



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**CERRADURA DIGITAL de ALTA
SEGURIDAD para CAJAS FUERTES**

Autor: Pablo Castillo Martínez

Tutor(a): Jorge Dávila Muro

Madrid, diciembre 2020

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: CERRADURA DIGITAL de ALTA SEGURIDAD para CAJAS FUERTES

Diciembre 2020

Autor: Pablo Castillo Martínez

Tutor:

Jorge Dávila Muro

Lenguajes y Sistemas Informáticos e Ingeniería de Software

ETSI Informáticos

Universidad Politécnica de Madrid

Resumen

A lo largo de la historia, numerosas innovaciones tecnológicas han revolucionado por completo la industria de la seguridad física y las cajas fuertes.

Desde mejores materiales, a mejor lógica interna, a funciones especializadas (para el uso en sucursal bancaria, privado, en comercios...), el último siglo ha traído con él grandes avances para desbaratar los intentos de robo de aquellas posesiones guardadas bajo llave.

Otros campos que van a la par son el de la criptografía y la seguridad informática, que en mucho menos tiempo han desarrollado cientos de métodos y estándares con los que mantener la seguridad de nuestra información y conexiones, y que hoy en día siguen expandiéndose. Proteger nuestros datos personales (cuentas bancarias, contraseñas...), posiblemente nuestros datos más preciados, siempre será necesario.

Este trabajo de fin de grado estudiará si es posible, y cómo, aplicar diferentes técnicas de la criptografía y la seguridad modernas para la creación de una cerradura digital de alta seguridad, además de los fundamentos detrás de cada método y la seguridad asociada.

Palabras clave: SHA, HMAC, HOTP, cerradura, seguridad

Abstract

Over time, several technological developments have revolutionized completely the security and safe industry.

From better building materials to an improved inner logic, to specialized functions for them (designed for banks, houses, stores...), the last century has brought with it new ways to stop the attempts from other people to steal our valuables.

Other areas that have also evolved are Cryptography and Computer Security, areas that in far less time have created hundreds of protocols and standards to secure our information and connections, and nowadays keeps expanding. Protecting our personal information (bank accounts, passwords), possibly our most important information, will always be necessary.

This TFG will study if it is possible and how to apply several modern Cryptography and Security algorithms for the creation of a high security lock, with the concepts behind these methods and the security they provide.

Keywords: SHA, HMAC, HOTP, lock, security

Tabla de contenidos

1	Introducción.....	1
2	Trabajos Previos	2
3	Especificaciones Físicas.....	3
3.1	Materiales	3
3.2	Cerradura	3
3.3	Mecanismo de Cierre	5
3.4	Conclusión sobre el mecanismo	6
4	Diseño del Sistema	7
4.1	Criptografía	7
4.2	Diffie-Hellman	8
4.3	Especificaciones de HMAC y HOTP	10
4.3.1	HMAC	10
4.3.2	HOTP	11
4.4	Función Hash SHA-512.....	14
4.5	Acceso al sistema	14
4.6	Seguridad asociada a la criptografía	16
4.6.1	Diffie-Hellman.....	16
4.6.2	HMAC	17
4.6.3	HOTP	17
5	Componentes del Sistema.....	19
5.1	Sistema Llave	19
5.2	Sistema Caja Fuerte	20
5.3	Circuitos	20
5.3.1	Seguridad en el circuito	21
6	Implementación en código	22
6.1	Clase diffieHellman.py	22
6.1.1	Funciones del archivo	23
6.2	Clase hash.py.....	26
6.2.1	Funciones del archivo	28
6.3	Clase administrador.py	35
6.3.1	Funciones del archivo	35
7	Implementación final y pruebas en Arduino	39
7.1	Curvas elípticas, HMAC y HOTP en Arduino	40
7.2	Código de la implementación	41
8	Vulnerabilidades del sistema.....	47
9	Resultados y conclusiones	49
10	Líneas futuras	51
11	Bibliografía	53

12	Anexo A: Pruebas sobre HMAC y HOTP.....	55
12.1	Pruebas iniciales	55
12.1.1	Grupo 15	55
12.1.2	Grupo 16	55
12.1.3	Grupo 17	56
12.1.4	Grupo 18	57
12.2	Pruebas avanzadas.....	57
12.2.1	Prueba 1	58
12.2.2	Prueba 2	61
12.2.3	Prueba 3	65
12.2.4	Prueba 4	67
12.2.5	Prueba 5	67

1 Introducción

La aplicación de nuevos avances tecnológicos a los mecanismos diseñados para mantener alejadas de nuestras posesiones a personas sin el permiso adecuado siempre ha estado a la orden del día. Desde las cerraduras más tradicionales, pasando por las que aprovechan los efectos del electromagnetismo y la corriente eléctrica para evitar un forzado tradicional, a la tecnología de lector de tarjetas o claves, sensores de biometría... para probar nuestra identidad, las cerraduras han avanzado a pasos agigantados.

Pero igual que ha avanzado la tecnología para darnos seguridad de la mano de la comodidad y los últimos avances (la posibilidad de no llevar llaves encima, de poder conectar nuestros teléfonos para abrir una puerta, usar nuestra huella como método para identificarnos...), también han avanzado las habilidades y conocimientos de aquellos interesados por romper estas defensas.

Pero ¿cómo se adaptan las cerraduras electrónicas a cajas fuertes? Si bien es cierto que las cerraduras más modernas [1][2] (con contraseña electrónica, con lector ID, con conexión Bluetooth al móvil...) bien aplicadas por marcas de confianza son igual de seguras que las cerraduras tradicionales, suelen destacar marcas comerciales menores, poseyendo unos materiales deficientes en comparación, o mayor facilidad para forzarlas por fallos de diseño. No por ser digitales proporcionan una seguridad adicional, y menos en tiempos del IoT ("Internet de las cosas").

Existe un amplio mercado basado en la venta de cajas fuertes (de distintas categorías, tamaños, funciones...) e instalación de bóvedas bancarias con características como contraseña, lector de ID, conexión móvil, adicional a las combinaciones o llaves tradicionales. La protección que otorgan está regulada por distintos estándares nacionales e internacionales.

Pero en esta área cabe aún espacio donde innovar. Es posible añadir componentes criptográficos basados en un secreto entre la cerradura y el usuario que haga que la combinación a introducir sea diferente en cada uso que se realice. Esta cerradura tiene que proporcionar la misma seguridad que los métodos tradicionales, ser difícil de sustraer o interceptar sus mensajes de clave y, además, registrar todas las combinaciones o claves usadas para no utilizarlas de nuevo.

Éste trabajo pretende diseñar e implementar un sistema de alta seguridad para una caja fuerte, con técnicas que son actualmente usadas en la seguridad y criptografía modernas.

También se pretende estudiar la fiabilidad que puedan proporcionar estas técnicas tras llevar varias décadas en uso.

2 Trabajos Previos

Si se pretenden aplicar principios de seguridad criptográficos a los sistemas tradicionales, primero es necesario una breve introducción sobre la historia de éstos en el mundo moderno.

Si lo que nos interesa es un sistema con una clave numérica secreta, una combinación, capaz de cambiar tras cada uso, debemos investigar los principales mecanismos que existen actualmente para generar contraseñas de un sólo uso.

En el mundo comercial destacan los servicios proporcionados por Google (y empresas que permiten el uso de éste en su software) y su Google Authenticator [3]. Aparte de la información normal de usuario y password, se genera una “contraseña” de un sólo uso que se manda a un canal seguro proporcionado por el usuario para probar que es él mismo el que accede, ya que dispone de información conocida para el acceso, y puede iniciar sesión en el canal seguro.

Éste es el mundo de la autenticación de múltiples factores, o AMF [4] (que también suele ser conocida por autenticación de dos factores, ya que en el uso habitual sólo se usan dos para confirmar la identidad del usuario).

Combinando diferentes componentes que debe poseer el usuario (algo que se tiene, algo que se sabe, o algo que se es), se consigue una seguridad total mayor.

Éste y otros métodos operan bajo los algoritmos HMAC (Hash-based message authentication code) y HOTP (HMAC-based one-time Password). Ambos operan bajo sus respectivos RFC, request for comments, documentos del IETF, Internet Engineering Task Force, que dictan con detalle cómo y con qué parámetros formar estos algoritmos, que se explicarán en más detalle en la parte de codificación del proyecto.

Los RFC de más interés para este trabajo son los siguientes: RFC 4226, RFC 6238 (HOTP y TOTP), RFC 2104 y RFC 4868 (ambos de HMAC). [5][6][7][8]

3 Especificaciones Físicas

A continuación, se pasará a describir con detalle las características físicas de los mecanismos de cerradura y seguridad actuales.

3.1 Materiales

Según la definición de caja fuerte por la Real Academia Española, una caja fuerte, o caja de caudales, es “una caja blindada para guardar dinero y cosas de valor” [9].

En su forma más simple, una caja fuerte consiste en una capa exterior de un material duradero, difícil de penetrar, con un espacio limitado en su interior, seguro y bien definido, y un mecanismo de apertura a través de un secreto que se tiene (una llave), se sabe (una combinación), o se es (biometría).

Los materiales exteriores pueden variar dependiendo de la función de la caja fuerte (uso doméstico, uso comercial, uso como armero, bóveda bancaria...), el tipo de objetos que pueda contener, o la resistencia a situaciones límite (a fuego, a explosivos...).

Los materiales de construcción pueden ir desde el plástico endurecido, pasando por el acero, hasta llegar al hormigón armado con fibras metálicas, capaces de romper taladros industriales. Numerosos estándares existen sobre los materiales necesarios y su grado de seguridad, medidas y demás elementos físicos necesarios para asegurar la seguridad exterior, como el BOE número 42, de 18/02/2011, sobre medidas de seguridad privada [10], la norma UNE EN-1143/1 o la norma EN-1300 sobre CAS (cerraduras de alta seguridad).

La puerta por la que se accede suele ser del mismo material que el exterior, dejando fuera del alcance de un atacante cualquier posible punto de acceso obvio, así como el mecanismo con el que se abre y se cierra. Adicionalmente, también cuenta con una placa adicional de un material capaz, como antes, de romper la mayoría de los taladros, protegiendo así la “lógica” interna de la cerradura.

Como este trabajo se centra más en la lógica interna que en las especificaciones físicas, la elección de la caja fuerte se deja a elección del lector o de quien desee llevar los mecanismos descritos en adelante a una caja concreta.

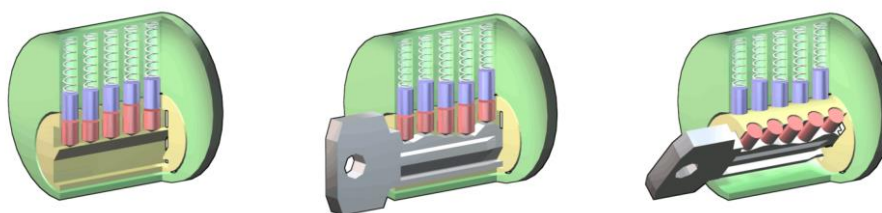
3.2 Cerradura

La cerradura a lo largo de la historia [11] es un mecanismo mecánico o eléctrico que permite el acceso a lo que guarda a través de la apertura del mecanismo cuando se proporciona algún tipo de información secreta (una llave, una combinación, la biometría del usuario, token...) conocida de antemano entre usuario y el propio sistema.

Lo más habitual en el día a día es el uso de una cerradura de tambor de pines en nuestras puertas.

Para abrir esta cerradura, es necesario tener una llave, un trozo de algún material duradero cuyo perfil haya sido moldeado y alterado para seguir un patrón que encaje con el interior de la cerradura.

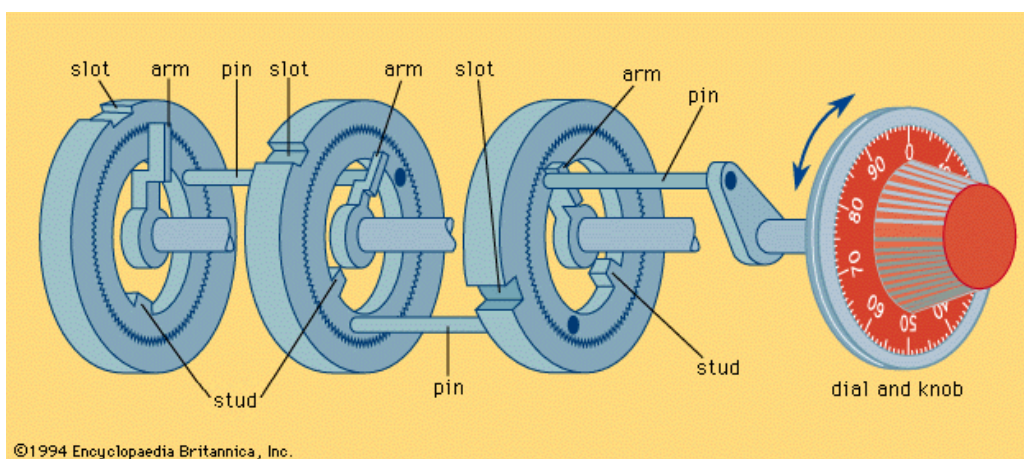
De manera sencilla, este interior consiste en una combinación de varios pines apoyados sobre sus respectivos muelles, todos a distintas alturas. En su estado habitual, prohíben el movimiento del cilindro interior, hasta que se introduce el perfil correcto, que hace que cada pin se levante a la altura adecuada para poder mover el cilindro, ejercer movimiento con la llave y poder abrir la cerradura.



1

Sin embargo, este método da lugar a varias posibles amenazas. Puede suceder que estas llaves sean sustraídas debido al factor físico que presentan; no poseen ningún tipo de secreto adicional, la apertura posterior se producirá sin depender de quién introduzca estas llaves. También puede suceder que se fuercen con la ayuda de ganzúas, a través de su apertura física.

Un mecanismo que no necesita de ningún token físico es la cerradura de combinación, que consiste en una línea giratoria externa que se conecta a unos discos interiores, normalmente tantos como "números" de la combinación haya que introducir.



2

¹ Ejemplo de funcionamiento de una cerradura de tambor de pines. Fuente: GWirken, Wikipedia Neerlandesa.

² Ejemplo de cerradura de combinación. Fuente: Encyclopedia Britannica, <https://www.britannica.com/technology/combination-lock>

Cada disco, a su vez, tiene un agujero. La alineación de todos los agujeros en una posición concreta hace que una pieza en el interior "salte" por la presión y el mecanismo se abra. Cada disco posee una conexión al siguiente disco, y un "diente" en cada lado.

Así, el movimiento de un disco puede mover el siguiente si nos pasamos de largo y reiniciar el sistema, teniendo que empezar de nuevo. Si no llegamos al número correcto, los agujeros no se alinean y continúa cerrado (éste es el principal motivo de que haya que alternar el sentido en el que giramos el dial por número).

Esta combinación podría estar dividida entre 2 o más personas de confianza cuando se trata de un entorno más profesional, para que nadie tenga todos los números a la vez y así disminuir riesgos de intrusiones.

De esta forma, la "clave" es nuestro conocimiento de la combinación. Este conocimiento no puede ser sustraído con tanta facilidad como una llave, pero no es perfecto. Una persona puede observar cómo se introduce la combinación y así obtener el secreto para futuros usos, hasta que se cambie la combinación, momento que puede ser demasiado tarde.

En el caso de los sistemas de alta seguridad profesionales, la apertura suele consistir en varios de estos métodos de cierre, para aumentar su seguridad.

Es posible tener una combinación de varios tipos de cerraduras distintas, teniendo por ejemplo varias llaves (tradicionales o electrónicas), cada una llevada por un empleado, y sólo la unión de todas las llaves da acceso a la cámara, junto al conocimiento de una combinación también, y que sólo sea posible la apertura de la caja/cámara en ciertos momentos del día.

Es por estos motivos anteriores por los que añadir un generador de secuencias aleatorias, usadas una sola vez en la vida del sistema, puede ayudar a crear combinaciones más seguras de un único uso. Además, a pesar de que alguien observe cómo se mete la combinación, al cambiar en cada uso esto no tendría mayor importancia.

El problema, es este nuevo caso, estaría en disponer de un dispositivo que pudiera intercambiar claves con el sistema. Al ser físico, como una llave o una tarjeta, nos encontraríamos de nuevo con un posible problema de sustracción del dispositivo, por lo que sería aconsejable que contara de forma adicional con otra forma de identificar al usuario del dispositivo.

3.3 Mecanismo de Cierre

El mecanismo de cierre en una caja de seguridad es similar al de una cerradura tradicional en una casa moderna, sólo que, en lugar de haber un número bajo de bulones (componentes de acero anti-sierra que se introducen en el marco de la puerta), hay decenas de ellos, a diferentes ángulos y alturas, dependiendo del tamaño y la masa de la puerta, para que sea imposible forzarla de esta forma.

Gran parte de los mecanismos internos modernos de las compuertas son secretos y no están a disposición del público, además de emplear medidas adicionales de seguridad (cámaras, cerraduras temporales que impidan su uso fuera de horario de apertura, seguridad privada...) que no se tratarán de manera directa en este trabajo.

Cuando la combinación introducida en el sistema es correcta y los agujeros de los discos están alineados, el bloqueo cesa en la manivela externa que no dejaba

contraer los bulones, se hace fuerza para recogerlos, y puede hacer fuerza para abrir la compuerta. Esto puede hacerse de manera manual, como los modelos más antiguos, o contar con la ayuda de un motor capaz de mover las toneladas de peso de la compuerta.

3.4 Conclusión sobre el mecanismo

Después de estudiar el mecanismo de cerradura de las cajas fuertes, se llega a la conclusión de que no es posible usar una cerradura tradicional en un sistema moderno de seguridad basado en elementos criptográficos.

Esto se debe a que un sistema con una cerradura "electrónica" no puede disponer de las piezas mecánicas tradicionales que componen una cerradura tradicional; no se puede diseñar un sistema de esta forma (Por no ser compatible y por posibles vulnerabilidades).

Si se desea diseñar un sistema de seguridad capaz de disponer de una contraseña o combinación que cambie en cada uso, se necesita disponer de:

- Token ("llave") portátil **T**. Es capaz de calcular la clave que el sistema central acepta. Esta clave se calcula a través de un reto (un mensaje) que el sistema manda a **T**. Adicionalmente, necesita dar energía al sistema, para que no haya dependencia en caso de que el sistema eléctrico deje de funcionar.
- Sistema central **S**. Manda el mensaje a través del que **T** genera la clave, y compara el resultado de la operación con el suyo propio. Dispone de un teclado en el exterior del sistema de seguridad a través del cual introducir la clave.

Ambos componentes deben de disponer del suficiente poder computacional para calcular las claves a partir de un mensaje inicial.

Al ser necesario un token físico para el acceso a la caja fuerte (la "llave"), es recomendable que se haga inventario regular de quiénes tienen permisos para acceder (si es un entorno bancario o profesional) y disponer de algún componente adicional que pruebe que el portador es quien dice ser.

Obtener el sistema token portátil, de otro modo, garantiza el acceso a la caja fuerte; sólo garantiza que ojos indiscretos no puedan reintroducir la clave que vieron para acceder.

4 Diseño del Sistema

En este nuevo sistema, cada dispositivo que se quiera conectar para pedir acceso tiene un ID. Este ID está asociado a cada usuario y no se podrá transferir (como la dirección física o MAC de las tarjetas de redes tradicionales). A su vez, cada usuario puede tener o no acceso al sistema dentro de la configuración de éste.

El protocolo que se empleará será uno de desafío respuesta, en el que el servidor (en nuestro caso, el controlador dentro de la caja fuerte) manda un desafío al usuario, y éste tiene que responder con la respuesta correcta. Éste desafío es muy flexible, siendo en el uso cotidiano una pregunta que el usuario sabe responder, una contraseña secundaria, etc.

En nuestro caso (y en el recomendado de forma oficial), se plantea calcular una función hash o similar asociada a un mensaje a través de una clave privada que solo conocen el sistema y la llave del usuario. Dicha clave secreta se debe poner en común antes de la comunicación, cuando se establece la conexión.

La parte criptográfica del sistema se describirá en el siguiente subapartado.

4.1 Criptografía

En primer lugar, es necesario hablar del principal componente de la criptografía simétrica.

En el mundo de la seguridad tradicional, se denomina criptografía simétrica a aquellos métodos que emplean la misma clave tanto para cifrar como para descifrar los mensajes intercambiados en la comunicación, a diferencia de la criptografía asimétrica, en la que una clave puede cifrar y otra clave puede descifrar.

A pesar de no cifrar nada en este TFG, y sólo calcular el hash y HMAC, es necesario mencionar esto para el intercambio de claves secretas.

Uno de los principales problemas que introduce este tipo de criptografía es el mencionado intercambio inicial de claves en un medio, y cómo llegar a un acuerdo entre las partes interesadas para acordar una clave en común, y, además, que no se entere nadie que pueda estar escuchando el intercambio por el medio usado.

Un razonamiento inicial para superar este problema puede ser tan sencillo como hacer este intercambio en un medio seguro. En nuestro caso, al disponer de un medio físico donde conectar el dispositivo que proporciona energía a la cerradura, podemos llegar a la conclusión de que, en un primer acceso físico, el medio es seguro para intercambiar la clave inicial que calcule el sistema, y en los siguientes accesos operar con ella como se explicará más tarde.

Sin embargo, ya existen métodos para hacer intercambios sobre un medio público partiendo de unos parámetros iniciales de manera sencilla, por lo que no es ningún problema implementar alguno de ellos.

Por ejemplo, Diffie–Hellman [12][13] (muy común y extendido en el campo desde su nacimiento a mediados de los años 70) puede añadir aún más seguridad al

sistema y generar claves en un medio público sin que nadie más se entere del resultado final.

4.2 Diffie-Hellman

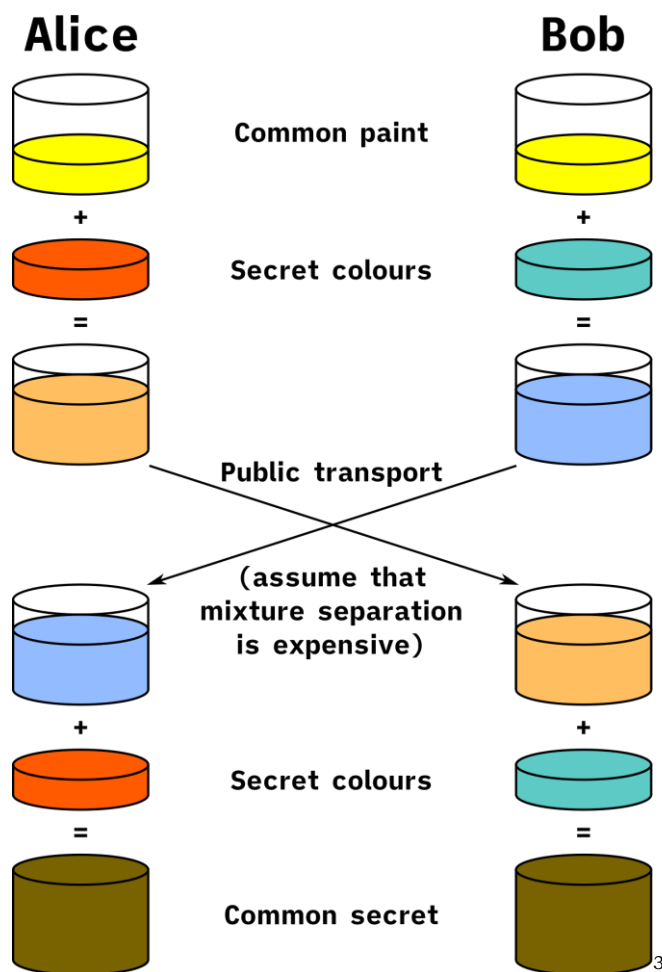
Mediante este sistema, dos interlocutores generarán una clave compartida que, aunque un tercer atacante esté escuchando el medio, no conseguirá ni la clave final, ni tampoco computar la clave a través de los mensajes intercambiados en claro.

Para conseguir esto, se eligen dos números que se encontrarán en público, definidos en el estándar.

Cada uno de los dos miembros que se quieren comunicar eligen un número secreto, que sólo conocen ellos.

Cada miembro hará una operación que combina su número secreto, y los dos números públicos, y guarda el resultado. Estos resultados se intercambian, teniendo en cuenta que es muy difícil conseguir el número secreto original de cada usuario a pesar de conocer el resultado final.

Finalmente, añaden su número secreto al resultado de su compañero, y obtienen el mismo número.



En detalle, los números públicos son un primo p y un generador de números g (enteros menores que p que son primos relativos de p). Son conocidos incluso por posibles adversarios; se encuentran al acceso de todos.

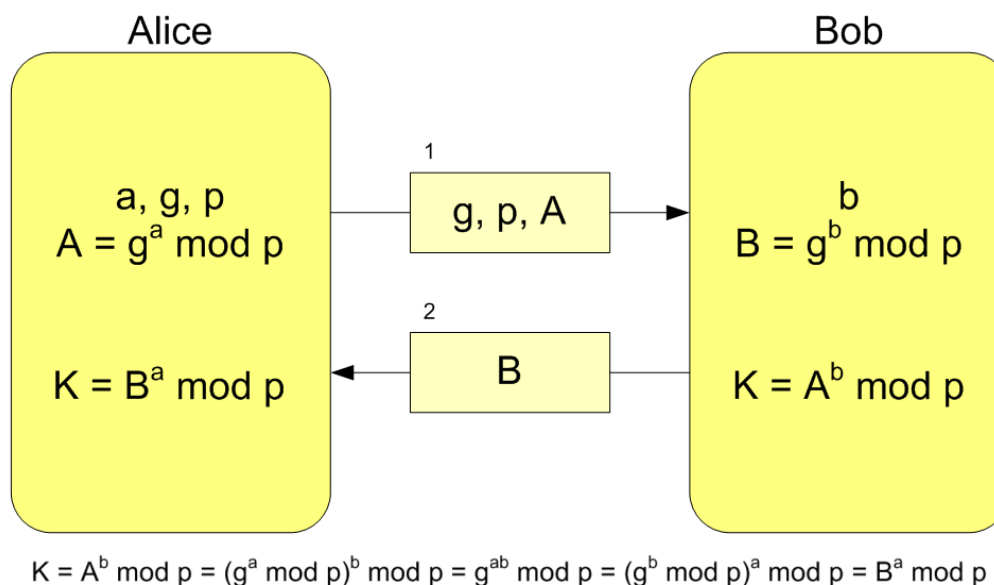
El tamaño de p , en este algoritmo concreto, suele ser de 2000 a 4000 bits, en este caso, pero también son posibles otros valores mucho mayores. El valor del generador suele ser 2, por motivos que se comentarán más en detalle en su apartado correspondiente.

Teniendo a Alice y Bob como miembros de la conexión, Alice elige un número privado a , en un rango de 1 a $p-1$, y calcula $A = g^a \bmod p$, y lo envía a Bob.

Bob hace lo mismo con b , y calcula $B = g^b \bmod p$, y lo envía a Alice. Ninguno de los dos comparte en ningún momento su número privado.

Por las propiedades matemáticas de la operación, ambos serán capaces de calcular $K = g^{(a*b)} \bmod p$ sin conocer directamente los números privados a o b por la otra parte.

³ Ejemplo simplificado de un intercambio entre Alice y Bob mediante Diffie-Hellman.
Fuente: A.J. Han Vinck, Introduction to public key cryptography



4

A continuación, se estudiarán HMAC y HOTP.

4.3 Especificaciones de HMAC y HOTP

HMAC (hash-based message authentication code) es un método de autenticación de mensajes mediante una función Hash y una clave secreta, que comprueba la integridad de los datos y que el mensaje no ha sido alterado.

HOTP (HMAC-based One-time Password Algorithm) por su parte es un algoritmo que trabaja sobre HMAC para generar claves o contraseñas de un sólo uso, que ha sido muy usado en los últimos años para inicios de sesión en compañías como Google.

4.3.1 HMAC

Antes de hablar de HMAC, es conveniente hablar de los MAC. MAC, o Message Authentication Code, es un mecanismo cuya función es la de autenticar un mensaje, habitualmente para comprobar que no ha sido alterado. HMAC nace con el mismo cometido, pero ofrece una seguridad mucho mayor, como se explicará algo más adelante.

Las especificaciones del HMAC se han sacado del RFC (Request For Comments) 2104, que especifica cómo y con qué parámetros se debe implementar para su uso de forma segura.

Como se ha comentado, HMAC es un algoritmo para autenticación de mensajes usando funciones Hash criptográficas. Se puede usar con cualquier función

⁴ Esquema sobre Diffie Hellman. Fuente: Benutzer Stern, Wikipedia

criptográfica Hash que en los últimos años se haya comprobado que sigue siendo robusta.

Los componentes necesarios para construir HMAC son los siguientes:

- Función criptográfica **H** con la que se "hashean" los datos bajo una función de compresión sobre bloques de tamaño B de datos (MD5 y SHA-1, por ejemplo, operan en bloques de 512-bit, o 64 Bytes).
- Clave Secreta **K**.
- Tamaño **B** bytes de bloques con los que se operan.
- Tamaño **L** bytes de salida del algoritmo.

Además, contamos con dos constantes, **ipad** = el byte 0x36 repetido B veces, y **opad** = el byte 0x5C repetido B veces. La operación que se realiza es la siguiente:

$$\text{HMAC}(K, m) = H \left((K' \oplus \text{opad}) \parallel H \left((K' \oplus \text{ipad}) \parallel m \right) \right)$$

$$K' = \begin{cases} H(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

5

Los pasos en detalle son los siguientes:

1. Insertar ceros al final de K para crear un string de B bytes (Si K tiene un tamaño de 20 Bytes y B=64, se insertan 44 Bytes de 0x00). Si es mayor, se hace el hash de K y se opera con él.
2. XOR del paso 1 con el ipad.
3. "Append"/insertar al string de texto junto al resultado del paso 2.
4. Aplicar H a la salida del paso 3.
5. XOR del paso 1 con el opad.
6. "Append"/insertar la salida H del paso 4 a la salida del paso 5.
7. Aplicar H a la salida anterior y devolver el resultado.

La clave usada en HMAC puede ser de cualquier tamaño, pero se desaconseja que sea menor de L, ya que podría disminuir la seguridad de la función H. Un tamaño muy grande de clave no añade más seguridad al sistema.

Las claves usadas, a su vez, tienen que ser elegidas al azar, de forma pseudoaleatoria, con algún tipo de generador, y tienen que ser cambiadas cada cierto tiempo.

Toda la seguridad del método recae sobre las propiedades de la función hash H (tamaño de bloques y output), por lo que conviene elegir una que no sea sensible a las colisiones.

4.3.2 HOTP

También se ha estudiado el funcionamiento de HOTP (HMAC-based One-time Password Algorithm), sacado del RFC 4226 del año 2005, algo más reciente.

⁵ Imagen de la fórmula de HMAC.

HOTP es un algoritmo para generar contraseñas de un solo uso, basadas en el método de HMAC. Es uno de los precursores de los métodos actuales de autenticación de dos pasos.

Este algoritmo presenta los siguientes requisitos en su RFC:

- Debe estar basado en un contador creciente o una secuencia numérica común entre el sistema llave T y el sistema de caja fuerte S, o que sea fácil de sincronizar en cada uso.
- El algoritmo debe ser económico a la hora de disponer del hardware necesario, como el uso de la batería y su consumo, el coste de su funcionamiento...
- Debe de poder trabajar sin ningún tipo de teclado físico, pero en dispositivos más avanzados puede usar teclados de tipo PIN (como nuestro caso).
- Si se hace display de la contraseña, debe de ser sencilla de leer por un usuario. Esto es, debe de tener un tamaño adecuado, y mayor a 6 caracteres, por ejemplo, 8 caracteres.
- Debe incluir mecanismos sencillos en caso de querer resincronizar el contador.
- El secreto debe de ser de un tamaño **MÍNIMO** de al menos 128 bits, siendo el **RECOMENDADO** de 160 bits

Los símbolos que utiliza son:

- **C**, contador de 8 bytes. Debe estar sincronizado entre cliente y servidor para operar de la misma manera.
- **K**, secreto compartido entre cliente y servidor, para calcular el HMAC sobre el que calcular HOTP.
- **T**, número de conexiones fallidas tras el cual el servidor rechaza las conexiones del cliente.
- **s**, parámetro de resincronización con el que el servidor intentará verificar una conexión recibida. Recalcula los s valores de contador seguidos hasta encontrar una clave que coincida con la salida HOTP.

El algoritmo funciona gracias a una clave secreta sólo conocida por el servicio de validación y el usuario, y un contador que se incrementa o cambia en cada uso.

Por defecto se usa SHA-1 en la implementación estándar del RFC, pero dado a que ha pasado cierto tiempo desde el nacimiento de HOTP, se deberían usar funciones más recientes o que sigan sin tener vulnerabilidades, al menos frente a ataques por colisión.

Si se usa SHA-1, el output es de 160 bits, ese valor se trunca entonces para obtener un valor que sea fácilmente introducible por el usuario, de la forma:

$$\text{HOTP}(K,C) = \text{Truncar}(\text{HMAC-SHA-1}(K,C))$$

Se trunca de la siguiente forma: Poniendo de ejemplo la salida de SHA-1 de 160 bits (20 Bytes), cogemos los 4 últimos bits del último Byte (El Byte 19, si se empieza a contar desde el Byte 0).

Con éstos últimos bits, los pasamos a decimal. Teniendo 2^4 números posibles para elegir desde un subgrupo (Del 0 al 15). Teniendo el grupo que nos haya salido, cogemos los 3 siguientes grupos de Bytes, y tenemos un número de 4 Bytes (Por ejemplo, 0x50ef7f19).

Ahora este número se puede pasar a decimal, se hace la operación de módulo igual a la longitud que queremos que introduzca el usuario y que calcule el sistema (Para 8 números de contraseña, $0x50ef7f19$ (pasado a decimal) $\% 10^8$).

Byte Number
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19
Byte Value
1f 86 98 69 0e 02 ca 16 61 85 50 ef 7f 19 da 8e 94 5b 55 5a

- We then take this number modulo 1,000,000 (10^6) to generate the 6-digit HOTP value 872921 decimal.

Si el valor calculado por el usuario no es igual al del sistema, puede haber un error en la sincronización entre los dos. Esto se debe a que el sistema sólo aumenta el contador cuando hay un acierto al calcular la contraseña HOTP,

13

mientras que el usuario incrementa su propio contador cada vez que lo pide. Esto puede dar lugar a una diferencia entre ambos valores.

Así, el sistema puede calcular los siguientes valores del contador y probar los siguientes valores de esta ventana. Si tras estos valores el valor sigue sin ser igual, se puede bloquear fuera del sistema al usuario y requerir medidas adicionales para probar su identidad.

También existen medidas para evitar ataques por fuerza bruta, como una espera de 5 segundos cada vez que se falla. Al primer fallo, 5 segundos. Al segundo, 10, luego 15, 20, etc.

4.4 Función Hash SHA-512

SHA-512 se encuentra dentro del conjunto de funciones criptográficas SHA-2. Como todas las funciones hash, crea una salida de longitud fija a partir de un conjunto de datos variable, como un mensaje o un documento de texto. Esto es posible debido a que se parte siempre de un estado interno definido por varios registros, y según se van añadiendo bits del mensaje, el resultado va cambiando.

El valor hash de, pongamos, un documento, puede ser usado para calcular si el documento ha sido alterado, al recalcular el valor del documento con el valor que se guardaba anteriormente (Este Hash también podría haberse cambiado, por lo que es útil calcular el HMAC de un mensaje, no sólo el Hash normal).

Este valor de salida es la “identidad” del archivo o mensaje, y es conveniente que sea altamente improbable que dos archivos totalmente diferentes generen una misma salida hash. Si esto ocurre, nos encontramos ante una colisión.

Además de ser un error, puede ser aprovechado por atacantes para realizar cambios en ficheros y, sabiendo las debilidades de la función hash, ocultar estos cambios bajo el mismo hash.

Las colisiones en cualquier función hash son un caso muy importante de estudio. Nos referimos con el término “resistencia a colisiones” a la dificultad que ofrece la función hash elegida para encontrar dos entradas (dos textos, dos archivos...) que generen una misma salida; esto es, teniendo A y B, y siendo A distinto a B, encontrar $H(A)$ igual a $H(B)$. Cuanto más difícil sea esto, más seguridad criptográfica se tiene en la función hash.

Finalmente, SHA-512 produce salidas de 512 bits, y opera en bloques de tamaño 1024 bits, valores que se deben tomar para implementar el protocolo HOTP comentado anteriormente.

4.5 Acceso al sistema

Los pasos para pedir acceso al sistema son los siguientes:

- El usuario se conecta al sistema desde su dispositivo eléctrico llave (más tarde se comentará el diseño) con el ID de su dispositivo.
- Se comprueba que el ID del usuario en cuestión tiene los permisos necesarios para acceder al sistema. Paso seguido, usuario y sistema calculan Diffie-Hellman: Cada uno toma los valores descritos antes como públicos, calcula su número privado (escoge un valor al azar en el rango $[1, p-1]$, siendo p el primo con el que se opera), su número público (de la

forma $A = g^a \bmod p$ o $B = g^b \bmod p$), comparte este último y, si todo ha ido de manera correcta, usuario y sistema llegan a un resultado común.

- El sistema genera un desafío **R**, un mensaje al azar formado por 8 Bytes aleatorios, garantizando que los valores tomados son al azar y están distribuidos de manera uniforme.
- El sistema espera como respuesta por parte del usuario a su desafío el resultado de calcular HOTP sobre el mensaje de 8 Bytes al azar con el resultado de Diffie-Hellman como clave. Cuando ambos coincidan, el dispositivo del usuario presentará la combinación del sistema de seguridad, que será el HOTP resultante tomando tantos dígitos de él como se deseen, entre 6 y 8 de forma habitual.
- Después del paso anterior, se sobrescribe la combinación de la caja por una nueva aleatoria, y también se cambia la clave asociada al usuario por otra al azar, y se pasa al dispositivo del usuario para que se actualice en el siguiente uso. Estas claves no deberían nunca de reutilizarse, pero sí que se pueden almacenar para a) comprobar que no se generan de nuevo (aunque puede escalar de manera negativa tras muchos usos) y b) añadir ruido a las claves que se generen para que no sea determinista para un atacante.
- Finalmente, la caja/puerta del sistema de seguridad puede ser abierta tras meter la combinación calculada en pasos anteriores.

Toda esta comunicación se deberá hacer sobre un medio físico seguro, que también sea capaz de proporcionar la energía necesaria para que el sistema funcione, pues su estado natural es esperando a una fuente de energía para iniciar la comunicación, y así no poder estar abierto a ataques que corten el suministro de energía.

El cálculo de HOTP como desafío significa también saber que ni el mensaje ni el hash del desafío ha sido modificado, una medida de seguridad adicional.

4.6 Seguridad asociada a la criptografía

Como se ha dicho antes, tres algoritmos principales rigen el comportamiento de todo el sistema: Diffie-Hellman, HMAC y HOTP. Estudiemos cada uno de ellos en detalle.

4.6.1 Diffie-Hellman

Descrito anteriormente, se encarga de establecer la conexión a partir de un número primo, un generador, y un número privado para cada miembro del par de la conexión.

Para que haya seguridad suficiente, en el RFC 3526 [14] habla de cómo generar los números primos públicos, y una de las mejores opciones computacionalmente hablando es partir de uno de los grupos predefinidos de primos que se nos ofrecen.

A mayor cantidad de bits del primo, mayor será la clave que se cree entre ambos al final. El mínimo recomendable en conexiones Internet es de 2048 bits, ya que se estima que romper el un sistema con un primo de 2048 bits es 10^9 veces más difícil que para un primo de 1024 bits.

Todo este algoritmo se basa en el *problema del logaritmo discreto*. Como se vio antes, partiendo de números iniciales privados y compartiendo cada parte pública, se concluye que se llega a $K = g^{(a*b)} \bmod p$ aunque no se conozca la clave privada de la otra parte, pues se conoce el valor público *compartido* $A = g^a \bmod p$ o $B = g^b \bmod p$.

Por las propiedades de la aritmética modular, en concreto de la propiedad de clases de equivalencia módulo n, ambas partes llegarán a la misma clave final K, ya que bajo un módulo p (primo) común:

$$A^b \bmod p = g^{(a*b)} \bmod p = g^{(b*a)} \bmod p = B^a \bmod p$$

Partiendo de los parámetros adecuados, teniendo un primo suficientemente grande, y un generador que también es un primo, revertir la operación de módulo es un proceso extremadamente complicado. Además, g podría tener el valor de cualquier primo que se desee, y sería igual de seguro con el valor de 2, o de cualquier raíz primitiva módulo p (el primo). Esto, comentado anteriormente, se debe al teorema de *random self-reductibility*, donde un algoritmo es igual de seguro para todos los valores que se presenten, en nuestro caso, g.

Esta parte se podría mejorar usando una Curva Elíptica [15], método que se tratará más adelante en detalle. Es igual de segura, más elegante, y sólo necesita la función de la curva y un solo primo con el que calcular el resultado de la operación.

Una de las vulnerabilidades más conocidas de Diffie-Hellman y que merece la pena mencionar es la basada en un ataque *man in the middle*, u hombre en el medio. El atacante se situaría entre las partes que se intentan comunicar y acuerda una clave simétrica con cada parte pasándose por la contraria. A partir de entonces sería posible comunicarse con cada parte de forma individual, descifrar las comunicaciones, y comunicarse con la otra parte.

Una forma sencilla de remediar esta vulnerabilidad es mediante una firma digital o un certificado emitido por una entidad de confianza que pruebe la identidad de cada parte. Esto es muy sencillo de hacer en una conexión en Internet, y suele estar incluido en el Diffie-Hellman que se use.

Todo lo comentado anteriormente depende de que se elijan los parámetros correctos para evitar fallos, siguiendo los pasos de los RFC que tratan Diffie-Hellman.

4.6.2 HMAC

Toda la seguridad del HMAC recae sobre la función Hash que se use. Lo único que se pide es que la clave usada para calcular el Hash no sea menor que la salida u output de la propia función.

Antes de meternos en más detalles, es necesario hablar en más detalle de MAC y HMAC.

Como se comentó antes, un MAC garantiza que el mensaje que ha enviado no ha sido alterado, por ejemplo, se puede mandar un mensaje y mandar con él el MAC que puede ser el hash del mensaje, como una especie de “checksum”, usando un simil de comunicación en redes.

Pero hay que tener en cuenta lo siguiente: ¿Qué ocurre si un adversario captura el mensaje, altera el mensaje, y recalcula el hash, y lo manda de nuevo? Sin tener en cuenta temas de cifrado, sólo mensajes en claro. Esto es un problema, el usuario recibe un mensaje totalmente distinto, y cuando recalcula el hash, es igual, y no sabría que ha sido alterado.

HMAC es la solución a estos problemas. Como se comentó antes, se hacen dos hash, uno sobre el mensaje con una clave **k1 con ipad**, y otro sobre lo anterior con la clave **k2 con opad**. Con esto, es imposible determinar el estado interno intermedio que existía tras la primera operación, y evitar ataques relacionados con él; a diferencia de un hash normal HMAC no es vulnerable a ataques por extensión.

Usando SHA-512, tenemos una salida de 512 bits, 64 Bytes. Usando la salida de Diffie-Hellman anterior, podemos estar seguros de que siempre será mayor a 2048 bits, que son 256 Bytes. Además, hasta la fecha no se ha encontrado ninguna forma de generar colisiones de manera determinada para SHA-512.

También hay que añadir que normalmente los HMAC a partir de SHA's (1,2 y 3) o MD5 son más resistentes que las funciones Hash singulares que los forman.

Según el RFC seguido, el único ataque conocido es el llamado “birthday attack”, para encontrar colisiones en la función hash. Con una construcción correcta y una función hash segura, esto no debería ser ningún problema.

HMAC es parte de HOTP, que se estudiará a continuación.

4.6.3 HOTP

En primer lugar, el RFC 4226 que habla de HOTP sugiere un tamaño mínimo de secreto compartido de 128 bits, siendo 160 bits el recomendado. Este secreto en nuestro caso es la clave generada por Diffie-Hellman, por lo que se cumple.

También cuenta con una interfaz sencilla de usar para un usuario, y presenta por pantalla una combinación sencilla de introducir y leer, de 8 dígitos en nuestro caso.

El algoritmo sigue lo indicado por el RFC 4226 de HOTP, excepto que en vez de disponer de una salida de 160 bits con SHA-1, tenemos 512 bits con SHA-512, por lo que se han tenido que hacer una serie de cambios en el algoritmo, pero la salida sigue dando valores aleatorios uniformemente distribuidos.

El ataque con más probabilidad de victoria es el de fuerza bruta, y a pesar de haber estudiado los mensajes intercambiados en la comunicación, la construcción de una nueva función F por parte del atacante que genere los valores de la contraseña a partir de estos mensajes, no será mejor que la probabilidad de acertar con un número al azar.

Si se intenta resolver por fuerza bruta, las posibilidades de acertar son:

$$Sec = \frac{sv}{10^{Digit}}$$

donde:

- **Sec** es la probabilidad de que el adversario consiga acceso.
- **s**, el tamaño de la ventana de look-ahead de sincronización (Cuántas veces se incrementa el contador de la parte del servidor en caso de no estar sincronizados).
- **v**, el número de intentos permitidos.
- **Digit**, el número de dígitos que queremos que tenga la contraseña.

Tanto Diffie-Hellman, HMAC y HOTP son algoritmos ampliamente usados hoy en día, pese a tener casi entre 15 y 20 años cada uno.

Como es costumbre, los métodos que con el tiempo se mantienen seguros y sólo requieren cambios en los tamaños de algunas funciones con el paso del tiempo son la mejor opción a tener en cuenta en un proyecto como éste.

5 Componentes del Sistema

Se pretenden emplear, de manera teórica, dos microcontroladores, del tipo Arduino y del mismo modelo: uno que actúe como dispositivo de tipo llave, otro que actúe como sistema central de la caja fuerte. Cada uno estará en el dispositivo que corresponda.

Para la construcción, se han seguido las especificaciones de la página oficial de Arduino (varios modelos comparten las mismas especificaciones excepto por algunas diferencias en sus pines y su tamaño) [16].

El voltaje recomendado para cada Arduino es “de entre 7 a 12 voltios”. La salida de los pines digitales funcionará a 5 voltios.

Estudiado esto, una batería de 9V debería ser suficiente para alimentar un sólo Arduino. En el sistema real, convendría además que fuera recargable y pudiera soportar cientos de recargas antes de dejar de funcionar.

Para conectar la batería, hay que conectar "tierra" de la batería al PIN de "GND" de la llave, y la otra parte al PIN de "Vin" ("Voltage in").

La memoria “tiene 32 KB (con 0.5 de éstos dedicados al bootloader)”, suficientes para almacenar los fragmentos de código de la llave y el sistema de la caja fuerte, y realizar las operaciones necesarias.

Para un diseño inicial, y puesto que la comunicación de forma inalámbrica no es suficientemente segura, se establecerá una conexión entre Arduinos directamente por sus puertos [17].

Este primer prototipo sobre papel es una prueba de concepto, para probar que cumple las funciones básicas de criptografía y es capaz de activar un actuador lineal cuando las claves coinciden.

5.1 Sistema Llave

Por parte del sistema llave o token se necesitarán los siguientes componentes:

- 1 x Microcontrolador (Arduino Uno, Nano, u otro)... Con el que realizar las operaciones
- 1 x Módulo botón para encender o apagar la llave cuando se necesite .
- 2 x Baterías li-ion recargable. Lo más apropiado es juntar 2 pilas de 9V, una para cada Arduino. Estas baterías se encuentran dentro de la llave. La llave se puede apagar y encender cuando se quiera, pero la segunda batería sólo da energía al cerrar el circuito con la caja fuerte.
- 1 x Pantalla LCD donde visualizar la información de interés para el usuario, como la contraseña que hay que introducir, los mensajes de ayuda que manda el sistema de la caja fuerte...
- (Opcional) 1 x Carcasa donde introducir el modelo de Arduino y sus componentes, con una apertura para quitar e introducir la batería y cargarla.

La función de esta llave es la de, mientras se encuentre encendida, buscar si encuentra encendido el Arduino del sistema de la caja fuerte.

No encontrará nada hasta que la propia llave proporcione parte de la energía en la pila de 9V que no se está usando.

Al proporcionar la energía necesaria por parte de la batería a la caja fuerte, se forma un circuito desde la segunda batería de 9 V hasta el otro módulo Arduino de dicha caja. El otro Arduino reacciona, se enciende y realiza sus operaciones hasta que activa el actuador lineal que bloquea la caja fuerte. Cada Arduino usa así 9V de cada batería.

5.2 Sistema Caja Fuerte

Para el Arduino de la caja fuerte, se necesitaría:

- 1 x Microcontrolador (Arduino Uno, Nano, u otro)... Igual que el de la llave.
- 1 x Unidad de almacenamiento extra (tarjeta SD), como espacio adicional permanente donde guardar información sobre claves, mensajes enviados...
- 1 x Actuador lineal con el que bloquear la compuerta de la caja fuerte [18].
- 1 x Teclado numérico a través del cual introducir la contraseña generada por HOTP de tantos dígitos como se desee.

En este caso, el sistema permanece apagado hasta que se le proporcione energía. En cuanto se enciende, intenta comunicarse con el Arduino de la llave.

Si lo detecta, puede comenzar todo el protocolo para el intercambio de mensajes según HOTP y la generación de la contraseña en común entre la llave y la caja fuerte.

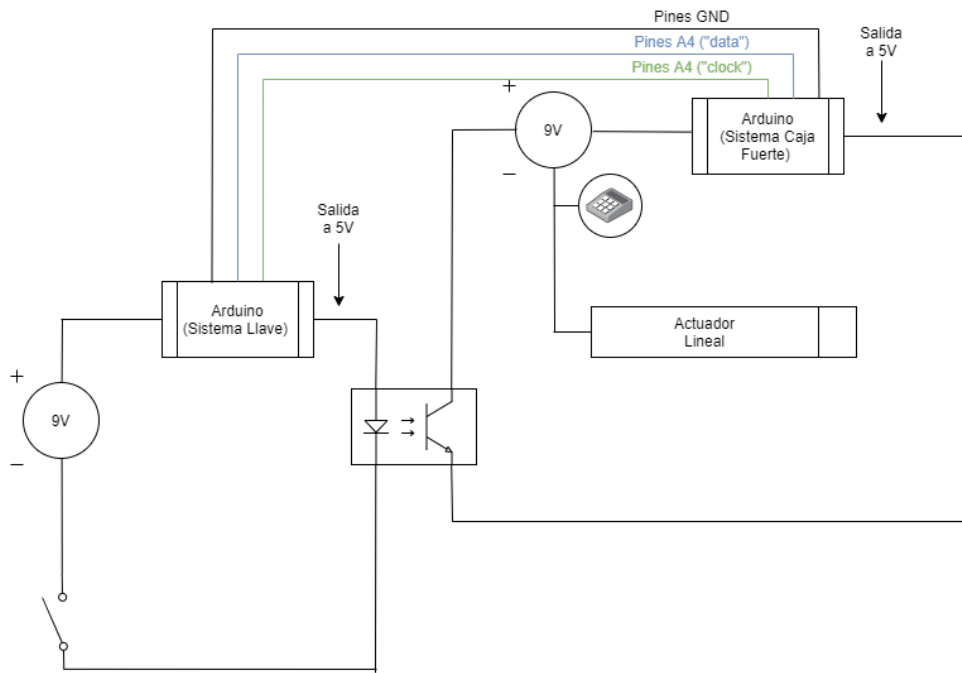
5.3 Circuitos

En primer lugar, hay que entender los fundamentos eléctricos sobre los que opera Arduino. El voltaje recomendado se encuentra entre 7 y 12 voltios. Ésta es la entrada, en nuestro caso los voltios de la batería hacia el Arduino.

Este voltaje pasa por un regulador del propio Arduino antes de llegar al chip interno, que lo regula a 5 voltios, y es el voltaje con el operan los pines digitales (del sistema llave) a los que los componentes estarán conectados.

Adicionalmente, para el circuito total, es necesario incorporar un *optoacoplador*, un componente que consigue comunicar dos circuitos sin contacto entre ellos.

Cuando por el circuito 1 de la llave pasa corriente, se activa un led dentro de este componente. El fotorreceptor del circuito 2 de la caja fuerte lo detecta, se activa y deja pasar la corriente por ese circuito. Un diseño simplificado sería:



7

Todo esto teniendo en cuenta que ambas baterías se encuentran en el dispositivo llave, y la entrada del Arduino de la caja fuerte se encuentra normalmente abierta hasta que no se conecta a la batería.

Como se comentó antes, éste es un diseño inicial que sólo prueba que la conexión y el envío de mensajes es posible, pero en caso de hacerse con fines comerciales en una caja fuerte real, habría que revisar este primer plano.

5.3.1 Seguridad en el circuito

Se recomienda incluir fusibles en cada circuito, tras cada batería. Así, si hay algún valor de las baterías que puedan causar una sobrecarga, la seguridad física de la caja fuerte seguiría en marcha, a pesar de que la caja estaría a partir de ese momento “rota”.

En caso de error interno en la caja fuerte, no quedaría otra opción que la de abrirla por fuerza bruta con herramientas para cambiar las piezas afectadas. Por tanto, queda descartada totalmente la inclusión de un mecanismo secundario de tipo mecánico en caso de fallo del sistema principal. Todo se realizará sobre el mecanismo electrónico, y en caso de un ataque para fundir el fusible, no se hace nada para evitar proporcionar ninguna ayuda al atacante.

Incluir un optoacoplador también ayuda a que, en un principio, ningún otro sistema aparte de la llave se puede conectar e iniciar la conexión, huyendo así de posibles ataques exteriores si no se dispone del mismo modelo de llave con los mismos componentes.

⁷ Ejemplo del circuito que forman los dos arduinos.

6 Implementación en código

Para la implementación inicial, el código principal se ha dividido en 2 archivos de python, *diffieHellman.py* y *hash.py*, y dos archivos de texto, *claves.txt* y *mensajes.txt*, con claves finales y mensajes que no se pueden volver a generar.

Todos los archivos y documentación del trabajo se pueden encontrar en el repositorio de Github del alumno [19].

Además, más tarde se creó un tercer archivo de administración llamado *administrador.py*, que se encarga de añadir permisos a usuarios, borrar permisos, y limpiar los archivos de mensajes y claves, y otro llamado *usuarios.txt* con los ID's de usuarios con permisos para operar sobre la caja.

6.1 Clase *diffieHellman.py*

En primer lugar, *diffieHellman.py* sigue los RFC 2631 (algoritmo básico para llegar a una misma clave a partir de un número primo de gran longitud) y 3526 (establece una serie de número primos por grupos, desde 1536 bits a 8192 bits).

Se da la opción de elegir cualquiera de los grupos, desde 15 a 18; cuanto mayor el grupo, mayor el primo, pero mayor el tiempo que tarda en calcular la clave común final. Nos interesan valores superiores a 4000 bits, como nos habla el RFC, por lo que se deberían usar siempre los grupos 16 (4096 bits), 17 (6144 bits) o 18 (8192 bits). El grupo 15 (3072 bits) sirve para hacer pruebas con rapidez.

La clase *diffieHellman.py* también es capaz de generar una clave privada de la forma *random.randint(1, rangoMax - 1)*, esto es, elegir un número aleatorio que va desde 1 al rango máximo, que es el primo elegido menos uno (para no elegir el propio primo de nuevo). Normalmente se usa una semilla o “seed” bien definida para hacer estos cálculos. Además, después del uso de cada clave privada, se pone en un archivo *claves.txt* que se comprueba antes de generar nuevas claves, para no tener que reusar una misma clave más de una vez.

Para generar claves públicas, se hace de la forma *pow(2, valorPrivado, primo)*, esto es, $2^{\text{valorPrivado}} \% \text{primo}$. Éste es el resultado que será compartido por cada miembro con el contrario. Al depender de la clave privada para su generación, mientras que no se reutilice una clave privada, nunca se creará una clave pública repetida.

Siendo *A* y *B* públicos que comparte el propio usuario A y usuario B, y *a* y *b* los valores privados que cada parte se queda para sí misma. *P* es el primo elegido al principio, y *g* suele ser un primo como 2, que es tan seguro como cualquier otro primo superior que cumpla las propiedades.

La seguridad de esta parte del sistema depende de que las claves privadas no se distribuyan con terceros. Si alguna se obtiene, pondría en peligro la integridad de las comunicaciones anteriores o posteriores. Si se pierde alguna clave privada también sería imposible realizar más comunicaciones.

Al salir de esta clase, tenemos una clave en común del mismo tamaño que el primo que se ha usado.

6.1.1 Funciones del archivo

elegirGrupo(self, grupo): A partir de un string con el grupo, devuelve el primo asociado. Ninguno de los primos entra en pantalla por ocupar miles de caracteres. En caso de no ser una key del diccionario de primos, devuelve la de 15 por defecto.

```
import random
import os
import csv

class diffieHellman:

    def elegirGrupo(self, grupo):
        """Devuelve el primo que se haya elegido dependiendo del grupo
        La longitud en bits de cada primo es:
        Group 15 (3072 bit)
        Group 16 (4096 bit)
        Group 17 (6144 bit)
        Group 18 (8192 bit)
        """
        # No se puede aplicar el límite de 80 caracteres por línea en los primos
        Primos = {
            [ESTOS VALORES TIENEN MILES DE BITS, Y NO ENTRAN EN
            EL DOCUMENTO. CONSULTAR https://tools.ietf.org/html/rfc3526]
            "15" : [...],
            "16" : [...],
            "17" : [...],
            "18" : [...]
        }

        # Si el grupo existe, devuelve el primo asociado
        if grupo in primos:
            return primos[grupo]

        # Si no existe, devuelve el primo asociado al grupo 15 por defecto
        else:
            print("El grupo introducido no existe, usando el primo por defecto del grupo 15")
            return primos["15"]
```

generarClavePrivada(self, rangoMax): A partir de un rango máximo, que es el primo elegido en el paso anterior, elige una clave aleatoria privada en el rango $[1, \text{rangoMax} - 1]$ (para la caja o para el usuario).

Se comienza con una seed de 30 para tener algo de determinismo; las claves generadas se insertan al final del archivo de claves. Si la clave está en el archivo, se vuelve a generar hasta que no esté.

```
def generarClavePrivada(self, rangoMax):  
    # Semilla estática para obtener resultados iguales tras  
    # varios usos,  
    # conviene quitarla cuando se use de verdad  
    random.seed(30)  
    if not os.path.exists('claves.txt'):  
        os.mknod('claves.txt')  
    """Devuelve la clave privada de cualquiera de las dos partes,  
    un numero secreto desde 1 al primo elegido - 1, el rango máximo"""  
    claveLocal = random.randint(1, rangoMax - 1)  
    # Abrir el fichero de claves  
    archivo = open('claves.txt', 'r')  
    lineas = archivo.readlines()  
    for linea in lineas:  
        if claveLocal == int(linea):  
            claveLocal = random.randint(1, rangoMax - 1)  
  
    f=open("claves.txt", "a+")  
    f.write(str(claveLocal))  
    f.write("\n")  
    f.close()  
    return claveLocal
```

generarClavePublica(self, valorPrivado, primo): Genera la clave pública para el intercambio de claves con su contraparte, como explica el docstring.

```
def generarClavePublica(self, valorPrivado, primo):  
    """Genera la clave pública a intercambiar, de la forma:  
    ya = g ^ xa mod p / yb = g ^ xb mod p,  
    yx es el resultado que se intercambia con la otra parte,  
    g el generador, por defecto 2,  
    xx la clave privada,  
    p el primo usado"""  
    return pow(2, valorPrivado, primo)
```

conexionCorrecta(self): Devuelve si los valores finales de la clave común K es igual para cada parte.

```
def conexionCorrecta(self):
    """Devuelve si la conexión es correcta; si los valores finales son iguales"""
    if self.aFinal == self.bFinal:
        print("Los valores son iguales, comparten la misma clave","\n")
        return True
    # Si es distinto, la comunicación se acaba
    else:
        print("Los valores difieren, error","\n")
        return False
```

presentarResultados(self): Debug que da la opción de imprimir por pantalla los valores establecidos en la conexión inicial.

```
def presentarResultados(self):
    """Print por pantalla para debug de todos los valores:
    valor privado de a y b,
    valores públicos de cada parte,
    valor final, que debería ser el mismo para ambos.
    Devuelve si a y b generan la misma clave con la que trabajar"""
    print("Valor privado de a: ",self.a,"\n")
    print("Valor privado de b: ",self.b,"\n")
    print("Valor público de a: ",self.A,"\n")
    print("Valor público de b: ",self.B,"\n")
    print("Valor de la clave para a: ",self.aFinal,"\n")
    print("Valor de la clave para b: ",self.bFinal,"\n")
    # Si es el mismo valor, acierto
```

__init__(self, grupo): Constructor de la clase. A partir del grupo, genera los valores privados, públicos y final

```
def __init__(self, grupo):
    """Objeto diffieHellman, que consta de
    primo: numero primo elegido con el que operar al inicio de la conexión
    a: clave privada de a
    b: clave privada de b
    A: clave publica de a
    B: clave publica de b
    aFinal: clave final de a
    bFinal: clave final de b
    """
    self.primo = self.elegirGrupo(grupo)
```



```
self.a = self.generarClavePrivada(self.primo)
self.b = self.generarClavePrivada(self.primo)
self.A = self.generarClavePublica(self.a, self.primo)
self.B = self.generarClavePublica(self.b, self.primo)
self.aFinal = pow(self.B, self.a, self.primo)
self.bFinal = pow(self.A, self.b, self.primo)
```

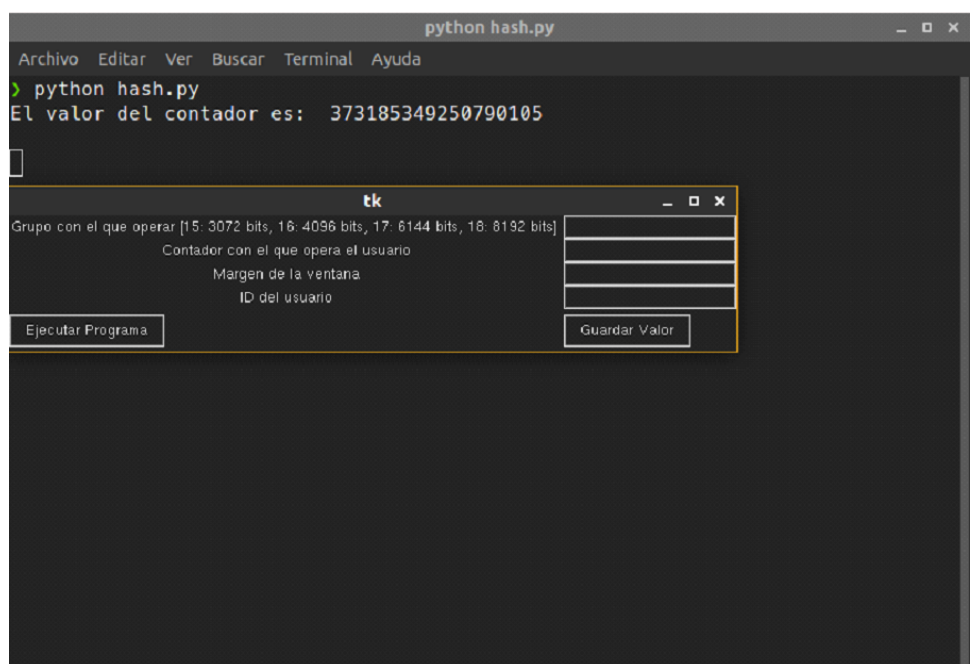
6.2 Clase hash.py

En esta clase se calcula con HMAC y HOTP el valor de la clave a introducir en cada uso. Seguimos los RFC oficiales para ambos.

A pesar de tener al usuario y a la caja en local en este modelo inicial, se usa una semilla predefinida para obtener siempre los mismos valores en cada prueba. Estos mensajes, como en el caso de *diffieHellman.py*, nunca se repetirán, pues se incluyen en un archivo *mensajes.txt* que comprueba que el mensaje actual no es repetido, y si lo es, genera otro nuevo hasta que no esté repetido.

Ésta semilla sirve para todos los valores pseudoaleatorios que se usarán, como el contador. Es un número de 8 Bytes que se genera al azar, uniformemente distribuido, que usaremos como “mensaje” sobre el cual calcular el HMAC con la clave común de Diffie-Hellman.

Se pide al usuario que elija uno de los grupos de primos disponibles, preferiblemente con un output de más de 4000 bits, a partir del cual se calculan todos los parámetros de Diffie-Hellman. También se pide que elija un contador con el que opere el usuario, y una ventana en caso de que los contadores de cada uno no coincidan.



(En el mundo real, el contador del usuario es un estado interno propio, no algo que se elija, pero para realizar pruebas en un modelo inicial es útil. Con la ventana ocurre lo mismo, sería un valor fijo, como 6 o 10, por ejemplo).

A partir de la clave común, y con el contador como mensaje, calculamos el HMAC con SHA-512. El tamaño del HMAC será de 64 Bytes (512 bits), y se calcula de la forma `hmac.new(clave, mensaje, hashlib.sha512)`.

A partir de ahora será cuando calculemos HOTP.

Como la salida es de 64 Bytes, no será posible hacer lo mismo que con la salida de SHA-1, que era de 20 Bytes, así que es necesario realizar una adaptación.

Queremos una salida de 4 Bytes aleatorios para calcular la clave que presentar al usuario, y que la clave final sea sencilla de leer y escribir también. Para esto, como se explicó anteriormente, cogemos el Byte final del HMAC, y lo pasamos a binario. Esto nos dejaría con 8 bits frente a los 4 que utiliza SHA-1, sobre los que se podrían elegir hasta 255 grupos (11111111 máximo), pero nos pasamos.

Con 4 bits podemos elegir hasta 15 (1111 máximo) grupos, no llegamos. Tampoco con 5 bits y 31 (11111 máximo) grupos, necesitamos 6 bits y 60 grupos, por lo que nos sobran grupos (111111 máximo), porque no podemos coger desde el 61 al 63.

Al ser un valor aleatorio, podemos obtener un valor de 0 (cogemos desde el grupo 0 al 3), 1 (grupo 1 a 4), 2 (2 a 5)... El último valor válido es 60, que nos da desde el grupo 60 al 63, el último. Si se obtiene alguno de los valores sobrantes, se calcula el módulo de 60 para obtener un grupo válido, hasta 60.

Se elige el grupo del Byte, y los tres siguientes hasta obtener 4 Bytes, se pasan a decimal, y a partir de estos números, cogemos 8 de ellos, por ejemplo, presentamos la contraseña que es igual para usuario y sistema.

⁸ Ejemplo de uso del script hash.py

Adicionalmente, si el contador del usuario no es igual al contador del sistema (caso común, el contador del usuario incrementa tras cada uso correcto, el del usuario siempre que se quiera conectar, por lo que pueden estar no sincronizados), se calcula HOTP del rango introducido en la ventana. Por ejemplo, si el sistema tiene el contador $i=3$, el usuario el contador local $j=5$, y la ventana es 10, calcula HOTP con los 10 últimos valores generados de contador, acertando cuando el contador de la caja tiene el valor 5.

6.2.1 Funciones del archivo

establecerConexion(grupo): A partir de una cadena con el grupo que se quiere usar, devuelve un objeto de la clase `diffieHellman` con el que seguir la conexión.

```
# Script principal que calcula el hash-hmac de la clave diffie hellman
# y luego genera la contraseña común con HOTP

import hashlib
import random
import hmac
import sys
import os
from tkinter import *
from diffieHellman import diffieHellman

def establecerConexion(grupo):
    """A partir del grupo del primo, devuelve un objeto
    DiffieHellman con el que comunicar entre ambos."""
    return diffieHellman(grupo)
```

ImprimirPantallaGuardar(): Función auxiliar para obtener valores introducidos en la interfaz que usa el usuario.

```
def imprimirPantallaGuardar():
    """Imprime por pantalla el número que se haya introducido por
    Tkinter y lo guarda en una variable para usarlo en el futuro"""

    print("El grupo introducido ha sido el %s" % (n.get()))
    print("El contador del usuario es %s" % (n1.get()))
    print("El margen con el que se opera es %s" % (n2.get()))
    return n.get()
```

GenerarInput(): Función que presenta por pantalla la interfaz al usuario, con opciones de:

- Grupo con el que operar.
- Contador con el que opera el usuario.
- Ventana en caso de que el contador no sea igual para ambos.
- ID del usuario.

Que tiene la apariencia presentada en la última imagen, con el shell. El código es el siguiente:

```
def generarInput():
    """Genera una ventana por tkinter para que el usuario
    introduzca el grupo con el que operar.
    Posee botones para guardar y ejecutar el programa"""

    ventana = Tk()
    Label(ventana, text='Grupo con el que operar [15: 3072 bits, 16: 4096 bits, 17:
6144 bits, 18: 8192 bits] ').grid(row=0)
    Label(ventana, text='Contador con el que opera el usuario').grid(row=1)
    Label(ventana, text='Margen de la ventana').grid(row=2)
    Label(ventana, text='ID del usuario').grid(row=3)

    global n
    global n1
    global n2
    global n3

    n = Entry(ventana)
    n1 = Entry(ventana)
    n2 = Entry(ventana)
    n3 = Entry(ventana)

    n.grid(row=0, column=1)
    n1.grid(row=1, column=1)
    n2.grid(row=2, column=1)
    n3.grid(row=3, column=1)

    Button(ventana, text='Ejecutar Programa', command=ventana.quit).grid(row=4,
column=0, sticky=W, pady=4)
    Button(ventana, text='Guardar Valor',
command=imprimirPantallaGuardar).grid(row=4, column=1, sticky=W, pady=4)
    mainloop()
    return n.get()
```

hmac_sha512(clave, mensaje): Función que genera el HMAC SHA-512 a partir de la clave en común (que varía en tamaño dependiendo del primo máximo

usado) y el mensaje o reto generado al azar. Utiliza la librería de Python hashlib para el HMAC.

```
def hmac_sha512(clave, mensaje):
    """Devuelve el hmac con sha512 a partir de:
    clave: clave de la comunicación entre DH,
    mensaje: contador de 8 Bytes generado al azar,
    devuelve el resumen hmac"""

    # Se convierten la clave y el mensaje, que eran enteros,
    # a un conjunto de bytes con el que operar
    clave = bytes(str(clave), "UTF-8")
    mensaje = bytes(str(mensaje), "UTF-8")

    # Se devuelve el resultado del hmac
    digester = hmac.new(clave, mensaje, hashlib.sha512)

    print("Tamaño del HMAC resultante: ", digester.digest_size, " Bytes\n")
    return digester.hexdigest()
```

EliminarPrefijo(stringNumero): Función auxiliar que se utiliza más tarde para quitar el prefijo “0b” de los números de tipo Byte.

```
def eliminarPrefijo(stringNumero):
    """Quita el prefijo 0b de los Bytes que se introducen"""
    prefijo = "0b"
    if stringNumero.startswith(prefijo):
        return stringNumero[len(prefijo):]
```

calcularHOTP(contador, grupo, diffie): A partir de un contador, un grupo, y la conexión Diffie-Hellman, devuelve el valor HOTP para quien lo pida, usuario o caja fuerte.

Expliquémoslo por partes. En primer lugar, comprueba que la conexión es correcta, que los valores finales de a y de b son el mismo. Si no lo son, error, si lo son, acierto.

A continuación, se calcula el HMAC del mensaje generado al azar (en la función main del programa, luego se verá) con la clave usada siendo la clave generada por Diffie-Hellman.

Posteriormente, se coge el último Byte del conjunto, representado por dos caracteres. Estos dos caracteres, que forman el susodicho Byte final, se pasan a binario por separado y se juntan. Este valor es en bits del conjunto final, por lo que, si es impar, sólo contará la segunda parte del Byte; se pasa a un valor par y se representa el valor del grupo de Bytes.

Si el grupo obtenido es más de 60, se hace pues el módulo para obtener un grupo de 4 Bytes válido, y se cogen también los 3 siguientes.

Éstos valores de los 4 Bytes se pasan a continuación a valores decimales, y se quitan valores del final hasta que haya sólo 8, o se rellena con ceros si no hay suficientes.

Ya tenemos los 8 valores; ya tenemos la contraseña, generada para la caja fuerte o para el usuario.

```
def calcularHOTP(contador, grupo, diffie):
    """Método que se encarga de calcular y devolver el HOTP"""
    # Guardamos un objeto con los parámetros a partir del grupo del primo elegido
    # ¿Ambos usuarios presentan la misma clave final?
    if not diffie.conexionCorrecta():
        # No: error y el sistema para
        print("Los valores finales de Diffie-Hellman no encajan, error en la comunicacion")
        return -1
    # Si: el sistema continua
    # Calculamos el HMAC del mensaje contador a partir de la clave en común
    resumenHmac = hmac_sha512(diffie.aFinal, contador)

    # 128 caracteres, 64 Bytes en total
    print("Resumen HMAC resultante: ", resumenHmac, "\n")

    # Coger el último Byte del grupo (por defecto 5f) para elegir un grupo al azar
    lastByte = resumenHmac[-2:]

    # Imprimir por pantalla el último byte
    print("Último Byte del HMAC: ", lastByte, "\n") #95

    # lista auxiliar donde guardar los bits del último byte
    aux = []
    for byte in lastByte:

        binary_representation = bin(int(byte,16))
        print("Representacion binaria de", byte, ":", binary_representation, "\n")

        salida = eliminarPrefijo(binary_representation)

        aux.append(salida)

    # Unir los bits para obtener la representación binaria del último byte
```

```

final = "".join(aux)
print("Binario del último Byte: ",final, "\n")

# Pasar a decimal para calcular el grupo a elegir; se divide entre dos y
# se pasa a entero porque final puede ser la segunda parte del Byte (impar),
# y queremos el inicio del grupo entero (par).
final = int(final,2)
final = int(final/2)
print("Valor en decimal del grupo de BITS resultante:",final*2,"\n")

# Si es un valor superior a 60 (61, 62 o 63), ponemos el final al máximo
# disponible, que es 60.

while final > 60:
    print("El grupo",final,"se encuentra fuera del rango de Bytes, que tiene como
máximo del 60 al 63","\n")
    final = final - 60

print("Se elige el grupo de Bytes",final,"\n")
# Cogemos el grupo de Bytes calculado antes y los 3 siguientes,
# hasta disponer de 4 Bytes
modulo = resumenHmac[final*2:final*2+8]
print("4 Bytes que nos salen:",modulo,"\n")
# Se pasan los valores, que están en hexadecimal, a decimal
# para ser un input fácil de introducir para un usuario
modulo = int(modulo,16)
print("Resultado decimal de los Bytes obtenidos:",modulo,"\n")

# Si el resulta tiene menos de 8 números, se añaden al final tantos
# 0's como sean necesarios hasta que haya 8.
while len(str(modulo)) <= 8:
    modulo *= 10

# Si el resultado tiene más de 8 números, se quitan al final tantos
# valores como sean necesarios hasta que haya 8.
while len(str(modulo)) > 8:
    modulo = modulo // 10

# Se devuelve el valor de la contraseña en común entre usuario
# y caja fuerte.
print("Primeros 8 dígitos de la contraseña:",modulo,"\n")
return modulo

```

Main(): Finalmente, la función main. Partimos de la misma semilla que en *diffieHellman.py*, y como ese caso, podemos recalculamos el mensaje todas las veces que sea necesario, comparando la salida con el archivo *mensajes.txt*.

Después de generar la interfaz para el usuario, establecemos la conexión entre usuario y caja fuerte.

Por defecto, calculamos el HOTP automáticamente de la caja fuerte, que es la contraseña adaptada por el sistema en ese momento.

Si son iguales, calcula el HOTP y lo compara, y si es distinto el del usuario, es que hay un caso de desincronización, y se pueden calcular tantos valores de HOTP para la caja fuerte, siempre dentro de la ventana introducida.

```
def main():
    # Semilla/Seed definida en cada inicio para obtener resultados consistentes = 30
    random.seed(30)

    if not os.path.exists('mensajes.txt'):
        os.mknod('mensajes.txt')
    # Contador del sistema para sincronizar, tiene 8 Bytes aleatorios,
    # como define el RFC de HOTP.
    # Sirve como mensaje del que calcular el HMAC a partir de la clave común
    contador = random.getrandbits(64)
    archivo = open('mensajes.txt', 'r')
    lineas = archivo.readlines()

    # Ya funciona, repasar diffie hellman
    for linea in lineas:
        if contador == int(linea):
            contador = random.getrandbits(64)

    f=open("mensajes.txt", "a+")
    f.write(str(contador))
    f.write("\n")
    f.close()
    print("El valor del contador es: ", contador,"\n")

    # entrada del usuario, que se espera sea un entero
    # en caso de no ser, error y sale
    # genera la pantalla para elegir grupo, guarda en n el valor introducido
    n = generarInput()
    print("\n")
```



```

# Abrir el archivo con los ID's de los usuarios

archivo_usuario = open('usuarios.txt', 'r')
lineas = archivo_usuario.readlines()
aux = False
# Comparar el id del usuario para ver que el usuario tiene los permisos necesarios.
# Se compara contra cada línea del archivo
for linea in lineas:
    if int(n3.get()) == int(linea):
        aux = True
# Si no los tiene, error y return -1

if not aux:
    print("ERROR; El usuario no tiene permisos")
    return -1
# Continúa de manera normal de lo contrario

# El valor no es numérico y devuelve un error
if not n.isdecimal():
    print("Introduzca un valor numérico la próxima vez")
    return -1

# Establecer conexion inicial a partir de un objeto diffieHellman
# se usa el grupo anterior
conexion = establecerConexion(n)
conexion.presentarResultados()
# Genera todos los parámetros de DH a partir del primo del grupo,
# los guarda en un objeto de tipo diffieHellman con todos los demás parámetros
contador_usuario = int(n1.get())
ventana = int(n2.get())
# Contadores desincronizados, el usuario puede estar adelantado
valorHOTPcaja = calcularHOTP(contador, n, conexion)

# Atencion, el usuario actual puede estar por detrás en sus aleatorios que el
usuario anterior
if contador_usuario != contador:
    # Dejamos el valor del usuario parado, la caja es la que va cambiando
    valorHOTPusuario = calcularHOTP(contador_usuario, n, conexion)
    print("EL CONTADOR DEL USUARIO NO ENCAJA. EL VALOR HOTP DEL USUARIO
ES",valorHOTPusuario)
    # Leer el archivo de mensajes aleatorios a partir de la seed de antes
    f = open("mensajes.txt", "r")
    # Leer todas las líneas menos la última porque no nos interesa el nuevo valor
    lines = f.readlines()

```

```

f.close()
lines = lines[:-1]
f = open("mensajes.txt", "w")
for linea in lines:
    f.write(linea)
f.close()

# Coger los últimos VENTANA valores del archivo, sin contar el generado para
esta ejecución
for aleatorio in lines[-ventana:]:
    # Print del valor con el que se opera
    print("Calculando HOTP para valor del contador",aleatorio)
    # Re-calcular el valor HOTP de la caja con el aleatorio
    valorHOTPcaja = calcularHOTP(int(aleatorio), n, conexion)
    # Si coinciden, se hace print de toda la info y se termina
    if valorHOTPcaja == valorHOTPusuario:
        print("EL SISTEMA ENCUENTRA EL MISMO VALOR HOTP PARA EL CONTADOR
GENERADO ANTES EN LA CAJA", aleatorio)
        print("VALOR DEL USUARIO:",valorHOTPusuario)
        print("VALOR DE LA CAJA FUERTE:",valorHOTPcaja)
        time.sleep(10)
        return 0

    print("NO SE CONSIGUE GENERAR LA CONTRASEÑA")
    return -1

# Los contadores son iguales, no hay problema
else:
    print("Los contadores de ambos son iguales, calculando HOTP para ambos\n")
    valorHOTPusuario = calcularHOTP(contador_usuario, n, conexion)
    if valorHOTPcaja == valorHOTPusuario:
        print("Los valores coinciden, se da acceso")
        return 0

if __name__ == "__main__":
    main()

```

6.3 Clase administrador.py

Esta clase se encarga de gestionar las funciones de las que se debe encargar el administrador del sistema, como añadir ID's de dispositivos con permisos para acceder al inicio de sesión y hacer todo el protocolo de HOTP.

6.3.1 Funciones del archivo

AñadirUsuario(): Añade un usuario, definido por el ID entero de su dispositivo, al sistema. Si ya se encuentra dado de alta devuelve -1 y un mensaje de error, de lo contrario realiza la operación con éxito y devuelve un 1.

```
from tkinter import *
from tkinter import messagebox

def añadirUsuario():
    """Función capaz de añadir un usuario al archivo de usuarios.
    Si el usuario ya existe, no hace nada y devuelve un -1.
    Si no existe, escribe el usuario y devuelve un 1"""
    try:
        user = int(n1.get())
    except:
        print("Introduzca un valor entero")
        return -1

    if type(user) != int:
        print("Introduzca un valor entero la próxima vez")
        return -1

    archivo = open('usuarios.txt', 'r')
    lineas = archivo.readlines()
    for linea in lineas:
        # El usuario ya existe en el archivo
        if user == int(linea):
            print("Usuario",user,"ya existente en el sistema")
            return -1

    f=open("usuarios.txt", "a+")
    f.write(str(user))
    f.write("\n")
    f.close()

    print("Usuario",user,"introducido de manera correcta en el sistema")
    return 1
```

borrarUsuario(): Borra un usuario del sistema. Si no se encuentra dado de alta devuelve -1 y un mensaje de error, de lo contrario realiza la operación con éxito y devuelve un 1.

```
def borrarUsuario():
    """Función capaz de buscar un usuario en el sistema y borrarlo.
    Si lo encuentra, lo borra y devuelve 1.
```

```

Si no lo encuentra, no hace nada y devuelve un -1"""
try:
    user = int(n2.get())
except:
    print("Introduzca un valor entero")
    return -1
if type(user) != int:
    print("Introduzca un valor entero la próxima vez")
    return -1
aux = False;
with open("usuarios.txt", "r") as f:
    lineas = f.readlines()
with open("usuarios.txt", "w") as f:
    for linea in lineas:
        if linea.strip("\n") != str(user):
            f.write(linea)
        else:
            aux = True;
if aux:
    print("Usuario encontrado en el sistema. Borrando")
    return 1
print("Usuario no encontrado")
return -1

```

borrarMensajes(): Borra por completo el archivo mensajes.txt

borrarClaves(): Borra por completo el archivo claves.txt

borrarUsuarios(): Borra por completo el archivo usuarios.txt

```

def borrarMensajes():
    """Función capaz de limpiar por completo el fichero de mensajes"""
    open('mensajes.txt', 'w').close()

def borrarClaves():
    """Función capaz de limpiar por completo el fichero de claves"""
    open('claves.txt', 'w').close()

def borrarUsuarios():
    """Función capaz de limpiar por completo el fichero de usuarios"""
    open('usuarios.txt', 'w').close()

```

Main(): Se encarga de presentar la interfaz gráfica que verá el usuario para operar con ella.

```
if __name__ == "__main__":
    ventana = Tk()
    ventana.title('Panel de Administrador')

    Label(ventana, text='Usuario a introducir:').grid(row=0)
    Label(ventana, text='Usuario a quitar:').grid(row=1)

    global n1
    global n2

    n1 = Entry(ventana)
    n2 = Entry(ventana)

    n1.grid(row=0, column=1)
    n2.grid(row=1, column=1)

    Button(ventana, text='Añadir Usuario', command=añadirUsuario).grid(row=4,
column=0, sticky=W, pady=4)
    Button(ventana, text='Quitar Usuario', command=borrarUsuario).grid(row=4,
column=1, sticky=W, pady=4)
    Button(ventana, text = 'Borrar Usuarios', command = borrarUsuarios).grid(row=4,
column=2, sticky=W, pady=4)
    Button(ventana, text = 'Borrar Mensajes', command = borrarMensajes).grid(row=4,
column=3, sticky=W, pady=4)
    Button(ventana, text = 'Borrar Claves', command = borrarClaves).grid(row=4,
column=4, sticky=W, pady=4)
    Button(ventana, text='Salir', command=ventana.quit).grid(row=4, column=6,
sticky=W, pady=4)

    ventana.mainloop()
```

7 Implementación final y pruebas en Arduino

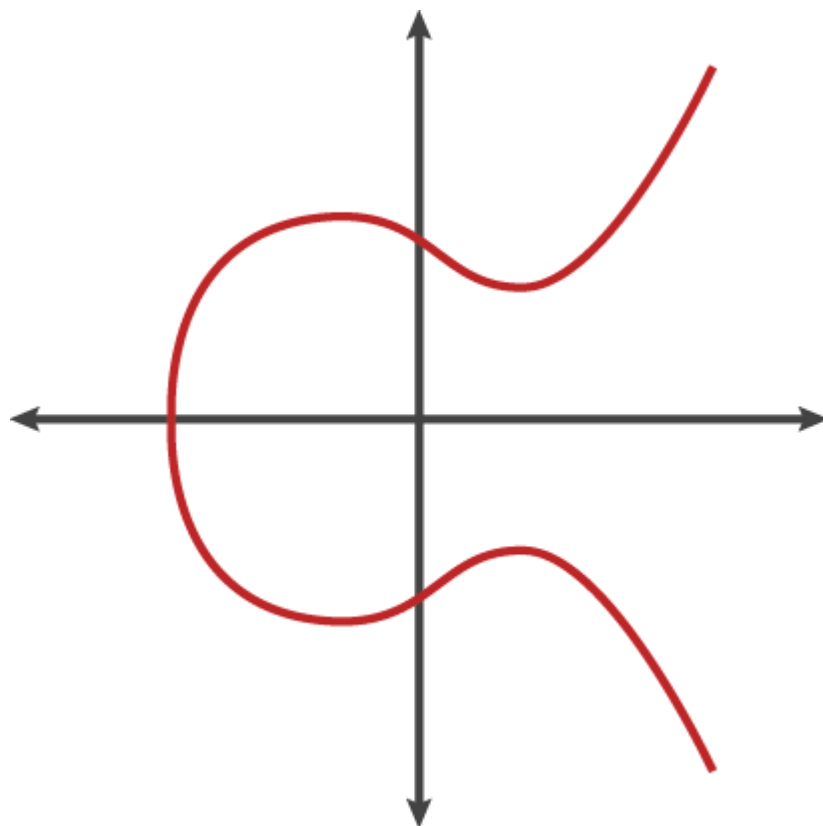
Para comprobar el funcionamiento del código de forma concreta con un microcontrolador, se ha usado un Arduino Uno y un simple LED de color verde para simular el comportamiento de un usuario del sistema y de la caja fuerte, en lugar del entorno de pruebas habitual, más abstracto.

Se ha conectado el LED al pin 13 del Arduino, y se ha transformado el código desarrollado en Python para transformarlo en código nativo de Arduino.

En primer lugar, se ha introducido código de varias fuentes externas, como por ejemplo Arduino Cryptography Library [20], una librería externa que introduce decenas de protocolos diferentes, relacionados con la criptografía, como funciones Hash, generadores de números aleatorios aprovechando los componentes de Arduino... En segundo lugar, se utiliza la librería SimpleHOTP [21], que implementa funciones HMAC y HOTP. Curiosamente, sólo se usa para generar HMAC-SHA1, y no HOTP (esto se hace de manera manual).

Entrando más en detalle en el programa para Arduino, no se diferencia tanto de su forma anterior. Para acordar una clave privada común entre usuario y caja fuerte, se usa Diffie-Hellman (de nuevo) sobre curva elíptica.

Éste es el momento oportuno para hablar más en detalle de las curvas elípticas para llegar a una clave común con la que iniciar una comunicación privada.



9

⁹ Ejemplo de una curva elíptica sencilla. La forma concreta de la curva puede variar dependiendo de los parámetros de su fórmula.

7.1 Curvas elípticas, HMAC y HOTP en Arduino

Las curvas elípticas comenzaron a ser estudiadas a mediados de los años 80, pero no fue hasta estas últimas dos décadas que empezaron a ser usadas con propósitos criptográficos a gran escala.

El uso de Diffie-Hellman sobre una curva elíptica es una forma muy interesante, además de elegante, de obtener el resultado de Diffie-Hellman, comentado al principio de este documento, para mantener una comunicación entre dos usuarios.

Este tipo de curvas tienen una propiedad interesante que las hace ideales para esta tarea: Son simétricas respecto al eje X. Esta propiedad hace que cualquier línea que atravesase esta curva tenga como máximo 3 puntos en los que corta.

Siguiendo el ejemplo de Diffie-Hellman tradicional, antes de comenzar la comunicación se llega a un acuerdo entre los dos usuarios para elegir una **curva elíptica común** (con una ecuación característica, como $y^2 = x^3 + ax + b$), un punto generador **G**, y un **primo**.

Cada parte escoge un número privado en el rango $[1, p - 1]$, y comparte, como antes, su clave pública de la forma $Pa = ka * G$, o $Pb = kb * G$. A partir de esto, es sencillo calcular una clave final con la parte pública del otro comunicante y nuestro número privado.

Entremos en más detalle. Se siguen aplicando los principios del problema del logaritmo discreto, donde realizar la operación de “ida” para sacar la parte pública es muy sencillo, pero determinar la parte privada de cada miembro, o el resultado final sólo con datos públicos, es una operación con un coste muy elevado, además de ser muy compleja.

Es una forma muy interesante de llegar a una clave común, porque al realizar la operación $P = k * G$, lo que hacemos es dar revoluciones o “vueltas” sobre la curva. Como la operación de módulo de Diffie-Hellman, ambas maneras de llegar a una clave común se basan en lo mismo.

Además, la curva sólo opera con valores enteros, para que los números privados y públicos funcionen de la misma forma que un Diffie-Hellman normal y no haya que operar con decimales en las operaciones

En concreto, se usa Curve25519 sobre éste Arduino. Curve25519 es llamado así por usar una curva basada en el número primo $2^{255} - 19$. Con este método se consigue un número final de 256 bits. Ésta curva elíptica es uno de los métodos criptográficos más novedosos de los últimos años para llegar a una clave común.

Tras 30 años de estudio, aún no se ha encontrado ninguna técnica que se aproveche de alguna debilidad subyacente para romper un criptosistema basado en ECC, o “Elliptic Curve Cryptography”. Es mucho más difícil que romper un criptosistema basado en las matemáticas tradicionales, como RSA (otro sistema criptográfico para generar claves públicas entre usuarios), y mucho más caro computacionalmente, por lo que es una buena elección.

Recientemente, algunos métodos criptográficos de curvas elípticas, como P-256, han sido vistos con malos ojos tras existir indicios de que poseen “puertas traseras” para que las agencias de seguridad del país en cuestión descifren las

comunicaciones entre usuarios. Curve25519 es la alternativa de facto a este P-256.

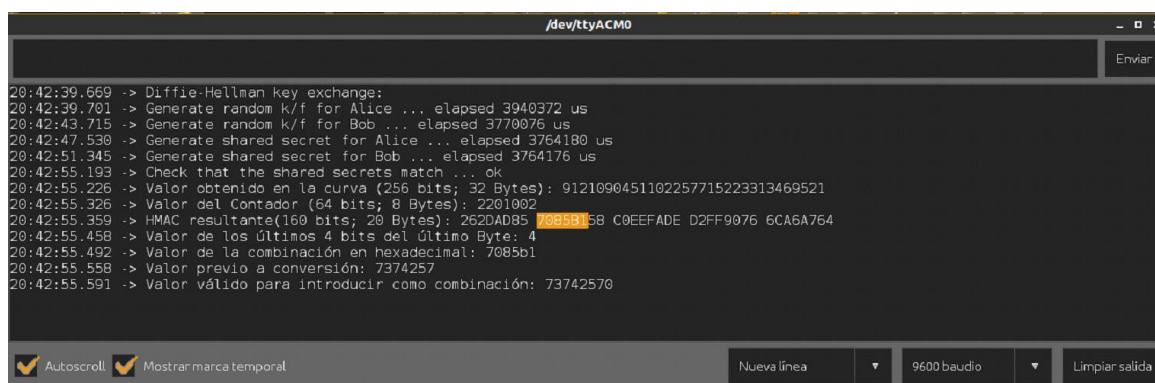
El siguiente paso es calcular el HMAC de los 256 bits anteriores. Dado que no se dispone de una capacidad casi ilimitada como en un ordenador normal, se ha optado por usar un HMAC-SHA1. Su funcionamiento es exactamente igual a un SHA-512, mencionado anteriormente, sólo que da un output de 160 bits en lugar de 512 bits. Esta decisión también ha sido influenciada tras investigar las funciones HMAC y SHA, y comprobar que las funciones HMAC no pueden sufrir ataques de extensión de longitud, como algunas de las SHA. Además, como lo que se busca es generar una secuencia “aleatoria” para la caja fuerte, no se prevé que los problemas presentes en SHA-1 afecten al sistema.

Esto es, puede ser un problema si se utilizase como forma de verificar hashes de archivos, o identidades digitales en la red, pero si lo único que se busca es calcular el HMAC con una clave secreta de un conjunto de bits para tener “ruido” y números pseudoaleatorios, no se ve nada negativo en el uso de SHA-1. No es tan bueno como usar HMAC-SHA-512 u otro método más reciente, pero sirve para probar las capacidades de cualquier Arduino.

Finalmente, el método HOTP no cambia mucho. Se cogen los últimos 4 bits del último grupo generado en HMAC, y dependiendo del valor (De 0 hasta 15), se coge ese grupo y los siguientes para llegar a una combinación de 8 dígitos final que presentar al usuario.

En este caso, cogemos en total sólo 3 grupos, es decir, 3 Bytes, porque coger 4 Bytes y pasarlos al rango decimal puede superar el rango permitido de los números de tipo “long” de Arduino. Si al pasar de hexadecimal a decimal no se llegan a 8 números de combinación, se rellena a 0’s. Por ejemplo, si la secuencia aleatoria da como resultado 000000, 000001... que son 0, 1... la combinación sería 00000000 y 10000000. Dentro de la aleatoriedad que presenta, es un caso posible.

A continuación, una imagen de este procedimiento desde el monitor de Serie:



```
/dev/ttyACM0
20:42:39.669 -> Diffie-Hellman key exchange:
20:42:39.791 -> Generate random k/f for Alice ... elapsed 3940372 us
20:42:43.715 -> Generate random k/f for Bob ... elapsed 3770076 us
20:42:47.530 -> Generate shared secret for Alice ... elapsed 3764180 us
20:42:51.345 -> Generate shared secret for Bob ... elapsed 3764176 us
20:42:55.193 -> Check that the shared secrets match ... ok
20:42:55.226 -> Valor obtenido en la curva (256 bits; 32 Bytes): 9121090451102257715223313469521
20:42:55.326 -> Valor del Contador (64 bits; 8 Bytes): 2201002
20:42:55.359 -> HMAC resultante(160 bits; 20 Bytes): 262DAD85 7085b158 C0EEFADE D2FF9076 6CA6A764
20:42:55.458 -> Valor de los últimos 4 bits del último Byte: 4
20:42:55.492 -> Valor de la combinación en hexadecimal: 7085b1
20:42:55.558 -> Valor previo a conversión: 7374257
20:42:55.591 -> Valor válido para introducir como combinación: 73742570
```

No se incluye como imagen el encendido del LED durante 10 segundos tras conseguir llegar a la clave final.

7.2 Código de la implementación

```
#include <Crypto.h>
#include <Curve25519.h>
```



```

#include <RNG.h>
#include <string.h>
#include <SimpleHOTP.h>
#include <stdlib.h>

static uint8_t alice_k[32];
unsigned long startTime;
uint32_t code[5];
uint8_t contador[8];

//Funcion de ejemplo sacada de los ejemplos de la propia biblioteca SimpleHOTP que
se usa
void testDH()
{
    static uint8_t alice_f[32];
    static uint8_t bob_k[32];
    static uint8_t bob_f[32];

    Serial.println("Diffie-Hellman key exchange:");
    Serial.print("Generate random k/f for Alice ... ");
    Serial.flush();
    unsigned long start = micros();
    Curve25519::dh1(alice_k, alice_f);
    unsigned long elapsed = micros() - start;
    Serial.print("elapsed ");
    Serial.print(elapsed);
    Serial.println(" us");

    Serial.print("Generate random k/f for Bob ... ");
    Serial.flush();
    start = micros();
    Curve25519::dh1(bob_k, bob_f);
    elapsed = micros() - start;
    Serial.print("elapsed ");
    Serial.print(elapsed);
    Serial.println(" us");

    Serial.print("Generate shared secret for Alice ... ");
    Serial.flush();
    start = micros();
    Curve25519::dh2(bob_k, alice_f);

```

```

    elapsed = micros() - start;
    Serial.print("elapsed ");
    Serial.print(elapsed);
    Serial.println(" us");

    Serial.print("Generate shared secret for Bob ... ");
    Serial.flush();
    start = micros();
    Curve25519::dh2(alice_k, bob_f);
    elapsed = micros() - start;
    Serial.print("elapsed ");
    Serial.print(elapsed);
    Serial.println(" us");

    Serial.print("Check that the shared secrets match ... ");
    if (memcmp(alice_k, bob_k, 32) == 0)
        Serial.println("ok");
    else
        Serial.println("failed");
}

void setup() {
    Serial.begin(9600);
    // Dejar el led del puerto 13 para iluminar
    pinMode(13, OUTPUT);
    // Ejecutar el DH de arriba
    testDH();
    // Generar un número aleatorio de 8 Bytes y guardarlo en "contador"
    RNG.rand(contador, sizeof(contador));
    // Fragmento para pasar de String a char []
    char salidaContador[8];
    String textoMensaje = "";
    for (int i = 0; i < sizeof(contador); i++){
        textoMensaje += String(contador[i]);
    }
    textoMensaje.toCharArray(salidaContador, sizeof(salidaContador));
    // Fragmento para pasar de String a char []
    char salidaAlice[32];
    String textoAlice = "";
    for (int i = 0; i < sizeof(alice_k); i++){

```

```

        textoAlice += String(alice_k[i]);
    }
    textoAlice.toCharArray(salidaAlice, sizeof(salidaAlice));

    // Presentar por la consola de DEBUG los valores de DH y del contador
    Serial.print("Valor obtenido en la curva (256 bits; 32 Bytes): ");
    Serial.println(salidaAlice);
    Serial.print("Valor del Contador (64 bits; 8 Bytes): ");
    Serial.println(salidaContador);
    // Hacer que la clave sea el valor de DH presentado anteriormente
    Key key(salidaAlice, sizeof(salidaAlice)-1);
    // Generar HMAC. Clave es la de la línea anterior, mensaje es el contador de 8 Bytes
    SimpleHMAC::generateHMAC(key, salidaContador, (sizeof(salidaContador)-1)*8, code);
    // Presentar por consola de DEBUG los valores del HMAC en HEX
    Serial.print("HMAC resultante(160 bits; 20 Bytes): ");
    for (int i = 0; i < 5; i++) {
        Serial.print(code[i], HEX);
        Serial.print(" ");
    }
    Serial.println();
    // Guardar el último grupo, para acceder a los 4 últimos bits de manera sencilla
    String lastGroup;
    lastGroup = String(code[4], HEX);
    // Acceder a la longitud del último grupo, en caso de que sean menos de 4 Bytes
    int len;
    len = lastGroup.length();

    char valorInicio = lastGroup[len-1];
    // Unir los 5 grupos del HMAC total
    String total = "";
    total += String(code[0], HEX);
    total += String(code[1], HEX);
    total += String(code[2], HEX);
    total += String(code[3], HEX);
    total += String(code[4], HEX);
    // Necesario presentar el código desde la posición valorInicio hasta 3 más adelante.
    Serial.print("Valor de los últimos 4 bits del último Byte: ");
    Serial.println(valorInicio);
    // Código para pasar el número a un entero
    int numero = String(valorInicio).toInt();
    switch (valorInicio){

```

```

    case 'a':
        numero=10;
        break;
    case 'b':
        numero=11;
        break;
    case 'c':
        numero=12;
        break;
    case 'd':
        numero=13;
        break;
    case 'e':
        numero=14;
        break;
    case 'f':
        numero=15;
        break;
}

// Agrupar los Bytes que toca coger para la combinación desde el grupo que dicen
los últimos

// 4 bits. Va desde 0 (valor HEX 0) a 15 (valor HEX f).
String combinacion;
combinacion += total[numero*2];
combinacion += total[numero*2+1];
combinacion += total[numero*2+2];
combinacion += total[numero*2+3];
combinacion += total[numero*2+4];
combinacion += total[numero*2+5];
combinacion += total[numero*2+6];
combinacion += total[numero*2+7];
char salidaCombinacion[7];
combinacion.toCharArray(salidaCombinacion,sizeof(salidaCombinacion));
long respuestaFinal = strtol(salidaCombinacion, NULL, 16);
Serial.print("Valor de la combinación en hexadecimal: ");
Serial.println(salidaCombinacion);
Serial.print("Valor previo a conversión: ");
Serial.println(respuestaFinal);

int numero_digitos = floor(log10(respuestaFinal)) + 1;
while (numero_digitos < 8){

```

```
    respuestaFinal *= 10;
    numero_digitos = floor(log10(respuestaFinal)) + 1;
}
Serial.print("Valor válido para introducir como combinación: ");
Serial.println(respuestaFinal);
// Encender el LED
digitalWrite(13, HIGH);
delay(10000);
digitalWrite(13, LOW);
delay(10000);
}

void loop() {

}
```

8 Vulnerabilidades del sistema

En este apartado se comentarán algunas de las posibles formas que se han detectado por las cuales el sistema podría ser atacado y accedido sin seguir los protocolos correspondientes.

En primer lugar, está la opción de un forzado clásico con herramientas capaces de comprometer los elementos físicos de la caja. Al menos con una implementación electrónica podemos estar seguros de que no será posible el uso de ganzúas u otros elementos con los que acceder a una cerradura externa, ni tampoco con el sonido, como aquellos profesionales capaces de evaluar en qué estado se encuentra la cerradura sólo por los sonidos internos al mover el dial de combinaciones.

Lo cierto es que, ante un ataque físico real, una caja sólo puede aguantar un tiempo determinado antes de ceder por completo, y esto es evaluado por la categoría en la que se encuentre. Disponiendo de tiempo ilimitado y un equipamiento moderno, ninguna caja es segura ante un atacante bien equipado, por eso son necesarias medidas disuasorias.

Pasemos ahora a formas en las que es posible forzar el sistema electrónico o el propio software.

Una de las más sencillas formas de hacer esto, por ejemplo, es proporcionar al Arduino del sistema central un voltaje que no pueda soportar. Siendo el límite 20 voltios, habría que comprobar cuál es el voltaje que sobrecarga por completo el Arduino. Tras esto, la caja quedaría sin forma de poder acceder a ella, dado que no existe ningún tipo de backdoor mecánico que nos deje acceder (como debe ser, por otra parte), y habría que recurrir a herramientas manuales de forzado para abrir la caja y obtener los objetos de valor que alberga. Este proceso destruiría la caja en el proceso, puesto que no es buena idea la de incluir ninguna puerta trasera.

El acceso al sistema central estará dividido en usuarios con y sin permisos, y la conexión por cable en un caso ideal no permitiría sobrescribir nada en el Arduino, pero podemos contemplar una serie de casos con los que acceder a él tras, por ejemplo, taladrar la compuerta exterior y tener acceso a los puertos.

Normalmente un usuario normal nunca debería tener acceso a estos controles, igual que no podría leer los ficheros con los mensajes, claves o usuarios, pero en el caso de un atacante con acceso a los mensajes o claves generados de forma pseudoaleatoria, podría ser capaz de, mediante ingeniería inversa y suficientes mensajes y claves, sacar posibles valores futuros. Esta sería una tarea que necesitaría de tiempo y acceso continuado a la caja o sistema hasta sacar una lista de valores posibles, además del equipo con el que taladrar la compuerta. Incluso podría alterar el generador de números aleatorios para cambiar el mensaje y la clave secreta a su antojo y poseer ya las respuestas de antemano.

Sería preocupante también que se pudiera acceder al sistema de otra forma aparte de la de la entrada física (Bluetooth, Wi-Fi...), por ejemplo, como los dispositivos IoT que se suelen usar hoy en día, y como bien explica “Security Analysis and Exploitation of Arduino devices in the Internet of Things” [22], ya existen métodos para provocar heap y stack buffer overflow en ciertos modelos de Arduino en el Internet de las Cosas, pudiendo de esta forma extraer

información de ficheros, como mensajes o claves, para generar claves y mensajes futuros, o tomar control del programa en ejecución. Con un punto de entrada adecuado, se podría conseguir toda esa información, por lo que es adecuado que cuantos menos (sólo un punto de entrada físico), mejor.

Otra posible vulnerabilidad se encontraría a la hora de encender el sistema cada vez tras hacer reset en cada uso. De no existir unos ficheros de mensajes y claves, al reiniciar el sistema (apagado automático después de cada uso), se conocería siempre el estado inicial de éste, y se podría romper de manera muy sencilla la implementación. Al acceder a estos ficheros para comprobar si las claves generadas al inicio del programa ya han sido usadas, nos libramos de este defecto, pero somos dependientes de un generador de números pseudoaleatorios que debe estar sincronizado entre el sistema y cada usuario. En resumen, es necesario esconder los mensajes generados, si se guarda constancia de ellos, o contar con un generador de número aleatorios que garantice con gran probabilidad que un reset no afectará al RNG.

También se puede imaginar un caso en el que un atacante pretenda forzar el sistema mediante miles de peticiones por segundo, para intentar obtener claves, mensajes... O desbordar con peticiones para encontrar un fallo por estrés. El propio RFC de HOTP habla de implementar un sistema que aumenta en varios segundos el tiempo de respuesta a un usuario que falle varios accesos seguidos, para no poder realizar este tipo de ataque, además de la ventana capaz de recalcular el HMAC en los casos que se definan antes de dejar fuera del sistema al usuario.

9 Resultados y conclusiones

Llegados al final de este trabajo, podemos decir con certeza que el desarrollo ha terminado de manera exitosa, y la experiencia ha sido muy positiva.

El proyecto no ha presentado ningún tipo de problema que merezca especial mención. Se comenzó con una idea básica de lo que es la seguridad aplicada a un sistema físico como una caja fuerte, y de ahí se comenzaron a estudiar distintos tipos de algoritmos que se pudieran aplicar al desarrollo, gracias a la ayuda de mi tutor, Jorge Dávila, que en los momentos en los que no sabía cómo seguir, sus palabras me sirvieron de gran ayuda para probar ideas nuevas en este trabajo.

Todo el trabajo realizado ha sido posible gracias a los RFC mencionados hasta la fecha, y a la dedicación de la Internet Engineering Task Force, cuyos investigadores han dejado al alcance de todo aquel interesado algoritmos y protocolos sobre los que seguir innovando en el tiempo.

Todos los grandes avances en las áreas de la criptografía y la seguridad de los últimos 50 años han sido documentados en esta serie de RFC's; y como es habitual en una área como la de la seguridad, nunca es buena idea idear nuevos algoritmos de la nada, siempre es una opción mucho mejor utilizar aquellos algoritmos que han sido probados una y otra vez en el tiempo, pues nos apoyamos en los hombros de aquellos gigantes que en el pasado los idearon.

Estos RFC se han seguido y adaptado para nuestro uso de la manera más conveniente posible usando un lenguaje de programación actual como es, en el primer caso, Python: fácil de entender (y modificar) para cualquier persona interesada, en caso de querer seguir con el desarrollo, todo esto bajo una licencia GPLv3. En C está una versión de prueba capaz de hacer funcionar el programa por sí mismo.

Por añadir sencillez a un sistema que en una aplicación real trabajaría sobre un microcontrolador, se han diseñado varios ficheros de configuración en texto plano en lugar de diseñar complejos sistemas de bases de datos para mensajes, claves y usuarios con acceso.

Se considera que los resultados obtenidos son muy positivos. La capacidad de no repetir mensajes ni claves de nuevo, obtener un valor HOTP diferente en cada uso de una caja fuerte... Todo esto siguiendo las medidas de seguridad de los que hablan los RFC's anteriores, haciendo múltiples tests y pruebas sobre el código, y un desarrollo continuo, como se puede observar en el repositorio de Github.

A pesar de ser mi primera entrada práctica al mundo de la seguridad aplicada a sistemas informáticos, se considera que se ha hecho una buena labor de investigación, que es la parte que más tiempo ha llevado: recabar información, contrastar la seguridad que proporciona cada módulo del trabajo, cómo funciona en detalle, explicarlo de manera que todo el público pueda interesarse por igual... La parte de desarrollo ha sido más sencilla en comparación con la anterior: teniendo la información necesaria, es tan sencillo como consultar librerías de criptografía seguras y operar de manera correcta con los resultados.

En lo personal, el trabajo ha sido muy gratificante y enriquecedor. Cada momento de investigación, de pruebas, de desarrollo de código... Ha sido

bastante agradable, y una de las mejores experiencias que he vivido en toda la carrera; esto es, partir desde 0 con una idea inicial y conseguir crear un sistema que se puede aplicar a la vida real, o que se puede presentar a cualquier empresa de seguridad para continuar con el desarrollo, si esto se presentara.

10 Líneas futuras

Después de completar el desarrollo de este trabajo, se hablará sobre cómo se puede continuar en el futuro con este proyecto.

En primer lugar, se debería disponer de los componentes físicos necesarios, y de una caja fuerte con la que probar todo el sistema.

Dado que el coste de todos estos componentes es bastante alto, se ha desarrollado todo de más abstracta, centrándonos en el código en Python y C y la seguridad que nos ofrecen. Numerosas pruebas deberían de realizarse sobre todos los materiales físicos, como el máximo voltaje que son capaces de soportar en una situación real, la facilidad de poder forzarlos para intentar acceder a los contenidos de la caja elegida... La resistencia de cada componente.

Relacionado con el tema anterior, si se desea continuar con el desarrollo también sería necesario formar un equipo de profesionales en áreas como diseño de circuitos y diseño de cajas fuertes, además de expertos en el área de la seguridad tradicional, física y electrónica, para revisar el proyecto de nuevo y, en caso de haber algo que no cumpla los requisitos, tomar las medidas necesarias.

Estas cuestiones son necesarias puesto que es un proyecto orientado a una implementación real, quizá incluso orientado a su desarrollo en una empresa de seguridad y cajas fuertes. Debe de cumplir con los estándares de las zonas del globo a las que se lleve la idea.

Tras comprobar esto, también se deberían probar distintos microcontroladores que ofrezcan un mejor rendimiento y sean específicos para el tipo de tareas (paso de mensaje y cálculos criptográficos) que se especifican.

El aspecto de la interfaz visual ofrecida como apoyo visual no preocupa mucho, ya que en un sistema real todo esto se realizaría de manera secreta entre las partes involucradas en la comunicación. Se ha incluido sólo de forma ilustrativa en la presentación.

Finalmente, se debería presentar la idea a cualquier parte privada interesada en un mecanismo de seguridad de este estilo. Como se comentó en este documento anteriormente, la mayor parte de la lógica interna de las cerraduras de alta seguridad modernas se encuentran bajo secreto, o sólo mencionan de forma superficial su funcionamiento, por lo que comparar otros modelos actuales con el desarrollado en éste TFG es una tarea complicada. Es una muy buena idea la de llevar esta idea a posibles compradores que ya tengan una experiencia previa manteniendo otros sistemas de seguridad, ya que la implementación de HOTP para contraseñas de un sólo uso puede ser algo novedoso e interesante para el campo, y puede proporcionar una mayor seguridad y un factor de aleatoriedad en cada uso.

Con una serie de ajustes para implementarlo en el uso diario a través de una App móvil específica, puede ser una idea bastante interesante de llevar a buen puerto.

Como se puede ver, presenta mucha versatilidad, se quiera llevar a un terreno de la seguridad doméstica más tradicional, o a un entorno más profesional y experimentado.

11 Bibliografía

- [1] Definición de cerradura inteligente, https://en.wikipedia.org/wiki/Electronic_lock
- [2] Ha, Ilkyu. (2015). Security and Usability Improvement on a Digital Door Lock System based on Internet of Things. International Journal of Security and Its Applications. 9. 45-54. 10.14257/ijisia.2015.9.8.05, http://article.nadiapub.com/IJSIA/vol9_no8/5.pdf
- [3] What is Google Authenticator, Margaret Rouse, 2014, <https://searchsecurity.techtarget.com/definition/Google-Authenticator#:~:text=Google%20Authenticator%20is%20a%20mobile,masquerade%20as%20an%20authorized%20user>.
- [4] What is Multi-Factor Authentication (MFA), Vince Lujan, 2018, [https://securityboulevard.com/2019/10/what-is-multi-factor-authentication-mfa/#:~:text=Multi%2Dfactor%20authentication%20\(MFA\),biometrics%2C%20time%2C%20and%20location](https://securityboulevard.com/2019/10/what-is-multi-factor-authentication-mfa/#:~:text=Multi%2Dfactor%20authentication%20(MFA),biometrics%2C%20time%2C%20and%20location).
- [5] RFC 4226 (2005), <https://tools.ietf.org/html/rfc4226>
- [6] RFC 6238 (2011), <https://tools.ietf.org/html/rfc6238>
- [7] RFC 2104 (1997), <https://tools.ietf.org/html/rfc2104>
- [8] RFC 4868 (2007), <https://tools.ietf.org/html/rfc4868>
- [9] Caja de caudales, definición de la RAE, <https://dle.rae.es/caja#2s7HuNa>
- [10] Orden INT/317/2011, de 1 de febrero, sobre medidas de seguridad privada, <https://www.boe.es/buscar/act.php?id=BOE-A-2011-3171>
- [11] An Introduction to the History of locks, <https://www.locks.ru/germ/informat/schlagehistory.htm>
- [12] Diffie, Whitfield & Hellman, Martin. (1976). New Directions in Cryptography. Information Theory, IEEE Transactions on. 22. 644 - 654. 10.1109/TIT.1976.1055638, <https://www.nku.edu/~christensen/092mat483%20DH%20paper.pdf>
- [13] RFC 2631 (1999), <https://tools.ietf.org/html/rfc2631>
- [14] RFC 3526 (2003), <https://tools.ietf.org/html/rfc3526>
- [15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. 2018. Imperfect forward secrecy: how Diffie-Hellman fails in practice. Commun. ACM 62, 1 (January 2019), 106–114. DOI:<https://doi.org/10.1145/3292035>, <https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf>
- [16] Especificaciones físicas de Arduino, <https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD>
- [17] Ejemplo de comunicación entre dos Arduinos, Master y Slave, <https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter>

- [18] Ejemplo de activación de actuador lineal, Firgelli Automations Team, 2020, <https://www.firgelliauto.com/blogs/tutorials/how-do-you-control-a-linear-actuator-with-an-arduino>
- [19] Repositorio de Github asociado al trabajo, <https://github.com/PCMSec/TFG>
- [20] Arduino Cryptography Library, rweather, 2012, <https://rweather.github.io/arduinolibs/crypto.html>
- [21] SimpleHOTP Library, jlusPrivat, 2019, <https://github.com/jlusPrivat/SimpleHOTP>
- [22] Alberca, Carlos & Pastrana, Sergio & Suarez-Tangil, Guillermo & Palmieri, Paolo. (2016). Security analysis and exploitation of arduino devices in the internet of things. 437-442. 10.1145/2903150.2911708, <https://cosec.inf.uc3m.es/docs/papers/2016mal-iot.pdf>

12 Anexo A: Pruebas sobre HMAC y HOTP

Por comodidad, todos los resultados que normalmente se imprimen por terminal se encuentran en formato de texto.

12.1 Pruebas iniciales

En un primer momento, antes de automatizar un sistema de pruebas aparte, se ha comprobado que ninguno de los HMAC iniciales (para los primos de los grupos 15, 16, 17 y 18) colisiona con el resto, para comprobar que esta parte inicial es correcta.

Los resultados de la operación también se han comprobado con una página web de un tercero, freeformatter.com/hmac-generator.html, capaz de generar un HMAC SHA-512 a partir de un mensaje y la clave. Al ser una página de un tercero, se descartan también posibles errores que la propia máquina local podría haber dado como correctos.

Esta primera tanda de tests se ha hecho sobre una semilla aleatoria predefinida igual a 30.

12.1.1 Grupo 15

Mensaje:

14906391684844699610

Clave secreta:

18753840255348843856775405748813838180603814521195350221694303167813405912
88909256317795025529973070732929637823090809913093072193671265230795498055
87308470180753352787867351111224591519307174544714965871118077056750993144
24159386014610157549979995114970135980672751671878786253060013870621431780
57287180358661213674739650939802155566167587419091532543128109626387487177
99243797987032053855849637839588354080083282874095466279930219083069380756
00731425184438555086150464273119080936621827295589293091601711356987297474
18536074710327045945687382950662039130947554777303792983413536641828925510
35097562644615776525085512535944425553199974812053651924250370283688747120
85985829474687335736628248112102963408840352789705592023396042647892786652
00853032047346620652276374077437898360121324191255307597184438017305835153
92152129149419052035852922693008629251860271400055061618576286709319118829
4561043595918107255633043920402181358

Resultado:

70f28d460d12490322919865ee3dab25fc558ac161fe232d2a4c1f77c70b1ff621db18591f4472f
8e23f7b1b2a3a1538c10bebbbbb612dc206b80b2ff09cf71f6

12.1.2 Grupo 16

Mensaje:

14906391684844699610

Clave secreta:

81959933953197952991903375534751808408335781783896732151257485508750585401
64758031303080691856889675698735549963430329170954068445221757588768213841
51411405090098998480068381105525703834147936666968959193183752616283388683
57521591129615991434313436514990325836640248109331790417747261443063620390
32082795829106570888019621839591712757031918749112308645262193186433486712
90318484608061592674939660642690097510667738997140611576189432703811471318
06881599138845743462241734348166757233057077928916062042280972741517085396
89337745881221282994077812807448134511116587943375059647447863219281004606
10361674537362337590250973534020130160757097404497197654815678481672650171
43915609563869639930715711320339920395473299896054783230304220979495210023
99730086226296136166137251887916596876755566820805729365676525224949874873
60839277181889369394669164994676520251631785667619771585415686185460658637
37622638657383973568599289505245636969997596920470487877968238196262993418
93561710911247259538207518345159138056272396896151921817372091524247739631
68956485992809059375768838124382316707034390353221090303681008144675948827
33338038808472295565401734864272304069701799924815781699325768481664372607
6353435234049667676783308654221492561491754599414

Resultado:

a8b9c890d4789f8d7b169de699fd1743386a2ab97f21d3993b725c63830e43e3706d420c52ba
8a57c985651e21c768c15ed53935954fa2329d91f66e84db4fdf

12.1.3 Grupo 17

Mensaje:

14906391684844699610

Clave secreta:

22946890134272630780177385643597323696957853782697812827488048961096665517
50999655337399528160200222749913321663880383768108586122947825251786535703
17105225655501357163151348260258429315793308543802516460617949328077947630
90942559003731203918110081190301916494790260018372155504459059296191747394
92976158730622270971427700574085055790661891852569083219626664102555353737
32489529349040725603722387963432325516855387438541194419548462765773946884
05417497671567656372669466920467713695226556033168263336900696834929585460
00625371716031037425129714911172541718383336462798892341247328269515877851
44254322156677956191146644621312715384052278387645376581188118259837908321
37701105948630233688161375347897813684744400634425673516883407998970371103
55535359686468172731618722629084071860672737278983421257190866571761597156
18421492512532228469408654864920394117705233540043084957546639448197364524
94536111061760076174607435069051784980086000066136851697158766639252597691
19587045483885195489214092977449099063143493493468842489585540903492268041
46705565976672441648477881765958295510107900755151243939188045514621267715
51267377795182341941913750267621475221626575509217092142852700620435178475
02874483885397664073517288545917035727870491502340018350782555627374161830
32470845703237044268192280371897581607727665319662878903180132425337567779
11895898883460252389471948368599508701172043397354477241678971998205695611
99995933686566915306043259626779013183176248548946872960067997179543367441
11057540261918122799616205317858405059395534480732035411198810192908804098
70260749876005511304218408955256865229057298403940673783131632835414399210
08935714853759799936304714061900678236595364348795781428170199813135775422
7875262064202103870731411662082356499499322804958067965560546078962013402
44654697413353024164867993399922779358072621683974122771093964490137968777

Resultado:

19a00b800e0cc2847d17d0fc80f44ef9656b3df86a4c93c933a081a16b6eeb3b3fd4a88a8db47a
42f5613862327870e6bb11496ce32fa8e37c932a2cc557516c

12.1.4 Grupo 18

Mensaje:
14906391684844699610
Clave secreta:
43956265485896723728633825955716014144185707430104593377719479054654214820 29393567461990218725372119680477102262583088479020805891089719800186744207 76284238510382772618018277933157724449134321887192414205951145358491188169 39993840676764666925236728036052996993686887266217996759193441462289697313 68207320832609637766282190912905213454122926793982303467007244681998129605 45169194407339063506746078264642511618048611118417194935065490632888145295 50940968324137260021949390892599478042216967768499718895644764109350789742 74345313079132722192537270755166580122442569163807396351056962679236297425 70031757405051251563528545635216481545557051496841432731560400949597206290 22188693324053559676253778353029507024950765113043950529214462358380223270 89725271665759692156775674711442724644511464477577132178964378937926687714 10604721375463087288507921918086792214517026272545572898477954729496678587 36803405785139675619617941532852973422110090749478688177715315730359460032 29466290245662516117441660698362441855200063649930148271949072404374664452 87020087801405350805985221356908017836210572399780327689665358117783924305 83642314163278573210784571216359620205429638053335635504834951966172036535 17775771015131553722318251646346969353003971579014786804025028076962541815 28475476895803903969637154880577125443245561334196488895851445816177040343 99810912752816934978854515292651125082328009815738113672572529961056098864 14366015120246746217613099165008109401497764047047384322568793196159315390 72518967043619187186801035344861942724751250768423138965750173226420820855 40466833846817042022407816169569237792390864978603255742310449769255577268 33476098251137292069410801622827633381264994872996176967611609157777169844 96721893315552827301193789348747923658963473606317280545894987253756550057 40601436160333662679482993007062812127712070614381661141729343695993494300 19218344270340837307361622426465846342493002807818955469534583289415171975 99195921606419746951988373442509672938396764958713751104084682316475038389 06174004447326017635012431609966855235339172161949922899701777890560505615 98510031273779159205515889486799390372323940833284091943659666076573757403 15989374608948289682556597962658663069770845320812040746030199870813666382 57070076514746279605337085437580699801127262870405799518856236675675751003 20052554010581610348840423549055355324672611171587744122774823457249235576 06553226249341751463664443074155483907530908658870174589199600561806519442 687686194294431417207470
Resultado:
4737e7ffce3c906f7a94593f259f82cdf98610a46b5030e72c7578cae026ce960a546f107239c3d 490bd508bfa2ef727c4de0137fc6537e761a3a1e5f50f3323

Todos los HMAC anteriores son correctos para cada mensaje con cada clave secreta.

12.2 Pruebas avanzadas

Tras implementar la parte de aleatoriedad para que no se generen dos mensajes o dos claves iguales, se han procedido a hacer más pruebas sobre valores aleatorios, tanto con el mismo valor para el contador, como para contadores distintos, pero dentro de la ventana.

12.2.1 Prueba 1

Para la ejecución actual, se obtiene un contador de la caja fuerte con un valor de 9791974547769813817. Introducimos como contador de la llave el mismo, una ventana de 10 en caso de que no hubieran sido iguales, y operamos con el grupo 15.

En primer lugar, establecemos la clave en común mediante Diffie-Hellman (asignar una clave privada a cada parte de la comunicación, calcular las claves públicas, intercambiar, y calcular la clave final):

Valor	privado	de	a:
46368474962158838557949212789398280824266999646174050229198950031041847015			
53621277856412331763646548520304647905284014067898225672602277954587844928			
41976834482907051551385839373528850140535340865680058743047501517874987488			
88016287674561229070960603450146454443457949134978174717907708943875715071			
24810622702647852205499385559333016440566410618430211082794955695364311847			
50073021968713021008160903576231991511599469092511452339586616701843940719			
76033189096830655357677294960123411529262174200804639956401267384190436274			
22099425852208695167573875428649342961177944149547591267829848640783456985			
06847111150018071641397248097274323501563471782435382082302525191956969102			
49040561322819895490325929438886376748686465415065178376185055613068162285			
74213697828696459304345751510881697867317475780874944982848058821215942474			
11738687838946898799646152629943906170275204523524730515454616391612390266			
3924658889016234105648418031867385856			
Valor	privado	de	b:
22091967352039524862750914308946268065967838504654867972535659244965221775			
13428261529761474549353467256712542614133713537905268595356116009004520203			
21997123491100713462694039305334257685560843883600169066573100228911960090			
98487478916119142043520422523973739497954114118065604731734563022974053161			
29386567072529272306703348135530010984830258594718525899024610950262881509			
51502003670863139829706302786200352416379935245695219629911668657252983656			
76972712862113221608803961979835957881512282566409395351166523307716608809			
80571944401770600825044965527660175691843629530995863832409126991028622904			
72656361884114040136911583953730975977836292305969842914687536055378548446			
55361854385310804814074899187197284981340763346188873527779926547437668500			
24446505804673969490366822051292023644615064140304131167643535419397711255			
08697470536290630286722378670182867018301330059337152693436340182237613157			
2785762340340949399772531143528131717			
Valor	público	de	a:
48860787194289593925796780498697341839660565862183377631010482764168518594			
88544093604589444908447177636785567740920278833316217428135685323924780804			
02043417805147695972724722435708065966276768608083969474619199783864677288			
72308669908111072911802698057793230372289470808589412493874119762742535357			
94585088565091362593170593707670958332055572750377707049721991245708848132			
66816388123126034075827538894152417296691377441412900703403169079141589829			
93452660186844564001523729417206229999158541597367320960001449954292700634			
21721334030683977440943268727250661667350447674409426742751452303418296168			
82316060267362971799308340018596309628410615991043545578701749616238710253			
18067066415738712397433430341633826248311257535670622055197906375012825066			
52396695144408380407628313886040911459577087900570622266853045802290699324			
34178996996253120464360755809907420018201636402908604530162527527150184255			
1952705358324068637315919886728804651			
Valor	público	de	b:
17235531688809373874898676976348026557821949264659663277043519212429864588			
87860692380516883708935926878686906992564376076187792212430210399511554293			

17169472357857364021741601225358185977940886289381953333383762373049180066
24571170834184513966020763995322873963483243082616800253208469281489555618
88245266741895540379742111320807440766393294482941688240954797556684116134
39023394781668435420449844380994181319433190981726192206122223372770831495
86937577567587009271203848994633757782768122806337722736389801900018917241
14222230962668016719434222342901286364659111928374130433106439578800503238
83676453087815334440623685503221214794404760375119863012591944951669549867
86738029685781138544890075715021262357588454335690820052000483062825607802
63839172095377419228885353130609830232479739321750290431141043367813196304
70530054447485915356890787198365694147186458767488413134746922924175333714
3544376701892061681412792135485045945

Valor de la clave para a:
46876624213734238698099612968235547426428666709234502316012486818712944063
61046772048547602643175176647332395142834702536829241461845881310250367249
53612187543216892920703127313702826852518502399145448081199257295261753190
73280817318804261786695880599759796155003960891222581428792150994376423501
89227872072115114433708068536489408225381586849963704532625633182772817123
80726447592943713384349144371922359262186589735446425528037026034222206898
30645534879220390252538809376619330751112192865316027364414564198937679423
95027368765659990675323968665102740545294897993975289791834773544073455375
74049499974081196299879305978008336600833868932592448964667447371423726272
62996437797941319556855373445897449261200580360623341965966279802854430909
76957752308745165790598769007630255137527918410416771213146691255509188036
92543473847501787098577417159638612038196635822754053614695455085354936588
6483080846548262959255664297928501249

Valor de la clave para b:
46876624213734238698099612968235547426428666709234502316012486818712944063
61046772048547602643175176647332395142834702536829241461845881310250367249
53612187543216892920703127313702826852518502399145448081199257295261753190
73280817318804261786695880599759796155003960891222581428792150994376423501
89227872072115114433708068536489408225381586849963704532625633182772817123
80726447592943713384349144371922359262186589735446425528037026034222206898
30645534879220390252538809376619330751112192865316027364414564198937679423
95027368765659990675323968665102740545294897993975289791834773544073455375
74049499974081196299879305978008336600833868932592448964667447371423726272
62996437797941319556855373445897449261200580360623341965966279802854430909
76957752308745165790598769007630255137527918410416771213146691255509188036
92543473847501787098577417159638612038196635822754053614695455085354936588
6483080846548262959255664297928501249

Los valores son iguales, comparten la misma clave

A continuación, a partir del mensaje contador, calculamos HMAC y HOTP para la caja fuerte, y si los valores del contador coinciden para ambos, calculamos HMAC y HOTP también para el sistema de llave del usuario:

Tamaño del HMAC resultante: 64 Bytes

Resumen	HMAC	resultante:
094877f400d0723dd2f5f48aafdfc5d849a833719535348c0ec75a69a0960146ae5e14f6b1c91c27cb6c7f2cd3aa061a28b7edcc32b77ae1aafe9031e86719da		

Último Byte del HMAC: da

Representacion binaria de d : 0b1101

Representacion binaria de a : 0b1010

Binario del último Byte: 11011010

Valor en decimal del grupo de BITS resultante: 218

El grupo 109 se encuentra fuera del rango de Bytes, que tiene como máximo del 60 al 63

Se elige el grupo de Bytes 49

4 Bytes que nos salen: b7edcc32

Resultado decimal de los Bytes obtenidos: 3085814834

Primeros 8 dígitos de la contraseña: 30858148

Los contadores de ambos son iguales, calculando HOTP para ambos

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen	HMAC	resultante:
094877f400d0723dd2f5f48aafdfc5d849a833719535348c0ec75a69a0960146ae5e14f6b1c91c27cb6c7f2cd3aa061a28b7edcc32b77ae1aafe9031e86719da		

Último Byte del HMAC: da

Representacion binaria de d : 0b1101

Representacion binaria de a : 0b1010

Binario del último Byte: 11011010

Valor en decimal del grupo de BITS resultante: 218

El grupo 109 se encuentra fuera del rango de Bytes, que tiene como máximo del 60 al 63

Se elige el grupo de Bytes 49

4 Bytes que nos salen: b7edcc32

Resultado decimal de los Bytes obtenidos: 3085814834

Primeros 8 dígitos de la contraseña: 30858148

Los valores coinciden, se da acceso

12.2.2 Prueba 2

Para el siguiente caso, el contador del usuario estará adelantado con respecto al de la caja fuerte (utilizada un valor aleatorio generado anteriormente, 476862029414530019), pero está dentro de la ventana de sincronización. La caja tendrá un contador con el valor de 7411514733785699652, la ventana tendrá un valor de 20, se operará con el contador del usuario 11316404850165261876 (10 valores adelantados), y con el grupo 15.

De nuevo, se llega a una clave a partir de los dos números privados:

Valor	privado	de	a:
85953781726812398477575915190976345518399180873645183249985823237419208797			
83638866536876383964568451860928552304621348012909696662511926526926941811			
09167931154249328722995607041542774156902369159487845571334345510707691173			
74979421429530424487890869600512700614838862725633444141282342667049250754			
54989998206234494633142320793242572369037603312014203616534946034531388802			
41517240946021863975944346129848513544813015724746169706873259876952277930			
76066230829869499160204775855766253286039934959815341553852546159100050145			
47310271734130950119635597397776502692886729132049655108086388552936842692			
67407093584260415726952831435050640593692786427344364799956985746322337854			
85220730668182498552545034667978576278980346766037164040103062574273560914			
74451915856547257339191249414149283037946493022957458685455716562157040895			
93862949432856229478497404041775788542966606847430792356478955216813986425			
200983442942474502649461484312456843			

Valor	privado	de	b:
30080475359797676779499593480086052647712579175844948580787927096311863596			
59899528452580892742880066497668406370935852653376024800927476266405516443			
60326439510934074671867737593707251606375270532498772693415096625265876915			
42648401216579228171101746264972954022802888359000378966139893770482151767			
66130326066144854978975527229898301929255547385016675543071621196293654318			
01991618847817898790871839807046844270306361586509691784421510693781288798			
76765333967971851195602190795765660473615773608517253843270059084110783406			
44221934352782073060500269061484514335510831729331285069474215014442827001			
26972461769129599235526949897722005509662518919879824136216012273248182774			
15662750090165596213470164883293939903514591298198975008941620006654132633			
09568281423623903193437805723757181774147513569299618453278877183119453485			
82207853743006627279973951868237985494163368181500898741883094994540250695			
1228094454302441747453312134131985223			

Valor	público	de	a:
90677395265571863556672592339213798146305397298253588921904107841124158059			
76304812551246411485029778790110286340756983715125321866554915684730353844			
31163486002496017125954346360587081249956244554374794133382525901675914756			
59958605824548692590170561996316028505215228411194389743314939656747169224			
55551909968422405523867053462800529625537809137643141846014721466060345052			
34123349181734071505657671625412321903441126640306062167270494621976555047			
79260086152041149017603387716814191493081790400565524398925961550458720102			
34329661060504218026039693430610837960379262896897682309814991789377362205			
85779910527351288485981121276680488067673037717577339746959096191028239873			
78574251427569951798125289221769752673538065809881394267285504618834925242			
52298445328229124120049749433304952856443631187996091309245110367097229692			
44752792370577378069328750243684778148260912270069461042440666349686374986			
580695207613332294752389606357112831			

Valor	público	de	b:
48204231071112488477986117422492847106909953786582437659454082649912906791			
97487168800523899093583238840069712730770222899247453341438726371320373514			
05304899078890697354905531936530905075621923842608022342409340593643054918			
92584786683697346542395148194352022204970192271498305097831624238617546600			
28237666020225676439292322776025750480525884671667644526578556444335089682			
88026190114934973524171877550963577329109437514729329515274268678118653455			
31958414249142202083371479573466165254861851024303867442656175620589558096			
38940616498689872622203710859174290999794661289661551055975109405580577752			
36529997215517484293626354744874231760769982466371663788474761479896832844			
19068645551985101803476719055638894950735073007992875922508299338082056729			
17190151137088107021445281502404410524338078065508990884787993084837453577			
74254034845840588495717889977555254624973084018985665492708639994560935672			
5833463943576116959808518391575331268			

Valor	de	la	clave	para	a:
32303438607219396995427734384294673449814762293237715896276583485081032127					
15433347476447102972497703121226995229695732621052599194620582160847951944					
83816292763120555607104695547877412222986866141554640080063793898822296965					
72404960324711363315268192765802061639403156467870858400506068124116377110					
23238288047070813678218298663198161557974908316639271159336273531893830924					
73067550226411526936514875062899007718498645642522650642606341961492974896					
96381155697576172273692077835670058656039398573526708781511166366916054346					
04304065326376341329184100698314138107204639636351797891902814232259157070					
16345120163710594261932733913197782396239294665985805855913360368879098451					
15551231680764583285477132359122608945860580305082931950003869928812936752					
59629456307799840476754925396464750864804112334670063399511478494757386048					
39386543804704869461636461481930879877861844505164380163582614332913077119					
3872593358421980191351443756309235316					

Valor	de	la	clave	para	b:
32303438607219396995427734384294673449814762293237715896276583485081032127					
15433347476447102972497703121226995229695732621052599194620582160847951944					
83816292763120555607104695547877412222986866141554640080063793898822296965					
72404960324711363315268192765802061639403156467870858400506068124116377110					
23238288047070813678218298663198161557974908316639271159336273531893830924					
73067550226411526936514875062899007718498645642522650642606341961492974896					
96381155697576172273692077835670058656039398573526708781511166366916054346					
04304065326376341329184100698314138107204639636351797891902814232259157070					
16345120163710594261932733913197782396239294665985805855913360368879098451					
15551231680764583285477132359122608945860580305082931950003869928812936752					
59629456307799840476754925396464750864804112334670063399511478494757386048					
39386543804704869461636461481930879877861844505164380163582614332913077119					
3872593358421980191351443756309235316					

Los valores son iguales, comparten la misma clave.

Calculamos el HOTP de la caja (pero como no son iguales los contadores, habrá que recalcularlo más tarde).

Tamaño del HMAC resultante: 64 Bytes

Resumen	HMAC	resultante:
e6bdd0b4e46bd92d4151b236bc8225f9b9f50f6aa26cf0d853cfddfea52176f084f22620f5bcae591af6352a99dc4103e390f60590f16ca51bb5e2aa80326055		

Último Byte del HMAC: 55

Representacion binaria de 5 : 0b0101

Representacion binaria de 5 : 0b0101

Binario del último Byte: 01010101

Valor en decimal del grupo de BITS resultante: 84

Se elige el grupo de Bytes 42

4 Bytes que nos salen: 352a99dc

Resultado decimal de los Bytes obtenidos: 891984348

Primeros 8 dígitos de la contraseña: 89198434

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen	HMAC	resultante:
bc191a8384dc5ad7049be507282adcc4989dd9420a31b9a943c3521d99ba73fa8e0f5a460eba9270f11144f0ee989cddb9b755ba5aaabee2f1e96e60d256d3a		

Último Byte del HMAC: 3a

Representacion binaria de 3 : 0b0011

Representacion binaria de a : 0b1010

Binario del último Byte: 00111010

Valor en decimal del grupo de BITS resultante: 58

Se elige el grupo de Bytes 29

4 Bytes que nos salen: ba73fa8e

Resultado decimal de los Bytes obtenidos: 3128162958

Primeros 8 dígitos de la contraseña: 31281629

EL CONTADOR DEL USUARIO NO ENCAJA. EL VALOR HOTP DEL USUARIO ES 31281629

Calculando HOTP para valor del contador 14906391684844699610

Como el valor del usuario está retrasado, y entra dentro de la ventana de 20, la caja fuerte cambia su valor HOTP para coincidir con la llave del usuario, y recalcula hasta coincidir con la contraseña del usuario:

[Resumido con el último valor para no ocupar tanto espacio]

Calculando HOTP para valor del contador 476862029414530019

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen	HMAC	resultante:
bc191a8384dc5ad7049be507282adcc4989dd9420a31b9a943c3521d99ba73fa8e0f5a460eba9270f11144f0ee989cddb9b755ba5aaabee2f1e96e60d256d3a		

Último Byte del HMAC: 3a

Representacion binaria de 3 : 0b0011

Representacion binaria de a : 0b1010

Binario del último Byte: 00111010

Valor en decimal del grupo de BITS resultante: 58

Se elige el grupo de Bytes 29

4 Bytes que nos salen: ba73fa8e

Resultado decimal de los Bytes obtenidos: 3128162958

Primeros 8 dígitos de la contraseña: 31281629

EL SISTEMA ENCUENTRA EL MISMO VALOR HOTP PARA EL CONTADOR GENERADO ANTES EN LA CAJA 476862029414530019

VALOR DEL USUARIO: 31281629

VALOR DE LA CAJA FUERTE: 31281629

12.2.3 Prueba 3

En ésta prueba, se toma un valor para el usuario que no encaje dentro de la ventana establecida.

La caja tendrá el valor de contador 7411514733785699652, el usuario 100 (para fallar), el grupo será el 15 y la ventana será igual a 2.

Los pasos iniciales son como los casos anteriores, se recalcula el valor de la caja fuerte para los 2 valores aleatorios generados anteriormente (la ventana), y como el usuario está desincronizado, hay un error.

Valor	privado	de	a:
54166933283021665597108837322070400722798845975749840116763826201885662808			
13851381977770738023316452068004932889080576778436994020191049006049715327			
56901012922291216717318569603194919298207248444927220639790073802662126448			
13872356950168366571885656629341469442754527474040116899665600494959786631			
88027907782738785580326252022110946589999874822230975886728506441119544443			
44260366460573339499279210204822336210401738323760870143874918772480124887			
18354506721616848484428689562274211463739670488030791956876214008613440164			
0948823980222284948334826330146705722810917213626120108067169564255828075			
37535938629551002702012424969942946773084380789721743617983714366971412478			
28753918697496703527865015165641694750985909090835728002070142623458753679			
34242798513135479622670634169008040907179295613013467962253557028069960168			
85823533758893731652711282093795800892424731753648165944423888112456850937			
8515721641560037993352478718034288483			

Valor	privado	de	b:
48190724014465837462510858958210527070760206427838388518158418690359465476			
19687965349709767058099342891551968751869143628799642061765611565589389914			
12665617803845304943191131378542370669113798054326364482779923607383125595			
50895570853981514198024284914728625099599019324087645887807671269964449557			
43437543091875565057134218307896672248091434982132839263406278628657258536			
20815787250124292365292428346390629935007912750025870308550639240072720354			
51243587332331897738469310004988302269195249727379219812588407182204926308			
18272310752112332128351137372773357957327500525233998764807880682552532571			
85366482699677855778284269010506657447363063421252974074847311759788616564			
75975575030624333467841714251342949000825632609661484399331000104598902659			
32096146773747343657295906175131317848946204059088749305971781734235587580			
47164514232557617688575483301781723151472290966104754837547851961297817452			
6680193815856751403081136148632149088			

Valor	público	de	a:
57327771309317037417014163755842662332255994156122628000714407312816302199			
94467978321420461669942366431105830924643312879087144925171036286155570269			
59575362743354689159717353665313806458209103161263787322309907797290213705			
29597675583470357224637897497090516307461728551723253386882982068834175085			
39570803673553272662149558409666147940742571544983595310244282996807947780			
20863983896386236123320069602990409908484311712827166629417795759682898610			
80165984412290360367869847421708227705907536080110769349738605759190875201			
88792040600041476613829253923287929864617127969761055262874831704834388745			
69133404497066670135801873393049592115583983067757152303332134247088563800			
65783945940722342441473808130716208479188699260627508960034529664624968838			
16688089510779568854728441241534397231381808549949907923824545392032134343			
38117689987860827553071404781855292404037809536304852665222291475628851764			
3420703459365640344512610836423166211			

Valor	público	de	b:
34523892045008911476821737426920548948692974898302183793037790898019831737			
77837874346990327618281179763240392416079821129401083824540590741995627116			
70298133696919503360731174110866886446956710248571587961193636790413019595			
77389364210833719826063296128261401483805324022728918895374220916549486214			
79175085676531431847079098594647749000357966758088251987491493840179687829			
64719764189875964512489529369886828563321939176785996391406575156939177934			
70123071524834445550884395490643369320212940523551837992978407235181411538			
20245874819873991986346896595363041825786803426770358605522199539757365415			
28488412513751641206277896536771263563392451317074568483954401887720167731			
09344048112132037357184822850353502085136512653815684031522697147209079306			
40740599368452043731802436380571491982815168761481979599634630458773779091			
10693186911649315299832670630689591197096900572157491986288510289469157988			
1457094299052466732638579405774137029			

Valor	de	la	clave	para	a:
27812643160396274477848253727135767168753541549228289875567461556256000985					
30266585498324841216961764661118474417322795043478975433472009948199601521					
84281349288575652109633708203744605236953056620075259832896676804931981822					
56160749499039902831037593870953767471350441233277190437888008184370276057					
93091126819064614257821884224557883432290477348923173549238271623415091008					
25666216730740401011330288424125566487982096575065216772500096861410820118					
65738950748369711284973613629195957421014895817972519323030178085804713632					
07340074947323803424488271350467223136082626434042445756156856668165503203					
12143358984105913055513286666249246101001369016385393147111574686281589595					
18653904526877903791237699079847493396444916888636345881264449756134201003					
85359230277719753964056924680863561930715503054616861983030439153443445759					
16802208741616590499809815550417587971360644055947371698723335715517414480					
8394930250928768912446364457557324974					

Valor	de	la	clave	para	b:
27812643160396274477848253727135767168753541549228289875567461556256000985					
30266585498324841216961764661118474417322795043478975433472009948199601521					
84281349288575652109633708203744605236953056620075259832896676804931981822					
56160749499039902831037593870953767471350441233277190437888008184370276057					
93091126819064614257821884224557883432290477348923173549238271623415091008					
25666216730740401011330288424125566487982096575065216772500096861410820118					
65738950748369711284973613629195957421014895817972519323030178085804713632					
07340074947323803424488271350467223136082626434042445756156856668165503203					
12143358984105913055513286666249246101001369016385393147111574686281589595					
18653904526877903791237699079847493396444916888636345881264449756134201003					
85359230277719753964056924680863561930715503054616861983030439153443445759					
16802208741616590499809815550417587971360644055947371698723335715517414480					
8394930250928768912446364457557324974					

Los valores son iguales, comparten la misma clave

[Resumido]

EL CONTADOR DEL USUARIO NO ENCAJA. EL VALOR HOTP DEL USUARIO ES 12060706

Calculando HOTP para valor del contador 15054378804418375653

[Resumido]

Calculando HOTP para valor del contador 9791974547769813817

[Resumido]

NO SE CONSIGUE GENERAR LA CONTRASEÑA

12.2.4 Prueba 4

En las siguientes pruebas se comprueba que los usuarios tienen o no permisos para acceder la sistema. En éste primer caso, no se tendrá acceso.

Se dará permiso en un fichero .txt, con permisos para los ID's 5, 6, 7 y 8. Introducimos un usuario que no tendrá acceso, con un ID 1, por ejemplo. El resultado obtenido será:

El valor del contador es: 6128953485477967316

El grupo introducido ha sido el 15

El contador del usuario es 6128953485477967316

El margen con el que se opera es 15

El ID del usuario es 1

ERROR; El usuario no tiene permisos

12.2.5 Prueba 5

En ésta prueba, se introduce un usuario con el ID igual a 5, uno de los que tienen permisos. El resultado obtenido será:

El valor del contador es: 6228590875159555504

El grupo introducido ha sido el 15

El contador del usuario es 6228590875159555504

El margen con el que se opera es 14

El ID del usuario es 6

Valor privado de a:
13736027981824370971309248134224865682709553971455602273352883779046259594
36159863594032528371312897211847372386389430166898639529769519957261757429
60877887648489876090713756395730705731435959166764003040640124619877738672
20630334884387842729079580382594155729372857980774641992647313059480869324
13034856330615251139953562734858779005230628729999556069915885829230215363
72899883883289312396949599277553092319148057627530673863852367690574333296
31238934284521605971204365121682779641116692310505064195643496849085159344
98609233619899420489214827514305217444871940495192401881545246521432919282
75917572147969051088188546287629042306377503159931719856143099005200058307
49812630132851660055196451984797389626443246376808661909508356620803810337
76777353687383319888213724510088722984891361740308327568275286042592485184
58343366698413989374022772886446826173450925481930705842659778487814481145
9662529696316264026498109205623137407

Valor privado de b:
50880719604918457385404054465754161244517326370797203345804513969170218403
60161312582174382213051237079398447094387807956689006813319196989810215760
09808913383162979443918341652597590258564724225147446981751599484287090414
62148984570552685307127291213795833100354589097517393474668243349068262418
00432257384670140343328204865071044460599624026711773051885146538319390929
80258991459979752306426778679195842127342255412255234158471020830190955064
29826227831969625841125422614395633731869717411804969707936127755444567272
08317151062386558304376120551278202037210854327153472509411128255203805796
00585005603582845639819668792250902651406794000315055668147505830157827120
65221659351077693727035262261519094599598609925331489505997982941062644410
01791716836541584087224250940236687187215526479074296261125917554073987031
34375800701401439589631285748483584033809768786307978914222390101688875221
3291638949339895232480692805432411277

Valor público de a:
37257189543286827418186008457962095126401704407046538316266544007876861061
31023309168149662024650504966321393017609513265244555598646278080131372755
63008865504911761588619423638363035021866921278751293280096184808825791584
08336957708213905095603326545010008064172497114545854263645580218087292169
69189124575048448407060392415482056674028587751414546639330639576823637935
00897011877703633782024600654742858127736874594010025451583074294885985477
6119731336222242370750331080552279996701895530376842769436603149519508693
26468666892444707027915652143016499403555642989989594916894008083283151111
08690066081849953822802868851756796210051906445497184150940841878262317207
33691215997111316051735810422836182388143851635156057019923535958159046248
39522187504539751925003428394877772928626032289817128105931863506439033984
96997709647439817176137682092403498351367817831743987358546230347566038885
5905108385864145154878427594615484410

Valor público de b:
46527700699951522909164297497190458492494225678849458675454232470204800278
57800402620692894435870232090633688829220982422941886931702189163862552922
94676747761097675515410609923134745972200635794215540500922106871237174819
85353811535993195647523459853317536910215894027913897954822317958619363713
02744129712626235546240434513709699097122773942746491031114388045026316771
26822859905171664232371475654377281273497392190422001235335471883277460798
89106114295509049666405384971122796650004919976627093136059710714060762500
56108149993628779694690392615136500200804751508978923377589852683019710251
26422689965017836377734816922843764421144097230747116827230713958551146976
79164300747590184058258525209340562078976326606640498294744406854995932060
35885261292447953339408665966797106791863500459172174553637925470173280826
14784498331022037347200381875735543382378380383450199347061605154680897616
0536092364721412154824746895345472631

Valor de la clave para a:
10234444496854323126053165737869690285967968700214148215949308198388712400
6454218686226631650759922261647959418185583262666577868190769555382440814
45819468470313063458114854547245897726022524371287223985194954531663823820
87676542183307772496694309634304007667366660636425481151622179991706862508
24026823438170971525928449815221672677292439724738291527547136635166875576
05936278719976391500905189391633946904985713750414216125157849848934911973
17876548203470071095532999522312445487763604968911597680649924953328730660
97277089653132560765979451904740477033435541390986787371965066065093766994
60312961169491713957642105963053288642546705230911090499896907666132668150
76042209263503725813710220055773124568290541371066232067630914092948366459
70180867111957563417785631040149332465109795724326724863356121450409967833
04344782332849430453832282809730310966483637802783319267081334548242182948
0406519249825305342084270912472835040

Valor de la clave para b:
10234444496854323126053165737869690285967968700214148215949308198388712400
6454218686226631650759922261647959418185583262666577868190769555382440814
45819468470313063458114854547245897726022524371287223985194954531663823820
87676542183307772496694309634304007667366660636425481151622179991706862508
24026823438170971525928449815221672677292439724738291527547136635166875576
05936278719976391500905189391633946904985713750414216125157849848934911973
17876548203470071095532999522312445487763604968911597680649924953328730660
97277089653132560765979451904740477033435541390986787371965066065093766994
60312961169491713957642105963053288642546705230911090499896907666132668150
76042209263503725813710220055773124568290541371066232067630914092948366459
70180867111957563417785631040149332465109795724326724863356121450409967833
04344782332849430453832282809730310966483637802783319267081334548242182948
0406519249825305342084270912472835040

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
46b7d8ae6bcbeaf212dc2a7010667b3fe15432f297078515902bf8ab9aaba5b383d8aa4c11ea8
71f3c7bcbfccfe1ede8204877c3a5c9be4845fc39ef87b1709

Último Byte del HMAC: 09

Representacion binaria de 0 : 0b0

Representacion binaria de 9 : 0b1001

Binario del último Byte: 01001

Valor en decimal del grupo de BITS resultante: 8

Se elige el grupo de Bytes 4

4 Bytes que nos salen: 6bcbeaf2

Resultado decimal de los Bytes obtenidos: 1808526066

Primeros 8 dígitos de la contraseña: 18085260

Los contadores de ambos son iguales, calculando HOTP para ambos

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen	HMAC	resultante:
46b7d8ae6bcbeaf212dc2a7010667b3fe15432f297078515902bf8ab9aaba5b383d8aa4c11ea871f3c7bcbfcfcfe1ede8204877c3a5c9be4845fc39ef87b1709		

Último Byte del HMAC: 09

Representacion binaria de 0 : 0b0

Representacion binaria de 9 : 0b1001

Binario del último Byte: 01001

Valor en decimal del grupo de BITS resultante: 8

Se elige el grupo de Bytes 4

4 Bytes que nos salen: 6bcbeaf2

Resultado decimal de los Bytes obtenidos: 1808526066

Primeros 8 dígitos de la contraseña: 18085260

Los valores coinciden, se da acceso

Este documento esta firmado por



Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Tue Jan 26 14:47:13 CET 2021
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)