se hayan escrito los parámetros correctamente, lo que haremos será crear una sal criptográfica aleatoria, la cual llamaremos Security Key. Una vez realizado ese paso, utilizaremos la función de derivación de claves PBKDF2 para derivar una clave criptográfica segura a partir de la unión

del correo electrónico y la contraseña maestra, utilizando como sal criptográfica la SecurityKey generada y la función pseudoaleatoria la función SHA256, y un número de iteraciones de alrededor de 650000 iteraciones. Ese número de iteraciones es el recomendado [18] por OWASP. Con esto derivamos una clave criptográfica segura, que será utilizada como clave simétrica del algoritmo AES256, que se utilizará para encriptar el archivo con extensión .db3, el cual es la base de datos con los datos. A su vez de utilizar PBKDF2 para obtener una clave criptográfica segura, y AES256 para cifrar la Passguard Vault, también encriptaremos los datos de la tabla de contraseñas con otra clave criptográfica segura derivada con PBKDF2 para AES256. Esto asegurará que, aunque consigas la Passguard Vault descriptada, tengas que descriptar los datos de la misma. Para derivar la clave para el contenido de la Passguard Vault seguiremos el mismo procedimiento, pero esta vez derivaremos esa clave a partir de la unión de la anterior clave que cifra la Passguard Vault junto con la contraseña maestra y el correo electrónico, y obtendremos bytes para la sal criptográfica de esta derivación a partir de la unión de la anterior sal criptográfica con la clave generada para descriptar la Passguard Vault. A continuación, un diagrama donde se ve mejor cómo se deriva cada clave criptográfica:

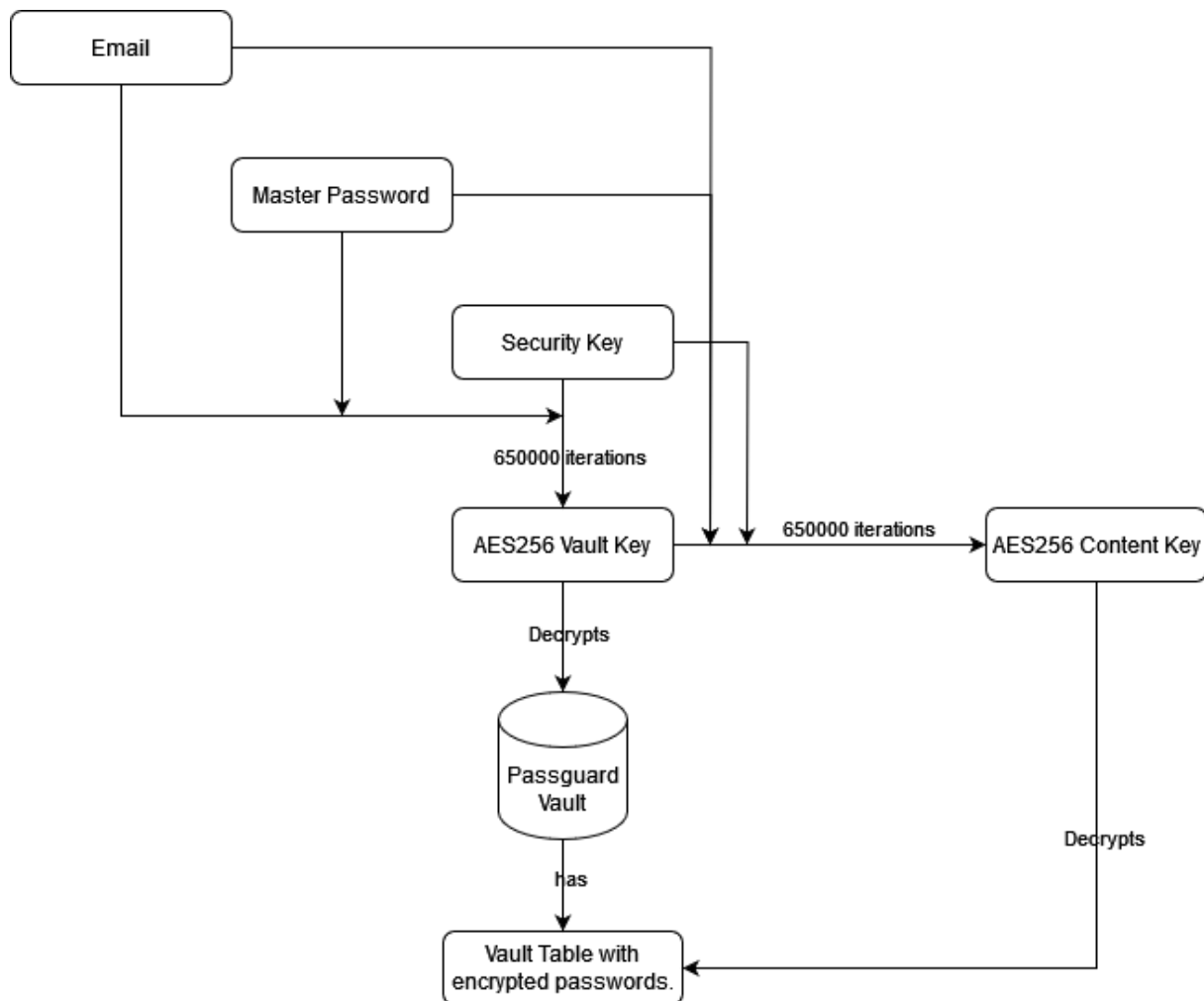


Figura 3.12: Diagrama de derivación de claves y descriptado de contenido de un Passguard Vault.

Como nota sobre las preferencias de implementación escogidas, se ha elegido AES debido a que no necesitamos claves público-privadas en este caso, nos vale con la misma clave para encriptar y descriptar, y AES es el estándar a utilizar por su rapidez, seguridad y eficiencia, de hecho, está integrado en los procesadores actuales así que es muy rápido independientemente del número de contraseñas que tenga el usuario. Por otra parte, se ha escogido PBKDF2 frente a otras funciones de derivación de claves como

Argon2id, scrypt o bcrypt. Pese a que OWASP recomienda estas tres últimas funciones por delante de PBKDF2, he escogido PBKDF2 debido a que tiene una librería nativa de Microsoft en donde se implementa, a diferencia del resto. Es muy importante que los algoritmos estén implementados de manera correcta para prevenir ataques a esta implementación, y puedo llegar a confiar con cierta facilidad en que una empresa como Microsoft ha implementado correctamente esta función de derivación de claves, sin embargo, me cuesta más confiar en que proveedores de librerías que no conozco y que implementan las otras tres funciones lo han hecho perfectamente sin ninguna falla de seguridad o puerta trasera. Para este proyecto he preferido una implementación segura de una función de derivación de claves no tan recomendada (aunque segura a día de hoy) antes que implementaciones quizá no tan seguras de otras funciones de derivación de claves más recomendadas. Aun así, en caso de no poder utilizar Argon2id, scrypt o bcrypt, OWASP da unas directrices y recomienda el uso de PBKDF2, y tales directrices como por ejemplo el número de iteraciones a utilizar se han seguido en este proyecto.

A nivel de implementación, en cuanto tenemos las cuatro entradas necesarias con datos válidos, lo que haremos es crear una base de datos SQLite vacía, que su representación va a ser un archivo con extensión .db3, y después de eso conectarnos a esa base de datos y ejecutaremos un comando SQL para crear la tabla dentro de la base de datos, la cual contendrá las contraseñas encriptadas. Una vez realizado estos pasos, lo que haremos es encriptar el archivo con extensión .db3 con la clave que hemos derivado de los datos introducidos por el usuario, de tal manera que el archivo .db3 se sustituirá por un archivo con extensión .encrypted.

La implementación de PBKDF2 en C# es sencilla, basta con llamar a la función Rfc2898DeriveBytes pasándole la contraseña, la sal criptográfica, el número de iteraciones y la función hash a utilizar. Esta implementación en forma de función tiene el siguiente aspecto:

```

1  /// <summary>
2  /// Class that implements Interface IKDF as it holds functions for PBKDF2 Key
    Derivation Function....
3  /// </summary>
4  internal class PBKDF2Function : IKDF
5  {
6      /// <summary>
7      /// Function to derive a 256 bit key (with PBKDF2) given a password and a salt.
8      /// </summary>
9      /// <param name="password"></param>
10     /// <param name="salt"></param>
11     /// <param name="bytes"></param>
12     /// <returns></returns>
13     public byte[] GetVaultKey(String password, byte[] salt, int bytes)
14     {
15         Rfc2898DeriveBytes d1; //Given bits, calculate PBKDF2 with corresponding
            hash algorithm and iterations
16         switch (bytes*8)
17         {
18             case 256:
19                 d1 = new Rfc2898DeriveBytes(password, salt, 650060, HashAlgorithmName.
                    SHA256);
20                 break;
21             case 384:
22                 d1 = new Rfc2898DeriveBytes(password, salt, 650060, HashAlgorithmName.
                    SHA384);

```



```

23         break;
24     case 512:
25         d1 = new Rfc2898DeriveBytes(password, salt, 650060, HashAlgorithmName.
                SHA512);
26         break;
27     default:
28         d1 = new Rfc2898DeriveBytes(password, salt, 650060, HashAlgorithmName.
                SHA256);
29         break;
30     }
31
32     return d1.GetBytes(bytes); //Get key...
33 }
34 }

```

Listado 3.1: Código de implementación de PBKDF2 en C#

La implementación de AES256 en C# es algo más complicada, ya que dependiendo de si lo que queremos encriptar es texto o es un archivo se implementa de manera distinta. En este paso queremos implementar un fichero con extensión .db3, así que simplemente utilizaremos la interfaz base Aes de C# para que automáticamente se divida en bloques y se vaya encriptando. La implementación de funciones para encriptar y desencriptar archivos con AES tiene el siguiente aspecto:

```

1  /// <summary>
2  /// Function to encrypt a src file into a dst file given a key with AES.
3  /// </summary>
4  /// <param name="key"></param>
5  /// <param name="src"></param>
6  /// <param name="dst"></param>
7  public void Encrypt(byte[] key, String src, String dst)
8  {
9      // Encrypt the source file and write it to the destination file.
10     using (var sourceStream = File.OpenRead(src))
11     using (var destinationStream = File.Create(dst))
12     using (var provider = Aes.Create())
13     {
14         if (key != null)
15         {
16             provider.Key = key; //Set key
17         }
18
19         using (var cryptoTransform = provider.CreateEncryptor())
20         using (var cryptoStream = new CryptoStream(destinationStream,
                cryptoTransform, CryptoStreamMode.Write))
21         {
22             destinationStream.Write(provider.IV, 0, provider.IV.Length); //Writes a
                block of bytes from offset 0 to the length.
23             sourceStream.CopyTo(cryptoStream);
24         }
25     }
26 }
27

```

```

28    /// <summary>
29    /// Function to decrypt a src file into a dst file given a key with AES.
30    /// </summary>
31    /// <param name="key"></param>
32    /// <param name="src"></param>
33    /// <param name="dst"></param>
34    public void Decrypt(byte[] key, String src, String dst)
35    {
36        // Decrypt the source file and write it to the destination file.
37        using (var sourceStream = File.OpenRead(src))
38        using (var destinationStream = File.Create(dst))
39        using (var provider = Aes.Create())
40        {
41            var IV = new byte[provider.IV.Length];
42            sourceStream.Read(IV, 0, IV.Length);
43            using (var cryptoTransform = provider.CreateDecryptor(key, IV))
44            using (var cryptoStream = new CryptoStream(sourceStream, cryptoTransform,
45                CryptoStreamMode.Read))
46            {
47                cryptoStream.CopyTo(destinationStream);
48            }
49        }

```

Listado 3.2: Implementación de encriptado y desencriptado de un archivo con AES

A la hora de generar la nueva Passguard Vault tenemos la siguiente implementación:

```

1    . . .
2    String path = VaultPathTextbox.Text + "\\\" + VaultNameTextbox.Text + ".
        db3"; //Path for the vault.
3    SqlConnection.CreateFile(path); //Create 0-byte file that will be
        modeled when it is opened, if it already exists then it is
        substituted.
4
5    query = new Query(path);
6    query.CreateNewVault();
7
8    //Vault Encryption
9    //Deal with paths for files.
10   List<String> saveVaultPath = path.Split('\\').ToList();
11   saveVaultPath[0] = saveVaultPath[0] + "\\\";
12
13   List<String> saveEncryptedVaultPath = VaultPathTextbox.Text.Split('\\').
        ToList();
14   saveEncryptedVaultPath[0] = saveEncryptedVaultPath[0] + "\\\";
15   saveEncryptedVaultPath.Add(VaultNameTextbox.Text + ".encrypted");
16
17   //Encrypt New Vault
18   //Generate random salt for AES file encryption.
19   Random rnd = new();
20   byte[] salt = new byte[16];

```

```

21         rnd.NextBytes(salt);
22         string rndsalt = Convert.ToBase64String(salt);
23         //Encrypt and delete previous decrypted file.
24         crypt.Encrypt(key: kdf.GetVaultKey(password: (VaultEmailTextbox.Text +
                VaultPassTextbox.Text), salt: Convert.FromBase64String(rndsalt),
                bytes: 32), Path.Combine(saveVaultPath.ToArray()), Path.Combine(
                saveEncryptedVaultPath.ToArray()));
25         File.Delete(Path.Combine(saveVaultPath.ToArray()));
26         . . .

```

Listado 3.3: Implementación de encriptado y desencriptado de un archivo con AES

Al terminar la creación de la base de datos y encriptación se le provee al usuario los datos que debe guardar y mantener, o de lo contrario, no podrá acceder a su Passguard Vault. Los datos provistos son el nombre de la Passguard Vault, el nombre del archivo encriptado, el correo electrónico, la contraseña maestra y la Security Key, y sabiendo los últimos cuatro de ellos podremos acceder y entrar a nuestra Passguard Vault.

La segunda funcionalidad principal es la de poder acceder a una Passguard Vault mediante las credenciales mencionadas anteriormente. Para implementar esto requeriremos los datos que hemos mencionado antes, y el proceso de implementación es sencillo. Obtendremos los datos y volveremos a realizar el proceso de derivación de claves con esos datos, si los datos introducidos son correctos, cuando derivemos la clave y la utilicemos la Passguard Vault se desencriptará, si los datos son diferentes, entonces a la hora de desencriptar los datos habrá un error y no se podrá desencriptar. La implementación es la siguiente:

```

1         . . .
2         //Calculate key to decrypt vault
3         var key = kdf.GetVaultKey(password: (VaultEmailTextbox.Text + VaultPassTextbox.
                Text), salt: Convert.FromBase64String(SecurityKeyTextbox.Text), bytes: 32);
4         . . .

```

Listado 3.4: Implementación de encriptado y desencriptado de un archivo con AES

Una vez se han dado las credenciales correctas, podemos ver cómo implementar la tercera funcionalidad principal, es decir, acceder a una vista con una tabla donde se muestran todos los datos de las contraseñas guardadas en texto plano. Este proceso de mostrar los datos en texto plano se realiza básicamente desencriptando la Passguard Vault, obteniendo todos los datos de la tabla e iterando por cada fila desencriptando los datos con la clave adecuada. Esto se ha unificado en una función llamada LoadContent(), la cual pasándole un orden y una columna nos pondrá los datos ordenados en la tabla. El campo de contraseña está censurado con asteriscos, y no se puede cambiar, sin embargo, si hacemos clic sobre esa contraseña se nos copiará al portapapeles. A continuación, se muestra la implementación principal que maneja la muestra de datos cuando accedemos a nuestra Passguard Vault:

```

1         . . .
2         else if (order == Order.Normal) //Order is normal, or it is first time
                loading content in the table...
3         {
4             List<String[]> fullResults = query.GetAllData();
5
6             VaultContentDGV.Rows.Clear(); //Clear previous content in the list and in
                the table.
7             foreach (String[] row in fullResults)

```

```

8      {
9          var tempImportant = crypt.DecryptText(key: cKey, src: row[6]); //Not to
              decrypt two times same string....
10      VaultContentDGV.Rows.Add(
11          crypt.DecryptText(key: cKey, src: row[0]),
12          crypt.DecryptText(key: cKey, src: row[1]),
13          crypt.DecryptText(key: cKey, src: row[2]),
14          String.Concat(Enumerable.Repeat("*", 15)), //Hide the password
15          crypt.DecryptText(key: cKey, src: row[4]),
16          crypt.DecryptText(key: cKey, src: row[5]),
17          tempImportant == "1" //If decrypts to "1" it is important, else is
              not.
18      );
19  }
20 }
21
22 crypt.Encrypt(vKey, (Environment.GetFolderPath(Environment.SpecialFolder.
    MyDocuments) + "\\\" + (lastValue[0] + "." + lastValue[1])), Path.Combine
    (saveEncryptedVaultPath)); //Encrypt Vault
23 File.Delete(Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
    + "\\\" + (lastValue[0] + "." + lastValue[1])); //Delete decrypted vault
24 . . .

```

Listado 3.5: Implementación de encriptado y desencriptado de un archivo con AES

Para implementar la tercera funcionalidad principal se ha seguido un proceso similar para añadir, editar o borrar una o todas las contraseñas del Passguard Vault, ese proceso es obtener los datos, validarlos y en caso de que cumplan ciertos requisitos operaremos sobre la Passguard Vault realizando los cambios, volviendo a la vista con la tabla de contraseñas y viéndose los datos actualizados. Para añadir una contraseña a la Passguard Vault se abrirá una nueva ventana en la que el usuario podrá definir todos los datos necesarios, es decir, definir la url, categoría, notas, importancia, nombre, nombre de usuario y contraseña. Las reglas para definir una nueva contraseña es que tanto el nombre como el nombre de usuario y la contraseña no sean vacíos, y que el nombre definido para la contraseña no puede ser un nombre ya guardado en la Passguard Vault, esto se comprueba pasándole la lista de nombres ya definidos a la vista para añadir contraseñas. En caso de cumplir todos esos requisitos, se desencriptará la Passguard Vault y se hará una consulta SQL para añadir un nuevo registro a la tabla de contraseñas, el cual se compondrá por los datos de la nueva contraseña encriptados. La implementación del añadido de contraseñas a la Passguard Vault tiene el siguiente aspecto en código C#:

```

1  . . .
2      if (add.AddedSuccess) //Exited add dialog from the add button, so we have
          valid data to insert. We didnt exit through AltF4 or X button.
3      {
4          String newUrl = add.Url;
5          String newName = add.name;
6          String newUsername = add.Username;
7          String newPassword = add.Password;
8          String newCategory = add.Category;
9          String newNotes = add.Notes;
10         String newImportant = add.Important;
11

```

```

12         query.InsertData(newUrl, newName, newUsername, newPassword, newCategory,
13             newNotes, newImportant); //Insert in decrypted vault.
14
15         //Encrypt the decrypted vault with the new changes (Encrypted vault now
16         has old data), so that then LoadContent decrypts it and loads
17         updated data.
18         crypt.Encrypt(vKey, (Environment.GetFolderPath(Environment.SpecialFolder
19             .MyDocuments) + "\\\" + (vaultpath[0] + ".db3")), encryptedVaultPath)
20         ; //Encrypt changes
21         File.Delete(Environment.GetFolderPath(Environment.SpecialFolder.
22             MyDocuments) + "\\\" + (vaultpath[0] + ".db3")); //Delete old data
23
24         //If there was a search ongoing, redo the search...
25         if(isSearched)
26         {
27             SearchButton.PerformClick();
28         }
29         else
30         {
31             VaultContentDGV.Rows.Clear(); //Clear previous content in the list and
32             in the table.
33             LoadContent(actualOrder, actualColumn);
34         }
35     }
36 }

```

Listado 3.6: Implementación de encriptado y desencriptado de un archivo con AES

En el caso de editar una contraseña, se accederá a una vista en donde el usuario podrá elegir la contraseña a editar a partir de una lista desplegable con los nombres de las mismas, ya que el nombre de la contraseña debe ser único para cada contraseña. En la misma vista, en cuanto se seleccione una contraseña se podrán editar los datos de la misma, y al darle clic al botón de editar se comprobarán las mismas condiciones con los nuevos datos que si estuviésemos añadiendo una nueva contraseña a la Passguard Vault. En caso de que se cumplan los requisitos, se editará la contraseña actualizándola con los nuevos datos y se volverá a la vista con la tabla de contraseñas, la cual se actualizará con los nuevos datos. La implementación del editado de contraseñas de una Passguard Vault se realiza de la siguiente manera en C#:

```

1  . . .
2      if (edit.EditedSuccess) //Exited add dialog from the add button, so we
3      have valid data to insert. We didnt exit through AltF4 or X button.
4      {
5          String newUrl = edit.Url;
6          String newName = edit.name;
7          String newUsername = edit.Username;
8          String newPassword = edit.Password;
9          String newCategory = edit.Category;
10         String newNotes = edit.Notes;
11         String newImportant = edit.Important;

```

```

12      query.UpdateData(newUrl, newName, newUsername, newPassword, newCategory,
        newNotes, newImportant, edit.GetHashofName(name: edit.
        NameToBeEdited));
13      //Encrypt the decrypted vault with the new changes (Encrypted vault now
        has old data), so that then LoadContent decrypts it and loads
        updated data.
14      crypt.Encrypt(vKey, (Environment.GetFolderPath(Environment.SpecialFolder
        .MyDocuments) + "\\\" + (vaultpath[0] + ".db3")), encryptedVaultPath)
        ; //Encrypt changes
15      File.Delete(Environment.GetFolderPath(Environment.SpecialFolder.
        MyDocuments) + "\\\" + (vaultpath[0] + ".db3")); //Delete old data
16
17
18      //If there was a search ongoing, redo search....
19      if (isSearched)
20      {
21          SearchButton.PerformClick();
22      }
23      else
24      {
25          VaultContentDGV.Rows.Clear(); //Clear previous content in the list and
        in the table.
26          LoadContent(actualOrder, actualColumn);
27      }
28  }
29  . . .

```

Listado 3.7: Implementación de encriptado y desencriptado de un archivo con AES

Si lo que queremos es borrar una contraseña de una Passguard Vault, se puede realizar de dos formas. Si hacemos clic al botón de borrar una contraseña ubicado en la tabla de contraseñas (hay un botón por cada contraseña), se nos pedirá confirmación y en caso de aceptarla se borrará esa contraseña. Se mostrarán cómo se ven esos botones de borrado de contraseñas más adelante. Esa parte tiene la misma lógica de implementación que la otra forma de eliminar contraseñas, la cual es hacer clic en el botón de borrar ubicado en la parte inferior de la tabla de contraseñas. Al acceder a la vista correspondiente veremos un aspecto muy similar (aunque diferente) a la vista de edición de contraseñas, es decir, habrá una lista desplegable donde se podrá elegir la contraseña a borrar y unos campos donde, en cuanto se seleccione una contraseña, se mostrarán los datos de la misma, aunque esos datos no se podrán editar. La vista de borrado de contraseñas es diferente visual y operativamente a la vista de edición de contraseñas debido a que en la vista de borrado de contraseñas también hay una casilla seleccionable, la cual nos permite activar un botón. Si le damos clic a ese botón, podremos borrar todas las contraseñas de golpe de la Passguard Vault. Cuando se haya pulsado el botón, ya sea borrando una contraseña o todas, nos pedirá una confirmación, ya que la acción es irreversible. En caso de aceptar esa confirmación, se desencriptará la Passguard Vault y se mandará una consulta SQL para borrar de la tabla esa contraseña en concreto, o todos los datos de la tabla, según el caso en el que nos encontremos. La implementación del borrado de contraseñas de una Passguard Vault se ha programado de la siguiente manera en C#:

```

1  . . .
2      if (del.DeletedSuccess) //If valid data is for deleting one row (we didnt
        exit through AltF4 or X button).
3      {

```

```

4      query.DeletePassword(del.NameToBeDeleted);
5
6      }
7      else if (del.DeletedAllSuccess) //If valid data is for deleting all
          contents in the Vault (we didnt exit through AltF4 or X button).
8      {
9          query.DeleteAllData();
10     }
11
12     //Encrypt the decrypted vault with the new changes (Encrypted vault now
        has old data), so that then LoadContent decrypts it and loads updated
        data.
13     crypt.Encrypt(vKey, (Environment.GetFolderPath(Environment.SpecialFolder.
        MyDocuments) + "\\\" + (vaultpath[0] + ".db3")), encryptedVaultPath);
        //Encrypt changes
14     File.Delete(Environment.GetFolderPath(Environment.SpecialFolder.
        MyDocuments) + "\\\" + (vaultpath[0] + ".db3")); //Delete old data
15
16     //If there was a search going on, redo the search....
17     if (isSearched)
18     {
19         SearchButton.PerformClick();
20     }
21     else
22     {
23         VaultContentDGV.Rows.Clear(); //Clear previous content in the list and
            in the table.
24         LoadContent(actualOrder, actualColumn);
25     }
26     . . .

```

Listado 3.8: Implementación de encriptado y desencriptado de un archivo con AES

Una de las funcionalidades secundarias que se habían visto para este proyecto es dar la posibilidad de crear contraseñas seguras, fuertes y personalizadas. Esta funcionalidad se ha desarrollado en Passguard. Un usuario puede elegir que su contraseña tenga tanto letras mayúsculas como letras minúsculas, además de números y una variedad de símbolos que el usuario puede elegir. El usuario además puede generar desde 1 hasta 512 contraseñas aleatorias, y su tamaño puede ser desde 5 caracteres o 16 caracteres hasta 52 caracteres. Se deben mencionar dos aspectos, el primero de ellos es que, aunque el usuario elija, por ejemplo, diez símbolos distintos para su contraseña es improbable que la contraseña generada contenga los diez símbolos, la mayoría de las veces contendrá uno o dos de estos símbolos. Esto es debido a que se utiliza un generador criptográfico pseudoaleatorio y se pueden dar combinaciones de parámetros para la generación de contraseñas que hagan que la aplicación tarde demasiado tiempo en cumplir la petición, por eso se recomienda generar muchas contraseñas y, de entre ellas, seguramente hay alguna que se ha generado con el símbolo que desea el usuario.

El otro aspecto a comentar es que el usuario puede generar contraseñas de un tamaño mínimo de 16 caracteres en caso de que una casilla esté activada, y de 5 caracteres en caso de que no esté activada. La casilla que determina el inicio del rango de longitud de las contraseñas es si las contraseñas que se van a generar son completamente aleatorias, entonces el mínimo es de cinco caracteres, o debemos generar contraseñas que no hayan sido encontradas en previas brechas de seguridad, nos referiremos a

ellas como contraseñas comprometidas o pwneadas por el término informal del inglés, además de por cómo se comprueba esto en Passguard. La técnica para comprobar si las contraseñas generadas se han encontrado previamente en brechas de seguridad es una aproximación sencilla de lo que hacen gestores de contraseñas actuales del mercado. Estos gestores para saber si tu contraseña ha sido comprometida lo que hacen es investigar en diferentes lugares (como la dark web) por brechas de seguridad, de tal manera que obtienen esos datos y comprueban si la contraseña se ha encontrado ahí. En este proyecto no se realizará de esa forma, pero sí que comprobaremos a través de una página web [94] disponible para todo el mundo si nuestras contraseñas han sido comprometidas, su nombre es Have I Been Pwned. Esta web es creada por el consultor de seguridad australiano Troy Adam Hunt, y lo que hace es recolectar brechas de seguridad en diferentes lugares y comprobar si algún dato que le introduzcamos, como puede ser un correo electrónico o una contraseña, se ha encontrado en alguna brecha de seguridad. En resumen, le damos nuestra contraseña a la página web y nos dirá si se encontró en una brecha de seguridad. El lector más atento se habrá dado cuenta de que esta última frase, por lo menos, parece sospechosa. Esta página podría tener algún fin maléfico, almacenando nuestro correo electrónico y contraseña podría probar en diferentes páginas web comunes (por ejemplo, Gmail, Netflix....entre otras), descifrando más combinaciones correo-contraseña de las que podríamos creer. El concepto que utiliza esta página web es el de k-anonimato [95], y lo que hace es [96] anonimizar parcialmente la contraseña. Nos centraremos en el caso de Passguard, en su página web deberían anonimizar la contraseña y después comprobarla si ha sido comprometida. Lo que haremos en Passguard es calcular los 40 caracteres que componen el valor hash de la contraseña a comprobar con la función SHA1. Pese a que dijimos que SHA1 está roto, en este caso el hecho de que se pueda crear un texto que tenga el mismo hash SHA1 que la contraseña que estemos comprobando no afecta mucho [97], además, por ejemplo, con SHA256 que es actualmente seguro, tardarían más las respuestas del servidor y no ganamos mucho en anonimidad. Una vez tengamos los 40 caracteres del valor hash de SHA1 de la contraseña a comprobar, lo que haremos es hacerle una petición HTTP a la API de Pwned Passwords, pero lo que haremos será pasarle los primeros cinco caracteres (si le mandamos cuatro puede comprometerse el servicio, y si le mandamos seis tenemos menos anonimidad), nos guardaremos el sufijo compuesto por los treinta y cinco caracteres restantes. Pwned Passwords responderá a nuestra petición HTTP con una serie de sufijos hash junto con las veces que han sido comprometidos esos hashes, siempre va a haber un mínimo de una aparición en brechas de seguridad para un sufijo dado, por cuestiones obvias. Nosotros lo que haremos es iterar por esa respuesta juntando nuestro prefijo con cada sufijo retornado, y si alguno de ellos coincide con el hash SHA1 que calculamos, entonces tenemos certeza de que nuestra contraseña ha sido comprometida. De esta forma nosotros comprobamos nuestra contraseña y los dueños de la página web no pueden saberla.

La implementación del creador de contraseñas aleatorias es sencilla, obtenemos los caracteres que desea el usuario, creamos contraseñas hasta que se genere una que cumpla con los requisitos del usuario, y en caso de hacerlo se muestra en pantalla para que la pueda copiar al portapapeles. Para el apartado de comprobar si las contraseñas han sido comprometidas, se ha realizado la siguiente implementación:

```
1  . . .
2  /// <summary>
3  /// Makes an HTTP Request (curl) with the Hash Header so that the API returns
   us all the pwned passwords that have as SHA1 header hash that hash header.
4  ///
5  /// Obtain all the hashes that start with headHash from an API of pwned
   passwords.
6  /// </summary>
7  /// <param name="headHash"></param>
8  /// <returns></returns>
```



```

9      internal static async Task<string> GetHashes(string headHash)
10     {
11         string instruction = "https://api.pwnedpasswords.com/range/";
12         string url = instruction + headHash.ToUpper();
13         HttpClient client = new();
14         try
15         {
16             //HTTP request to the API and await for the list of hashes and their
17             appearances in the API....
18             string response = await client.GetStringAsync(url);
19             return response;
20         }
21         catch (HttpRequestException) //Error control
22         {
23             return "-1";
24         }
25     }
26
27     //Compound the SHA1 of password, get all the hashes with a headhash, and if
28     the compound is in the hashes returns true as password has been pwned, if
29     not returns false
30     /// <summary>
31     /// Checks if password has been pwned by computing the SHA1 of the given pass
32     and checking if in the API that hash has been pwned in a data breach...
33     /// </summary>
34     /// <param name="password"></param>
35     /// <returns></returns>
36     /// <exception cref="Exception"></exception>
37     internal static async Task<bool> CheckPwnage(string password)
38     {
39         IHash sha1 = new SHA1Algorithm();
40
41         string hash = sha1.Compute(password, 160); //Compute SHA1
42         string headhash = hash.Substring(0, 5); //Compute first part of hash in
43         order to check hashes.
44         string PwnedHashes = await GetHashes(headhash);
45
46         if((PwnedHashes == "-1") || (String.IsNullOrEmpty(PwnedHashes))) {
47             throw new Exception(); } //No need of Else because throw Exception goes
48         out of the function.
49
50         StringReader reader = new(PwnedHashes);
51         string line, tailHash;
52         while ((line = reader.ReadLine()) != null) //Read all pwned hashes
53         {
54             tailHash = line.Substring(0, 35);
55             if (headhash.ToUpper() + tailHash.ToUpper() == hash.ToUpper()) //If match,
56             pass has been pwned before.
57             {
58                 return true;

```

```

52     }
53     }
54     return false;
55 }
56 . . .

```

Listado 3.9: Implementación de encriptado y desencriptado de un archivo con AES

La implementación del tema claro u oscuro en la aplicación es sencilla en general, por ejemplo, en el caso de la vista principal de la aplicación basta con cambiar el color de fondo de un panel. Sin embargo, en otras vistas algunos componentes como listas desplegables o cajas de texto pueden dar problemas debido a su programación, así que hay que programar para esos componentes en concreto el cambiarle el color de fondo. Cuando se inicia la aplicación se comprueba un parámetro para así iniciarla con el tema apropiado, y a medida que el usuario se mueve entre vistas se llaman dentro de las clases de esas vistas a las funciones que controlan los eventos de cambio de color de fondo (son funciones que cuando se cambia el color de fondo se ejecutan) para cambiar componentes conflictivos de color.

A la hora de implementar la funcionalidad de copia de datos de la tabla de contraseñas al portapapeles, el componente utilizado tiene eventos para saber cuándo un usuario ha hecho clic en una celda, y decírnos las coordenadas de esa celda. Nuestra tabla se compone de botones, entonces al pasar el ratón por encima de esos botones se sombrea el contenido. Una vez le demos clic a una celda, saltará la función que controla si se da clic a una celda, y ahí ubicaremos la celda y se copiará el contenido del texto del botón al portapapeles.

La funcionalidad de exportar el contenido de una Passguard Vault a un formato PDF se ha realizado de la siguiente forma. En caso de que el usuario no haya iniciado sesión en su Passguard Vault se le pedirán sus credenciales para que inicie sesión, y una vez se hayan validado esas credenciales se hará una consulta SQL para obtener todos los datos de la tabla de contraseñas, y se iterará sobre ellos para desencriptarlos y generar un archivo PDF con una tabla e ir rellenando esa tabla. Este PDF se generará en la carpeta Documentos del sistema operativo Windows OS con el nombre VaultTable-yyyymmdd-hhmmss, y la función en C# que la genera es ciertamente extensa y no se mostrará en este documento. Sin embargo, a continuación, se muestra una captura del aspecto del PDF generado:

PassGuard: Vault Content

Date: Thursday, August 31, 2023, 19:12:49

Vault Name: prueba

Vault Filename: prueba.encrypted

PassGuard Saved Email: prueba@gmail.com

PassGuard Saved Security Key (SK): LgEHYPfEAEQOkwzOsU+sqq==

Note: Saved Email and SK may not correspond to the Vault. Those values are the ones PassGuard had stored the day the backup was done.

Note2: If the column Important has value 1, it means it was saved as an important password. If it has value 0, it was not saved as an important password.

URL	Name	Site Username	Site Password	Category	Notes	Important
www.url1.es	name1	username1	password1	category1	notes1	1
	name2	username2	password2			0
www.url3.es	name3	username3	password3	category3		0

Figura 3.13: Tabla de contraseñas en formato PDF.

Para implementar la funcionalidad de crear un back-up de una Passguard Vault se accederá a una vista en la que se pedirá al usuario una ruta de la Passguard Vault a la que realizarle una copia de seguridad y una ruta en la que se guardará dicha copia de seguridad. El proceso es sencillo, se tomará la ruta del archivo con extensión .encrypted y se copiará ese archivo en la ruta provista para guardar el back-up. El proceso para realizar esta funcionalidad en código C# es ciertamente sencillo, habrá que comprobar que el archivo no exista ya en la dirección, y que esa dirección donde se quiere guardar exista en el sistema de archivos.

La cuestión de realizar copias de seguridad automáticas tiene algo más de complejidad, ya que debe ser un proceso recurrente según una frecuencia. El usuario accederá a una vista donde podrá elegir la ruta de la Passguard Vault, una ruta en donde ir guardando las copias de seguridad automáticas y una frecuencia, las cuales pueden ser cada día, cada semana, cada mes, antes de cerrar la aplicación o justo después de cualquier cambio en los contenidos del Passguard Vault. En estos dos últimos casos realizar el auto back-up es sencillo, cuando el usuario cierre la aplicación se llamará a un evento llamado FormClosing(), y ahí es donde comprobaremos si el auto back-up está activado y si la frecuencia es la indicada, en tal caso realizaremos un back-up y se le avisará al usuario.

En el caso de auto back-up para cualquier cambio en la Passguard Vault, consideraremos los cambios cuando se añaden, editar o borran datos de la misma, de tal forma que en las funciones que manejan esas situaciones, al final de las mismas se comprobará si auto back-up está activado, y en caso de estarlo y tener la frecuencia indicada se realizará la copia de seguridad automática.

Para los casos con frecuencia diaria, mensual y semanal no es tan sencillo ya que lo que debe haber por detrás es un proceso ejecutándose con Passguard que compruebe si es el momento de realizar una copia de seguridad. Se ha utilizado una Task en C#, la cual es una abstracción a alto nivel que es eficiente y provee facilidades para escribir código asíncrono. En este caso se ha programado una función que se ejecutará siempre para comprobar si entre la última fecha de realización de un back-up y este momento ha pasado el intervalo, y en caso de hacerlo crear una nueva copia de seguridad. Al iniciar la aplicación, si se tiene auto back-up activado se creará una Task cuyo objetivo es ejecutar esa función continuamente, de esta forma conseguiremos tener el auto back-up funcionando en segundo plano. Claramente y cómo se ha descrito el funcionamiento, auto back-up solo funciona cuando Passguard esté ejecutándose, en caso de tenerlo activado y no haber abierto la aplicación no se generarán copias de seguridad automáticas. Para la implementación de la Task basta con crear una nueva en caso de que las copias de seguridad automáticas estén activadas:

```

1  . . .
2      //Code for regulating AutoBackup when it is enabled and has a time
        frequency (every day, week or month).
3      int [] timeCodes = new int [] { 3, 4, 5 }; //Modes for everyday, week or
        month.
4      //If Autobackup is activated and has time frequency, start a task with the
        function to check every time if a backup has to be made.
5      if ((ConfigurationManager.AppSettings["AutoBackupState"] == "true") &&
        timeCodes.Contains(Int32.Parse(ConfigurationManager.AppSettings["
        FrequencyAutoBackup"])))
6      {
7          autobackup = Task.Factory.StartNew(() => Backup.SystemBackup.
            AutoBackupTime()); //Start a task with a method
8      }
9  . . .

```

Listado 3.10: Implementación de encriptado y desencriptado de un archivo con AES

La función que se ejecuta constantemente es ciertamente grande y en este documento no se mostrará. Sin embargo, mostraremos la parte de la función que maneja las copias de seguridad diarias y se confía en que sea fácil para el lector deducir el funcionamiento para las frecuencias semanales y mensuales:

```

1  . . .
2      var mode = ConfigurationManager.AppSettings["FrequencyAutoBackup"];
3      var pathVault = ConfigurationManager.AppSettings["PathVaultForAutoBackup"];
4      var dstPath = ConfigurationManager.AppSettings["dstBackupPathForSave"];
5      var lastDate = ConfigurationManager.AppSettings["LastDateAutoBackup"];
6      var active = ConfigurationManager.AppSettings["AutoBackupState"];
7      if (active == "true") //If autobackup is set...
8      {
9          switch (Int32.Parse(mode))
10         {
11             case 3: //Daily basis...
12                 if (DateTime.Now.Subtract(DateTime.Parse(lastDate)).TotalDays >= 1)
13                     //Difference between last backup and now is +1day
14                 {
15                     if (File.Exists(pathVault) && Directory.Exists(dstPath)) //If file
16                         and destination directory exists...
17                     {
18                         if (CreateBackup(pathVault, dstPath: dstPath)) //Create backup
19                             and set new lastdate of autobackup....
20                         {
21                             Configuration config = ConfigurationManager.
22                                 OpenExeConfiguration(ConfigurationUserLevel.None);
23                             config.AppSettings.Settings["LastDateAutoBackup"].Value =
24                                 DateTime.Now.ToString(); //Modify data in the config file
25                             for future executions.
26                             config.Save(ConfigurationSaveMode.Modified, true);
27                             ConfigurationManager.RefreshSection("appSettings");
28                             MessageBox.Show(text: "AutoBackup was created successfully.",
29                                 caption: "Success", buttons: MessageBoxButtons.OK, icon:
30                                 MessageBoxIcon.Information);
31                         }
32                     }
33                 }
34             else
35             {
36                 MessageBox.Show(text: "AutoBackup could not make a backup of
37                     the specified Vault, please try again later. \nThis
38                     message will be shown every 30 seconds until the issue is
39                     solved or AutoBackup is deactivated.", caption: "Error",
40                     buttons: MessageBoxButtons.OK, icon: MessageBoxIcon.Error)
41                 ;
42                 Thread.Sleep(30000); //30second to sleep
43             }
44         }
45     }
46     else
47     {
48         MessageBox.Show(text: "AutoBackup could not make a backup of the
49             specified Vault. Please review AutoBackup config and check

```

```

33         all the paths and files exist. \nThis message will be shown
34         every 60 seconds until the issue is solved or AutoBackup is
35         deactivated.", caption: "Error", buttons: MessageBoxButtons.
36         OK, icon: MessageBoxIcon.Error);
37         Thread.Sleep(60000); //60second to sleep
    }
}
break;
. . .

```

Listado 3.11: Implementación de encriptado y desencriptado de un archivo con AES

En caso de que no estuviese auto back-up establecido y al final de la ejecución de la vista no quiera establecerlo, o estuviese ya establecido y sigue estando establecido entonces no se hace nada. En caso de estar establecido y el usuario lo quita simplemente cortaremos la Task y, por tanto, el método de continua evaluación de las frecuencias. Por último, en caso de no estar establecido y establecerlo, iniciaremos la Task. En caso de que, con copias de seguridad automáticas activadas, la carpeta donde se guardan deje de existir o la Passguard Vault que estamos guardando se mueve de lugar, en cuanto auto back-up intente hacer una copia de seguridad no podrá y avisará al usuario de que ha habido un error, y se le mostrará este mensaje cada un minuto hasta que el error se arregle (la Passguard Vault vuelva a su lugar y la carpeta donde se guardan las copias exista) o que modifique la configuración o desactive auto back-up del menú de opciones.

La funcionalidad de un buscador de contraseñas dentro de la tabla de las mismas es bastante sencilla de implementar. Sabemos por cómo hemos diseñado la base de datos y la tabla de contraseñas que las contraseñas son únicas por el campo nombre, así que un buscador básico debe filtrar por nombre. Lo que se ha hecho a nivel de código C# es, sobre el componente de C# que representa la tabla de contraseñas, crear un filtro con el término de búsqueda, y se buscará cualquier nombre ignorando la posición del término a buscar y si viene en letras mayúsculas o minúsculas. Por ejemplo, para el termino de búsqueda ne, aparecerían tanto Netflix como eneba. La implementación en C# se ha realizado de la siguiente forma:

```

1  . . .
2  /// <summary>
3  /// Search that name in the table...applying a filter to the whole content of
4  table....
5  /// </summary>
6  /// <param name="sender"></param>
7  /// <param name="e"></param>
8  private void SearchButton_Click(object sender, EventArgs e)
9  {
10     TrimComponents();
11
12     //Reset content, so that you dont search content on the previously searched
13     ocntent....
14     VaultContentDGV.Rows.Clear();
15     LoadContent(actualOrder, actualColumn);
16
17     List<DataGridViewRow> matchingRows = new();
18
19     foreach (DataGridViewRow row in VaultContentDGV.Rows)
20     {

```

```

19         if (row.Cells["NameColumn"].Value != null)
20         {
21             string name = row.Cells["NameColumn"].Value.ToString();
22
23             if (name.Contains(SearchTextbox.Text, StringComparison.OrdinalIgnoreCase))
24             {
25                 matchingRows.Add(row);
26             }
27         }
28     }
29
30     // Display the matching rows in the DataGridView
31     VaultContentDGV.Rows.Clear();
32     VaultContentDGV.Rows.AddRange(matchingRows.ToArray());
33
34     //Enable reset and set to true the flag of searching something...
35     ResetButton.Enabled = true;
36     isSearched = true;
37 }
38 . . .

```

Listado 3.12: Implementación de encriptado y desencriptado de un archivo con AES

De cara a realizar unas estadísticas básicas sobre las contraseñas tendríamos primero que pensar en qué estadísticas querríamos generar. Hay bastantes estadísticas que se podrían calcular, así que definiremos unos aspectos sencillos que podrían ser útiles al usuario. En este caso, se ha pensado conveniente indicar del total de contraseñas la distribución de longitud y de composición de las mismas, y aprovechando que podemos comprobar si una contraseña ha sido encontrada en brechas de seguridad previas, también poder comprobar de las contraseñas guardadas cuántas han sido comprometidas, además de cuántas contraseñas guardadas son únicas (no se repiten) en comparación con el total de contraseñas guardadas. La implementación de esta parte en código C# básicamente consiste en desencriptar y obtener todos los datos de las contraseñas guardadas, y pasárselas a funciones que calcularán los valores numéricos de las estadísticas. Una vez obtenidos esos valores, crearemos diferentes gráficos circulares o de barras para mostrar estas estadísticas. Una vez se generen los gráficos se le dará la opción al usuario de descargar archivos JSON con más detalles de cada estadística, para que así pueda ver qué contraseñas han caído en qué parte de las estadísticas. La implementación en C# no se mostrará en este documento debido a la extensión de la misma, pero se realiza como se ha descrito y se mostrará en un futuro apartado cómo se ven. Cabe destacar que las estadísticas se generarán únicamente sobre Passguard Vaults que contengan al menos diez contraseñas guardadas, debido básicamente a que si habrá un mínimo de contraseñas con el que poder trabajar.

Como última funcionalidad, se ha implementado un gestor para manejar los colores de contorno de la aplicación. Se verá en un siguiente apartado cómo son estos colores de entorno. El gestor de colores de entorno es bastante similar al gestor de contraseñas, se verá una tabla con el nombre de la configuración de colores de entorno, sus valores en sistema RGB, un campo para ver el color, además de opciones para borrar la configuración de color desde la tabla o para marcarla como favorita. Las configuraciones de colores se pueden definir para utilizarse solo en la actual ejecución de la aplicación, o en las futuras ejecuciones. El gestor de configuraciones permite acciones como añadir un nuevo color, editarlo, borrarlo o borrar todas las configuraciones, exportar configuraciones de colores como archivos PDF o JSON,

importar configuraciones de colores a partir de archivos JSON, así como un buscador de configuraciones de colores a partir del nombre de las mismas. También existen ventanas para obtener ayuda acerca de cómo funciona el gestor de configuraciones y sobre cómo funciona el sistema RGB para elegir un color.

No se mostrarán las implementaciones de C# en este caso ya que son similares en cuanto a concepto con respecto al gestor de contraseñas, sin embargo, a continuación, se redactará cómo funciona cada funcionalidad de manera general. Al añadir una configuración de colores se pedirá el nombre, los valores RGB, su condición de favorito o no y si quiere seleccionar la configuración (en caso de hacerlo, si para sólo esta o para futuras ejecuciones) para su uso. Se podrán añadir configuraciones cuyo nombre o combinación RGB no estén ya guardadas, y además la combinación RGB tendrá que tener una luminosidad [98] de entre el 20 y el 90 por ciento ambos inclusive, para que esto no afecte a la visibilidad ni funcionamiento de la aplicación. Al editar colores deben cumplirse las condiciones de cuando añades una nueva configuración, y habrá una lista desplegable para elegir la configuración de color según el nombre. En el caso del borrado puedes elegir una configuración con la lista desplegable o borrar todas las configuraciones activando una casilla, de manera similar al gestor de contraseñas, solo que al borrar todas las contraseñas se añadirá una configuración de color por defecto. Al importar un archivo JSON con configuraciones se le avisará al usuario de que si importa configuraciones entonces las configuraciones guardadas se sustituirán por las importadas, en cuanto acepte esto se le abrirá una pestaña para elegir su archivo JSON, se le validará la forma y el contenido y se importará a la aplicación. Al exportar los datos podremos elegir exportarlos como PDF o como JSON y ambos se guardarán en la carpeta Documentos del sistema operativo Windows OS, con formato de nombre ColoursTable-yyymmdd-hhmmss. El buscador del gestor de configuraciones funciona por el nombre de la configuración y de manera similar al buscador en la vista de contraseñas, y habrá dos botones para obtener ayuda de cómo funciona el gestor de configuraciones en modo texto y se abrirá una página web para que el usuario escoja un color y obtenga sus valores RGB de manera fácil.

Las funcionalidades de establecer Passguard para que se ejecute cuando se inicia el ordenador y para que al abrirse o cerrarse lo haga minimizado básicamente añaden la aplicación a ciertas rutas y registros de Windows para lograr este comportamiento.

Con esto se acaba este apartado en el que se explica la implementación y el funcionamiento a nivel general de las principales funcionalidades de la aplicación, a continuación, veremos cómo se ve la aplicación final.

3.4.2 Interfaz de usuario final de la aplicación

A continuación, se mostrarán varias imágenes con las principales vistas finales de la aplicación:

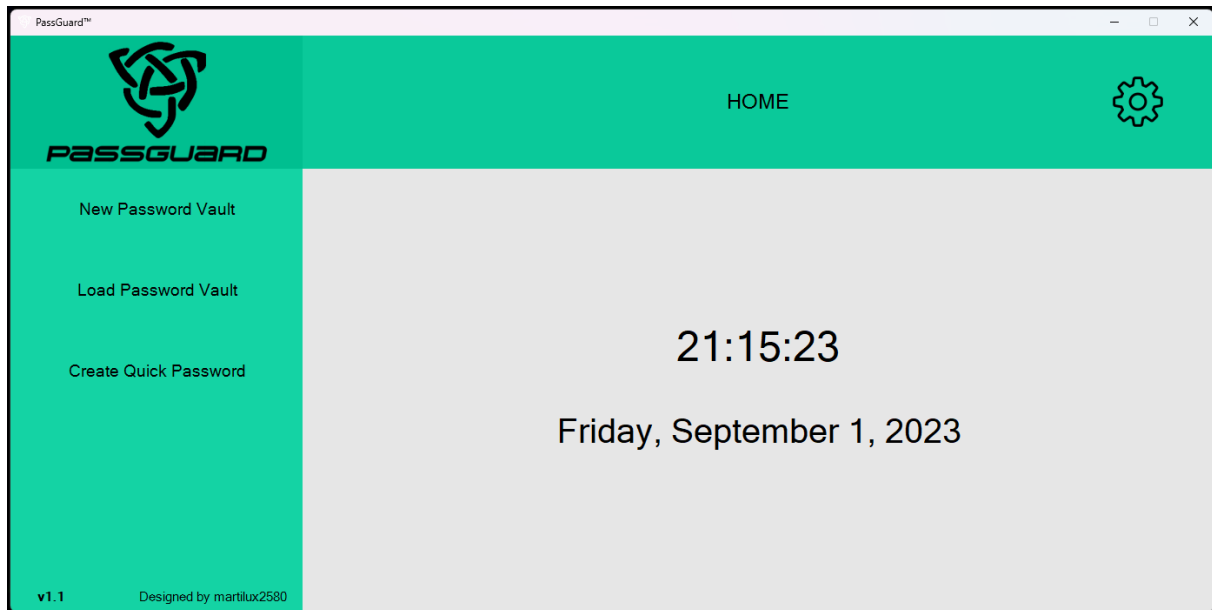


Figura 3.14: Vista principal de la aplicación.

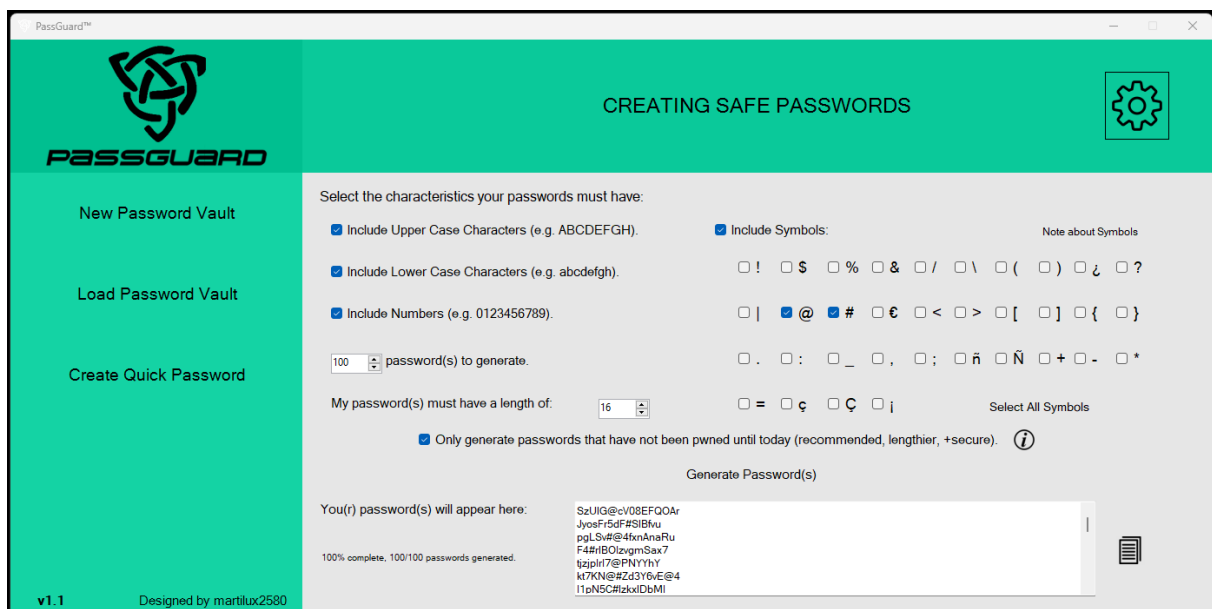


Figura 3.15: Vista de creación de contraseñas.

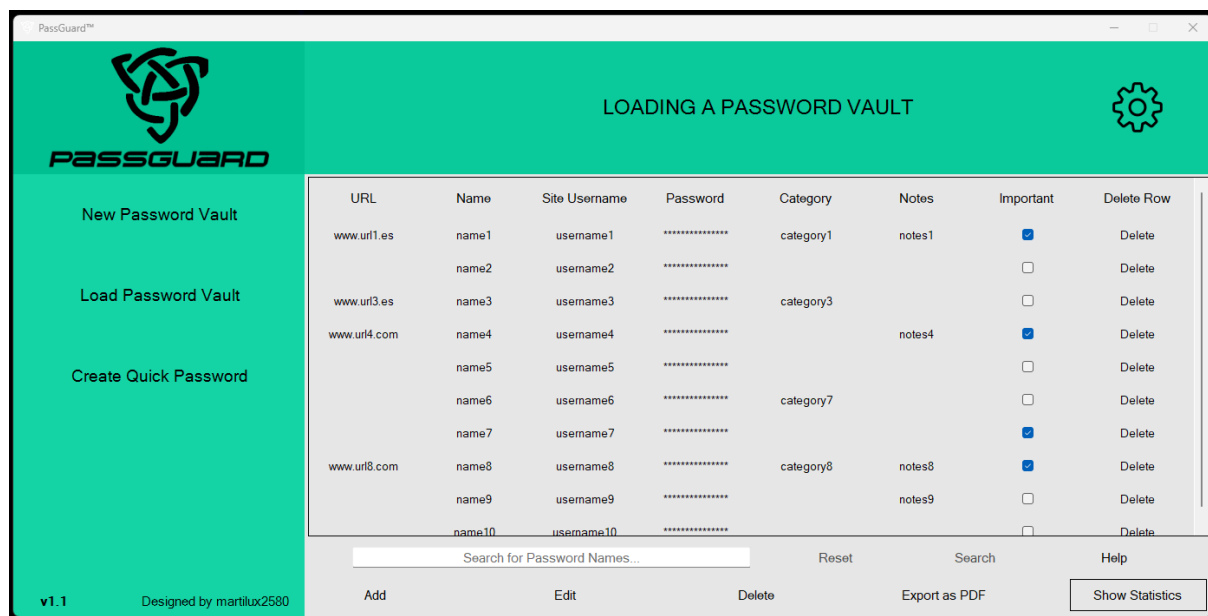


Figura 3.16: Vista con la tabla de contraseñas.

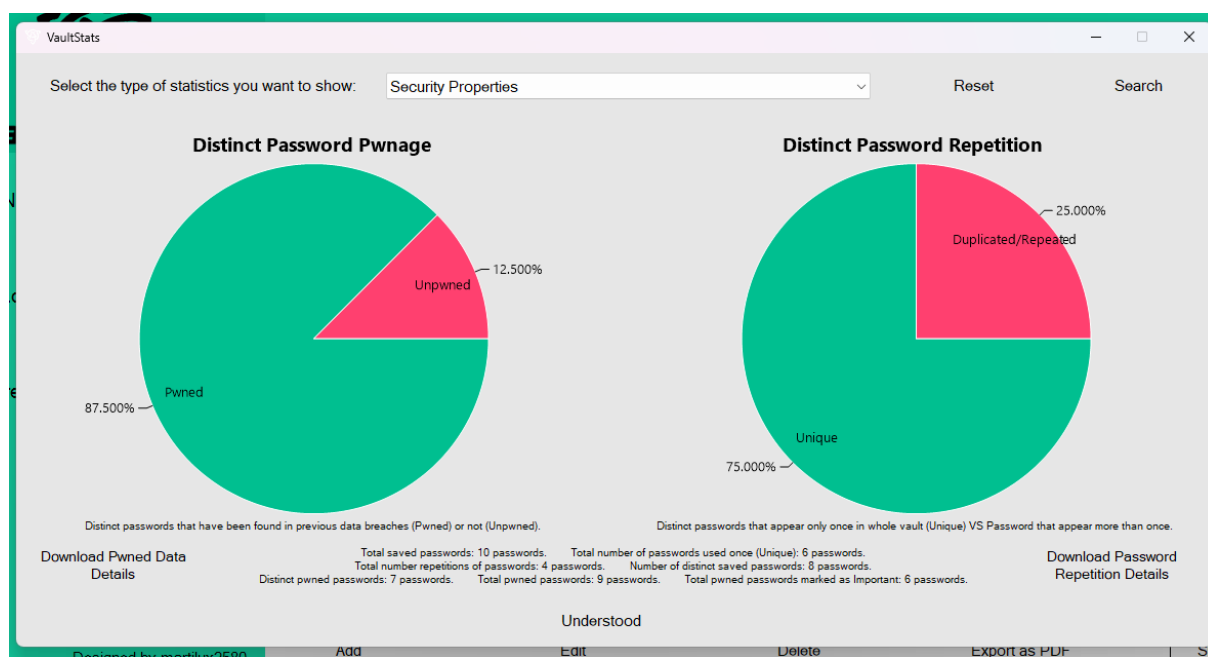


Figura 3.17: Vista de generación de algunas estadísticas.

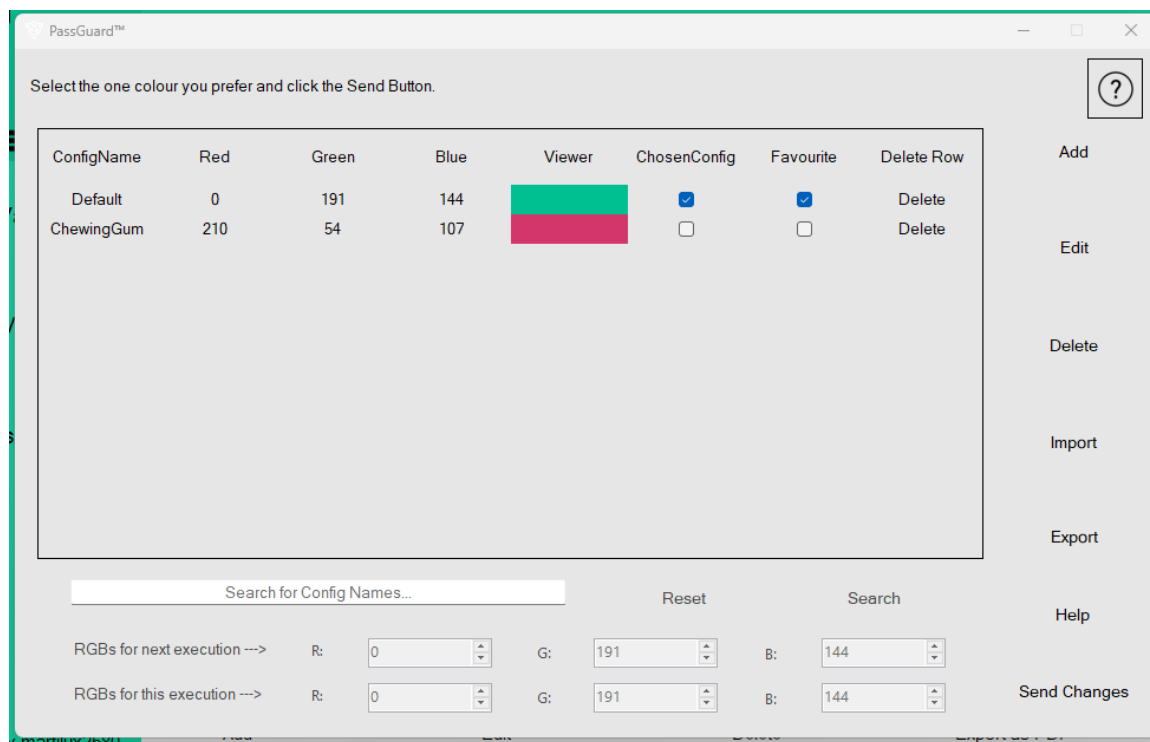


Figura 3.18: Vista del gestor de configuraciones de colores de contorno.

	ColoursTable-20230715-224058.pdf	15-Jul-23 22:41	Microsoft Edge PDF Document	4 KB
	ColoursTable-20230715-234318.json	15-Jul-23 23:43	JSON Source File	2 KB
	CompositionStats-20230722-230239.json	22-Jul-23 23:02	JSON Source File	1 KB
	LengthStats-20230722-230227.json	22-Jul-23 23:02	JSON Source File	1 KB
	PwnageStats-20230722-184644.json	22-Jul-23 18:46	JSON Source File	1 KB
	RepetitionStats-20230722-185335.json	22-Jul-23 18:53	JSON Source File	1 KB
	VaultTable-20230831-191248.pdf	31-Aug-23 19:12	Microsoft Edge PDF Document	3 KB

Figura 3.19: Listado de todos los archivos generables y descargables en Passguard.

Con esto se ha mostrado el aspecto general de la aplicación, en caso de querer más sobre la aplicación se puede mirar el manual de usuario o instalar la aplicación utilizando el manual de despliegue.

Capítulo 4

Conclusiones y líneas futuras

En este apartado se redactarán ciertas conclusiones que se han extraído del trabajo realizado, desde la base teórica obtenida hasta la implementación de la aplicación Passguard. Además, también se mencionarán posibles mejoras y trabajo futuro, de cara a hacer autocrítica y poder evolucionar el proyecto.

4.1 Sobre la base teórica

Derivaremos conclusiones sobre la base teórica obtenida. Considero que la base teórica redactada en este proyecto ha sido la necesaria en cuanto a diversidad de algoritmos y funciones, y ha sido correcta en cuanto a la longitud y la explicación de cada uno. En el apartado del Estado del Arte se ha visto cómo trabajan los gestores de contraseñas actuales y con eso se ha obtenido una base de su funcionamiento, sobre la cual se ha obtenido inspiración para crear Passguard. Se han estudiado estos gestores con cierto detalle y se ha obtenido lo bueno y malo de cada uno para implementar Passguard. Eso no es suficiente ya que se tiene que obtener una base teórica para saber lo que se está implementando, así que se han estudiado los algoritmos que podrían ser necesarios y entendido su funcionamiento para escoger los necesarios.

Respecto a los algoritmos de cifrado, considero que AES ha sido la mejor opción, es el estándar y es muy utilizado, y aún no tiene ataques prácticos sobre el mismo, creo que este algoritmo está destinado a durar unos cuantos años más con la seguridad que ofrece. Por otra parte, en los algoritmos hash se escogió el uso de SHA256 y PBKDF2 respectivamente, y considero que son algoritmos seguros y establecidos en la actualidad pero que, si se quiere seguir manteniendo a Passguard con mucha seguridad, habrá que cambiar estos algoritmos antes del momento en el que habría que cambiar AES. Actualmente en el caso de algoritmos hash SHA3 no es tan utilizado y PBKDF2 es utilizado pero otras alternativas como Argon2id se empiezan a ver con frecuencia, esos dos serían el futuro de los algoritmos en caso de que tener que cambiarlos.

Considero que se podría haber puesto algo más de investigación en la emergente computación cuántica y cómo es el futuro de las contraseñas, los gestores de las mismas y los algoritmos en general cuando la computación cuántica sea más accesible. En el documento se ha redactado brevemente sobre cómo puede afectar a ciertos algoritmos, pero considero que se puede expandir más.

Resumiendo, para este apartado considero que la base teórica redactada ha sido correcta y la necesaria para entender los algoritmos que se han utilizado, para una segunda versión del documento se podrían investigar algunos algoritmos más aunque considero que con los redactados ya se tiene una base teórica adecuada, y que podría haber dedicado algo más de investigación a la computación cuántica, sin embargo,

aún no está lo suficientemente asentada como para suponer una amenaza visible para los algoritmos de encriptación y criptográficos utilizados.

4.2 Sobre PassGuard

Ahora procederemos con algunas conclusiones de la implementación de Passguard. En primer lugar, me ha gustado mucho desarrollar este proyecto, debido a que al principio tenía la idea de implementar simplemente las funcionalidades básicas con una interfaz de usuario decente. Sin embargo, después se me iban ocurriendo multitud de funcionalidades secundarias que yo si fuese usuario de un gestor de contraseñas utilizaría, y al final la implementación se ha extendido en el tiempo, pero estoy contento de que Passguard sea un gestor de contraseñas con muchas otras funcionalidades interesantes.

Considero que programar Passguard en C# y utilizando SQLite3 ha sido un acierto, he aprendido C# y este proyecto es algo que se queda en mi portfolio de desarrollos. Considero que Passguard tiene un nivel de implementación adecuado y suficiente a una persona que está acabando el Grado de Ingeniería Informática, y me alegra haber desarrollado una aplicación que a nivel estético y de funcionalidad podría ser publicable y que no solo sea un proyecto personal de programación.

Hay algunos aspectos que considero que se pueden mejorarse en Passguard. En cuanto a utilización de algoritmos, utilizar AES creo que es un acierto, aunque se podrían haber dado a elegir otros algoritmos, como hacen algunos gestores de contraseñas en la actualidad. Por otra parte, quizá se debería haber utilizado SHA3 en vez de SHA256 y Argon2id en vez de PBKDF2. SHA3 no es tan utilizado como SHA256 hoy en día, así que me siento seguro por esta parte, y en caso de tener que cambiar a SHA3 por dudas en la seguridad de SHA256 se puede realizar de manera sencilla en una nueva versión de la aplicación. En caso de PBKDF2 y Argon2id se ha expuesto el por qué utilizar un algoritmo menos recomendado que el otro (aunque aún es seguro), pero definitivamente se debería por lo menos dar la opción al usuario de que elija el algoritmo que considere. El tema oscuro de la aplicación puede mejorarse de cara a que los datos escritos en campos de texto o seleccionados en listas desplegables sean claramente distinguibles. Hacer que la aplicación pueda cambiar de tamaño de ventana y que no sea fija sería un buen avance, no está implementada de esa manera debido a que he tenido problemas en vistas con muchos componentes para ajustar la localización de los mismos de manera precisa cuando la aplicación cambia de tamaño. Como último, mejorar algunas interacciones con la aplicación que pueden resultar tediosas para el usuario. En una futura versión de la misma se podría dar la opción al usuario de escoger otros algoritmos diferentes, por ejemplo, cifrar con ChaCha20 o derivar claves con Argon2id, o realizar auto exportados en PDF o JSON de contraseñas o configuraciones de colores de contorno. También para futuras versiones se podría desarrollar una versión en aplicación móvil para Passguard, de tal manera que el usuario conserve su Passguard Vault en la nube y que los cambios que realice a la misma en su ordenador los pueda abrir y ver en su aplicación móvil.

En definitiva, considero que Passguard es un buen proyecto para tener una aproximación inicial buena al funcionamiento de gestores de contraseñas, con el que he aprendido en detalle de criptografía y de los algoritmos utilizados hoy en día. Estoy al corriente de que se puede mejorar en pequeños detalles su implementación y que existe un trabajo futuro para mejorar esos pequeños detalles e implementar nuevas funcionalidades.

Bibliografía

- [1] A. Vigderman, *Guía de seguridad digital para 2023*. <https://www.security.org/digital-safety/> [Último acceso: Aug, 2023].
- [2] *LastPass, App Líder en Bóveda y Gestión de contraseñas con inicio de Sesión único Y soluciones MFA*. <https://www.lastpass.com/es> [Último acceso: Aug, 2023].
- [3] *LastPass, Wikipedia*. <https://en.wikipedia.org/wiki/LastPass> [Último acceso: Aug, 2023].
- [4] *Inside the DNA of Your Master Password - LastPass*. <https://blog.lastpass.com/2023/03/inside-the-dna-of-your-master-password/> [Último acceso: Aug, 2023].
- [5] *LastPass Zero knowledge Security*. <https://www.lastpass.com/security/zero-knowledge-security> [Último acceso: Aug, 2023].
- [6] *1Password - Gestor de contraseñas para familias, empresas, equipos | 1Password*. <https://1password.com/es> [Último acceso: Aug, 2023].
- [7] *How Many People Use 1Password in 2023?*. <https://earthweb.com/1password-users/> [Último acceso: Aug, 2023].
- [8] *Get started with 1Password*. <https://support.1password.com/explore/get-started/> [Último acceso: Aug, 2023].
- [9] *Move your data from other applications to 1Password*. <https://support.1password.com/import/> [Último acceso: Aug, 2023].
- [10] *How to choose a good 1Password account password*. <https://support.1password.com/strong-account-password/> [Último acceso: Aug, 2023].
- [11] *Enforce 1Password account password requirements in 1Password Business*. <https://support.1password.com/account-password-policy/> [Último acceso: Aug, 2023].
- [12] *Why you can trust 1Password's cloud-based storage and syncing | 1Password*. <https://blog.1password.com/why-trust-1password-cloud/> [Último acceso: Aug, 2023].
- [13] *Get to know your Emergency Kit*. <https://support.1password.com/emergency-kit/> [Último acceso: Aug, 2023].
- [14] *About the 1Password security model*. <https://support.1password.com/1password-security/> [Último acceso: Aug, 2023].
- [15] *Bitwarden Users: How Many People Use Bitwarden in 2023?*. <https://earthweb.com/bitwarden-users/> [Último acceso: Aug, 2023].
- [16] *Bitwarden Code*. <https://github.com/bitwarden> [Último acceso: Aug, 2023].
- [17] *Bitwarden Security Whitepaper | Bitwarden Help Center*. <https://bitwarden.com/help/bitwarden-security-white-paper/> [Último acceso: Aug, 2023].

- [18] *OWASP Password Hashing Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html [Último acceso: Aug, 2023].
- [19] *Bitwarden Review by Cybernews*. <https://cybernews.com/best-password-managers/bitwarden-review/> [Último acceso: Aug, 2023].
- [20] *Trust - KeePass*. <https://keepass.info/help/kb/trust.html> [Último acceso: Aug, 2023].
- [21] *Security Issues - KeePass*. https://keepass.info/help/kb/sec_issues.html [Último acceso: Aug, 2023].
- [22] *KeePass Review by Cybernews*. <https://cybernews.com/best-password-managers/keepass-review/> [Último acceso: Aug, 2023].
- [23] *Encriptación en reposo predeterminada / Documentación*. <https://cloud.google.com/docs/security/encryption/default-encryption?hl=es-419> [Último acceso: Aug, 2023].
- [24] *How Password Managers Work - Computerphile*. <https://www.youtube.com/watch?v=w68BBPDW8r8> [Último acceso: Aug, 2023].
- [25] *Cryptography / NIST*. <https://www.nist.gov/cryptography> [Último acceso: Aug, 2023].
- [26] *What is Cryptography? Definition, Importance, Types*. <https://www.fortinet.com/resources/cyberglossary/what-is-cryptography> [Último acceso: Aug, 2023].
- [27] *Cifrado (criptografía)*. [https://es.wikipedia.org/w/index.php?title=Cifrado_\(criptograf%C3%aA\)](https://es.wikipedia.org/w/index.php?title=Cifrado_(criptograf%C3%aA)) [Último acceso: Aug, 2023].
- [28] *Criptografía simétrica*. https://es.wikipedia.org/w/index.php?title=Criptograf%C3%aA_sim%C3%A9trica [Último acceso: Aug, 2023].
- [29] *Ransomware encryption techniques*. <https://medium.com/@tarcisioma/ransomware-encryption-techniques-696531d07bb9> [Último acceso: Aug, 2023].
- [30] *AES instruction set*. https://en.wikipedia.org/w/index.php?title=AES_instruction_set [Último acceso: Aug, 2023].
- [31] *Criptografía asimétrica*. https://es.wikipedia.org/w/index.php?title=Criptograf%C3%aA_asim%C3%A9trica [Último acceso: Aug, 2023].
- [32] *Public Key Cryptography - Computerphile*. https://www.youtube.com/watch?v=GSIDS_1vRv4 [Último acceso: Aug, 2023].
- [33] *Criptografía híbrida*. https://es.wikipedia.org/w/index.php?title=Criptograf%C3%aA_h%C3%ADrida [Último acceso: Aug, 2023].
- [34] *Ataques ransomware WannaCry*. https://es.wikipedia.org/w/index.php?title=Ataques_ransomware_WannaCry [Último acceso: Aug, 2023].
- [35] *How WanaCrypt Encrypts Your Files - Computerphile*. <https://www.youtube.com/watch?v=pLluFxFHrc30> [Último acceso: Aug, 2023].
- [36] *WannaCry ransomware attack*. https://en.wikipedia.org/w/index.php?title=WannaCry_ransomware_attack [Último acceso: Aug, 2023].
- [37] *Cifrador de flujo*. https://es.wikipedia.org/w/index.php?title=Cifrador_de_flujo [Último acceso: Aug, 2023].
- [38] *Libreta de un solo uso*. https://es.wikipedia.org/w/index.php?title=Libreta_de_un_solo_uso [Último acceso: Aug, 2023].
- [39] *Criptografía - Sesión 2, Parte 1 - Cifrado en Flujo*. <https://www.youtube.com/watch?v=zEZk-1wKvcY> [Último acceso: Aug, 2023].

- [40] *Cifrado por bloques*. https://es.wikipedia.org/w/index.php?title=Cifrado_por_bloques [Último acceso: Aug, 2023].
- [41] *Confusión y difusión*. https://es.wikipedia.org/w/index.php?title=ConfusiÃşn_y_difusiÃşn [Último acceso: Aug, 2023].
- [42] *Función hash criptográfica*. https://es.wikipedia.org/w/index.php?title=FunciÃşn_hash_criptogrÃafrica [Último acceso: Aug, 2023].
- [43] *Principio del palomar*. https://es.wikipedia.org/w/index.php?title=Principio_del_palomar [Último acceso: Aug, 2023].
- [44] *Hashing Algorithms and Security - Computerphile*. <https://www.youtube.com/watch?v=b4b8ktEV4Bg> [Último acceso: Aug, 2023].
- [45] *Salt (cryptography)*. [https://en.wikipedia.org/w/index.php?title=Salt_\(cryptography\)](https://en.wikipedia.org/w/index.php?title=Salt_(cryptography)) [Último acceso: Aug, 2023].
- [46] *Secure Salted Password Hashing - How to do it Properly*. <https://crackstation.net/hashing-security.htm> [Último acceso: Aug, 2023].
- [47] *Hash Functions, Chapter 11*. <https://joyofcryptography.com/pdf/chap11.pdf> [Último acceso: Aug, 2023].
- [48] *Pepper (cryptography)*. [https://en.wikipedia.org/w/index.php?title=Pepper_\(cryptography\)](https://en.wikipedia.org/w/index.php?title=Pepper_(cryptography)) [Último acceso: Aug, 2023].
- [49] *KDFs*. https://es.wikipedia.org/w/index.php?title=FunciÃşn_de_derivaciÃşn_de_clave [Último acceso: Aug, 2023].
- [50] *Pseudorandom function family*. https://en.wikipedia.org/w/index.php?title=Pseudorandom_function_family [Último acceso: Aug, 2023].
- [51] *Generador de números pseudoaleatorios criptográficamente seguro*. https://es.wikipedia.org/w/index.php?title=Generador_de_nũmeros_pseudoaleatorios_criptogrÃaficamente_seguro [Último acceso: Aug, 2023].
- [52] *Entropy - Glossary / CSRC*. <https://csrc.nist.gov/glossary/term/entropy> [Último acceso: Aug, 2023].
- [53] *Data Encryption Standard*. https://es.wikipedia.org/w/index.php?title=Data_Encryption_Standard [Último acceso: Aug, 2023].
- [54] *Cifrado de Feistel*. https://es.wikipedia.org/w/index.php?title=Cifrado_de_Feistel [Último acceso: Aug, 2023].
- [55] *Triple DES*. https://es.wikipedia.org/w/index.php?title=Triple_DES [Último acceso: Aug, 2023].
- [56] *Advanced Encryption Standard*. https://es.wikipedia.org/w/index.php?title=Advanced_Encryption_Standard [Último acceso: Aug, 2023].
- [57] *Cuerpo finito*. https://es.wikipedia.org/w/index.php?title=Cuerpo_finito [Último acceso: Aug, 2023].
- [58] *Computación cuántica*. https://es.wikipedia.org/w/index.php?title=ComputaciÃşn_cuÃantica [Último acceso: Aug, 2023].
- [59] *Algoritmo de Grover*. https://es.wikipedia.org/w/index.php?title=Algoritmo_de_Grover [Último acceso: Aug, 2023].

- [60] *Diffie-Hellman*. <https://es.wikipedia.org/w/index.php?title=Diffie-Hellman> [Último acceso: Aug, 2023].
- [61] *RSA*. <https://es.wikipedia.org/w/index.php?title=RSA> [Último acceso: Aug, 2023].
- [62] *Función Phi de Euler*. https://es.wikipedia.org/wiki/Funci%C3%A3n_%E1_de_Euler [Último acceso: Aug, 2023].
- [63] *Teorema de Euler*. https://es.wikipedia.org/w/index.php?title=Teorema_de_Euler [Último acceso: Aug, 2023].
- [64] *Números coprimos*. https://es.wikipedia.org/w/index.php?title=N%C3%BAmeros_coprimos [Último acceso: Aug, 2023].
- [65] *Algoritmo de Euclides*. https://es.wikipedia.org/w/index.php?title=Algoritmo_de_Euclides [Último acceso: Aug, 2023].
- [66] *Inverso modular*. [https://es.wikipedia.org/w/index.php?title=Inverso_multiplicativo_\(aritm%C3%A9tica_modular\)](https://es.wikipedia.org/w/index.php?title=Inverso_multiplicativo_(aritm%C3%A9tica_modular)) [Último acceso: Aug, 2023].
- [67] *Fermat Attack on RSA*. <https://fermatattack.secvuln.info/> [Último acceso: Aug, 2023].
- [68] *Researcher uses 379-year-old algorithm to crack crypto keys found in the wild*. <https://arstechnica.com/information-technology/2022/03/researcher-uses-600-year-old-algorithm-to-crack-crypto-keys-found-in-the-wild/> [Último acceso: Aug, 2023].
- [69] *Fermat Factorization Method*. https://es.wikipedia.org/w/index.php?title=M%C3%A9todo_de_factorizaci%C3%B3n_de_Fermat [Último acceso: Aug, 2023].
- [70] *Algoritmo de Shor*. https://es.wikipedia.org/w/index.php?title=Algoritmo_de_Shor [Último acceso: Aug, 2023].
- [71] *ECC*. https://es.wikipedia.org/w/index.php?title=Criptograf%C3%9A_de_curva_el%C3%94ptica [Último acceso: Aug, 2023].
- [72] *Curva elíptica*. https://es.wikipedia.org/w/index.php?title=Curva_el%C3%94ptica [Último acceso: Aug, 2023].
- [73] *Elliptic-curve Diffie-Hellman*. https://es.wikipedia.org/w/index.php?title=Elliptic-curve_Diffie-Hellman [Último acceso: Aug, 2023].
- [74] *Característica matemática*. [https://es.wikipedia.org/w/index.php?title=Caracter%C3%A1stica_\(matem%C3%A1tica\)](https://es.wikipedia.org/w/index.php?title=Caracter%C3%A1stica_(matem%C3%A1tica)) [Último acceso: Aug, 2023].
- [75] *NIST.SP.800-186*. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186.pdf> [Último acceso: Aug, 2023].
- [76] *Elliptic Curve Cryptography (ECC)*. <https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc> [Último acceso: Aug, 2023].
- [77] *128 Bit or 256 Bit Encryption? - Computerphile*. <https://www.youtube.com/watch?v=pgzWxOtk1zg> [Último acceso: Aug, 2023].
- [78] *Dual_EC_DRBG*., https://en.wikipedia.org/w/index.php?title=Dual_EC_DRBG [Último acceso: Aug, 2023].
- [79] *Secure Hash Algorithm*. https://es.wikipedia.org/w/index.php?title=Secure_Hash_Algorithm [Último acceso: Aug, 2023].
- [80] *FIPS 180 4 / NIST*. https://csrc.nist.gov/files/pubs/fips/180-4/final/docs/draft-fips180-4_feb2011.pdf [Último acceso: Aug, 2023].

- [81] *SHA-3*. <https://es.wikipedia.org/w/index.php?title=SHA-3> [Último acceso: Aug, 2023].
- [82] *Keccak Team*. https://keccak.team/keccak_specs_summary.html [Último acceso: Aug, 2023].
- [83] *PBKDF2*. <https://en.wikipedia.org/w/index.php?title=PBKDF2> [Último acceso: Aug, 2023].
- [84] *PKCS 5: Password-Based Cryptography Specification Version 2.0*. <https://datatracker.ietf.org/doc/rfc2898/> [Último acceso: Aug, 2023].
- [85] *Length extension attack*. https://en.wikipedia.org/w/index.php?title=Length_extension_attack [Último acceso: Aug, 2023].
- [86] *HMAC*. <https://es.wikipedia.org/w/index.php?title=HMAC> [Último acceso: Aug, 2023].
- [87] *Distancia de Hamming*. https://es.wikipedia.org/w/index.php?title=Distancia_de_Hamming [Último acceso: Aug, 2023].
- [88] *List of PBKDF2 implementations*. https://en.wikipedia.org/w/index.php?title=List_of_PBKDF2_implementations [Último acceso: Aug, 2023].
- [89] *Argon2*. <https://es.wikipedia.org/w/index.php?title=Argon2> [Último acceso: Aug, 2023].
- [90] *Password Hashing Competition*. https://en.wikipedia.org/w/index.php?title=Password_Hashing_Competition [Último acceso: Aug, 2023].
- [91] *Entropía (computación)*. [https://es.wikipedia.org/w/index.php?title=Entropía_\(computación\)](https://es.wikipedia.org/w/index.php?title=Entropía_(computación)) [Último acceso: Aug, 2023].
- [92] *Casos a incluir y casos a extender*. <https://www.abiztar.com.mx/articulos/casos-a-incluir-casos-a-extender.html> [Último acceso: Aug, 2023].
- [93] *URL length: how long can a URL be?*. <https://www.sistrix.com/ask-sistrix/technical-seo/site-structure/url-length-how-long-can-a-url-be/> [Último acceso: Aug, 2023].
- [94] *Have I Been Pwned: Pwned Passwords*. <https://haveibeenpwned.com/Passwords> [Último acceso: Aug, 2023].
- [95] *k-anonimato*. <https://es.wikipedia.org/w/index.php?title=K-anonimato> [Último acceso: Aug, 2023].
- [96] *I've Just Launched "Pwned Passwords" V2 With Half a Billion Passwords for Download | Troy Hunt*. <https://www.troyhunt.com/ive-just-launched-pwned-passwords-version-2/> [Último acceso: Aug, 2023].
- [97] *Understanding Have I Been Pwned's Use of SHA-1 and k-Anonymity | Troy Hunt*. <https://www.troyhunt.com/understanding-have-i-been-pwneds-use-of-sha-1-and-k-anonymity/> [Último acceso: Aug, 2023].
- [98] *Modelo de color HSL*. https://es.wikipedia.org/w/index.php?title=Modelo_de_color_HSL [Último acceso: Aug, 2023].

Apéndice A

Manual de usuario

A.1 Introducción

Este apartado comprenderá el manual de usuario de Passguard, de cara a un usuario de cualquier nivel sepa cómo utilizar la aplicación y cómo se comporta según las acciones realizadas.

A.2 Vista principal de la aplicación

Al iniciar la aplicación obtenemos la vista principal de la misma, la cual tiene cinco partes importantes con las que se puede interactuar:

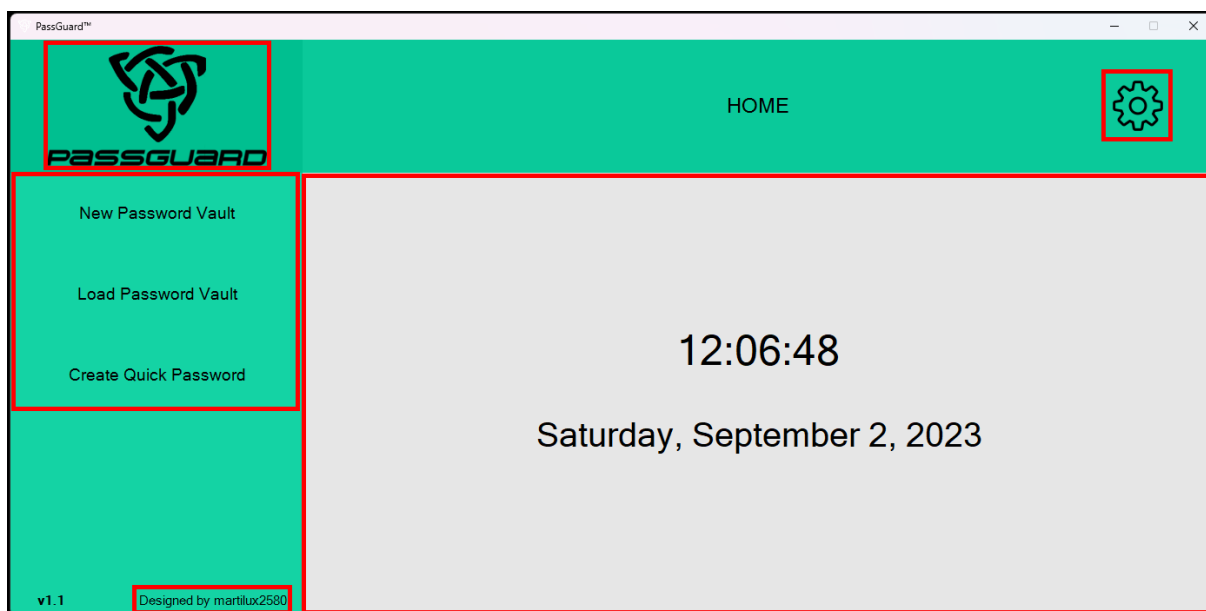


Figura A.1: Vista principal de la aplicación con cuatro partes importantes.

En la parte superior izquierda tenemos el logo y nombre de la aplicación, en cualquier momento de interacción con la aplicación si queremos volver a esta vista principal tan solo le daremos clic a ese logo con el nombre y volveremos a esta vista principal.

Debajo de ese logo tenemos un panel con un color ligeramente diferente, donde se encuentran botones para, en orden, crear una nueva Passguard Vault, cargar una nueva Passguard Vault o crear contraseñas

aleatorias rápidamente. En el futuro podrían aparecer más botones en esta parte, más adelante veremos qué pasa cuando se pulsa cada uno de esos tres botones. En la parte inferior izquierda tenemos un texto que dice Designed by martilux2580, si hacemos clic en ese texto veremos que es un vínculo que se abrirá en el navegador favorito del usuario y que dirige al perfil de GitHub del creador de esta aplicación.

Por otra parte, en la parte central de la aplicación y en tema claro en este caso tenemos el panel de contenidos de la aplicación. Debido a que nos encontramos en la vista principal de la aplicación lo que se nos muestra en este panel es la fecha y hora en tiempo real, conforme interactuemos con esta aplicación se mostrarán los contenidos en este panel. Cabe resaltar que se puede minimizar la aplicación, pero el tamaño de la ventana de la aplicación es fijo.

En la parte superior derecha de la aplicación tenemos un botón con una imagen de engranaje, se trata del botón de opciones, este botón y sus opciones se explicarán en un futuro apartado.

De toda la imagen anterior, las zonas con color de contorno verde son las que se mantendrán siempre visibles, la zona blanca es el panel de contenidos y cambiará según interactuemos con la aplicación.

A.3 Crear un nuevo PassGuard Vault

Estando en la vista principal de la aplicación, si le damos clic al botón de New Password Vault accederemos a la siguiente vista de la aplicación:

PassGuard™

CREATING A NEW PASSWORD VAULT

PassGUARD

New Password Vault

Load Password Vault

Create Quick Password

v1.1 Designed by martilux2580

User Email: ☐ Save Email

New Vault's Name:

New Vault's Password: ☐

Confirm New Vault's Password: ☐

New Vault's Path:

Create New Password Vault

Figura A.2: Vista de creación de un nuevo Passguard Vault.

Como se ve en la imagen, a la hora de crear un nuevo Passguard Vault se necesita un correo electrónico, un nombre para el archivo, una contraseña que habrá que confirmar y una ruta donde guardar el archivo. No se le enviarán correos electrónicos al correo proporcionado, y se puede hacer clic en la casilla de Save Email de cara a que luego cuando se quiera iniciar sesión en la Passguard Vault creada el usuario pueda copiar el correo electrónico en vez de teclearlo. En el caso de la contraseña debe contener letras mayúsculas y minúsculas, números y símbolos, y tener un mínimo de 16 caracteres. Los botones con imagen de ojos que se encuentran a la derecha de los campos de contraseña y confirmación de contraseña son para alterar la visibilidad de su respectivo campo de texto. Las contraseñas por defecto se muestran con caracteres escondidos, si hacemos clic en ese botón se mostrarán en texto plano el respectivo campo

de texto, si volvemos a hacer clic en el botón con la imagen de ojo cerrado se volverán a ocultar el valor de contraseñas. En el caso de la ruta, el usuario deberá hacer clic en el botón con ícono de carpeta y escoger una ruta de su dispositivo.

Una vez se introduzcan los datos se comprobará que la contraseña y el correo electrónico tengan la forma adecuada, que la contraseña y su confirmación coincidan y que el nombre no esté compuesto únicamente por símbolos, ya que podría ser un problema, y que no haya un Passguard Vault con ese nombre guardado ya en esa ruta. Si se cumplen las condiciones para estos campos nos saltará una ventana con información, esta información básicamente son los datos que no podemos perder para poder acceder a la Passguard Vault (el correo electrónico, la contraseña, y la Security Key (SK) generada), además de ciertas acciones que se han realizado al crearla. La ventana tiene el siguiente aspecto:

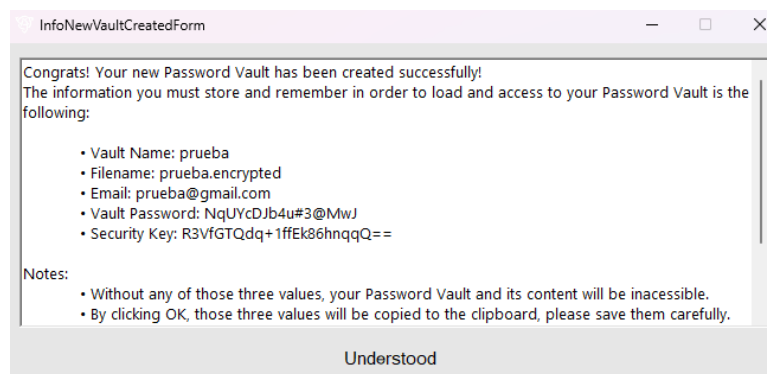


Figura A.3: Ventana emergente una vez se ha creado correctamente un Passguard Vault.

En cuanto sale la anterior ventana los contenidos importantes como el correo electrónico, la contraseña y la Security Key se copian al portapapeles para que el usuario las guarde de manera conveniente, de todas formas, se recomienda comprobar que efectivamente tenemos en el portapapeles esos datos copiados antes de cerrar la ventana con los datos. En caso de crear nuevas Passguard Vaults los datos se guardan en memoria y se descartan los que hubiesen guardados, así que el usuario debe tener cuidado en esos casos para no perder credenciales de acceso a sus Passguard Vaults.

En caso de que algún dato de entrada al crear la Passguard Vault no sea correcto el programa nos avisará con un mensaje emergente del siguiente estilo:

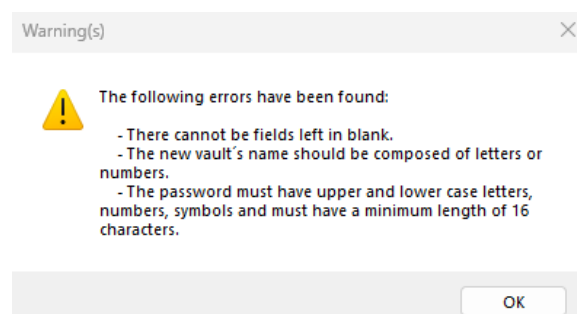


Figura A.4: Mensaje emergente en caso de errores en la creación de un Passguard Vault.

Una vez creada correctamente la Passguard Vault, se reseteará el contenido de los campos de texto (excepto el correo electrónico) por si el usuario desea crear otra Passguard Vault.

A.4 Cargar un Passguard Vault

Si nos encontramos en la vista principal de la aplicación y hacemos clic en el botón de Load Password Vault, nos llevará a la siguiente vista de inicio de sesión:

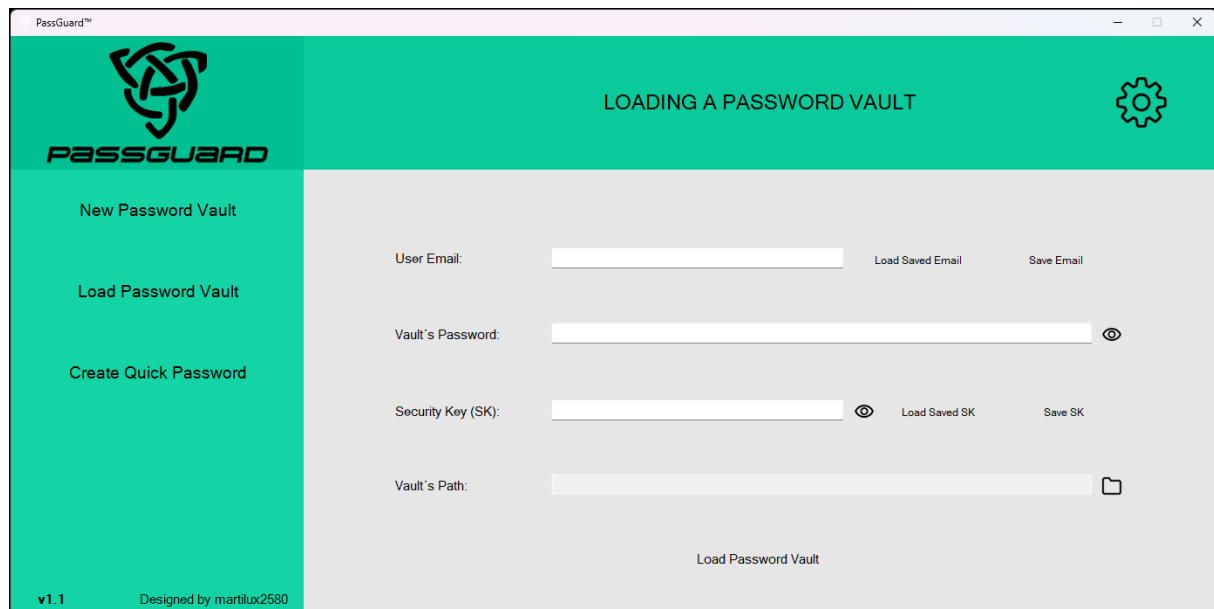
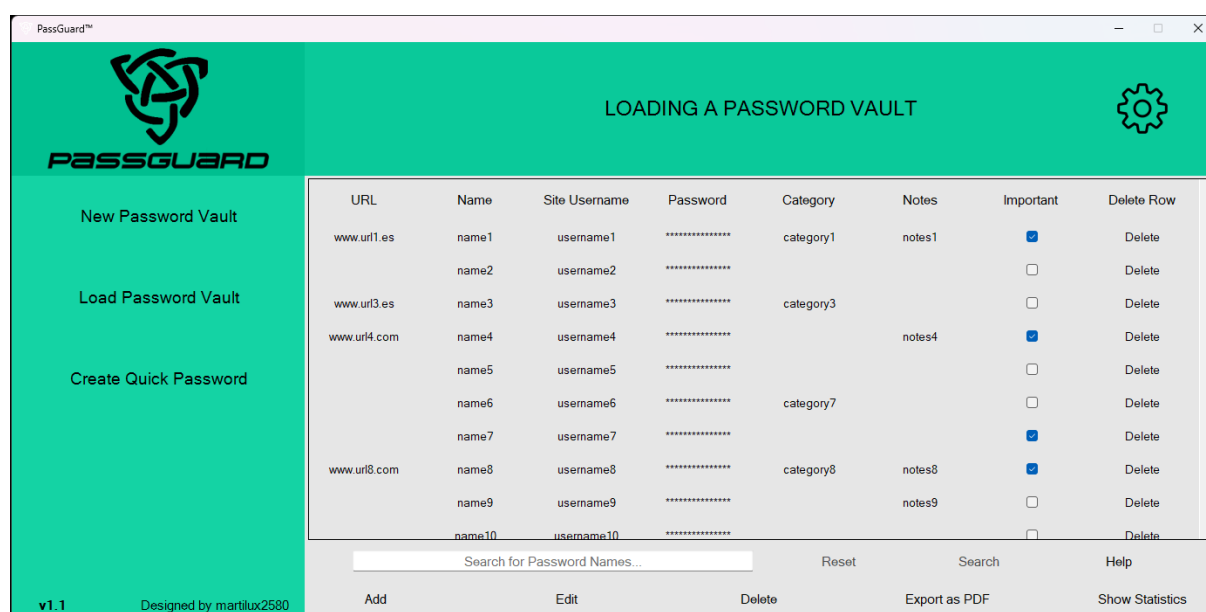


Figura A.5: Vista de inicio de sesión o carga de una Passguard Vault.

En esta vista deberemos introducir los tres campos fundamentales para acceder a la Passguard Vault y la ruta a la Passguard Vault específica. En el caso del correo electrónico, deberemos introducir el correo electrónico utilizado para crear la Passguard Vault, y en caso de tenerlo guardado podemos darle al botón Load Saved Email para que se pegue automáticamente en el campo de texto correspondiente. En caso de querer guardar un correo electrónico, primero deberemos escribir un correo electrónico en el campo de texto correspondiente y luego darle clic al botón Save Email para guardarlo. Los campos de contraseña y Security Key tienen los caracteres ocultos, pero con los botones con íconos de ojo abierto o cerrado podremos darle visibilidad en texto claro o no a esos campos. La Security Key es un campo que se generó de manera aleatoria al crear la Passguard Vault, y que justo después de crearla se guardó en el programa (aunque en la ventana emergente sale y se puede copiar y guardar), de tal forma que si le damos al botón Load Saved SK debería cargarse en el campo de texto correspondiente. De lo contrario, el usuario tendrá que escribir en el campo de texto correspondiente la Security Key asociada a ese Passguard Vault, y en caso de que quiera guardarla deberá darle clic al botón Save SK. Seguidamente el usuario escogerá la ruta de su Passguard Vault (es su archivo con extensión .encrypted) dándole clic al botón con ícono de carpeta y una vez seleccionada, podrá hacer clic en el botón Load Password Vault para cargarla. En caso de haber algún error con los datos de entrada o con las credenciales, al usuario le saltará un mensaje emergente de error con una descripción del mismo, de lo contrario se accederá a la vista de tabla con contraseñas, donde se mostrarán los datos de la misma.

A.4.1 Vista principal con los contenidos de un Passguard Vault

En cuanto el usuario inicie sesión o cargue su Password Vault a través de la vista de inicio de sesión, se accederá a una vista donde se mostrará la tabla de contraseñas descifradas, a su vez de varios botones y campos para interactuar con ella.



The screenshot shows the PassGuard application window titled "LOADING A PASSWORD VAULT". On the left is a sidebar with three options: "New Password Vault", "Load Password Vault", and "Create Quick Password". The main area displays a table of password entries. Below the table is a search bar and several action buttons.

URL	Name	Site Username	Password	Category	Notes	Important	Delete Row
www.url1.es	name1	username1	*****	category1	notes1	<input checked="" type="checkbox"/>	Delete
	name2	username2	*****			<input type="checkbox"/>	Delete
www.url3.es	name3	username3	*****	category3		<input type="checkbox"/>	Delete
www.url4.com	name4	username4	*****		notes4	<input checked="" type="checkbox"/>	Delete
	name5	username5	*****			<input type="checkbox"/>	Delete
www.url8.com	name6	username6	*****	category7		<input type="checkbox"/>	Delete
	name7	username7	*****			<input checked="" type="checkbox"/>	Delete
	name8	username8	*****	category8	notes8	<input checked="" type="checkbox"/>	Delete
	name9	username9	*****		notes9	<input type="checkbox"/>	Delete
	name10	username10	*****			<input type="checkbox"/>	Delete

Below the table, there is a search bar labeled "Search for Password Names..." with a "Reset" button. Below that is a row of buttons: "Add", "Edit", "Delete", "Export as PDF", and "Show Statistics".

Figura A.6: Vista de la tabla de contraseñas.

En este caso se le han añadido datos a la Passguard Vault para que así se vea el aspecto que tiene. En primer lugar, tenemos una tabla con ocho columnas y sus respectivas cabeceras. Debajo de esas cabeceras tenemos en la mayoría de las columnas botones, aunque en otra tenemos casillas. Debajo de la tabla tenemos una línea horizontal con un campo de búsqueda y tres botones, y debajo de esa línea horizontal tenemos otra con seis botones.

Si hacemos clic en algunas cabeceras de columnas nos saldrá una lista de opciones para ordenar los datos de la tabla por esa columna, los órdenes son normal (el orden con el que se muestran los datos al iniciar sesión), y ascendente o descendente alfabéticamente. La lista desplegable tiene el siguiente aspecto, y está disponible para algunas columnas:

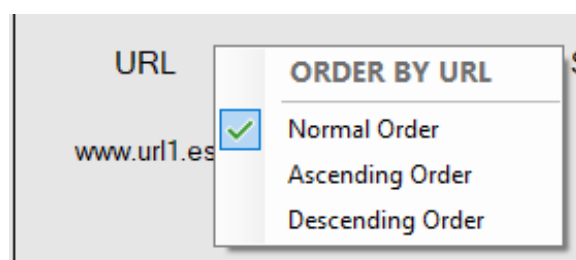


Figura A.7: Lista de opciones de ordenado.

Al hacer clic en alguna opción, los datos de la tabla de contraseñas se resetearán y se volverán a mostrar ordenados según se haya elegido.

La tabla de contraseñas está compuesta principalmente por botones, el texto de estos botones son los datos de las contraseñas guardadas, aunque no aparezcan al completo por motivos de tamaño están ahí. En todas las columnas de botones excepto en la columna Delete Row, en caso de hacer clic en un botón se copiará el contenido al portapapeles del usuario, el campo de contraseñas sale censurado con asteriscos, pero si se le hace clic también se copian sus datos en texto plano al portapapeles. En la columna Delete Row se encuentran botones para borrar esa contraseña (la de la fila en la que está asociada) de la Passguard Vault. Borrar los datos es una acción irreversible, por eso antes de borrar se pedirá confirmación con el siguiente mensaje emergente:

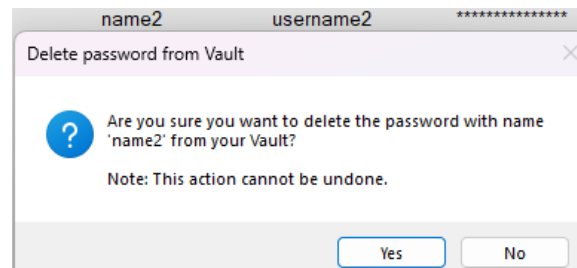


Figura A.8: Mensaje emergente de confirmación de borrado de una contraseña.

La columna Important indica las contraseñas que se han marcado como importantes, si la casilla está marcada indica que la contraseña es importante. Al ordenar las contraseñas por importancia siempre las importantes aparecerán arriba y se ordenarán según se haya especificado, en caso de empate se ordenan por nombre. A la hora de marcar o desmarcar una contraseña como importante se puede hacer clic en la casilla correspondiente y saldrá un mensaje de confirmación del cambio, en caso de que el usuario lo acepte se recargará la tabla de contraseñas con los datos actualizados.

El campo de búsqueda de contraseñas busca contraseñas por la columna nombre, ya que es la columna con valores únicos por cada contraseña. En cuanto haya texto en ese campo se activará el botón Search para ejecutar esa búsqueda, lo que se hará es que la tabla de contraseñas se actualizará mostrando el resultado de la búsqueda. El botón de Reset reestablecerá la tabla de contraseñas con los datos que tuviese guardados la Passguard Vault y vaciará el campo de texto de búsqueda. El botón de Help abrirá una ventana emergente con texto en el que se detalla cómo se puede utilizar esta vista de contraseñas y qué datos tiene, en caso de dudas se recomienda leer la información de esa ventana emergente.

Al hacer clic en el botón Add nos saltará una ventana con campos de texto para introducir la url, nombre, nombre de usuario, contraseña, categoría, notas y una casilla para seleccionar si la contraseña es importante o no. Los campos de texto del nombre, nombre de usuario y contraseña son obligatorios, y el nombre no debe ser un nombre que ya esté guardado en la Passguard Vault. En caso de que no se cumpla algún requisito se mostrará un mensaje emergente, de lo contrario se cerrará la ventana y se volverá a recargar la tabla de contraseñas con los datos actualizados. No se mostrará una captura de esta ventana debido a que es muy sencilla y se explica sola.

Al hacer clic en el botón Edit nos saltará una ventana que en el inicio solo tiene activada una lista desplegable, si se hace clic en esa lista desplegable se podrán ver todos los nombres de las contraseñas guardadas. Cuando el usuario seleccione un nombre de contraseña que no sea en blanco se mostrarán en los campos de texto que se encuentran debajo de la lista desplegable los datos relacionados con esa contraseña, es decir, la url, el nombre de la contraseña y el nombre de usuario, la contraseña y la categoría, las notas y la importancia de la contraseña. Esos datos se podrán editar, y en cuanto se dé clic al botón Edit Element se comprobará que el nombre de contraseña no existe en la base de datos (salvo que el nombre sea el nombre que estabas editando) y las comprobaciones que se realizaban cuando añades una nueva contraseña a la Passguard Vault.

Si le damos al botón de Delete accederemos a una ventana emergente similar a la del botón Edit, pero ligeramente distinta. En este caso tendremos también una lista desplegable para elegir la contraseña a borrar, y en cuanto la elijamos podremos borrarla. Esta ventana emergente tiene la diferencia de que hay una casilla que permite activar un botón para eliminar todas las contraseñas de la Passguard Vault. Tanto en el caso en el que borremos una contraseña como todas, al presionar el botón de Delete Element se pedirá confirmación para borrar, ya que es un proceso irreversible. La ventana emergente para borrar contraseñas es la siguiente, y con ella se puede deducir el aspecto de la ventana de editado:

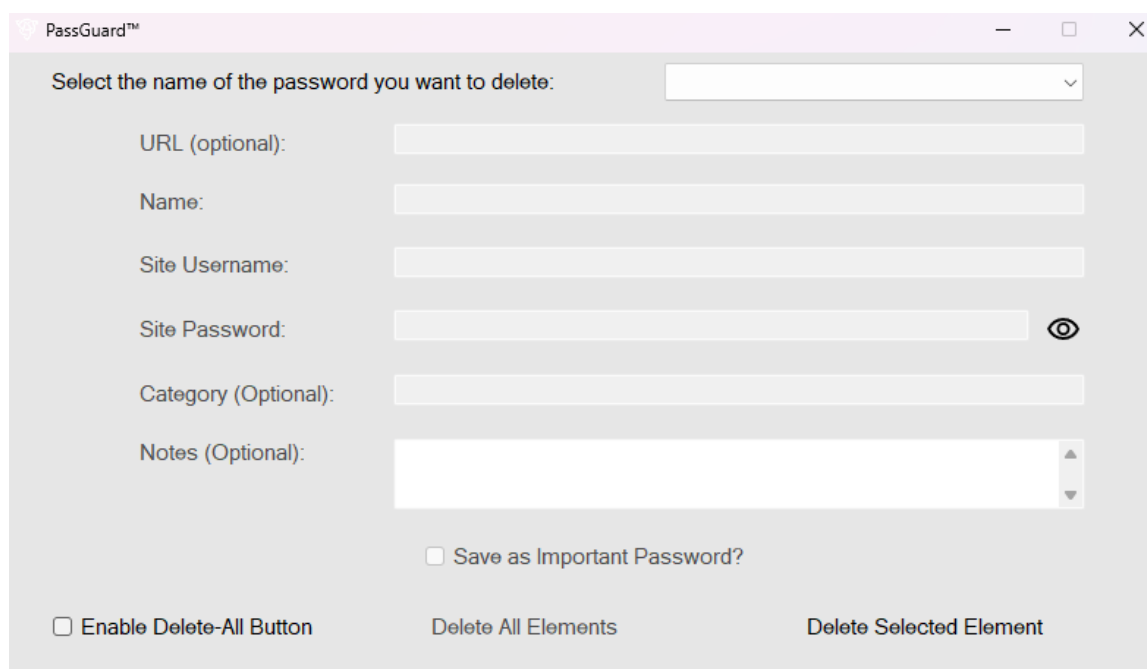


Figura A.9: Ventana emergente de borrado de contraseñas en un Passguard Vault.

En caso de darle al botón de Export as PDF la tabla de contenidos se exportará como documento PDF en la carpeta Documentos del sistema operativo Windows OS. En cuanto se haya generado ese documento saltará un mensaje emergente como el siguiente:

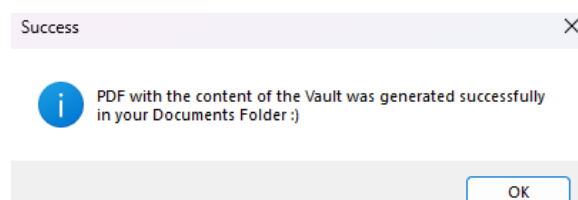


Figura A.10: Mensaje emergente del exportado correcto de una Passguard Vault como PDF.

Por último, si le damos al botón de Show Statistics accederemos a una ventana grande en donde tendremos una lista desplegable con tipos de estadísticas y botones Reset y Search. Podremos acceder a esta ventana en caso de que la Passguard Vault con la que estemos operando tenga un mínimo de diez contraseñas guardadas, de cara a tener un mínimo número de contraseñas con las que trabajar. En cuanto se haya seleccionado un tipo de estadística se habilitarán los botones de Reset y Search. El botón de Search busca los datos necesarios y genera los gráficos, el botón Reset por su parte reinicia la lista desplegable y los gráficos generados. La ventana grande de estadísticas una vez se han generado algunas tiene el siguiente aspecto:

Una vez se han generado las estadísticas, se mostrará debajo de cada gráfico correspondiente un botón en donde se podrán descargar detalles de las estadísticas en formato JSON en la carpeta Documentos del sistema operativo Windows OS. Estos detalles simplemente nos dirán los nombres de qué contraseñas han caído en cada parte de las estadísticas. Hay algunas estadísticas que pueden tardar mayor tiempo en generarse, esto depende del número de contraseñas que contenga la Passguard Vault.

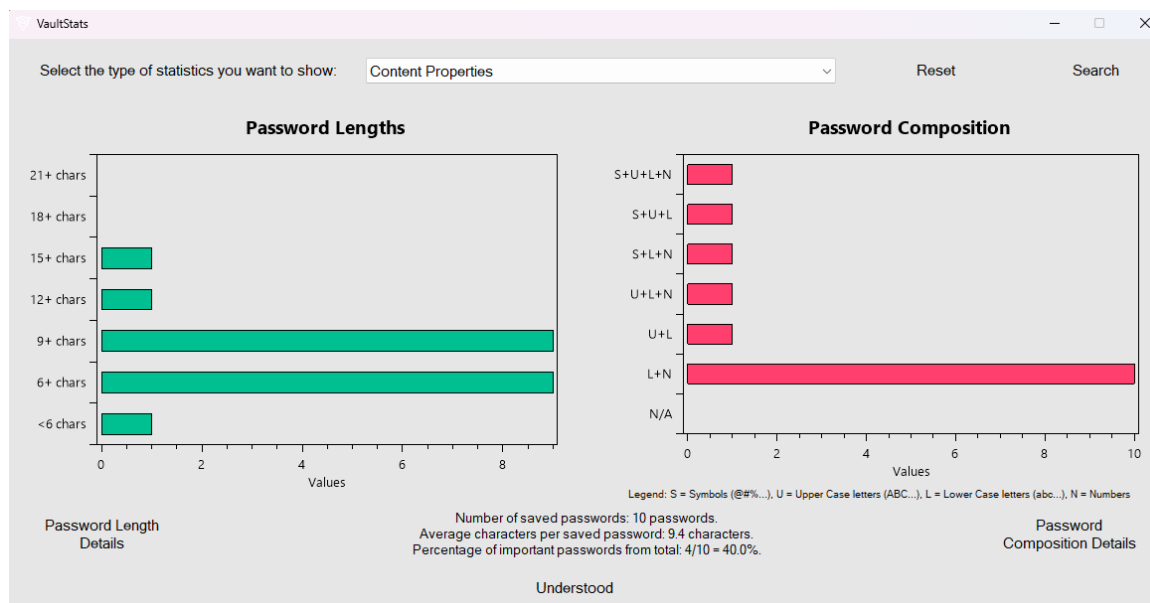


Figura A.11: Ventana de generación de estadísticas.

A.5 Creación de contraseñas aleatorias rápidas

En la vista principal de la aplicación si le damos clic al botón Create Quick Password accederemos a la siguiente vista con cinco partes diferenciadas:

The screenshot shows the PassGuard application window titled 'CREATING SAFE PASSWORDS'. The interface is divided into a left sidebar and a main content area.

Left Sidebar: Contains three buttons: 'New Password Vault', 'Load Password Vault', and 'Create Quick Password'.

Main Content Area:

- Select the characteristics your passwords must have:**
 - ☒ Include Upper Case Characters (e.g. ABCDEFGH).
 - ☒ Include Lower Case Characters (e.g. abcdefgh).
 - ☐ Include Numbers (e.g. 0123456789).
- Include Symbols:**
 - ☐ Include Symbols. (Note about Symbols)
 - Grid of symbol checkboxes: !, \$, %, &, /, \, (,), ^, ~, ?, |, @, #, €, <, >, [,], {, }, ., :, _ , , ;, ñ, Ñ, +, -, *.
 - ☐ =, ☐ ç, ☐ Ç, ☐ ¡
 - Buttons: 'Select All Symbols' and an information icon.
- password(s) to generate:** A text input field with '1' entered.
- My password(s) must have a length of:** A spin box set to '5'.
- ☐ Only generate passwords that have not been pwned until today (recommended, lengthier, +secure).
- Generate Password(s)** button.
- You(r) password(s) will appear here:** A large text area for the generated passwords.

Figura A.12: Vista de creación de contraseñas aleatorias rápidas con cinco partes importantes.

La primera parte importante contiene las casillas dedicadas a seleccionar letras mayúsculas, letras minúsculas o números. La segunda parte importante incluye una casilla para incluir símbolos, la cual si se activa entonces se habilitarán muchas casillas para seleccionar distintos símbolos, además de un botón para activar o desactivar todos los símbolos, y un botón con una nota respecto a los símbolos, la cual mostrará un mensaje emergente el cuál informa que, aunque se escojan múltiples símbolos, es altamente improbable que salgan todos esos símbolos en las contraseñas generadas, pero se garantiza que al menos uno de los símbolos generados saldrá.

La tercera parte importante contiene dos componentes NumericUpDown para elegir por una parte la cantidad de contraseñas a comprobar, la cual puede ser desde una hasta 512 contraseñas, y para elegir la longitud de las contraseñas, la cual puede ser desde 5 o 16 caracteres hasta 52 caracteres. El valor mínimo del tamaño de las contraseñas depende de la cuarta parte importante, la cual es la casilla en la que elegir si queremos que se generen contraseñas que no han sido comprometidas en previas brechas de seguridad. Al lado de esta casilla el usuario tiene un botón con ícono de información que si le da clic le aparecerá una ventana emergente con información de cómo se realiza este proceso. En caso de que esté activada esta casilla entonces el mínimo tamaño de las contraseñas deberá ser de 16 caracteres y al menos algún tipo de letras debe estar activado, en caso contrario el mínimo será de 5 caracteres.

La quinta parte importante es la caja de texto donde aparecerán las contraseñas en cuanto el usuario pulse el botón Generate Passwords. En caso de que algún parámetro no sea válido entonces se le notificará al usuario con un mensaje emergente, de lo contrario empezarán a salir las contraseñas hasta que se complete la petición del usuario. Una vez completada la petición, el usuario podrá hacer clic en el botón situado a la derecha de la caja de contraseñas con un ícono de un papel para copiar el contenido de la misma al portapapeles.

A.6 Menú de opciones en la vista principal

Si estamos en la vista principal de la aplicación y le damos clic al botón con un ícono de engranaje se nos desplegará el siguiente menú de opciones:

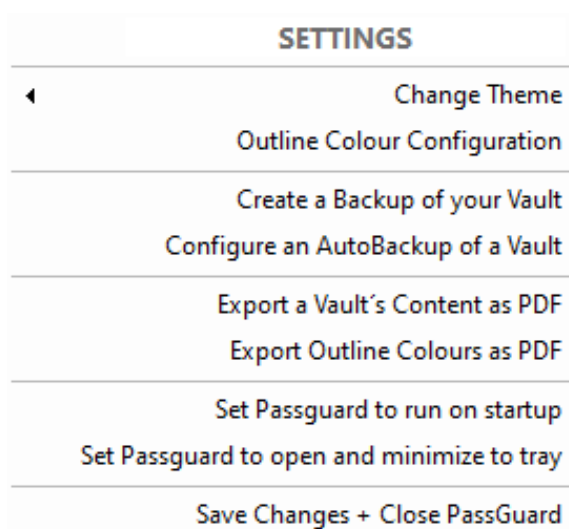


Figura A.13: Menú de opciones en la vista principal.

En caso de hacer clic en la opción Change Theme nos desplegará una segunda lista desplegable con las opciones para tema claro y tema oscuro. En cuanto hagamos clic en alguna de ellas el programa nos preguntará si queremos guardar esta configuración, además de para esta ejecución, para futuras ejecuciones de la aplicación. Se guardará la respuesta del usuario adecuadamente y se procederá a cambiar el tema.

Al hacer clic en las opciones para exportar contenidos como PDF, en el caso de exportar los colores de configuración como PDF se realizará de manera inmediata, sin embargo, para los contenidos de un Passguard Vault accederemos a una vista modificada pero muy similar a la vista de carga de una Passguard Vault, en donde deberemos introducir las credenciales y darle clic al botón de Export Vault as PDF para

exportar los datos como PDF. En ambos casos esos archivos PDF se exportarán en la carpeta Documentos del sistema operativo Windows OS.

La opción de establecer Passguard para que se inicie o no con Windows se puede activar desde este menú de opciones, y antes de hacerlo preguntará por confirmación al usuario, si esta opción está activada en cuanto el usuario encienda su dispositivo y cargue Windows se iniciará también Passguard, esta opción se puede desactivar en el mismo lugar donde se activó. Por su parte, la opción de establecer Passguard para que se abra y se minimice en segundo plano también preguntará por confirmación al usuario, si esta opción se activa en vez de cerrarse completamente la aplicación se pondrá en segundo plano, sabremos esto debido a que el ícono de Passguard se encontrará en el Tray de Windows, es decir, en la parte derecha de la barra de tareas. Si ambas opciones están abiertas, en cuanto se inicie la aplicación Passguard se iniciará en segundo plano. Esta opción también se puede desactivar donde se activó.

La opción de Save Changes + Close Passguard simplemente cierra la aplicación, que dependiendo del modo establecido la cerrará completamente o sólo la minimizará y se seguirá ejecutando en segundo plano.

En siguientes apartados se desarrollarán el resto de las opciones no comentadas hasta ahora del menú de opciones.

A.6.1 Crear una copia de seguridad de un Passguard Vault

Si le damos clic a la opción de Create a Backup of your Vault nos saldrá la siguiente ventana emergente:

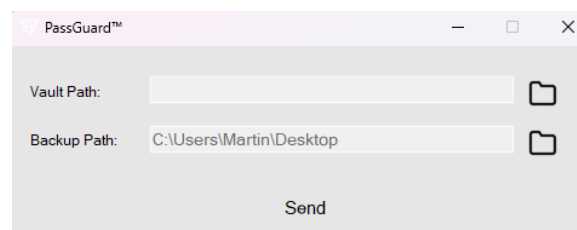


Figura A.14: Ventana emergente de creación de una copia de seguridad de una Passguard Vault.

Con los botones con íconos de carpetas podremos elegir en primer lugar la ruta de la Passguard Vault a la que realizarle la copia de seguridad, y en segundo lugar la ruta donde querremos guardar la copia de seguridad generada. Si los datos introducidos son correctos, se nos generará en la ruta especificada una copia de seguridad con nombre BackupNombreVault-yyyymmdd-hhmmss.encrypted, la cual será accesible con las mismas credenciales que la Passguard Vault original y el programa nos avisará con un mensaje emergente de completado con éxito. En caso de que algún parámetro de entrada sea inválido o que ya exista una copia de seguridad con el mismo nombre en la misma ubicación, entonces se avisará al usuario con un mensaje emergente de error.

A.6.2 Crear copias de seguridad automáticas de un Passguard Vault

En caso de darle clic a la opción de Configure an Autobackup of a Vault, nos aparecerá una ventana emergente de configuración. Lo primero que vemos en esta ventana es una casilla para poder activar o desactivar las copias de seguridad automáticas. Si esta casilla se encuentra desactivada las copias de seguridad también lo estarán, si se encuentra activada podremos acceder a los campos de configuración de Autobackup. En primer lugar, deberemos especificar la ruta donde se encuentra la Passguard Vault que queremos activarle las copias de seguridad automáticas, en segundo lugar, deberemos especificar la ruta

donde queremos que estas copias de seguridad se vayan almacenando. Estos dos pasos se pueden llevar a cabo con sus respectivos botones con íconos de carpeta. Después de esto tenemos una lista desplegable para elegir la frecuencia con la que queremos que se realicen las copias, hay cinco frecuencias disponibles: justo después de cualquier cambio en la Passguard Vault, al cerrar Passguard, cada un día, cada siete días (semana) o cada treinta días (mes).

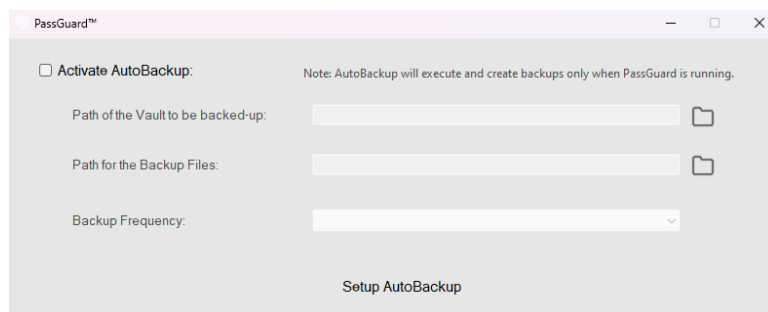


Figura A.15: Ventana emergente de configuración de copias de seguridad automáticas de un Passguard Vault.

Una vez elegidos todos los parámetros correctamente, al darle clic al botón de Setup nos lanzará un mensaje emergente de correcta configuración en ese caso, o un mensaje emergente de error en caso contrario. Cabe destacar dos aspectos importantes de cómo funciona Autobackup, el primero de ellos es que si al realizar la copia de seguridad automática hay algún problema, por ejemplo, que la Passguard Vault no se encuentre en la ruta especificada, o que la ruta de guardado de las copias automáticas desaparezca, entonces al usuario le saltará un mensaje emergente de error en el momento de realización de la copia de seguridad automática en el que se le informará de que ha habido un error al hacer la copia de seguridad y que revise los posibles motivos, este mensaje automático le saltará al usuario cada un minuto hasta que solucione los errores o desactive Autobackup en el menú de opciones. El segundo aspecto destacado es que las copias de seguridad automáticas solo se realizan cuando Passguard está en ejecución, es decir, si en un momento determinado toca hacer la copia de seguridad automática ya que ha pasado un intervalo concreto (un día, siete o treinta días), si el usuario no tiene abierto y ejecutándose Passguard esa copia de seguridad no se realizará, el usuario deberá abrir la aplicación y esta comprobará las fechas y en caso de ser tiempo de crear una copia de seguridad automática la creará y guardará apropiadamente, hay una nota en la parte superior derecha de la ventana donde se avisa de este funcionamiento.

A.6.3 Cambiar los colores de contorno de Passguard

Estando en el menú de opciones de la vista principal de la aplicación, si le damos clic a la opción Outline Colour Configuration nos saltará una ventana emergente con el gestor de configuración de colores de contorno. En la parte superior derecha de la imagen tenemos un botón con ícono de interrogación, si el usuario hace clic en ese botón se le abrirá su navegador web de preferencia con una página web para seleccionar un color de manera visual y saber su combinación RGB, que es al final con lo que funciona este gestor de colores. En la vista destaca por una parte una tabla con ocho columnas y dos configuraciones de colores preguardadas, debajo de esa tabla tenemos un campo de texto de búsqueda y dos botones asociados, y debajo de eso tenemos seis componentes NumericUpDown distribuidos en dos filas y tres columnas con valores RGB de colores. Desde la parte derecha tenemos siete botones para realizar diferentes acciones, excluyendo el botón con ícono de interrogación.

La tabla tiene ocho columnas, seis de ellas se componen de botones y dos de ellas de casillas: el nombre de la configuración, los valores de Red, Green y Blue, un visor donde se puede ver el color

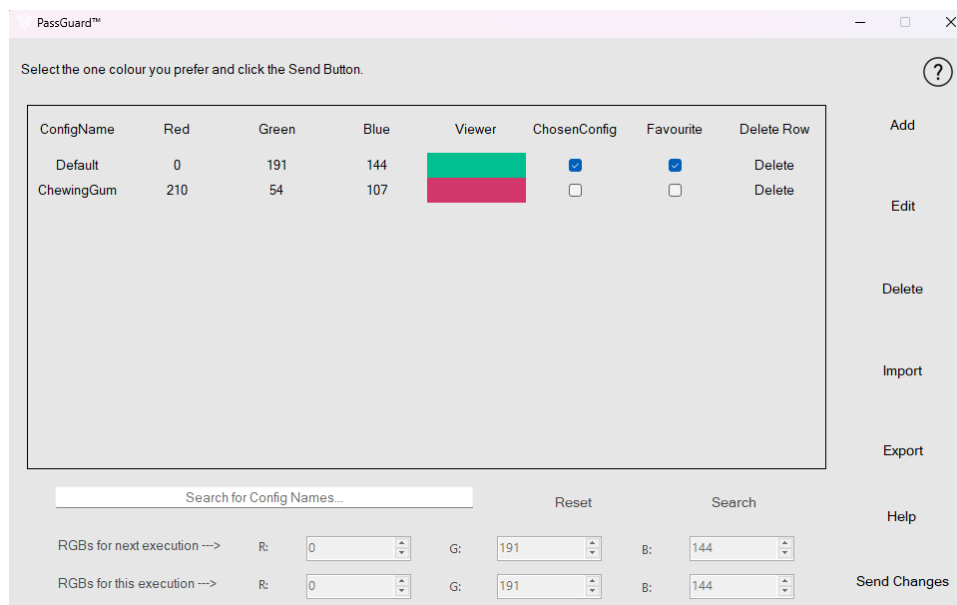


Figura A.16: Ventana emergente del gestor de configuraciones de colores de contorno.

descrito por las anteriores tres columnas, una columna para saber si ese color está siendo utilizado en esta ejecución o no, una columna para marcar una configuración como favorita, y por último una columna con botones para borrar esa fila (y, por tanto, esa configuración) guardada. Si hacemos clic en algunas cabeceras nos saltará una lista de opciones para ordenar el contenido de la tabla por esa columna y en el orden seleccionado, el cual puede ser normal, ascendente o descendente alfabéticamente. Esto funciona de manera similar a la ordenación de la tabla de contraseñas explicada en anteriores apartados. La tabla está compuesta por botones y casillas, en el caso de los botones se puede copiar el texto de los botones (que son los datos guardados de la configuración, por tema de tamaño es posible que no aparezca toda la información) al portapapeles haciendo clic en los botones que contengan texto, excepto en los botones de la columna Delete Row. Solo puede haber una configuración seleccionada en la columna ChosenConfig, que será la configuración utilizada en este momento (en caso de tenerla guardada). En caso de querer utilizar otra configuración guardada basta con hacer clic en la casilla asociada a su fila y nos saltará primero un mensaje emergente de confirmación de uso de esa configuración, y en caso de aceptar nos saltará un mensaje emergente preguntando si queremos mantener esta configuración sólo para esta ejecución o también para futuras ejecuciones de la aplicación, dependiendo lo que decida el usuario se guardarán sus preferencias adecuadamente. En el caso de hacer clic en la columna para marcar o desmarcar una configuración como favorita, nos saltará un mensaje emergente de confirmación de la acción, dependiendo de la respuesta del usuario se procederá a guardar sus preferencias. En el caso de querer borrar rápidamente una configuración guardada, el usuario podrá hacer clic en el botón Delete asociado a la fila de esa configuración, se le pedirá confirmación de la acción y dependiendo de su respuesta se recargará o no la tabla con las configuraciones guardadas.

Debajo de la tabla hay un campo de texto para buscar columnas por la columna ConfigName, en cuanto se introduzca texto en el mismo se activarán los dos botones asociados Reset y Search. El botón Search realizará la búsqueda, y los resultados se mostrarán en la tabla de configuraciones, el botón Reset vaciará el campo de texto y mostrará en la tabla de configuraciones las configuraciones de colores guardadas hasta el momento. Debajo de este campo de texto y sus dos botones tendremos los seis componentes NumericUpDown mencionados anteriormente. Los tres componentes localizados en la primera fila muestran el valor RGB guardado para utilizarse en futuras ejecuciones, los tres componentes localizados en la

segunda fila por su parte muestran el valor RGB guardado para utilizarse en esta ejecución, en cuanto se cierre el gestor de colores enviándose los cambios.

En la parte de la derecha comenzaremos a redactar acerca del funcionamiento cuando se pulsan cada uno de los siete botones. Si pulsamos el botón Add nos llevará a la siguiente ventana emergente:

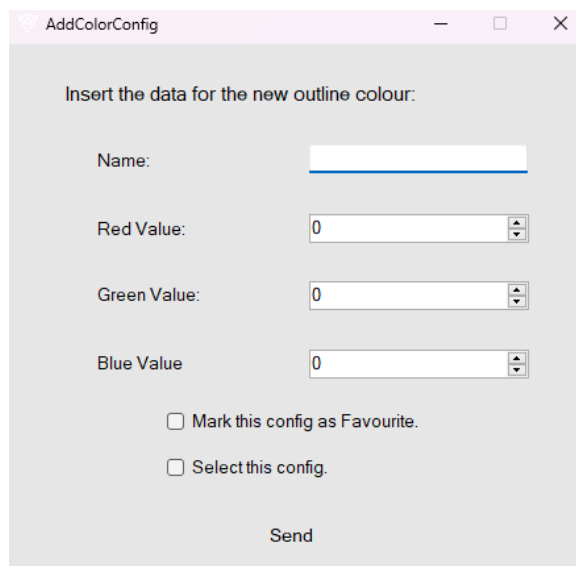


Figura A.17: Ventana emergente para añadir una configuración de colores.

En esta ventana deberemos introducir los datos para el nombre de la configuración (único, debe ser distinto a los nombres de configuraciones ya guardados), la combinación RGB (única, no puede haber dos configuraciones de colores con el mismo valor RGB) y se podrá marcar la configuración como favorita. Cabe resaltar que configuraciones RGB muy claras u oscuras no están permitidas ya que pueden dificultar el uso de Passguard, el criterio es que la luminosidad asociada al color esté entre un 20 y un 90 por ciento, y esto proviene de la escala HSL, donde la L significa luminosidad. Una vez realizado, si le damos al botón Send y hay algún problema se nos notificará con un mensaje emergente, de lo contrario se añadirá la configuración y se seleccionará automáticamente para utilizarse en esta ejecución de la aplicación, si se quiere utilizar para futuras ejecuciones en la ventana de Add se podrá marcar la opción Select this config.

En caso de hacer clic al botón Edit nos saltará una ventana emergente similar a la del mismo caso para la tabla de contraseñas. En esta ventana tendremos una lista desplegable con los nombres de las configuraciones, y en cuanto se seleccione uno se activarán los campos de abajo para modificar sus propiedades, una vez se modifiquen el usuario dará clic a Edit y se comprobarán los nuevos datos, estas comprobaciones son similares a las realizadas al añadir una nueva configuración. Si ha habido algún problema se le notificará al usuario con un mensaje emergente de error, de lo contrario se editará correctamente y se volverá a la anterior vista.

Si queremos borrar una configuración bastará o con darle al botón Delete ubicado dentro de la tabla y asociado a la fila de la configuración, o hacer clic en el botón Delete fuera de la tabla, el cual nos abrirá una ventana emergente similar a la que sale con la misma acción en la vista de tabla de contraseñas. En la lista desplegable seleccionaremos el nombre de la contraseña a borrar y podremos darle al botón de Delete Selected Element para borrar esa contraseña, o podremos activar la casilla para habilitar el botón que borra todas las configuraciones de colores guardadas. En ese caso le daremos clic al botón Delete All Elements para borrar todas las configuraciones. En ambos casos se le pedirá una confirmación de acción al usuario, ya que se tratan de operaciones irreversibles, y en caso de que se borren todas las configuraciones se añadirá la configuración por defecto.

Si lo que queremos es importar unas configuraciones que hemos exportado previamente en formato JSON, el usuario deberá dar clic al botón Import. En este caso, al usuario le saltará un mensaje emergente de aviso, en el que se indica que, si se importan datos de un JSON, todas las configuraciones previas que tuviese guardadas y que se muestran en su tabla de configuraciones se perderán irreversiblemente, ya que se sustituirán por las configuraciones que importe a través de ese archivo JSON, y se le preguntará al usuario si está de acuerdo en seguir una vez entendido esto. En caso de que esté de acuerdo, el usuario podrá elegir un archivo JSON y Passguard validará la forma y el contenido del mismo, en caso de ser correcto se importará el archivo y las configuraciones de ese archivo se mostrarán en la tabla de configuraciones, también se guardarán en memoria. En caso de que haya algún error con la forma o el contenido del archivo JSON indicado se le informará al usuario con un mensaje emergente de error.

El usuario podrá exportar el contenido de su tabla de configuraciones en un archivo dándole clic al botón Export. En este caso le saltará una ventana emergente con dos opciones para elegir, si exportar el contenido como archivo JSON o como archivo PDF. En cuanto elija uno y le dé al botón de Export los datos se exportarán en la carpeta Documentos de su sistema operativo Windows OS con el nombre de archivo ColoursTable-yyyyymmdd-hhmmss.pdf en caso de archivo PDF, y terminará en .json en caso de archivo JSON.

Si el usuario tiene dudas de cómo funciona esta vista con la tabla de configuraciones de colores, puede hacer clic en el botón Help, el cual le abrirá una ventana emergente en la que podrá leer información acerca de cómo funciona esta vista. Todas las vistas con información de funcionamiento tienen un aspecto similar al siguiente, en este caso se muestra la ventana emergente de ayuda para utilizar el gestor de configuraciones de colores de contorno:

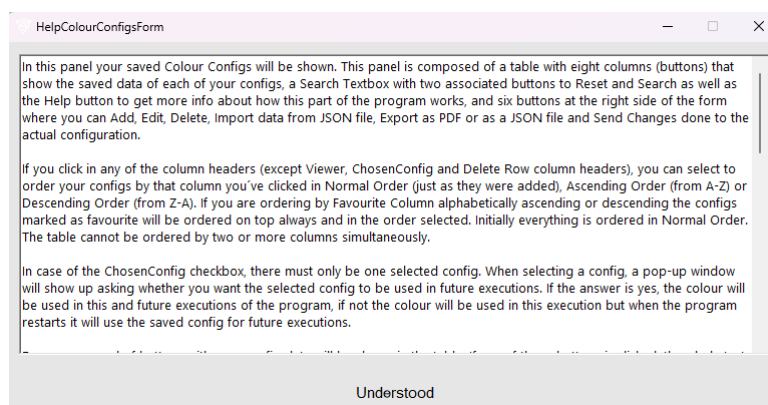


Figura A.18: Ventana emergente de ayuda al funcionamiento de la vista con tabla de configuraciones de colores de contorno.

El botón Send Changes, por último, cierra esta ventana con la tabla de configuraciones y aplica los cambios definitivamente, es decir, si el color para esta ejecución ha cambiado entonces refresca la ventana principal de la aplicación y muestra el nuevo color, y guarda definitivamente el color establecido para las futuras ejecuciones.

Apéndice B

Manual de Despliegue e Instalación de Passguard

B.1 Introducción

En este apartado se redactará acerca de cómo desplegar e instalar Passguard. Hay dos formas para realizar esto dependiendo del público, el público general simplemente querrá instalar la aplicación y que les funcione en su dispositivo, los desarrolladores por su parte querrán desplegar el código y probarlo desde su entorno de desarrollo integrado.

La aplicación funciona de manera probada tanto en los sistemas operativos Windows 10 como Windows 11, es posible que funcione en otras versiones de Windows, pero no se ha probado. Partiremos de un archivo con extensión .zip donde, entre otros archivos o carpetas, tendremos una carpeta Passguard que si abrimos tendremos dos carpetas llamadas SourceCode e Installers, además de un archivo de texto plano llamado Dependencies.txt. Dependiendo del tipo de usuario nos meteremos en una carpeta o en otra. A continuación, se explica el despliegue e instalación para cada tipo de usuario.

B.2 Instalación de Passguard para el público general.

Si simplemente queremos instalar la aplicación y que funcione en nuestro dispositivo, y no nos importa mucho el código que hay detrás de la misma, dentro de la carpeta Passguard nos meteremos en la carpeta Installers. En esa carpeta encontraremos, entre otros archivos, un archivo llamado Setup.exe. En el archivo Dependencies.txt mencionado al principio se muestran los frameworks y archivos que hay que tener instalados para que se ejecute la aplicación, de lo contrario no se podrá instalar:

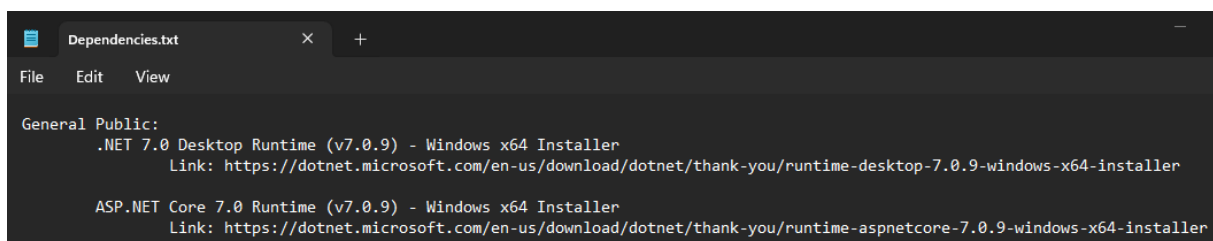


Figura B.1: Listado de dependencias para el instalador de Passguard.

Una vez el usuario haya instalado las dependencias contenidas en el archivo de texto, entonces podrá ejecutar el archivo Setup.exe para instalar Passguard. En caso de no tener instaladas las dependencias y hacer doble clic en el instalador, este le pedirá al usuario las dependencias, y aunque parezca que haya instalado Passguard, en verdad el instalador lo ha hecho de manera parcial, hasta que se instalen las dependencias y se pueda ejecutar de manera completa la aplicación.

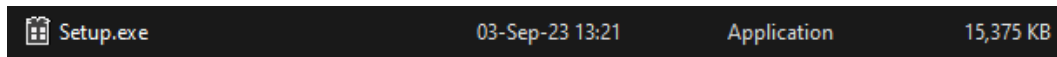


Figura B.2: Archivos con los que se trabajará la instalación de Passguard.

A continuación, se detallan los links de las dependencias:

- .NET 7.0 Desktop Runtime (v7.0.9) - Windows x64 Installer
 - Link: <https://dotnet.microsoft.com/en-us/download/dotnet/thank-you/runtime-desktop-7.0.9-windows-x64-installer>
- ASP.NET Core 7.0 Runtime (v7.0.9) - Windows x64 Installer
 - Link: <https://dotnet.microsoft.com/en-us/download/dotnet/thank-you/runtime-aspnetcore-7.0.9-windows-x64-installer>

El instalador no tiene pasos adicionales en sí, simplemente al ejecutarse instalará la aplicación y dejará un ícono en el escritorio. En caso de querer desinstalar la aplicación, el usuario podrá ir al Panel de Control de Windows, o al apartado de Agregar o Quitar programas en la configuración de Windows, buscar Passguard y desinstalarla de manera sencilla.

B.3 Despliegue de Passguard para desarrolladores.

En el caso de que queramos ver y probar el código de Passguard en nuestro entorno, deberemos en la carpeta Passguard abrir la carpeta SourceCode. La aplicación ha sido desarrollada con la versión 7.0 del SDK .NET de Microsoft, y para utilizar ese .NET se necesita tener Visual Studio 2022 como entorno de desarrollo integrado. En el archivo Dependencias.txt se anota las dependencias y programas necesarios para este tipo de usuarios:

```
Developers:
.NET SDK 7.0 - Windows x64 Installer
Link: https://dotnet.microsoft.com/es-es/download/dotnet/thank-you/sdk-7.0.400-windows-x64-installer
Visual Studio 2022 - Windows x64 Installer
Link: https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022
&source=VSLandingPage
```

Figura B.3: Lista de programas y dependencias para desarrolladores en Passguard.

A continuación, se detallan los links de las dependencias:

- .NET SDK 7.0 - Windows x64 Installer
 - Link: <https://dotnet.microsoft.com/es-es/download/dotnet/thank-you/sdk-7.0.400-windows-x64-installer>
- Visual Studio 2022 - Windows x64 Installer

- Link: <https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Release&version=VS2022&source=VSLandingPage>

Dentro de la carpeta SourceCode tendremos múltiples carpetas, entraremos en la carpeta Passguard y buscaremos el archivo llamado PassGuard.sln, este es el archivo que deberemos abrir con Visual Studio 2022 y .NET7.0. La ruta de este archivo será Passguard ->SourceCode ->PassGuard ->PassGuard.sln.

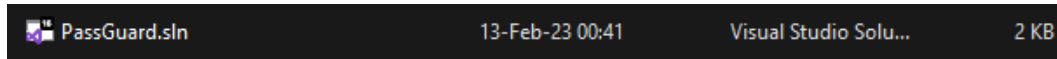


Figura B.4: Archivo solución de PassGuard, debe ser abierto con Visual Studio 2022.

Una vez abierto este archivo con Visual Studio se tendrá acceso a las diferentes clases y al proyecto Passguard. Para ejecutarlo, se escogerá la versión Any CPU y en modo Release y se le dará a Ejecutar sin depurar:

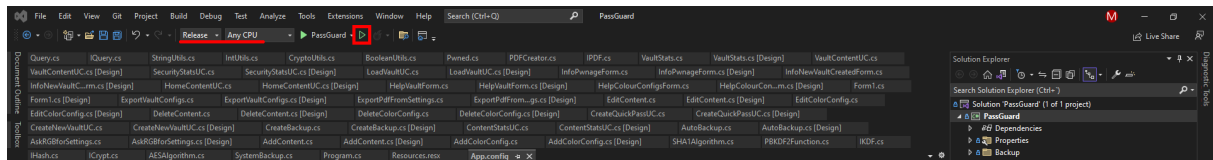


Figura B.5: Configuración de Visual Studio 2022 para ejecutar el proyecto Passguard.

Con esto, Passguard se habrá ejecutado correctamente utilizando un entorno de desarrollo integrado IDE y su kit de desarrollo de software SDK.

Universidad de Alcalá
Escuela Politécnica Superior



ESCUELA POLITECNICA
SUPERIOR



Universidad
de Alcalá