BORRADOR

31/10/20

Cerradura Digital de alta seguridad para cajas fuertes

Pablo Castillo Martínez
Tutor: Jorge Dávila Muro

Índice

Resumen

A lo largo del tiempo, 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 (sucursal bancaria, uso privado, uso en comercios...), el último siglo ha traído con él grandes avances para desbaratar los intentos de robo de aquellas posesiones guardadas bajo llave.

Otro campo que va a la par es el de la criptografía y la seguridad informática, que en mucho menos tiempo ha desarrollado cientos de métodos y estándares con los que mantener la seguridad de nuestra información y conexiones, y que a día de hoy sigue expandiéndose. Proteger nuestros datos personales (cuentas bancarias, contraseñas), posiblemente nuestros datos más preciados, siempre estará a la orden del día.

Este trabajo de fin de grado estudiará si es posible (y cómo) aplicar diferentes técnicas de criptografía para la creación de una cerradura digital de alta seguridad.

Abstract

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

From better materials to build them, 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 also have evolved are Criptography 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 Criptography algorithms for the creation of a high security lock.

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 cerraduras tradicionales, pasando por las que aprovechan los efectos del electromagnetismo y la corriente para evitar un forzado tradicional, a la tecnología de lector de tarjetas, biometría, claves... 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 (la posibilidad de no llevar llaves encima, de poder conectar nuestros teléfonos para abrir una puerta...), también han avanzado las habilidades y conocimientos de aquellos interesados por romper estas defensas.

Pero, ¿cómo se adaptan las cerraduras electrónicas a cajas fuertes? 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.

Si bien es cierto que las cerraduras más modernas [1] (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 digital proporciona una seguridad adicional.

Pero en esta área cabe aún espacio donde innovar. Por ejemplo, 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 los 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 éstas técnicas tras algunas de ellas llevar múltiples décadas en uso.

Trabajos previos

Si se pretenden aplicar componentes de seguridad de tipo criptográfico 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, por ejemplo, destacan los servicios proporcionados por Google (y empresas que permiten el uso de éste en su software) y su Authenticator [2]. 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 [3] (que también suele ser conocida por autenticación de dos factores, ya que normalmente 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 éstos algoritmos, que se explicarán en más detalle en la parte de codificación del proyecto.

Los RFC de más interés son los siguientes: RFC 4226, RFC 6238 (ambos de HOTP), RFC 2104 y RFC 4868 (ambos de HMAC). [4][5][6][7]

A continuación, se pasará a describir con detalle las características físicas de los mecanismos de caja fuerte/cerradura.

Especificaciones físicas

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.”[8]

En su forma más simple, una caja fuerte consiste de una capa exterior de un material duradero, difícil de penetrar, con un espacio limitado, seguro y bien definido en su interior, 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 valor de los objetos que pueda contener (seguros sobre los contenidos del interior contratando un seguro), la resistencia al medio en el que se encuentre (a fuego, a explosivos)...

Los materiales pueden ir desde el plástico endurecido, al 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 [9], la norma UNE EN-1143/1 o la norma EN-1300 sobre CAS(cerraduras de alta seguridad).

La puerta sobre la que se accede suele ser del mismo material que el exterior, dejando fuera del alcance de un atacante cualquier posible punto de acceso, 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 taladros, protegiendo así la “lógica” interna de la cerradura.

Cerradura

Una cerradura 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 por ambas partes [10].

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

Para abrir ésta cerradura, es necesario tener una llave, un trozo de algún material duradero cuyo perfil ha sido alterada para seguir un patrón que encaje con el interior de la cerradura. Este interior consiste de una combinación de varios pines apoyados sobre un muelle, 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, y poder abrir o cerrar la cerradura.

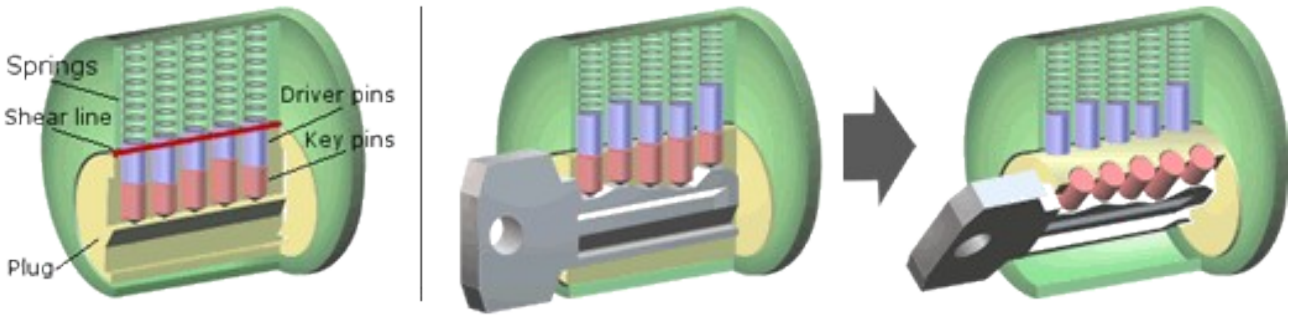


Ilustración 1: Fuente: www.safe.co.uk

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.

Un método que no necesita de ningún token físico es la cerradura de combinación, que consiste en un dial externo que se conecta a unos discos interiores (tantos como "números" haya que introducir).

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

Cada disco también posee una conexión al siguiente disco, y un "diente" en cada lado.

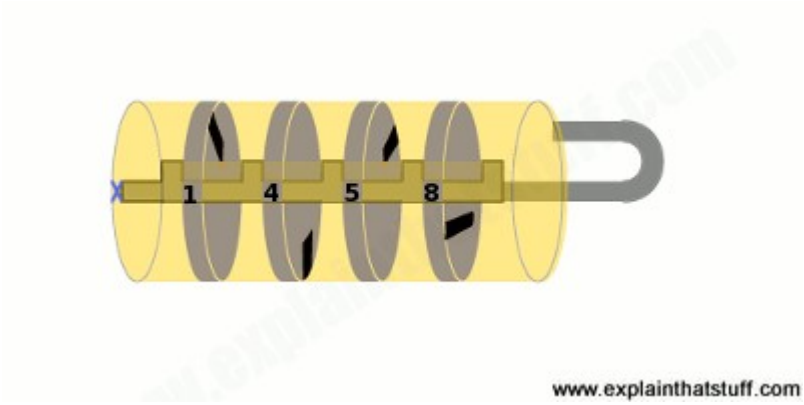


Ilustración 2: Fuente: https://www.explainthatstuff.com/yalelock.html

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).

A su vez, la combinación puede estar dividida entre 2 o más personas de confianza, para que nadie tenga todos los números a la vez.

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 como se introduce la combinación y así obtener el secreto para futuros usos.

En el caso de las cerraduras de sistemas de alta seguridad tradicionales, su apertura suele consistir de varios métodos de cierre, para aumentar su seguridad. Es posible tener una combinación de varias cerraduras basadas en varias llaves (tradicionales o electrónicas), cada una llevada por un empleado, y sólo la unión de todas las llaves da acceso a la cámara.

Añadir un generador de secuencias aleatorias, usadas una sola vez en la vida del sistema, puede ayudar a crear combinaciones más seguras de un solo uso. Además, a pesar de que alguien observe cómo se mete la combinación, al cambiar en cada uso esto no tendría gran 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.

Mecanismo de apertura

El mecanismo de apertura es similar al de una cerradura tradicional en una casa moderna, sólo que, en lugar de haber uno o dos "salientes", hay desde 5 a 20 que se encajan en el marco de la puerta para que sea imposible forzarla de esta forma. Gran parte de los mecanismos modernos de las compuertas son secretos y no están a disposición del público, pero se supone que estas cámaras usarán más de 5 discos por razones de seguridad (mayor número de combinaciones, mayor dificultad de encontrar la combinación desde cero...), además de medidas adicionales de seguridad (cámaras, cerraduras temporales que impidan su uso fuera de horario de apertura, seguridad privada...).

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 salientes, se hace fuerza para recogerlos, y puede hacer fuerza para abrir la compuerta.

Conclusión

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

Incluso si es modificada con componentes adicionales electrónicos, no se puede depender de éstos métodos mecánicos.

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 ésta forma.

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. Ésta 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 de la caja fuerte a través del cual introducir la clave.

Ambos componentes deben tener el 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, 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.

Diseño del sistema

En este sistema, cada dispositivo que se quiera conectar para pedir acceso tiene un ID. Este ID está asociado a cada usuario y no se puede 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.

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), se plantea calcular una función hash 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 detalle a continuación.

Criptografía

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. Uno de los principales problemas que introduce este tipo de criptografía es el intercambio inicial de claves, 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 intercambio en un medio público partiendo de unos parámetros secretos iniciales de manera sencilla.

Por ejemplo, Diffie–Hellman [11] [12] (muy común y extendido en el campo desde su nacimiento a finales 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.

Diffie-Hellman

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

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

También se eligen dos números que se encontrarán en público. 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.

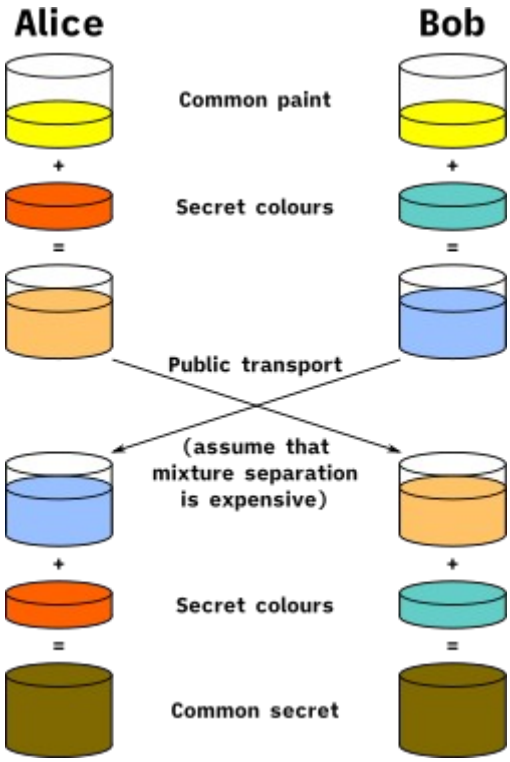


Ilustración 3: Esquema simplificado de cómo funciona Diffie-Hellman

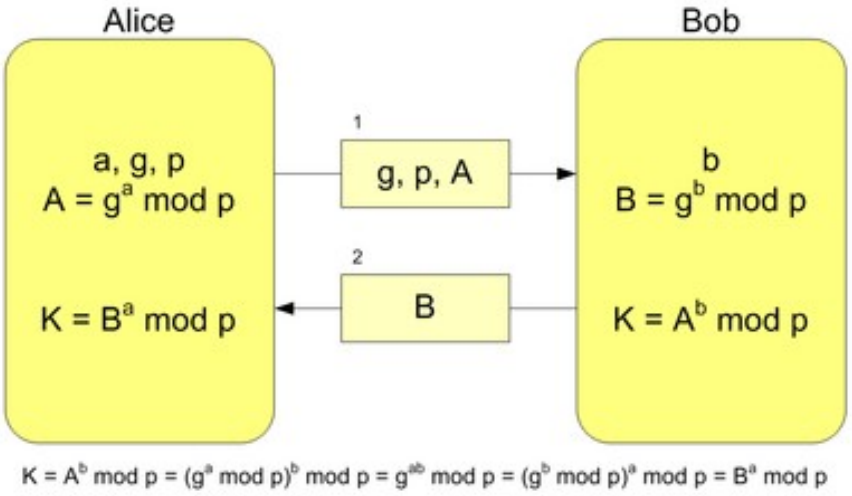
En detalle, los números públicos son un primo *p* y un generador de números *g* (enteros menores que *p* que son primos relativos de *p*). Son conocidos incluso por posibles adversarios.

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 mod p, y lo envía a Bob.

Bob hace lo mismo con b, y calcula B = g^b mod p y lo envía a Alice.

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

El tamaño de *p* suele ser de 2000 a 4000 bits, pero también son posibles otros valores mucho mayores.



A continuación se estudiarán HMAC y HOTP.

Especificaciones de HMAC/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 basado en HMAC para generar claves o contraseñas de un sólo uso, que ha sido muy usado en los últimos años para inicios de sesión.

HMAC

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.

En primer lugar, HMAC es un algoritmo para autenticación de mensajes usando funciones Hash criptográficas. Se puede usar con cualquier función criptográfica que en los últimos años se haya comprobado que sigue siendo robusta.

Los componentes necesarios para construir el 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 operan en bloques de 512-bit, 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 es la siguiente:

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

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

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).
2. XOR del paso 1 con el ipad.
3. "Append"/inserta 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"/inserta 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*. Las claves usadas, a su vez, tienen que ser elegidas al azar, de forma pseudo-aleatoria, 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.

HOTP

Sin embargo, el RFC de HMAC con su documentación es del año 1997, por lo que también se ha estudiado el funcionamiento de HOTP (HMAC-based One-time Password algorithm), sacado del RFC 4226 del año 2005, algo más reciente.

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:

1. Debe estar basado en un contador o una secuencia numérica creciente que conozcan el sistema llave T y el sistema de caja fuerte S, o que sea fácil de sincronizar tras cada uso.
2. El algoritmo debe ser económico a la hora de disponer del hardware necesario, como el uso batería y su consumo, el coste computacional...
3. Debe de poder trabajar sin ningún tipo de input físico, pero en dispositivos más avanzados puede usar teclados de tipo PIN.
4. 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.
5. Debe incluir mecanismos sencillos en caso de querer resincronizar el contador.
6. El secreto debe de ser de un tamaño **MÍNIMO** de de al menos 128 bits, siendo el **RECOMENDADO** de 160 bits.

Símbolos

- **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.
- **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 simétrica 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:

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

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 éste 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).

```
SHA-1 HMAC Bytes (Example)

-------------------------------------------------------------
| 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|
----------------------------------**********----------------++|
```



```
* The last byte (byte 19) has the hex value 0x5a.
* The value of the lower 4 bits is 0xa (the offset value).
* The offset value is byte 10 (0xa).
* The value of the 4 bytes starting at byte 10 is 0x50ef7f19,
  which is the dynamic binary code DBC1.
* The MSB of DBC1 is 0x50 so DBC2 = DBC1 = 0x50ef7f19 .
* HOTP = DBC2 modulo 10^6 = 872921.
```

El método de truncamiento (trucamiento dinámico), junto a un contador que aumenta y no es igual en cada uso, da lugar a salidas uniformes e independientemente distribuidas.

Éste RFC también ofrece una serie de requisitos que debe de cumplir un protocolo de comunicación para implementar HOTP, como:

- Autenticación de dos pasos entre algo que se tiene, y algo que se sabe.
- Un sistema para evitar ataques de fuerza bruta, como una ventana de accesos máxima que pueda dejar sin acceso al usuario tras varios intentos fallidos.

- Implementación sobre un canal seguro.

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, 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 **s** siguientes valores de ésta ventana. Si tras estos valores el valor sigue sin ser igual, se puede lockear 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.

De esta forma, sólo podemos sacar como **último** grupo el grupo 16-19. Ésto se puede ajustar para otras salidas con más grupos de Bytes.

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.

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 (Éste Hash también podría haberse cambiado, por lo que es útil calcular el HMAC, no sólo el Hash).

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 ésto 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. Es interesante comentar que no es imposible no encontrar colisiones con una función resistente a colisiones, sólo quiere decir que es muy difícil encontrar por medios tradiciones dos valores H(a) y H(b) que sean iguales.

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.

Acceso al Sistema

Los pasos para pedir acceso al sistema son los siguientes:

En el primer acceso al sistema

- Si es la primera vez que se intenta conectar al sistema, se comprueba el ID del dispositivo. Si tiene permiso, se genera una clave única mediante Diffie–Hellman, se almacena en el sistema para ese usuario, y se le distribuye la clave para iniciar el acceso.
- Se continúa con los siguientes pasos.

Siguientes accesos al sistema

- El usuario se conecta al sistema desde su el dispositivo eléctrico llave (que más tarde se diseñará) con el ID de su dispositivo.
- Se comprueba que el ID en cuestión tiene los permisos necesarios para acceder al sistema.
- El sistema genera un desafío **R**, un texto de longitud segura y que nunca se vaya a volver a generar. El sistema pasa a esperar una respuesta del usuario al desafío, una respuesta que puede ser calcular el HOTP del mensaje, con la fórmula que se mencionó anteriormente.
- El usuario responde al desafío. Si es igual a la respuesta del sistema central, da acceso al código de acceso/combinación de la caja.
- Después del paso anterior, se sobreescribe 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/cámara 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.

Seguridad asociada a la parte criptográfica

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.

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 [13] 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 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.

Teniendo un primo de 8192 bits en la implementación, se recomienda su uso habitual, a pesar de tardar más tiempo en calcularse que el resto.

Nuestra implementación cumple con los principios generales de DH, la clave final nunca se repite, siempre es diferente, puede utilizar primos de más de 2048 bits… Cabe mejorar en que podría usarse un algoritmo basado en curvas elípticas para generar primos, más recomendable que solamente primos estáticos. [15]

HMAC

Todas 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 de la propia función

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.

La implementación estándar usada en Python no tiene ningún defecto aparente hasta la fecha de hoy.

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

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 de sobra.

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.

Values	Probability that each appears as output
0,1,...,483647	2148/2^31 roughly equals to 1.00024045/10^6
483648,...,999999	2147/2^31 roughly equals to 0.99977478/10^6

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:

$$Sec = sv/10^{Digit}$$

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 de 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 implementar.

Componentes del sistema

Se pretenden emplear dos Arduinos, del mismo modelo, uno que actúe como llave, otro que actúe como sistema central de la caja fuerte.

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)[14].

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

Estudiado ésto, 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 puede no ser suficientemente segura, se establecerá una conexión entre Arduinos directamente por sus puertos [16].

Este primer prototipo 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.

Sistema Llave

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

- 1 x Arduino Uno, Nano...
- 1 x Módulo botón para encender o apagar la llave cuando se necesite .
- 1 x Batería li-ion recargable. Lo más apropiado es juntar 2 pilas de 9V, una para cada Arduino. Éstas 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...
- 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.

Sistema Caja Fuerte

Para el Arduino de la caja fuerte, se necesitaría:

- 1 x Arduino Uno, Nano... 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 [17].
- 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.

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

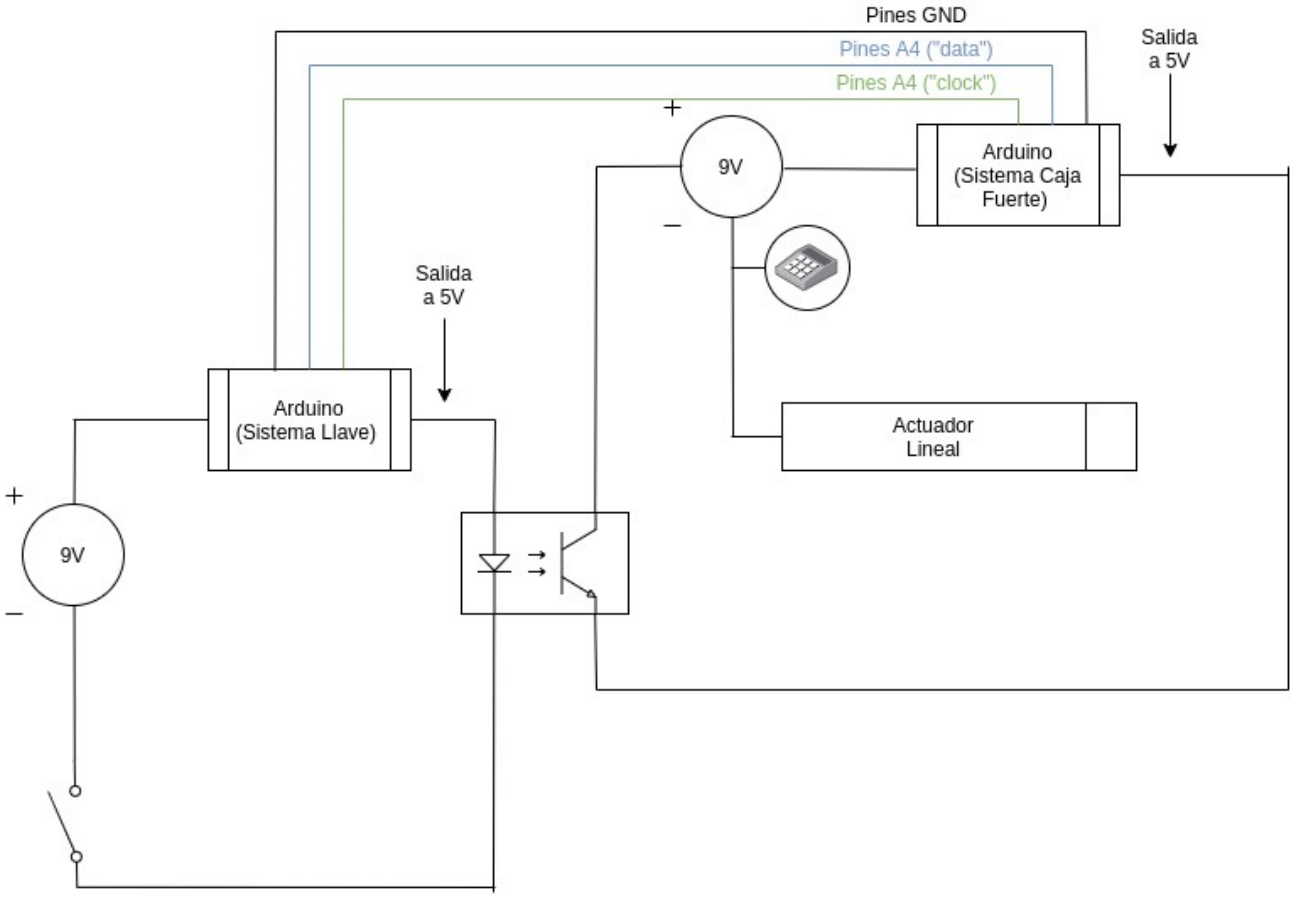


Ilustración 4: Esquema del circuito

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.

Una vista más en detalle de la conexión Arduino-Arduino sería la siguiente:

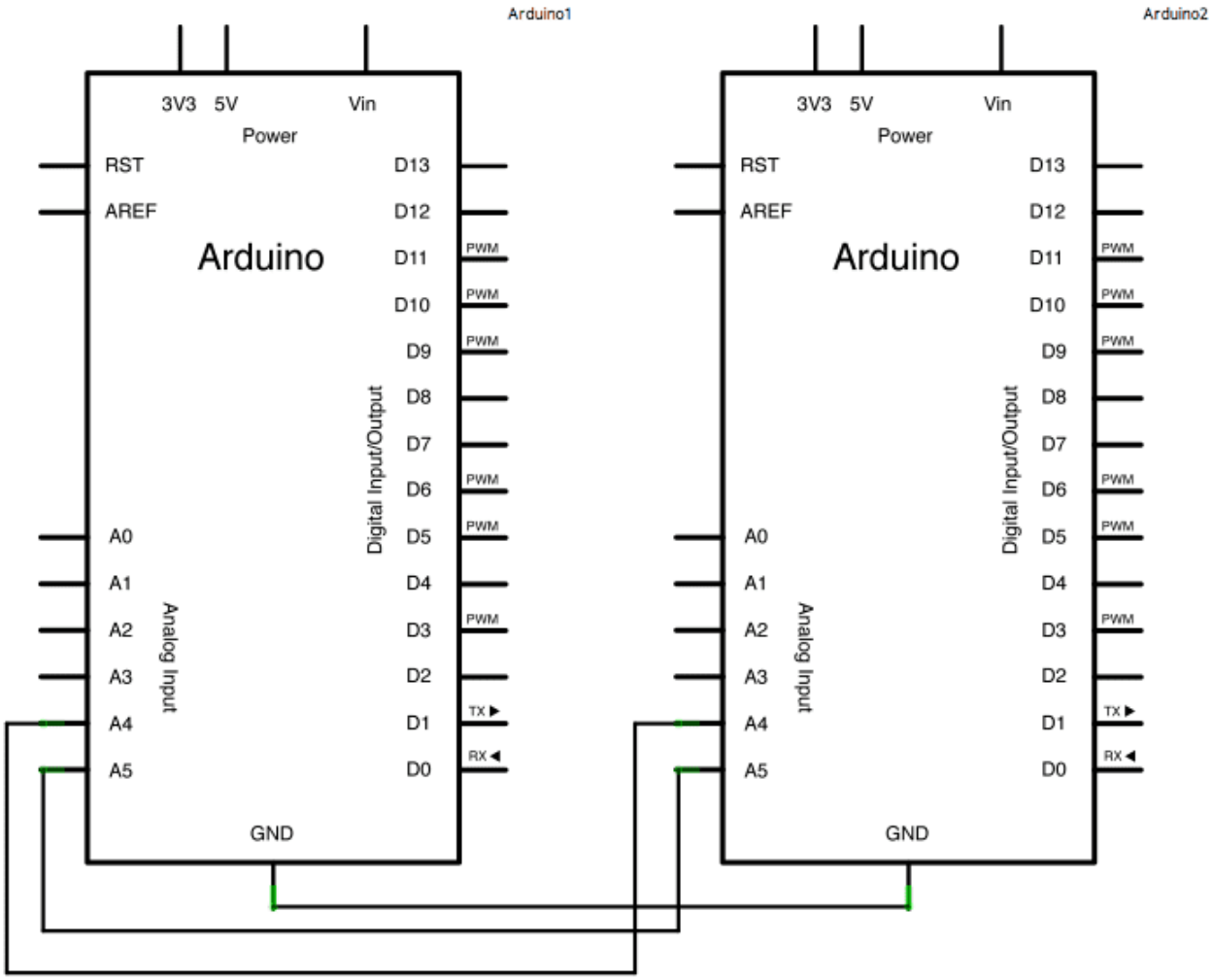


Ilustración 5: Detalle de la conexión. Fuente: https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter

Seguridad del circuito

Se recomienda incluir fusibles en cada circuito, tras cada batería. Así, si hay algún valor de las baterías que puedan causar una sobrecarga, los componentes seguirían intactos, y sólo habría que cambiar el fusible del circuito afectado.

En caso de querer reemplazar estos fusibles, debe de haber un mecanismo sencillo para hacer esto. El caso de la llave es simple, solo hace falta desmontar el aparato y cambiar las piezas necesarias (sólo el fusible en el mejor de los casos), todo esto por un profesional a cargo de ello y con seguridad de que nadie más accederá a la memoria del dispositivo.

Que éstos componentes sólo se encuentren antes de llegar a la caja fuerte sirve de ayuda, ya que, si estuvieran dentro, no quedaría otra opción que la de

abrirla por fuerza bruta con herramientas para cambiar las piezas afectadas. La idea de instalar una especie de puerta trasera mecánica (sin dependencia de la energía proporcionada) no encaja con el resto de medidas de seguridad especificadas hasta el momento.

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.

Implementación en código

Para la implementación inicial, el código 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 asociado [18], junto a los archivos que guardan las claves y mensajes que no se deben generar de nuevo.

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) ó 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 ^ valorPrivado % 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.

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:

$$A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$$

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.

La seguridad de ésta parte del sistema depende de que las claves privadas no se distribuyan. Si alguna se obtiene, pondría en peligro la integridad de las comunicaciones anteriores u 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.

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 = {
         [ÉSTOS VALORES DE MILES DE BITS NO ENTRAN EN WORD. 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, 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
        elemento generador es 2 puesto que:
        (g is often a small integer such as 2. Because of the random self-
reducibility of the discrete logarithm problem a small g is equally secure as any
other generator of the same group).
        """
        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)
```

Clase hash.py

En ésta 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 éste 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 que vuelve a comprobar si es o no 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, y lo usaremos como “mensaje” del 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.

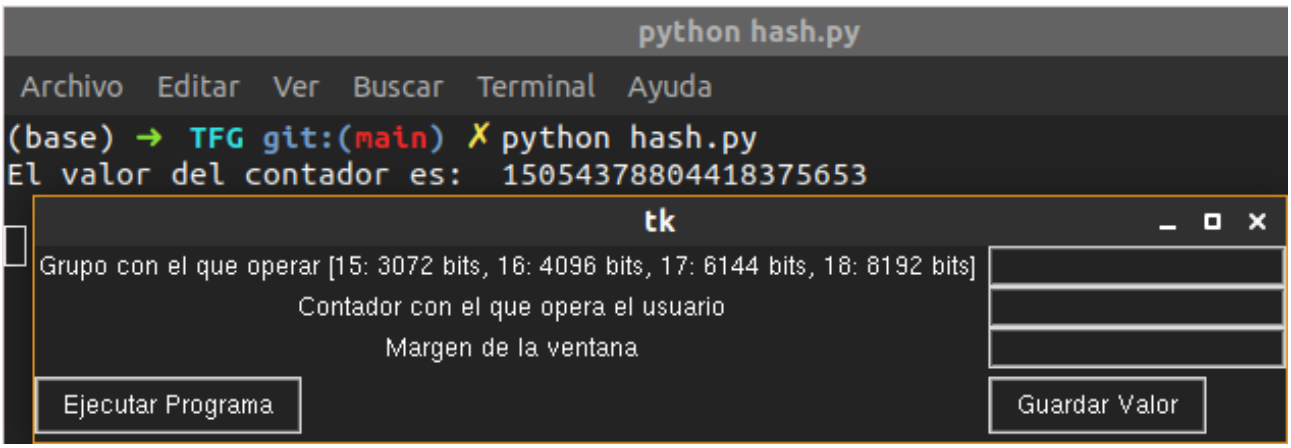


Ilustración 6: Interfaz sobre la que opera el usuario

(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, 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 cuando la salida de SHA-1 era de 20 Bytes, habrá que adaptarlo.

Queremos una salida de 4 Bytes aleatorios para calcular la clave que presentar al usuario, y que sea sencilla de leer para un usuario. Para ésto, como se explicó anteriormente, cogemos el Byte final del HMAC, y lo pasamos a binario. Esto nos dejaría con 8 bits, sobre los que se podrían elegir hasta 255 grupos, pero nos pasamos.

Con 4 bits podemos elegir hasta 16 grupos, no llegamos. Tampoco con 5 bits y 32, necesitamos 6 bits y 64 grupos, por lo que nos sobran 4 grupos, pero ésto se puede arreglar fácilmente.

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 éstos números, cogemos 8 por ejemplo, presentamos la contraseña que es igual para usuario y sistema.

Adicionalmente, si el contador del usuario está adelantado (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 3, el usuario el contador local 5, y la ventana es 10, calcula los 10 siguientes HOTP de cada valor de la ventana, acertando cuando el contador de la caja tiene el valor 5.

Funciones del archivo

establecerConexion(grupo): A partir de una cadena con el grupo que se quiere usar, devuelve un objeto de tipo 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)
```

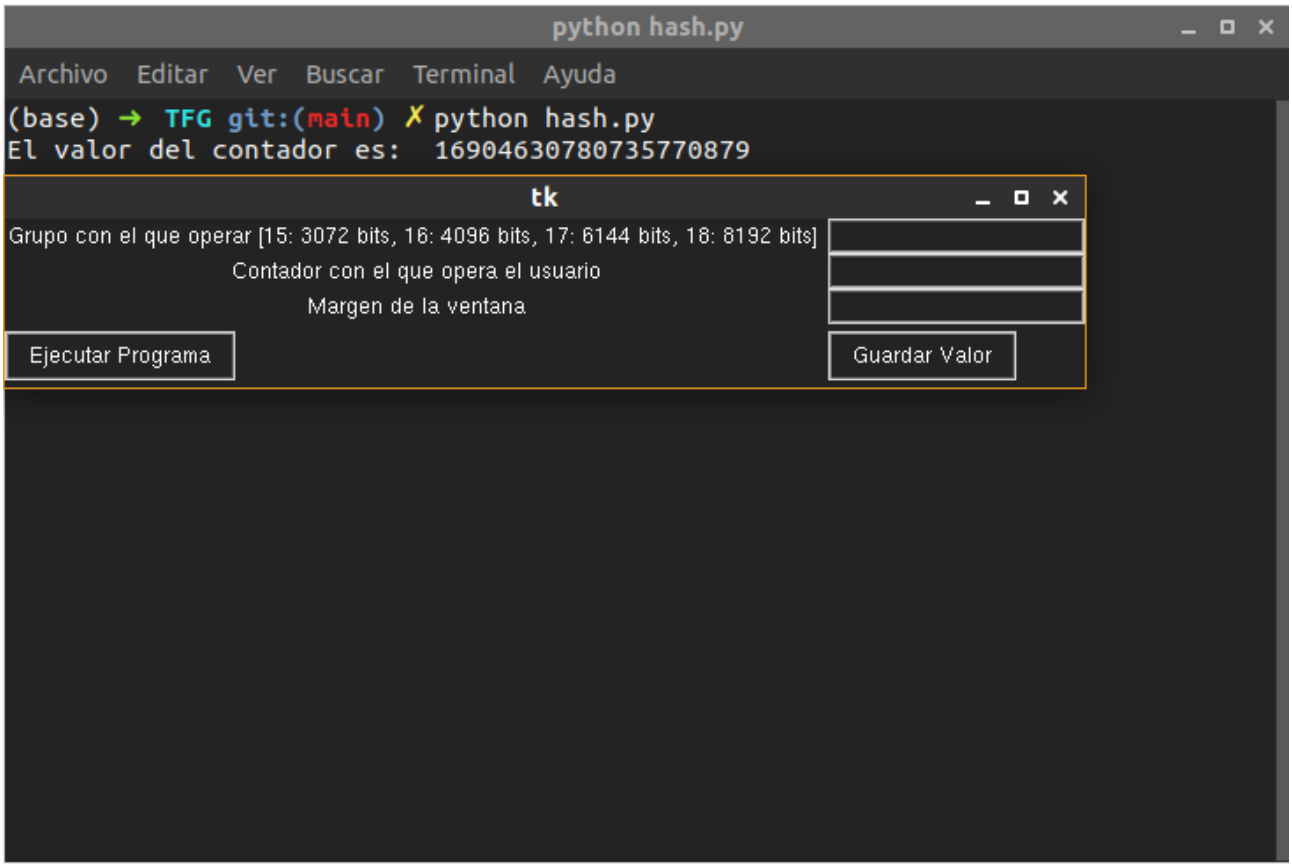
ImprimirPantallaGuardar(): Función auxiliar para obtener valores introducidos en la interfaz que usa el usuario.

```
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

Que tiene la siguiente apariencia:



```
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)

    global n
    global n1
    global n2

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

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

    Button(ventana, text='Ejecutar Programa',
command=ventana.quit).grid(row=3, column=0, sticky=W, pady=4)
    Button(ventana, text='Guardar Valor',
command=imprimirPantallaGuardar).grid(row=3, 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 y el mensaje o reto generado al azar. Utiliza la "library" 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.

Vamos a explicarlo 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, pasa. 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 de Diffie-Hellman.

A continuación, se coge el último Byte del conjunto, representado por dos caracteres. Estos dos caracteres se pasan a binario por separado y se juntan. Éste valor es en bits del conjunto final, no en Bytes, 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 (60 es el último grupo válido, porque toma 60, 61, 62 y 63. El grupo 61 tomaría 61, 62, 63 y 64, éste último no existente).

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 el usuario, la parte que sea.

```
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 recalcular 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 el valor del contador es menor que el de la caja fuerte, es un fallo automático. Si son iguales, calcula el HOTP y lo compara, y si es mayor 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 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")

    # 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)

    # solo el contador del usuario puede estar adelantado
    if contador_usuario > contador:
        # Dejamos el valor del usuario parado, aumentamos la caja
        valorHOTPusuario = calcularHOTP(contador_usuario, n, conexion)
        print("EL CONTADOR DEL USUARIO ESTÁ ADELANTADO. EL VALOR HOTP
DEL USUARIO ES",valorHOTPusuario)
        for i in range (1,ventana+1):
            print("Valor de la ventana",i)
            valorHOTPcaja = calcularHOTP(contador+i, n, conexion)
            if valorHOTPcaja == valorHOTPusuario:
                print("PARA EL CONTADOR DE LA CAJA",str(contador+i),"EL SISTEMA
ENCUENTRA EL MISMO VALOR HOTP")
                print("VALOR DEL USUARIO:",valorHOTPusuario)
                print("VALOR DE LA CAJA FUERTE:",valorHOTPcaja)
                return 0
        print("NO SE CONSIGUE GENERAR LA CONTRASEÑA")
        return -1
    # Los contadores son iguales, no hay problema
    elif contador_usuario < contador:
        print("CASO NO POSIBLE; ERROR")
        return -1
    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()
```

Pruebas

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

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 ésta 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 buenos.

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

Grupo 15

Mensaje:

14906391684844699610

Clave secreta:
187538402553488438567754057488138381806038145211953502216943031
678134059128890925631779502552997307073292963782309080991309307
219367126523079549805587308470180753352787867351111224591519307
174544714965871118077056750993144241593860146101575499799951149
701359806727516718787862530600138706214317805728718035866121367
473965093980215556616758741909153254312810962638748717799243797
987032053855849637839588354080083282874095466279930219083069380
756007314251844385550861504642731190809366218272955892930916017
113569872974741853607471032704594568738295066203913094755477730
379298341353664182892551035097562644615776525085512535944425553
199974812053651924250370283688747120859858294746873357366282481
121029634088403527897055920233960426478927866520085303204734662
065227637407743789836012132419125530759718443801730583515392152
129149419052035852922693008629251860271400055061618576286709319
118829456104359591810725563304392040218135 8

Resultado:

70f28d460d12490322919865ee3dab25fc558ac161fe232d2a4c1f77c70b1ff621
db18591f4472f8e23f7b1b2a3a1538c10bebbbb612dc206b80b2ff09cf71f6

Grupo 16

Mensaje:

14906391684844699610

Clave secreta:

81959933953197952991903375534751808408335781783896732151257485508750585401647580313030806918568896756987355499634303291709540684452217575887682138415141140509009899848006838110552570383414793666696895919318375261628338868357521591129615991434313436514990325836640248109331790417747261443063620390320827958291065708880196218395917127570319187491123086452621931864334867129031848460806159267493966064269009751066773899714061157618943270381147131806881599138845743462241734348166757233057077928916062042280972741517085396893377458812212829940778128074481345111165879433750596474478632192810046061036167453736233759025097353402013016075709740449719765481567848167265017143915609563869639930715711320339920395473299896054783230304220979495210023997300862262961361661372518879165968767555668208057293656765252249498748736083927718188936939466916499467652025163178566761977158541568618546065863737622638657383973568599289505245636969997596920470487877968238196262993418935617109112472595382075183451591380562723968961519218173720915242477396316895648599280905937576883812438231670703439035322109030368100814467594882733338038808472295565401734864272304069701799924815781699325768481664372607635343523404966767678330865422149256149175459941 4

Resultado:

99dae1d089cf7618fdbe825eded8d74b7d5c2596290fb8dd0f351aeadc5765f653
05061ff4662958424b5b1382d44d4a402baf2d8722eb7b33cae5427732b6ac

Grupo 17

Mensaje:

14906391684844699610

Clave secreta:

22946890134272630780177385643597323696957853782697812827488048961096665517509996553373995281602002227499133216638803837681085861229478252517865357031710522565550135716315134826025842931579330854380251646061794932807794763090942559003731203918110081190 3

0191649479026001837215550445905929619174739492976158730622270971427700574085055790661891852569083219626664102555353737324895293490407256037223879634323255168553874385411944195484627657739468840541749767156765637266946692046771369522655603316826333690069683492958546000625371716031037425129714911172541718383336462798892341247328269515877851442543221566779561911466446213127153840522783876453765811881182598379083213770110594863023368816137534789781368474440063442567351688340799897037110355535359686468172731618722629084071860672737278983421257190866571761597156184214925125322284694086548649203941177052335400430849575466394481973645249453611106176007617460743506905178498008600006613685169715876663925259769119587045483885195489214092977449099063143493493468842489585540903492268041467055659766724416484778817659582955101079007551512439391880455146212677155126737779518234194191375026762147522162657550921709214285270062043517847502874483885397664073517288545917035727870491502340018350782555627374161830324708457032370442681922803718975816077276653196628789031801324253375677791189589888346025238947194836859950870117204339735447724167897199820569561199995933686566915306043259626779013183176248548946872960067997179543367441110575402619181227996162053178584054059395534480732035411198810192908804098702607498760055113042184089552568652290572984039406737831316328354143992100893571485375979993630471406190067823659536434879578142817019981313577542278752620642021038707314116620823564994993222804958067965560546078962013402446546974133530241648679933999227793580726216839741227710939644901379687 77

Resultado:

19a00b800e0cc2847d17d0fc80f44ef9656b3df86a4c93c933a081a16b6eeb3b3fd4a88a8db47a42f5613862327870e6bb11496ce32fa8e37c932a2cc557516c

Grupo 18

Mensaje:

14906391684844699610

Clave secreta:

4395626548589672372863382595571601414418570743010459337771947905465421482029393567461990218725372119680477102262583088479020805891089719800186744207762842385103827726180182779331577244491343218871924142059511453584911881693999384067676466692523672803605299699368688726621799675919344146228969731368207320832609637766282190912905213454122926793982303467007244681998129605451691944073390635067460782646425116180486111184171949350654906328881457 45

295509409683241372600219493908925994780422169677684997188956447
641093507897427434531307913272219253727075516658012244256916380
739635105696267923629742570031757405051251563528545635216481545
557051496841432731560400949597206290221886933240535596762537783
530295070249507651130439505292144623583802232708972527166575969
215677567471144272464451146447757713217896437893792668771410604
721375463087288507921918086792214517026272545572898477954729496
678587368034057851396756196179415328529734221100907494786881777
153157303594600322946629024566251611744166069836244185520006364
993014827194907240437466445287020087801405350805985221356908017
836210572399780327689665358117783924305836423141632785732107845
712163596202054296380533356355048349519661720365351777577101513
155372231825164634696935300397157901478680402502807696254181528
475476895803903969637154880577125443245561334196488895851445816
177040343998109127528169349788545152926511250823280098157381136
725725299610560988641436601512024674621761309916500810940149776
404704738432256879319615931539072518967043619187186801035344861
942724751250768423138965750173226420820855404668338468170420224
078161695692377923908649786032557423104497692555772683347609825
113729206941080162282763338126499487299617696761160915777716984
496721893315552827301193789348747923658963473606317280545894987
253756550057406014361603336626794829930070628121277120706143816
611417293436959934943001921834427034083730736162242646584634249
300280781895546953458328941517197599195921606419746951988373442
509672938396764958713751104084682316475038389061740044473260176
350124316099668552353391721619499228997017778905605056159851003
127377915920551588948679939037232394083328409194365966607657375
740315989374608948289682556597962658663069770845320812040746030
199870813666382570700765147462796053370854375806998011272628704
057995188562366756757510032005255401058161034884042354905535532
467261117158774412277482345724923557606553226249341751463664443
074155483907530908658870174589199600561806519442687686194294431
417207470

Resultado:

4737e7ffce3c906f7a94593f259f82cdf98610a46b5030e72c7578cae026ce960a
546f107239c3d490bd508bfa2ef727c4de0137fc6537e761a3a1e5f50f3323

Pruebas avanzadas

Tras implementar la parte de aleatoriedad para que no se generen dos mensajes iguales 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.

Prueba 1

Para la ejecución actual, se obtiene un contador de la caja fuerte con un valor de 7910396204122684619. 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:
22462956782817433604260400411523661112381634408551837870395949045556469476614335970454823771413109765502687821714484554356227848128769294405576522479035630112220061611427509724015539723699386520853574226101747800554407362822586028164734487371243628235715253659389673109858962260810621035070429128204351576373105292842926957888865163481343488719091611227206761961852836071473485099776729960845119661435322839069261713631562601618040857388651253272107780315114708185590304559424114887356184566481130710341798439507325723610002587629477548864204909418606663839041105743943939791551392889374391563952433530551446276512264034193153360683877697002114355360748855098357663508367682290733623097639426775367014786052950193251947379266231678939077258411853698252776829572710327660776462085175429238244226863173771631091337284666539986166269234742795658138136882126238652544126719069424669070105835945942159770565848374698110483222188 0

Valor privado de b:
48136611804479334256905809375188542954258135454407132306317500256636349618583087158158595353025160531439380977254323584661192639694667809871654143380615102472381606629729047483917900531263352338061218247390852387807263287998886702729164640793245434491300414665452479764659753887405996786030550897455248340342936538792361379294909119271121151461871096493207631624728064866811605366195363446713180628207783505725186210362193378651028555292569186211309017940809614893869574837662626274371356264269203374725478962618904524035280584575140996095453904376376483926461276822367074569379777002099367659095248065581029290223188569494938981238389309207077095841714976405914323925569331667725078156974101179326184939675875817734430206636047362926872894404566262085763082709871644493506281198965871599397241545227714461292000962683765331932088521205631350799390200955534125706962848214991645928950351834800273271561161906584259343958561760

1

Valor público de a:
312912031898168046740037845856331709409847418414061341838223409
124100202597282804142046830266317127428228160339796711015866432
250926412568741282375901271921057593805953591706504549885158708
823443053315973437880766735785585492306361550083480120757043319
315026896340029604750411802510075697773430967528566866123538859
653380256131746050956970044267188621097553158457054361787301753
685213238504243174155605587889178614653689238238079076288657892
244633141423862025878461226834030132135723953019350699077767846
623004820463394683776480076430280794649624868368833417120518080
365649640976415002258412734175771101397751853633405474601728110
886353288585556012429732154631649779529467835438119245946391220
519372520361868913271178644188501003618389493250974763837359397
331662348895361469145663860383936317867895972372551094632623818
238320386224491806513705708362623059714229426151023123198315478
308444813427063516810578646834857472364880 0

Valor público de b:
193164956603585142229534672568860906815611918800111536207502554
627994139651772013576996084158973823068076714424820779501495459
684432891216869302192585823915470551465096688892546022577823669
731200922619665777939161601024090842200887450349598230614450822
900291489389128468907333051690030952523624186469485982629134919
373218859485871081233293045494431696046307859209023803530679891
809435227995826627704218159077464339640631687502289279551052174
062808405352050651427922207978950934086774959035118123611447 85
935687807570167910124955345529978583582937924482008775121300778
879428882919386929153747339135695386702612279999714085367379363
648393499771768555813618460296897291857736230004907204207356819
054676423872586133455424964490453616475566541633494240771924138
172572636866695546185550496778807162214687565468691043203012779
617766562538480484954179961904254021252435503643900482558039122
109215599514017942861518242864230025605 2534

Valor de la clave para a:
173612398381204929969401281842842363248748896435113452038365783
372813617031811887113961127711809502748724176375608410514666978
269755681192223401215278185198044064010999721621214790013079118
112636802812962848831675689702759145085703906763093863074231736
812874233869149633955646962708810079963290097148793249585918227

341115580750405433161029319356200083245658472087371637083640731
974999396245223789739247338519112030437649007807812896665503452
634274802416745082129349201166884718157763616017137033352432607
574617756714289159650956081552447387546431154795447950168022805
071642343162362785338409454671101467190295237821779694741499817
444214103382222045787150238853216147609107255887850554383657480
349448419488219074885402496250678221358689751910396942024045091
684542164185123804281504227935827553257260770754666017394765727
659473663990181152533470952961572677242153016646508871204601149
989399152750407108286766427169664876037538 8

Valor de la clave para b:
173612398381204929969401281842842363248748896435113452038365783
372813617031811887113961127711809502748724176375608410514666978
269755681192223401215278185198044064010999721621214790013079118
112636802812962848831675689702759145085703906763093863074231736
812874233869149633955646962708810079963290097148793249585918227
341115580750405433161029319356200083245658472087371637083640731
974999396245223789739247338519112030437649007807812896665503452
634274802416745082129349201166884718157763616017137033352432607
574617756714289159650956081552447387546431154795447950168022805
071642343162362785338409454671101467190295237821779694741499817
444214103382222045787150238853216147609107255887850554383657480
349448419488219074885402496250678221358689751910396942024045091
684542164185123804281504227935827553257260770754666017394765727
659473663990181152533470952961572677242153016646508871204601149
989399152750407108286766427169664876037538 8

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 sistem de llave del usuario:

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
f3e8197efe44cf2eaa5cca5a84ae20ddc2d067795eef474e3a0c3a4260d4286b47
4a6449da25163d1816e9c80243f93a850cee246c4ff001ce856a596dafeb5a

Último Byte del HMAC: 5a

Representacion binaria de 5 : 0b101

Representacion binaria de a : 0b1010

Binario del último Byte: 1011010

Valor en decimal del grupo de BITS resultante: 90

Se elige el grupo de Bytes 45

4 Bytes que nos salen: 43f93a85

Resultado decimal de los Bytes obtenidos: 1140406917

Últimos 8 dígitos de la contraseña: 11404069

Los contadores de ambos son iguales, calculando HOTP para ambos

Finalmente (recortando la parte en la que se recalcula Diffie-Hellman, HMAC y HOTP para la llave), vemos que los valores de la contraseña son iguales, y se puede garantizar el acceso:

Los valores coinciden, se da acceso

Prueba 2

Para el siguiente caso, el contador del usuario estará adelantado con respecto al de la caja fuerte, pero está dentro de la ventana de sincronización. La caja tendrá un contador con el valor de 11316404850165261866, 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:
20581778135358485115311926117298328935797312215979835894278132 2

88826524196448731497664274030815251007996378862077320748650936078998198519183267057447155689030684251694646410521434336606170360154047602543641114455989051537887437320307723775924699824153140350421232974408322917286702538874966274676227236429135864367616679502129702109013154112386976090191810233811766788013732650580641289731315250418142786010262055440310943218580693971795581022603063339787401991063721247147333174362825028598469847906924591157538953169731467465645243171839390591944248106961585790007135712257617177900514680290174115432128898798352860858659547839162434136644399364376200280426422099050085889042141818982547944276271411576506409202230430700854578086272373003843937779862287487223842287856367154715006655595054711817864177824178169385098361927170071853871316535098446564202559527226072229097298513882968312430142419015997235134578455631938067 66

Valor privado de b:
26240048660683613200168249022822964728319526242379437472561431506707775163547129475075605980031556722092434697765754708947608140359786468390481423992392901141255070998510266554343017470876417355217314420043334889170947837443689208526597490841611195442633769720674995076297025042809051764304892020523831495167278083340676463888037338333530763442533418302476798568709855008219180399960028326268588784925755274428254931375937333967995416663292203862201698955889875762600972999037645904294116363881182778036494984697700756645587113612118102404976410559181352722873853637367851025324766045984885686471013233966947986187490611946414558549604502163421466483680845002293314255006369704380690187427346811425378833059371549637154613050735530121662732078350582173254198441615081650505814480193285414473626371796386267423518357982746293738809893469483450785338601839108854773294893722198673485426544540001568423234885868178718639801047 34

Valor público de a:
15897493799393116164006980630637870372297565648772992007390169226451213035876803780974636786544431096922589318987821700979106547836568710350479037500627624036899414899070187538978744618810335417925682413439646598504426843292321310084369574328068547495814369691261373356821266240286283994645342973609612319958603827737432980347672454272228646925079206745155684163002267234237860920717736776152352228851948861246974881135026640411376400842107626808774006087958911641090921529643555945493454667686250092748 3

950600180863307591677809077994068655962084602930445206476733068
808207521823089197136666209159470978870623645179020612807400861
350718242825724507563709042720961555295254366736058258148643825
043236423348636179541887718266523656526124312745756396525436886
909024288077578124781967987557568356796144551618716277809499784
788073158066759623200695610736312593765714959644302946621302943
977970007350070914808437385025530401127304

Valor público de b:
540491022209181244546588359935127911015289005288957293434537174
697419052826981361612102155547282488288883010680230050333231255
065982117902984401301981094333590088968290171694712211009742380
054052767576035798364823414978259113640708010081820852313102374
146305262274298709248765374727283544903291255033280807017549686
433057799452412197663911540025092485930089425000545200622149102
448974861114064405393580410168373139130764528120786266411800504
427508381806066382339139528939837755084679071304758810525158341
113889758099958184245295709241080072590578392646219412017433212
819216308608780045351570826349777949662678428466956241855230036
927544193783438186603096078147775258228885295438250045922646894
339527529835967147738161504436705840787532501427300515648139839
921841105296441831038411324126401253760831498169569311811561425
580273951539891887613766612613799054767912739831703696867595842
308400101901255757868587567837292190086601 3

Valor de la clave para a:
150912455811774692266934930652244268373376150707980689060108776
104692246069515868031712455370171704104453300292035350709714208
494730897088531747654401952079156558321136768858245451997403782
137972899226342157851879185642330984983633716963638427408359196
855859272108920534455576977942646936418332455052256571255915776
905471516123318269609926192054606657693808717271017991035436478
301692036849401063559424582298976412962128607456888411090672609
860016786363485648237539852565595822610903647883831815083319403
035578355135231610698952301367230628489154618583023434524040482
023039148302446382502051238317936988366347973023566685980182675
092237455200881097150009117190652246823156081808745519610512471
345743143049693621637371368944008340116432154741328381825002526
025267905483874655099134407317437916592778827999921017249970851
346002632139490846550917670812051984090170543852495132714895039
710831922330655478930602760189500247329981 5

Valor de la clave para b:
150912455811774692266934930652244268373376150707980689060108776
104692246069515868031712455370171704104453300292035350709714208
494730897088531747654401952079156558321136768858245451997403782
137972899226342157851879185642330984983633716963638427408359196
855859272108920534455576977942646936418332455052256571255915776
905471516123318269609926192054606657693808717271017991035436478
301692036849401063559424582298976412962128607456888411090672609
860016786363485648237539852565595822610903647883831815083319403
035578355135231610698952301367230628489154618583023434524040482
023039148302446382502051238317936988366347973023566685980182675
092237455200881097150009117190652246823156081808745519610512471
345743143049693621637371368944008340116432154741328381825002526
025267905483874655099134407317437916592778827999921017249970851
346002632139490846550917670812051984090170543852495132714895039
710831922330655478930602760189500247329981 5

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:
555bda4167cbc7333650443c88c83e7e23e6ac83ad1e8a6223686c455f8fc98c2
e13204ecdb5351f64bf757fc07dea7e11f912ce56281efbdecea17a15b146c9

Último Byte del HMAC: c9

Representacion binaria de c : 0b1100

Representacion binaria de 9 : 0b1001

Binario del último Byte: 11001001

Valor en decimal del grupo de BITS resultante: 200

El grupo 100 se encuentra fuera del rango de Bytes, que tiene como máximo del 60 al 63

Se elige el grupo de Bytes 40

4 Bytes que nos salen: 64bf757f

Resultado decimal de los Bytes obtenidos: 1690269055

Últimos 8 dígitos de la contraseña: 16902690

Como el valor del usuario está adelantado 10 valores, 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:

EL CONTADOR DEL USUARIO ESTÁ ADELANTADO. EL VALOR HOTP DEL USUARIO ES 23207918

[...]

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
1c19daaeedb32dae09da2cbf2c7d0c7ccba3affbeb176ec97941718a547908984a738562f91d8a458a3b0751a3de422472ae8606855fa51577306f738b961e66

Último Byte del HMAC: 66

Representacion binaria de 6 : 0b110

Representacion binaria de 6 : 0b110

Binario del último Byte: 110110

Valor en decimal del grupo de BITS resultante: 54

Se elige el grupo de Bytes 27

4 Bytes que nos salen: 8a547908

Resultado decimal de los Bytes obtenidos: 2320791816

Últimos 8 dígitos de la contraseña: 23207918

PARA EL CONTADOR DE LA CAJA 11316404850165261876 EL SISTEMA ENCUENTRA EL MISMO VALOR HOTP

VALOR DEL USUARIO: 23207918

VALOR DE LA CAJA FUERTE: 23207918

Prueba 3

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

La caja tendrá el valor de contador 17224576119183002985, el usuario 17224576119183002990 (5 valores adicionales), 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 siguientes 2 valores (la ventana), y como el usuario está adelantando 5 valores, hay un error.

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
a637ca766a28a0f6aa6eb1adca6f5d6d0d425511f6db24de7e864b85a095365c1
3b8bf317e7d4f5ee7b20cd3433a915417dbe0b1a7c109411f7aa36031d2c493

Último Byte del HMAC: 93

Representacion binaria de 9 : 0b1001

Representacion binaria de 3 : 0b11

Binario del último Byte: 100111

Valor en decimal del grupo de BITS resultante: 38

Se elige el grupo de Bytes 19

4 Bytes que nos salen: 11f6db24

Resultado decimal de los Bytes obtenidos: 301390628

Últimos 8 dígitos de la contraseña: 30139062

NO SE CONSIGUE GENERAR LA CONTRASEÑA

En caso de que la clave del usuario sea menor (caso imposible, nunca debería ser menor), se da un error y termina la ejecución.

Bibliografía

[1] Cerradura inteligente, https://en.wikipedia.org/wiki/Electronic_lock

[2] Google Authenticator, https://es.wikipedia.org/wiki/Google_Authenticator

[3] Autenticación de múltiples factores, https://es.wikipedia.org/wiki/Autenticaci%C3%B3n_de_m%C3%BAltiples_factores

[4] RFC 4226, https://tools.ietf.org/html/rfc4226

[5] RFC 6238, https://tools.ietf.org/html/rfc6238

[6] RFC 2104, https://tools.ietf.org/html/rfc2104

[7] RFC 4868, https://tools.ietf.org/html/rfc4868

[8] Caja de caudales, definición de la RAE, https://dle.rae.es/caja#2s7HuNa

[9] 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

[10] Información en detalle sobre las cerraduras, https://www.locks.ru/germ/informat/schlagehistory.htm

[11] Intercambio mediante Diffie-Hellman, https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[12] RFC 2631, https://tools.ietf.org/html/rfc2631

[13] RFC 3526, https://tools.ietf.org/html/rfc3526

[14] Especificaciones de Arduino, https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD

[15] Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice, https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf

[16] Ejemplo de comunicación entre Arduinos, https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter

[17] Ejemplo de activación de actuador lineal, https://www.firgelliauto.com/blogs/tutorials/how-do-you-control-a-linear-actuator-with-an-arduino

[18] Repositorio de Github asociado al trabajo, https://github.com/PCMSec/TFG

Anexo A: Componentes

Lista de componentes que se necesitarían para montar el sistema de forma físico.

Arduino:

[https://store.arduino.cc/arduino-genuino/boards-modules]

Llave:

[https://www.amazon.com/Gikfun-12x12x7-3-Tactile-Momentary-Arduino/dp/B01E38OS7K/ref=sr_1_3?dchild=1&keywords=arduino+buttons&qid=1602004206&sr=8-3]

[https://www.amazon.com/BONAI-Rechargeable-Batteries-Ultra-Efficient-Lithium-ion/dp/B0718WPPN8]

[https://www.amazon.es/AZDelivery-Display-Pantalla-Caracteres-Arduino/dp/B07DDKBCY7/ref=sr_1_9?dchild=1&keywords=Arduino+Led+Display&qid=1602413015&sr=8-9]

Caja fuerte:

[https://www.actuonix.com/Arduino-Linear-Actuators-s/1954.htm]

[https://www.amazon.es/Teclado-Interruptor-Membrana-Teclas-Arduino/dp/B018CGKAYY/ref=asc_df_B018CGKAYY/?tag=googshopes-21&linkCode=df0&hvadid=170884034566&hvpos=&hvnetw=g&hvrand=3258163074243580011&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9061041&hvtargid=pla-219554677732&psc=1].