BORRADOR

3/12/20

Cerradura Digital de alta seguridad para cajas fuertes

Pablo Castillo Martínez
Tutor: Jorge Dávila Muro

Índice

Resumen

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

Desde mejores materiales, a mejor lógica interna, a funciones especializadas (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 la criptografía y la seguridad modernas para la creación de una cerradura digital de alta seguridad.

Abstract

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

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

Other areas that have also evolved are 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 and Security 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 eléctrica para evitar un forzado tradicional, a la tecnología de lector de tarjetas o claves, sensores de biometría... para probar nuestra identidad, las cerraduras han avanzado a pasos agigantados.

Pero igual que ha avanzado la tecnología para darnos seguridad de la mano de la comodidad (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 nacionales e internacionales.

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 este 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 por la que se accede suele ser del mismo material que el exterior, dejando fuera del alcance de un atacante cualquier posible punto de acceso, 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.

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

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 las partes interesadas [10].

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

Para abrir ésta cerradura, es necesario tener una llave, un trozo de algún material duradero cuyo perfil ha sido alterado y alterado 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, ejercer movimiento con la llave y poder abrir la cerradura.

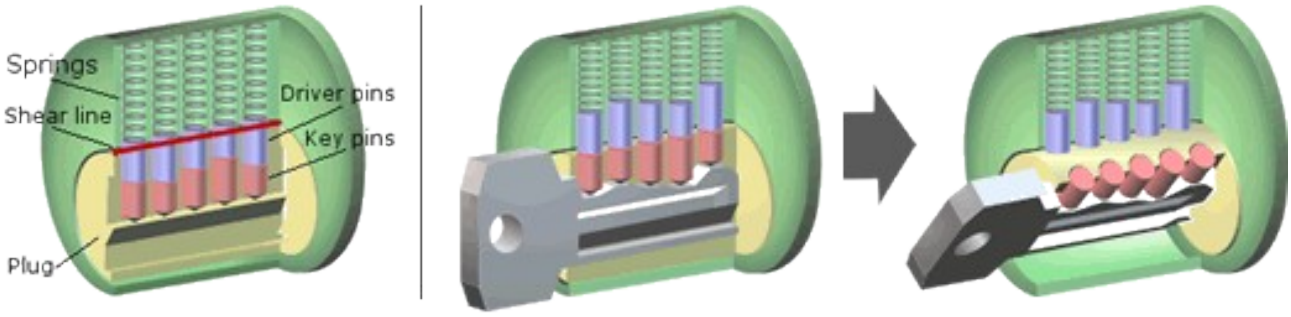


Ilustración 1: Fuente: www.safe.co.uk

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

Un método que no necesita de ningún token físico es la cerradura de combinación, que consiste en una línea giratoria externa que se conecta a unos discos interiores (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.

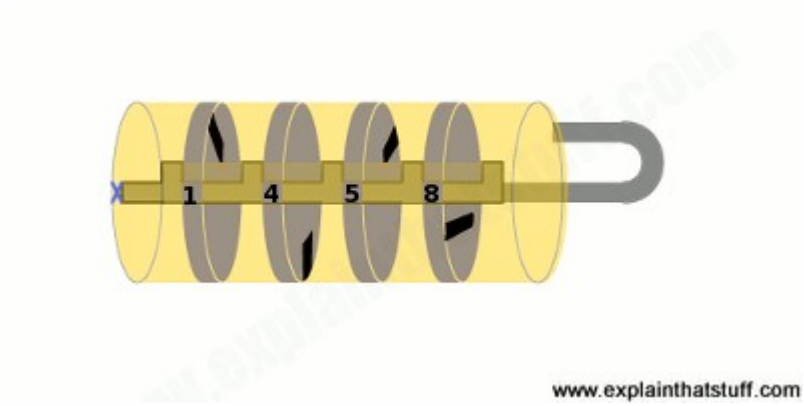


Ilustración 2: Fuente: https://www.explainthatstuff.com/yalelock.html

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).

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

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 de éstos métodos de cierre, para aumentar su seguridad.

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

Es por éstos motivos por los que añadir un generador de secuencias aleatorias, usadas una sola vez en la vida del sistema, puede ayudar a crear combinaciones más seguras de un 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 en una caja/cámara es similar al de una cerradura tradicional en una casa moderna, sólo que, en lugar de haber un número bajo de "salientes", hay hasta 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 la mayoría de éstas 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…) que no se tratarán de manera directa en este trabajo.

Cuando la combinación introducida en el sistema es correcta y los agujeros de los discos están alineados, el bloqueo cesa en la manivela externa que no dejaba contraer los salientes, se hace fuerza para recogerlos, y puede hacer fuerza para abrir la compuerta.

Conclusión sobre el mecanismo

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

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 (Por no ser compatible y por posibles vulnerabilidades previas).

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 dentro de un fichero de configuración.

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

En nuestro caso (y en el recomendado de forma oficial), se plantea calcular una función hash 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 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. Aunque nosotros no cifremos nada, y sólo calculemos el hash, es interesante conocer ésto para el intercambio de claves secretas.

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 exista atacante esté escuchando el medio, no conseguirá ni la clave final, ni tampoco computar la clave a través de los mensajes intercambiados 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, definidos en el estándar. 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.

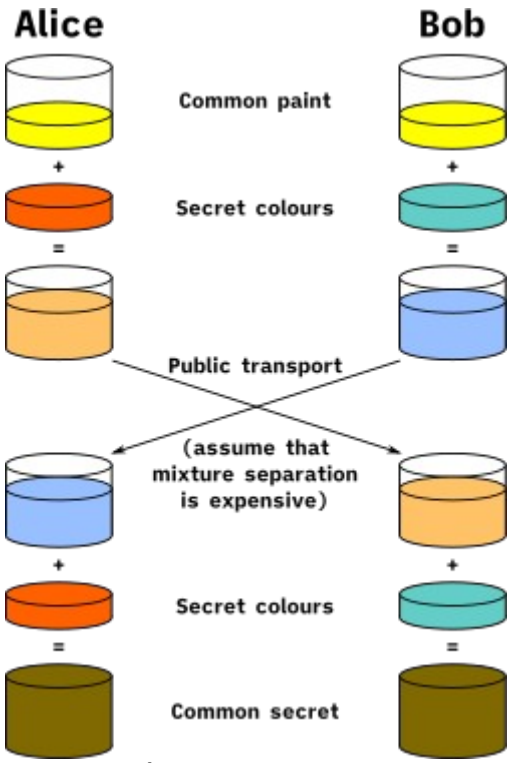


Ilustración 3: Esquema simplificado de cómo funciona Diffie-Hellman

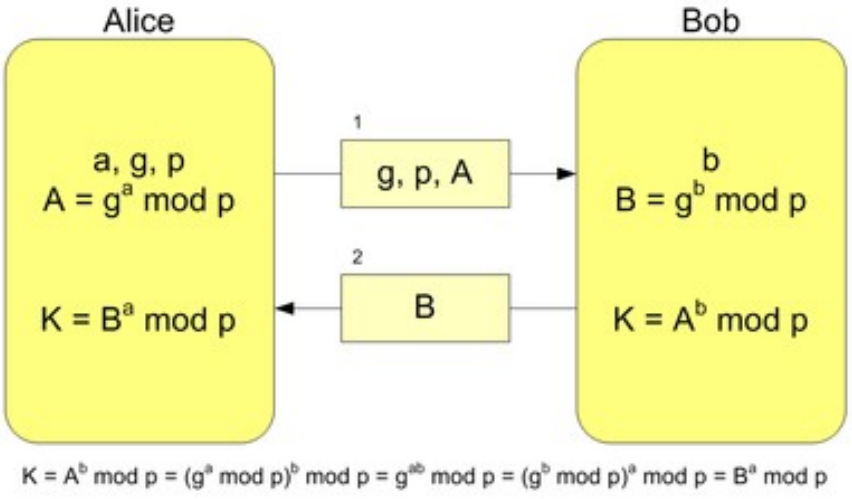
En detalle, los números públicos son un primo *p* y un generador de números *g* (enteros menores que *p* que son primos relativos de *p*). Son conocidos incluso por posibles adversarios.

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 en compañías como Google.

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 Hash 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). Si es mayor, se hace el hash de K y se opera con él.
2. XOR del paso 1 con el ipad.
3. "Append"/insertar al string de texto junto al resultado del paso 2.
4. Aplicar *H* a la salida del paso 3.
5. XOR del paso 1 con el *opad*.
6. "Append"/insertar la salida *H* del paso 4 a la salida del paso 5.
7. Aplicar *H* a la salida anterior y devolver el resultado.

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

Las claves usadas, a su vez, tienen que ser elegidas al azar, de forma 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

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 en su RFC:

1. Debe estar basado en un contador creciente o una secuencia numérica común entre el sistema llave T y el sistema de caja fuerte S, o que sea fácil de sincronizar en cada uso.
2. El algoritmo debe ser económico a la hora de disponer del hardware necesario, como el uso de la batería y su consumo, el coste de su funcionamiento...
3. Debe de poder trabajar sin ningún tipo de teclado físico, pero en dispositivos más avanzados puede usar teclados de tipo PIN (como nuestro caso).
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, por ejemplo, 8 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).

En éste sistema, con SHA-1, sólo podemos sacar como **último** grupo el conjunto [15-18]. Ésto se puede ajustar para otras salidas con el número de Bytes que se necesiten.

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, truncamiento dinámico, junto a un contador que aumenta, o al menos no es igual en cada uso, da lugar a salidas uniformes e independientemente distribuidas en cada uso del sistema.

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.

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 de un mensaje, 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 importante comentar que no es totalmente 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, o sacar un “exploit” para que un documento tenga el mismo hash de otro mensaje.

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

Acceso al Sistema

Los pasos para pedir acceso al sistema son los siguientes:

- El usuario se conecta al sistema desde su el dispositivo eléctrico llave (más tarde se comentará el diseño) 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 (8 Bytes) 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/puerta 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 se nos ofrecen.

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

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], pero nuestra elección sigue siendo una opción válida.

Una de sus vulnerabilidades conocidas y que merece la pena mencionar es la basada en un ataque “man in the middle”, u hombre en el medio. El atacante se situaría entre las partes que se intentan comunicar y acuerda una clave simétrica con cada parte pasándose por la contraria. A partir de entonces sería posible comunicarse con cada parte de forma individual, descifrar las comunicaciones, y comunicarse con la otra parte. Una forma sencilla de remediar ésta vulnerabilidad es mediante una firma digital que pruebe la identidad de cada parte.

HMAC

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

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

Además, HMAC no es vulnerable a ataques por extensión como un hash normal.

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 tener en cuenta en un proyecto como éste.

Componentes del sistema

Se pretenden emplear, de forma teórica, dos Arduinos o microncontroladores, del mismo modelo, uno que actúe como dispositivo de tipo 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 Microcontrolador (Arduino Uno, Nano, u otro)... Con el que realizar las operaciones
- 1 x Módulo botón para encender o apagar la llave cuando se necesite .

- 2 x 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...
- (Opcional) 1 x Carcasa donde introducir el modelo de Arduino y sus componentes, con una apertura para quitar e introducir la batería y cargarla.

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

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

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

Sistema Caja Fuerte

Para el Arduino de la caja fuerte, se necesitaría:

- 1 x Microcontrolador (Arduino Uno, Nano, u otro)... Igual que el de la llave.
- 1 x Unidad de almacenamiento extra (tarjeta SD), como espacio adicional permanente donde guardar información sobre claves, mensajes enviados...
- 1 x Actuador lineal con el que bloquear la compuerta de la caja fuerte [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:

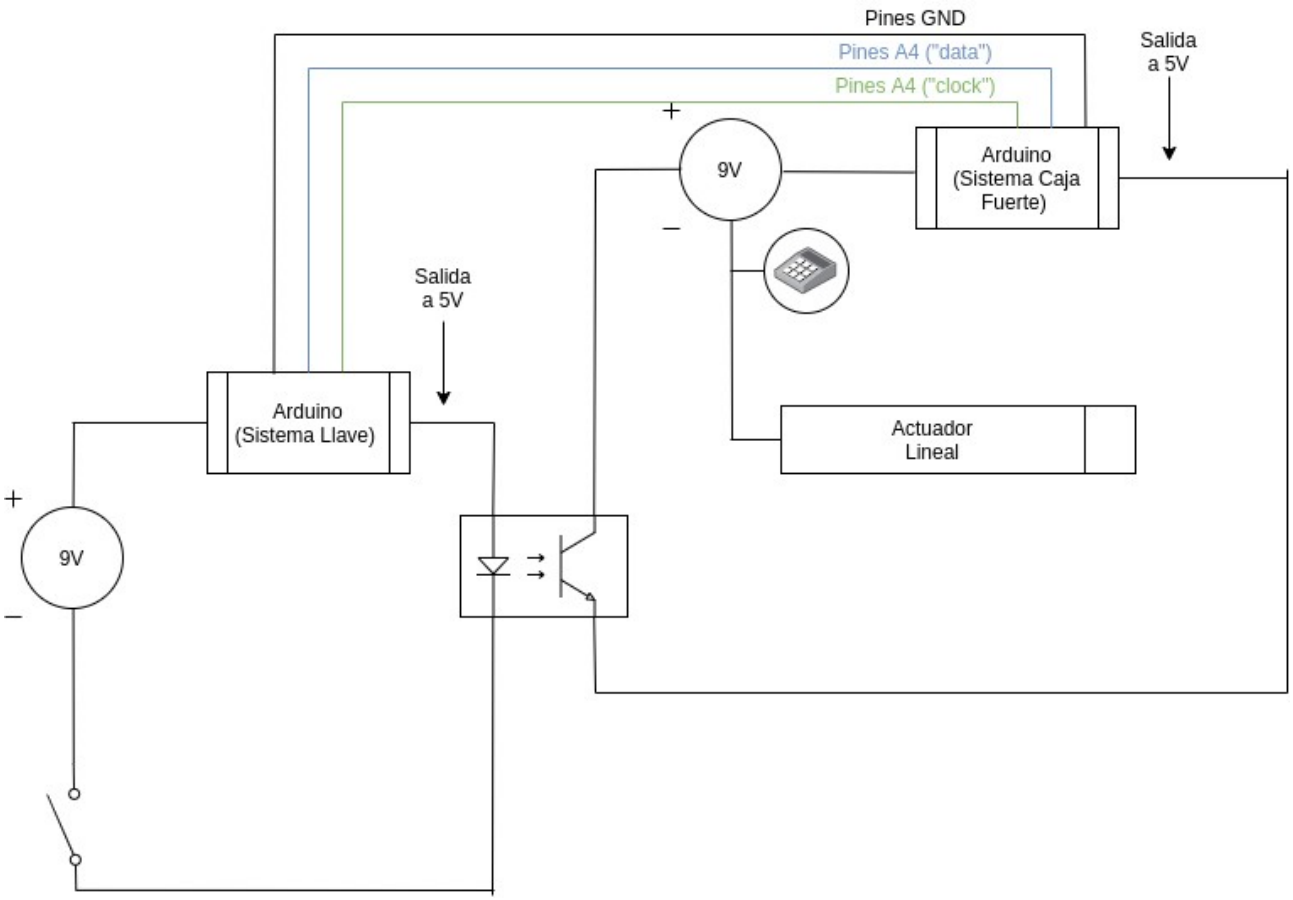


Ilustración 4: Esquema del circuito

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:

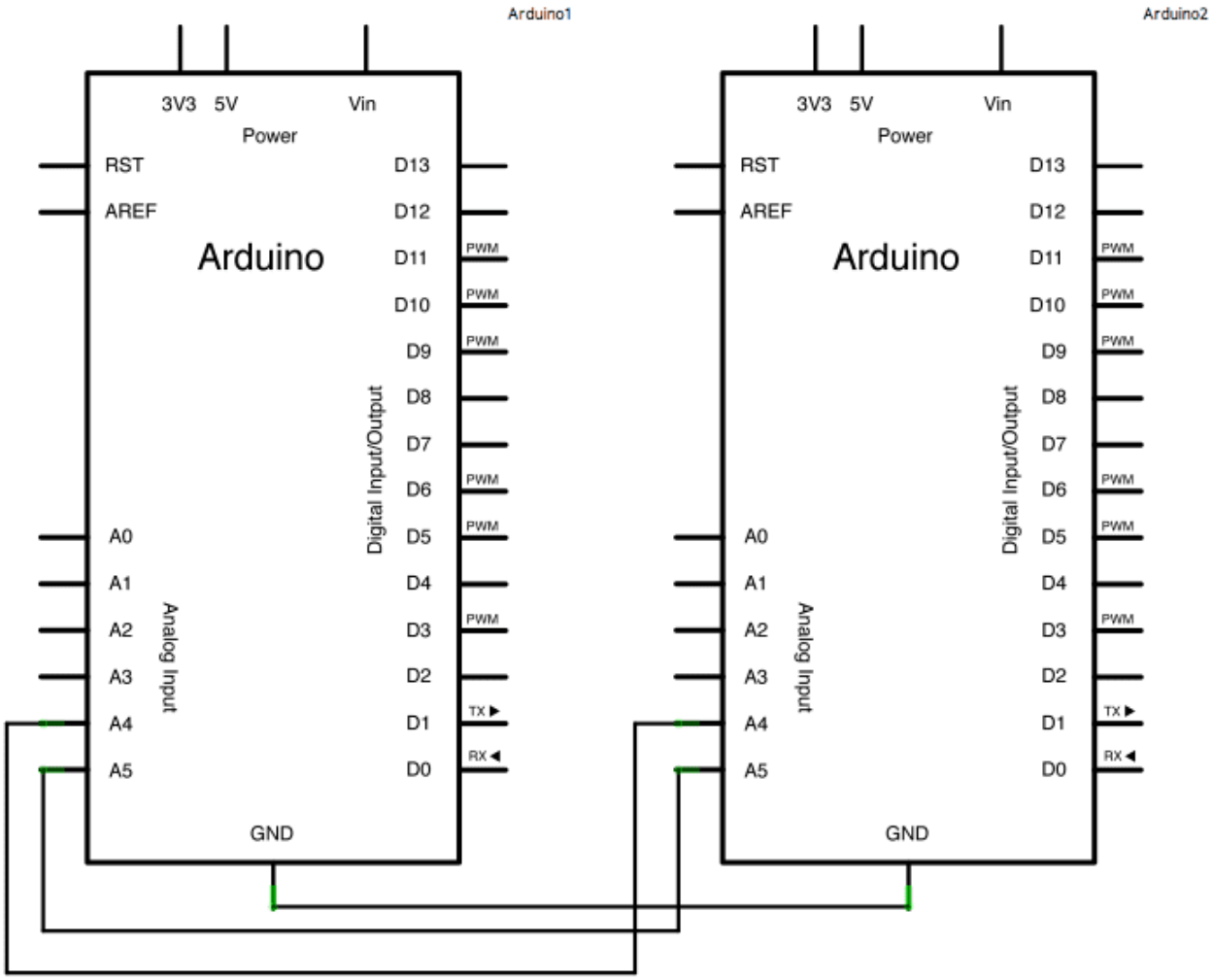


Ilustración 5: Detalle de la conexión. Fuente: https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter

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.

Además, existe un tercer archivo de administración llamado *administrador.py*, que se encarga de añadir permisos a usuarios, borrar permisos, y limpiar los archivos de mensajes y claves.

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 hasta que no esté repetido.

Ésta semilla sirve para todos los valores pseudoaleatorios que se usarán, como el contador. Es un número de 8 Bytes que se genera al azar, 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.

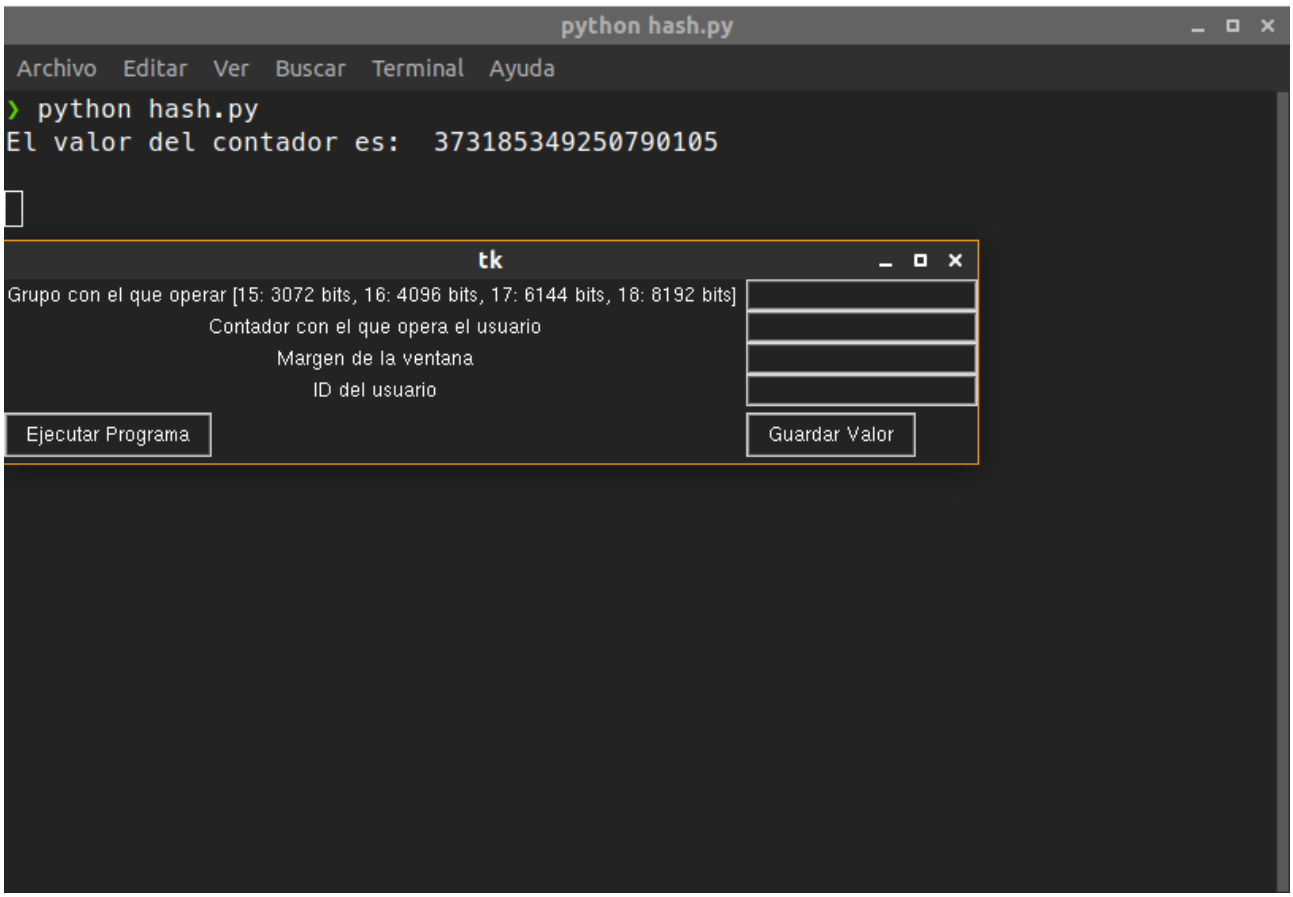


Ilustración 6: Interfaz sobre la que opera el usuario

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

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
- ID del usuario

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

```
def generarInput():
    """Genera una ventana por tkinter para que el usuario introduzca
    el grupo con el que operar.
    Posee botones para guardar y ejecutar el programa"""
    ventana = Tk()
    Label(ventana, text='Grupo con el que operar [15: 3072 bits, 16: 4096 bits,
17: 6144 bits, 18: 8192 bits] ').grid(row=0)
    Label(ventana, text='Contador con el que opera el usuario').grid(row=1)
    Label(ventana, text='Margen de la ventana').grid(row=2)
    Label(ventana, text='ID del usuario').grid(row=3)

    global n
    global n1
    global n2
    global n3

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

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

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

hmac_sha512(clave, mensaje): Función que genera el HMAC SHA-512 a partir de la clave en común 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 son iguales, calcula el HOTP y lo compara, y si es distinto el del usuario, es que hay un caso de desincronización, y se pueden calcular tantos valores de HOTP para la caja fuerte, siempre dentro de la ventana introducida.

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

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

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

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

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

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

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

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

    # El valor no es numérico y devuelve un error
    if not n.isdecimal():
         print("Introduzca un valor numérico la próxima vez")
         return -1
    # Establecer conexion inicial a partir de un objeto diffieHellman
    # se usa el grupo anterior
    conexion = establecerConexion(n)
    conexion.presentarResultados()
```

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

    # Atencion, el usuario actual puede estar por detrás en sus aleatorios que
el usuario anterior
    if contador_usuario != contador:
        # Dejamos el valor del usuario parado, la caja es la que va cambiando
        valorHOTPusuario = calcularHOTP(contador_usuario, n, conexion)
        print("EL CONTADOR DEL USUARIO NO ENCAJA. EL VALOR HOTP DEL
USUARIO ES",valorHOTPusuario)
        # Leer el archivo de mensajes aleatorios a partir de la seed de antes
        f = open("mensajes.txt", "r")
        # Leer todas las lineas menos la ultima porque no nos interesa el nuevo
valor
        lines = f.readlines()
        f.close()
        lines = lines[:-1]
        f = open("mensajes.txt", "w")
        for linea in lines:
         f.write(linea)
        f.close()
        # Coger los últimos VENTANA valores del archivo, sin contar el generado
para esta ejecución
        for aleatorio in lines[-ventana:]:
         # Print del valor con el que se operra
           print("Calculando HOTP para valor del contador",aleatorio)
           # Re-calcular el valor HOTP de la caja con el aleatorio
           valorHOTPcaja = calcularHOTP(int(aleatorio), n, conexion)
           # Si coinciden, se hace print de toda la info y se termina
           if valorHOTPcaja == valorHOTPusuario:
               print("EL SISTEMA ENCUENTRA EL MISMO VALOR HOTP PARA EL
CONTADOR GENERADO ANTES EN LA CAJA", aleatorio)
               print("VALOR DEL USUARIO:",valorHOTPusuario)
               print("VALOR DE LA CAJA FUERTE:",valorHOTPcaja)
               board.digital[13].write(1)
               time.sleep(10)
               board.digital[13].write(0)
               return 0
        print("NO SE CONSIGUE GENERAR LA CONTRASEÑA")
        return -1
    # Los contadores son iguales, no hay problema
    else:
        print("Los contadores de ambos son iguales, calculando HOTP para
ambos\n")
        valorHOTPusuario = calcularHOTP(contador_usuario, n, conexion)
        if valorHOTPcaja == valorHOTPusuario:
            print("Los valores coinciden, se da acceso")
        return 0

if __name__ == "__main__":
    main()
```

Clase administrador.py

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

Funciones del archivo

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

```
from tkinter import *
from tkinter import messagebox


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

    if type(user) != int:
        print("Introduzca un valor entero la próxima vez")
        return -1
    archivo = open('usuarios.txt', 'r')
    lineas = archivo.readlines()
    for linea in lineas:
        # El usuario ya existe en el archivo
        if user == int(linea):
            print("Usuario",user,"ya existente en el sistema")
            return -1
    f=open("usuarios.txt", "a+")
    f.write(str(user))
    f.write("\n")
    f.close()
    print("Usuario",user,"introducido de manera correcta en el sistema")
    return 1
```

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

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

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

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

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

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

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

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

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

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

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

    global n1
    global n2

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

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

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

    ventana.mainloop()
```

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

29550940968324137260021949390892599478042216967768499718895644764109350789742743453130791327221925372707551665801224425691638073963510569626792362974257003175740505125156352854563521648154555705149684143273156040094959720629022188693324053559676253778353029507024950765113043950529214462358380223270897252716657596921567756747114427246445114644775771321789643789379266877141060472137546308728850792191808679221451702627254557289847795472949667858736803405785139675619617941532852973422110090749478688177715315730359460032294662902456625161174416606983624418552000636499301482719490724043746644528702008780140535080598522135690801783621057239978032768966535811778392430583642314163278573210784571216359620205429638053335635504834951966172036535177757710151315537223182516463469693530039715790147868040250280769625418152847547689580390396963715488057712544324556133419648889585144581617704034399810912752816934978854515292651125082328009815738113672572529961056098864143660151202467462176130991650081094014977640470473843225687931961593153907251896704361918718680103534486194272475125076842313896575017322642082085540466833846817042022407816169569237792390864978603255742310449769255577268334760982511372920694108016228276333812649948729961769676116091577771698449672189331555282730119378934874792365896347360631728054589498725375655005740601436160333662679482993007062812127712070614381661141729343695993494300192183442703408373073616224264658463424930028078189554695345832894151719759919592160641974695198837344250967293839676495871375110408468231647503838906174004447326017635012431609966855235339172161949922899701777890560505615985100312737791592055158894867993903723239408332840919436596660765737574031598937460894828968255659796265866306977084532081204074603019987081366638257070076514746279605337085437580699801127262870405799518856236675675751003200525540105816103488404235490553553246726111715877441227748234572492355760655322624934175146366444307415548390753090865887017458919960056180651944268768619429443141720747 0

Resultado:

4737e7ffce3c906f7a94593f259f82cdf98610a46b5030e72c7578cae026ce960a546f107239c3d490bd508bfa2ef727c4de0137fc6537e761a3a1e5f50f3323

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 9791974547769813817. Introducimos como contador de la llave el mismo, una ventana de 10 en caso de que no hubieran sido iguales, y operamos con el grupo 15.

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

Valor privado de a:
463684749621588385579492127893982808242669996461740502291989500
310418470155362127785641233176364654852030464790528401406789822
567260227795458784492841976834482907051551385839373528850140535
340865680058743047501517874987488880162876745612290709606034501
464544434579491349781747179077089438757150712481062270264785220
549938555933301644056641061843021108279495569536431184750073021
968713021008160903576231991511599469092511452339586616701843940
719760331890968306553576772949601234115292621742008046399564012
673841904362742209942585220869516757387542864934296117794414954
759126782984864078345698506847111150018071641397248097274323501
563471782435382082302525191956969102490405613228198954903259294
388863767486864654150651783761850556130681622857421369782869645
930434575151088169786731747578087494498284805882121594247411738
687838946898799646152629943906170275204523524730515454616391612
390266392465888901623410564841803186738585 6

Valor privado de b:
220919673520395248627509143089462680659678385046548679725356592
449652217751342826152976147454935346725671254261413371353790526
859535611600900452020321997123491100713462694039305334257685560
843883600169066573100228911960090984874789161191420435204225239
737394979541141180656047317345630229740531612938656707252927230
670334813553001098483025859471852589902461095026288150951502003
670863139829706302786200352416379935245695219629911668657252983
656769727128621132216088039619798359578815122825664093953511665
233077166088098057194440177060082504496552766017569184362953099
586383240912699102862290472656361884114040136911583953730975977
836292305969842914687536055378548446553618543853108048140748991
871972849813407633461888735277799265474376685002444650580467396
949036682205129202364461506414030413116764353541939771125508697
470536290630286722378670182867018301330059337152693436340182237
613157278576234034094939977253114352813171 7

Valor público de a:
48860787194289593925796780498697341839660565862183377631010482764168518594885440936045894449084471776367855774092027883331621742813568532392478080402043417805147695972724722435708065966276768608083969474619199783864677288723086699081110729118026980577932303722894708085894124938741197627425353579458508856509136259317059370767095833205557275037770704972199124570884813266816388123126034075827538894152417296691377441412900703403169079141589829934526601868445640015237294172062299991585415973673209600014499542927006342172133403068397744094326872725066166735044767440942674275145230341829616882316060267362971799308340018596309628410615991043545578701749616238710253180670664157387123974334303416338262483112575356706220551979063750128250665239669514440838040762831388604091145957708790057062226685304580229069932434178996996253120464360755809907420018201636402908604530162527527150184255195270535832406863731591988672880465 1

Valor público de b:
17235531688809373874898676976348026557821949264659663277043519212429864588878606923805168837089359268786869069256437607618779221243021039951155429317169472357857364021741601225358185977940886289381953333383762373049180066245711708341845139660207639953228739634832430826168002532084692814895556188824526674189554037974211132080744076639329448294168824095479755668411613439023394781668435420449844380994181319433190981726192206122223372770831495869375775675870092712038489946337577827681228063377227363898019000189172411422223096266801671943422234290128636465911192837413043310643957880050323883676453087815334440623685503221214794404760375119863012591944951669549867867380296857811385448900757150212623575884543356908200520004830628256078026383917209537741922888535313060983023247973932175029043114104336781319630470530054447485915356890787198365694147186458767488413134746922924175333714354437670189206168141279213548504594 5

Valor de la clave para a:
46876624213734238698099612968235547426428666709234502316012486818712944063610467720485476026431751766473323951428347025368292414618458813102503672495361218754321689292070312731370282685251850239914544808119925729526175319073280817318804261786695880599759796155003960891222581428792150994376423501892278720721151144 3

370806853648940822538158684996370453262563318277281712380726447
592943713384349144371922359262186589735446425528037026034222206
898306455348792203902525388093766193307511121928653160273644145
641989376794239502736876565999067532396866510274054529489799397
528979183477354407345537574049499974081196299879305978008336600
833868932592448964667447371423726272629964377979413195568553734
458974492612005803606233419659662798028544309097695775230874516
579059876900763025513752791841041677121314669125550918803692543
473847501787098577417159638612038196635822754053614695455085354
936588648308084654826295925566429792850124 9

Valor de la clave para b:
468766242137342386980996129682355474264286667092345023160124868
187129440636104677204854760264317517664733239514283470253682924
146184588131025036724953612187543216892920703127313702826852518
502399145448081199257295261753190732808173188042617866958805997
597961550039608912225814287921509943764235018922787207211511443
370806853648940822538158684996370453262563318277281712380726447
592943713384349144371922359262186589735446425528037026034222206
898306455348792203902525388093766193307511121928653160273644145
641989376794239502736876565999067532396866510274054529489799397
528979183477354407345537574049499974081196299879305978008336600
833868932592448964667447371423726272629964377979413195568553734
458974492612005803606233419659662798028544309097695775230874516
579059876900763025513752791841041677121314669125550918803692543
473847501787098577417159638612038196635822754053614695455085354
936588648308084654826295925566429792850124 9

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:
094877f400d0723dd2f5f48aafdfc5d849a833719535348c0ec75a69a0960146ae
5e14f6b1c91c27cb6c7f2cd3aa061a28b7edcc32b77ae1aafe9031e86719da

Último Byte del HMAC: da

Representacion binaria de d : 0b1101

Representacion binaria de a : 0b1010

Binario del último Byte: 11011010

Valor en decimal del grupo de BITS resultante: 218

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

Se elige el grupo de Bytes 49

4 Bytes que nos salen: b7edcc32

Resultado decimal de los Bytes obtenidos: 3085814834

Primeros 8 dígitos de la contraseña: 30858148

Los contadores de ambos son iguales, calculando HOTP para ambos

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
094877f400d0723dd2f5f48aafdfc5d849a833719535348c0ec75a69a0960146ae
5e14f6b1c91c27cb6c7f2cd3aa061a28b7edcc32b77ae1aafe9031e86719da

Último Byte del HMAC: da

Representacion binaria de d : 0b1101

Representacion binaria de a : 0b1010

Binario del último Byte: 11011010

Valor en decimal del grupo de BITS resultante: 218

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

Se elige el grupo de Bytes 49

4 Bytes que nos salen: b7edcc32

Resultado decimal de los Bytes obtenidos: 3085814834

Primeros 8 dígitos de la contraseña: 30858148

Los valores coinciden, se da acceso

Prueba 2

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

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

Valor privado de a:
859537817268123984775759151909763455183991808736451832499858232
374192087978363886653687638396456845186092855230462134801290969
666251192652692694181109167931154249328722995607041542774156902
369159487845571334345510707691173749794214295304244878908696005
127006148388627256334441412823426670492507545498999820623449463
314232079324257236903760331201420361653494603453138880241517240
946021863975944346129848513544813015724746169706873259876952277
930760662308298694991602047758557662532860399349598153415538525
461591000501454731027173413095011963559739777650269288672913204
965510808638855293684269267407093584260415726952831435050640593
692786427344364799956985746322337854852207306681824985525450346
679785762789803467660371640401030625742735609147445191585654725
733919124941414928303794649302295745868545571656215704089593862
949432856229478497404041775788542966606847430792356478955216813
986425200983442942474502649461484312456843

Valor privado de b:
300804753597976767794995934800860526477125791758449485807879270
963118635965989952845258089274288006649766840637093585265337602
480092747626640551644360326439510934074671867737593707251606375
270532498772693415096625265876915426484012165792281711017462649
729540228028883590003789661398937704821517676613032606614485497
897552722989830192925554738501667554307162119629365431801991618
847817898790871839807046844270306361586509691784421510693781288
798767653339679718511956021907957656604736157736085172538432700
590841107834064422193435278207306050026906148451433551083172933
128506947421501444282700126972461769129599235526949897722005509
662518919879824136216012273248182774156627500901655962134701648
832939399035145912981989750089416200066541326330956828142362390
319343780572375718177414751356929961845327887718311945348582207
853743006627279973951868237985494163368181500898741883094994540
250695122809445430244174745331213413198522 3

Valor público de a:
90677395265571863556672592339213798146305397298253588921904107841124158059763048125512464114850297787901102863407569837151253218665549156847303538443116348600249601712595434636058708124995624455437479413338252590167591475659958605824548692590170561996316028505215228411194389743314939656747169224555519099684224055238670534628005296255378091376431418460147214660603450523412334918173407150565767162541232190344112664030606216727049462197655504779260086152041149017603387716814191493081790400565524398925961550458720102343296610605042180260396934306108379603792628968976823098149917893773622058577991052735128848598112127668048806767303771757733974695909619102823987378574251427569951798125289221769752673538065809881394267285504618834925242522984453282291241200497494333049528564436311879960913092451103670972296924475279237057737806932875024368477814826091227006946104244066634968637498658069520761333229475238960635711283 1

Valor público de b:
48204231071112488477986117422492847106909953786582437659454082649912906791974871688005238990935832388400697127307702228992474533414387263713203735140530489907889069735490553193653090507562192384260802234240934059364305491892584786683697346542395148194352022204970192271498305097831624238617546600282376660202256764392923227760257504805258846716676445265785564443350896828802619011493497352417187755096357732910943751472932951527426867811865345531958414249142202083371479573466165254861851024303867442656175620589558096389406164986898726222037108591742909997946612896615510559751094055805777523652999721551748429362635474487423176076998246637166378847476147989683284419068645551985101803476719055638894950735073007992875922508299338082056729171901511370881070214452815024044105243380780655089908847879930848374535777425403484584058849571788997755525462497308401898566549270863999456093567258334639435761169598085183915753312681 2

Valor de la clave para a:
32303438607219396995427734384294673449814762293237715896276583485081032127154333474764471029724977031212269952296957326210525991946205821608479519448381629276312055560710469554787741222298686614155464008006379389882229696572404960324711363315268192765802061639403156467870858400506068124116377110232382880470708136 3

821829866319816155797490831663927115933627353189383092473067550
226411526936514875062899007718498645642522650642606341961492974
896963811556975761722736920778356700586560393985735267087815111
663669160543460430406532637634132918410069831413810720463963635
179789190281423225915707016345120163710594261932733913197782396
239294665985805855913360368879098451155512316807645832854771323
591226089458605803050829319500038699288129367525962945630779984
047675492539646475086480411233467006339951147849475738604839386
543804704869461636461481930879877861844505164380163582614332913
077119387259335842198019135144375630923531 6

Valor de la clave para b:
323034386072193969954277343842946734498147622932377158962765834
850810321271543334747644710297249770312122699522969573262105259
919462058216084795194483816292763120555607104695547877412222986
866141554640080063793898822296965724049603247113633152681927658
020616394031564678708584005060681241163771102323828804707081367
821829866319816155797490831663927115933627353189383092473067550
226411526936514875062899007718498645642522650642606341961492974
896963811556975761722736920778356700586560393985735267087815111
663669160543460430406532637634132918410069831413810720463963635
179789190281423225915707016345120163710594261932733913197782396
239294665985805855913360368879098451155512316807645832854771323
591226089458605803050829319500038699288129367525962945630779984
047675492539646475086480411233467006339951147849475738604839386
543804704869461636461481930879877861844505164380163582614332913
077119387259335842198019135144375630923531 6

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:
e6bdd0b4e46bd92d4151b236bc8225f9b9f50f6aa26cf0d853cfddfea52176f084f
22620f5bcae591af6352a99dc4103e390f60590f16ca51bb5e2aa80326055

Último Byte del HMAC: 55

Representacion binaria de 5 : 0b0101

Representacion binaria de 5 : 0b0101

Binario del último Byte: 01010101

Valor en decimal del grupo de BITS resultante: 84

Se elige el grupo de Bytes 42

4 Bytes que nos salen: 352a99dc

Resultado decimal de los Bytes obtenidos: 891984348

Primeros 8 dígitos de la contraseña: 89198434

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
bc191a8384dc5ad7049be507282adcc4989dd9420a31b9a943c3521d99ba73fa
8e0f5a460eba9270f11144f0ee989cddbf9b755ba5aaabee2f1e96e60d256d3a

Último Byte del HMAC: 3a

Representacion binaria de 3 : 0b0011

Representacion binaria de a : 0b1010

Binario del último Byte: 00111010

Valor en decimal del grupo de BITS resultante: 58

Se elige el grupo de Bytes 29

4 Bytes que nos salen: ba73fa8e

Resultado decimal de los Bytes obtenidos: 3128162958

Primeros 8 dígitos de la contraseña: 31281629

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

Calculando HOTP para valor del contador 14906391684844699610

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

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

Calculando HOTP para valor del contador 476862029414530019

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
bc191a8384dc5ad7049be507282adcc4989dd9420a31b9a943c3521d99ba73fa
8e0f5a460eba9270f11144f0ee989cddbf9b755ba5aaabee2f1e96e60d256d3a

Último Byte del HMAC: 3a

Representacion binaria de 3 : 0b0011

Representacion binaria de a : 0b1010

Binario del último Byte: 00111010

Valor en decimal del grupo de BITS resultante: 58

Se elige el grupo de Bytes 29

4 Bytes que nos salen: ba73fa8e

Resultado decimal de los Bytes obtenidos: 3128162958

Primeros 8 dígitos de la contraseña: 31281629

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

VALOR DEL USUARIO: 31281629

VALOR DE LA CAJA FUERTE: 31281629

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 7411514733785699652, el usuario 100 (para fallar), el grupo será el 15 y la ventana será igual a 2.

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

Valor privado de a:
5416693328302166559710883732207040072279884597574984011676382620188566280813851381977770738023316452068004932889080576778436994020191049006049715327569010129222912167173185696031949192982072484449272206397900738026621264481387235695016836657188565662935414694427545274740401168996656004949597866318802790778273878558032625202211094658999987482223097588672850644111954444344260366460573339499279210204822336210401738323760870143874918772480124887183545067216168484844286895622742114637396704880307919568762140086134401640948823980222228494833482633014670572281091721362612010806716956425582807537535938629551002702012424969942946773084380789721743617983714366971412478287539186974967035278650151656416947509859090908357280020701426234587536793424279851313547962267063416900804090717929561301346796225355702806996016885823533758893731652711282093795800892424731753648165944423888112456850937851572164156003799335247871803428848 3

Valor privado de b:
4819072401446583746251085895821052707076020642783838851815841869035946547619687965349709767058099342891551968751869143628799642061765611565589389914126656178038453049431911313785423706691137980543263644827799236073831255955089557085398151419802428491472862509959901932408764588780767126996444955743437543091875565057134218307896672248091434982132839263406278628657258536208157872501242923652924283463906299350079127500258703085506392400727203545124358733233189773846931000498830226919524972737921981258840718220492630818272310752112332128351137372773357957327500525233998764807880682552532571853664826996778557782842690105066574473630634212529740748473117597886165647597557503062433346784171425134294900082563260966148439933100010459890265932096146773747343657295906175131317848946204059088749305971781734235587580471643 2

514232557617688575483301781723151472290661047548375478519612978174526680193815856751403081136148632149088

Valor público de a:
573277713093170374170141637558426623322559941561226280007144073128163021999446797832142046166994236643110583092464331287908714492517103628615557026959575362743354689159717353665313806458209103161263787322309907797290213705295976755834703572246378974970905163074617285517232533868829820688341750853957080367355327266214955840966614794074257154498359531024428299680794778020863983896386236123320069602990409908484311712827166629417795759682898610801659844122903603678698474217082277059075360801107693497386057591908752018879204060004147661382925392328792986461712796976105526287483170483438874569133404497066670135801873393049592115583983067757152303332134247088563800657839459407223424414738081307162084791886992606275089600345296646249688381668808951077956885472844124153439723138180854994990792382454539203213434338117689987860827553071404781855292404037809536304852665222291475628851764342070345936564034451261083642316621 1

Valor público de b:
345238920450089114768217374269205489486929748983021837930377908980198317377783787434699032761828117976324039241607982112940108382454059074199562711670298133696919503360731174110866886446956710248571587961193636790413019595773893642108337198260632961282614014838053240227289188953742209165494862147917508567653143184707909859464774900035796675808825198749149384017968782964719764189875964512489529369886828563321939176785996391406575156939177934701230715248344455508843954906433693202129405235518379929784072351814115382024587481987399198634689659536304182578680342677035860552219953975736541528488412513751641206277896536771263563392451317074568483954401887720167731093440481121320373571848228503535020851365126538156840315226971472090793064074059936845204373180243638057149198281516876148197959963463045877377909110693186911649315299832670630689591197096900572157491986288510289469157988145709429905246673263857940577413702 9

Valor de la clave para a:
278126431603962744778482537271357671687535415492282898755674615562560009853026658549832484121696176466111847441732279504347897543347200994819960152184281349288575652109633708203744605236953

056620075259832896676804931981822561607494990399028310375938709
537674713504412332771904378880081843702760579309112681906461425
782188422455788343229047734892317354923827162341509100825666216
730740401011330288424125566487982096575065216772500096861410820
118657389507483697112849736136291959574210148958179725193230301
780858047136320734007494732380342448827135046722313608262643404
244575615685666816550320312143358984105913055513286666249246101
001369016385393147111574686281589595186539045268779037912376990
798474933964449168886363458812644497561342010038535923027771975
396405692468086356193071550305461686198303043915344344575916802
208741616590499809815550417587971360644055947371698723335715517
414480839493025092876891244636445755732497 4

Valor de la clave para b:
278126431603962744778482537271357671687535415492282898755674615
562560009853026658549832484121696176466111847441732279504347897
543347200994819960152184281349288575652109633708203744605236953
056620075259832896676804931981822561607494990399028310375938709
537674713504412332771904378880081843702760579309112681906461425
782188422455788343229047734892317354923827162341509100825666216
730740401011330288424125566487982096575065216772500096861410820
118657389507483697112849736136291959574210148958179725193230301
780858047136320734007494732380342448827135046722313608262643404
244575615685666816550320312143358984105913055513286666249246101
001369016385393147111574686281589595186539045268779037912376990
798474933964449168886363458812644497561342010038535923027771975
396405692468086356193071550305461686198303043915344344575916802
208741616590499809815550417587971360644055947371698723335715517
414480839493025092876891244636445755732497 4

Los valores son iguales, comparten la misma clave

[Resumido]

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

Calculando HOTP para valor del contador 15054378804418375653

[Resumido]

Calculando HOTP para valor del contador 97919745477698l3817

[Resumido]

NO SE CONSIGUE GENERAR LA CONTRASEÑA

Prueba 4

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

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

El valor del contador es: 6128953485477967316

El grupo introducido ha sido el 15

El contador del usuario es 6128953485477967316

El margen con el que se opera es 15

El ID del usuario es 1

ERROR; El usuario no tiene permisos

Prueba 5

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

El valor del contador es: 6228590875159555504

El grupo introducido ha sido el 15

El contador del usuario es 6228590875159555504

El margen con el que se opera es 14

El ID del usuario es 6

Valor privado de a:
13736027981824370971309248134224865682709553971455602273352883779046259594361598635940325283713128972118473723863894301668986395297695199572617574296087788764848987609071375639573070573143595916676400304064012461987773867220630334884387842729079580382594155729372857980774641992647313059480869324130348563306152511399535627348587790052306287299995560699158858292302153637289988388328931239694959927755309231914805762753067386385236769057433329631238934284521605971204365121682779641116692310505064195643496849085159344986092336198994204892148275143052174448719404951924018815452465214329192827591757214796905108818854628762904230637750315993171985614309900520005830749812630132851660055196451984797389626443246376808661909508356620803810337767773536873833198882137245100887229848913617403083275682752860425924851845834336669841398937402277288644682617345092548193070584265977848781448114596625296963162640264981092056231374 07

Valor privado de b:
50880719604918457385404054465754161244517326370797203345804513969170218403601613125821743822130512370793984470943878079566890068133191969898102157600980891338316297944391834165259759025856472422514744698175159948428709041462148984570552685307127291213795833100354589097517393474668243349068262418004322573846701403433282048650710444605996240267117730518851465383193909298025899 1

459979752306426778679195842127342255412255234158471020830190955
064298262278319696258411254226143956337318697174118049697079361
277554445672720831715106238655830437612055127820203721085432715
347250941112825520380579600585005603582845639819668792250902651
406794000315055668147505830157827120652216593510776937270352622
615190945995986099253314895059979829410626444100179171683654158
408722425094023668718721552647907429626112591755407398703134375
800701401439589631285748483584033809768786307978914222390101688
875221329163894933989523248069280543241127 7

Valor público de a:
372571895432868274181860084579620951264017044070465383162665440
078768610613102330916814966202465050496632139301760951326524455
559864627808013137275563008865504911761588619423638363035021866
921278751293280096184808825791584083369577082139050956033265450
100080641724971145458542636455802180872921696918912457504844840
706039241548205667402858775141454663933063957682363793500897011
877703633782024600654742858127736874594010025451583074294885985
477611973133622224237075033108055222799967018955303768427694366
031495195086932646866689244470702791565214301649940355564298998
959491689400808328315111108690066081849953822802868851756796210
051906445497184150940841878262317207336912159971113160517358104
228361823881438516351560570199235359581590462483952218750453975
192500342839487777292862603228981712810593186350643903398496997
709647439817176137682092403498351367817831743987358546230347566
038885590510838586414515487842759461548441 0

Valor público de b:
465277006999515229091642974971904584924942256788494586754542324
702048002785780040262069289443587023209063368882922098242294188
693170218916386255292294676747761097675515410609923134745972200
635794215540500922106871237174819853538115359931956475234598533
175369102158940279138979548223179586193637130274412971262623554
624043451370969909712277394274649103111438804502631677126822859
905171664232371475654377281273497392190422001235335471883277460
798891061142955090496664053849711227966500049199766270931360597
107140607625005610814999362877969469039261513650020080475150897
892337758985268301971025126422689965017836377734816922843764421
144097230747116827230713958551146976791643007475901840582585252
093405620789763266066404982947444068549959320603588526129244795
333940866596679710679186350045917217455363792547017328082614784

4983310220373472003818757355433823783803834501993470616051546808976160536092364721412154824746895345472631

Valor de la clave para a:
1023444449685432312605316573786969028596796870021414821594930819838871240064542186862266316507599222616479594181855832626665778681907695555382440814458194684703130634581148545472458977260225243712872239851949545316638238208767654218330777249669430963430400766736666063642548115162217999170686250824026823438170971525928449815221672677292439724738291527547136635166875576059362787199763915009051893916339469049857137504142161251578498489349119731787654820347007109553299952231244548776360496891159768064992495332873066097277089653132560765979451904740477033435541390986787371965066065093766994603129611694917139576421059630532886425467052309110904998969076661326681507604220926350372581371022005577312456829054137106623206763091409294836645970180867111957563417785631040149332465109795724326724863356121450409967833043447823328494304538322828097303109664836378027833192670813345482421829480406519249825305342084270912472835040

Valor de la clave para b:
1023444449685432312605316573786969028596796870021414821594930819838871240064542186862266316507599222616479594181855832626665778681907695555382440814458194684703130634581148545472458977260225243712872239851949545316638238208767654218330777249669430963430400766736666063642548115162217999170686250824026823438170971525928449815221672677292439724738291527547136635166875576059362787199763915009051893916339469049857137504142161251578498489349119731787654820347007109553299952231244548776360496891159768064992495332873066097277089653132560765979451904740477033435541390986787371965066065093766994603129611694917139576421059630532886425467052309110904998969076661326681507604220926350372581371022005577312456829054137106623206763091409294836645970180867111957563417785631040149332465109795724326724863356121450409967833043447823328494304538322828097303109664836378027833192670813345482421829480406519249825305342084270912472835040

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
46b7d8ae6bcbeaf212dc2a7010667b3fe15432f297078515902bf8ab9aaba5b38
3d8aa4c11ea871f3c7bcbfcfcfe1ede8204877c3a5c9be4845fc39ef87b1709

Último Byte del HMAC: 09

Representacion binaria de 0 : 0b0

Representacion binaria de 9 : 0b1001

Binario del último Byte: 01001

Valor en decimal del grupo de BITS resultante: 8

Se elige el grupo de Bytes 4

4 Bytes que nos salen: 6bcbeaf2

Resultado decimal de los Bytes obtenidos: 1808526066

Primeros 8 dígitos de la contraseña: 18085260

Los contadores de ambos son iguales, calculando HOTP para ambos

Los valores son iguales, comparten la misma clave

Tamaño del HMAC resultante: 64 Bytes

Resumen HMAC resultante:
46b7d8ae6bcbeaf212dc2a7010667b3fe15432f297078515902bf8ab9aaba5b38
3d8aa4c11ea871f3c7bcbfcfcfe1ede8204877c3a5c9be4845fc39ef87b1709

Último Byte del HMAC: 09

Representacion binaria de 0 : 0b0

Representacion binaria de 9 : 0b1001

Binario del último Byte: 01001

Valor en decimal del grupo de BITS resultante: 8

Se elige el grupo de Bytes 4

4 Bytes que nos salen: 6bcbeaf2

Resultado decimal de los Bytes obtenidos: 1808526066

Primeros 8 dígitos de la contraseña: 18085260

Los valores coinciden, se da acceso

Pruebas en Arduino

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

Se ha conectado el LED al pin 13 del Arduino, y se ha retocado el código anterior en Python para que, cuando se consigue un resultado correcto, el LED se encienda. Se ha mantenido el código en Python para enseñar resultados por pantalla.

Actualmente se usa el módulo de Python *pyfirmata*, que utiliza el protocolo firmata [19] para comunicarse con el Arduino y enviar y recibir señales.

El LED se enciende durante 10 segundos, emulando así la señal eléctrica que llegaría al actuador lineal, en caso de coincidir los valores de apertura. De lo contrario, el LED no se activa y continúa apagado.

Nada más ha sufrido cambios respecto al programa original.

En el futuro se intentará cambiar el lenguaje para que ejecute de forma nativa en la versión de C que usa Arduino. Se han hecho una serie de pruebas iniciales con un código retocado en C, pero el Arduino del que se dispone actualmente carece de espacio para almacenar en variables los grupos de más de 2000 bits, y problemas con su generación de números aleatorios, necesidad de más módulos adicionales para que funcione el Arduino...

Sería necesario utilizar otro microcontrolador que cumpla con los requisitos necesarios de memoria.

Vulnerabilidades del sistema

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

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

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

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

Una de las más sencillas formas de hacer esto, por ejemplo, es proporcionar al Arduino del sistema central un voltaje que no pueda soportar. Siendo el límite 20 voltios, habría que comprobar cual es el voltaje que rompe por completo el Arduino. Tras romperlo, la caja quedaría sin forma de poder acceder a ella, dado que no existe ningún tipo de backdoor mecánico que nos deje acceder, y habría que recurrir a herramientas manuales de forzado para abrir la caja de nuevo y cambiar las piezas afectadas.

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

Normalmente un usuario normal nunca debería tener acceso a estos controles, igual que no podría leer los ficheros con los mensajes, claves o usuarios, pero en el caso de un atacante con acceso a los mensajes o claves generados de forma pseudoaleatoria, podría ser capaz de, mediante ingeniería inversa y suficientes mensajes y claves, sacar posibles valores futuros. Esta sería una tarea que necesitaría de tiempo y acceso continuado a la caja o sistema hasta sacar una lista de valores posibles, además del equipo con el que taladrar la compuerta.

Sería preocupante también que se pudiera acceder al sistema de otra forma aparte de la de la entrada física (Bluetooth, Wi-Fi…), por ejemplo, como los

dispositivos IoT que se suelen usar hoy en día, y como bien explica “Security Analysis and Exploitation of Arduino devices in the Internet of Things” [20], ya existen métodos para provocar heap y stack buffer overflow en ciertos modelos de Arduino, pudiendo de esta forma extraer información de ficheros, como mensajes o claves, para generar claves y mensajes futuros, o tomar control del programa en ejecución. Con un punto de entrada adecuado, se podría conseguir toda esa información, por lo que es adecuado que cuantos menos (sólo un punto de entrada físico), mejor.

Otra posible vulnerabilidad se encontraría a la hora de encender el sistema cada vez tras hacer reset en cada uso. De no existir unos ficheros de mensajes y claves, al reiniciar el sistema (apagado automático después de cada uso), se conocería siempre el estado inicial de éste, y se podría romper de manera muy sencilla la implementación. Al acceder a éstos ficheros para comprobar si las claves generadas al inicio del programa ya han sido usadas, nos libramos de éste defecto, pero somos dependientes de un generador de números pseudoaleatorios que debe estar sincronizado entre el sistema y cada usuario.

También se puede imaginar un caso en el que un atacante pretenda forzar el sistema mediante miles de peticiones por segundo, para intentar obtener claves, mensajes… O desbordar con peticiones para encontrar un fallo por estrés. El propio RFC de HOTP habla de implementar un sistema que aumenta en varios segundo el tiempo de respuesta a un usuario que falle varios accesos seguidos, para no poder realizar éste tipo de ataque.

Resultados y conclusiones

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

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

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

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

Estos RFC se han seguido y adaptado para nuestro uso de la manera más conveniente posible usando un lenguaje de programación actual como es Python, fácil de entender (y modificar) para cualquier persona interesada, en caso de querer seguir con el desarrollo, todo esto bajo una licencia GPLv3.

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

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

A pesar de ser mi primera entrada práctica al mundo de la seguridad aplicada a sistemas informáticos, se considera que se ha hecho una buena labor de investigación y de desarrollo.

En lo personal, el trabajo ha sido muy interesante y enriquecedor. Cada momento de investigación, de pruebas, de desarrollo de código… Ha sido de lo más agradable, y ha sido una de las mejores experiencias que he vivido en toda la carrera; esto es, partir desde 0 con una idea inicial y conseguir crear un sistema que se puede aplicar a la vida real, o que se puede presentar a cualquier empresa de seguridad para continuar con el desarrollo.

Líneas futuras

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

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

Dado que el coste de todos éstos componentes es bastante alto, se ha desarrollado todo de más abstracta, centrándonos en el código y la seguridad que éste ofrece. Numerosas pruebas deberían de realizarse sobre todos los materiales físicos, como el máximo voltaje que son capaces de soportar en una situación real, o la facilidad de poder forzarlos para intentar acceder a los contenidos de la caja elegida.

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

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

Además, se deberá portar el código desarrollado en Python a otro lenguaje que sea más óptimo, como C. Tras comprobar ésto, también se debería probar distintos microcontroladores que ofrezcan un mejor rendimiento y sean específicos para el tipo de tareas (paso de mensaje y cálculos criptográficos) que se especifican.

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

Finalmente, se debería presentar la idea a cualquier parte privada interesada en un mecanismo de seguridad de éste estilo. Como se comentó en éste documento anteriormente, la mayor parte de la lógica interna de las cerraduras de alta seguridad modernas se encuentran bajo secreto, o sólo mencionan de forma superficial su funcionamiento, por lo que comparar otros modelos actuales con el desarrollado en éste TFG es una tarea complicada.

La implementación de HOTP para contraseñas de un sólo uso puede ser algo novedoso e interesante para el área, y puede proporcionar una mayor seguridad y un factor de aleatoriedad en cada uso.

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

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

[19] TODO https://github.com/firmata/protocol

[20] TODO https://cosec.inf.uc3m.es/docs/papers/2016mal-iot.pdf

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].