BORRADOR

22/10/20

Cerradura Digital de alta seguridad para cajas fuertes

Pablo Castillo Martínez
Tutor: Jorge Dávila Muro

Índice

Estado del arte

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

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

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

Si bien es cierto que las cerraduras más modernas [1] (con contraseña electrónica, con lector ID, con conexión Bluetooth al móvil...) bien aplicadas por marcas de confianza son igual de seguras que las cerraduras tradicionales, suelen destacar marcas menores, poseyendo unos materiales deficientes en comparación, o facilidad para forzarlas por fallos de diseño. No por ser digital proporciona una seguridad adicional.

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

Éste trabajo pretende diseñar e implementar un sistema de alta seguridad para una caja fuerte, con técnicas que son actualmente usadas en la seguridad informática y criptografía modernas. También se pretende estudiar la fiabilidad que puedan proporcionar éstas técnicas tras algunas de ellas llevar múltiples décadas en uso.

Especificaciones físicas

Materiales

Según la definición de caja fuerte por la RAE, una caja fuerte (o caja de caudales) es “una Caja blindada para guardar dinero y cosas de valor.”[2]

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) o se conoce (una combinación).

Los materiales exteriores pueden variar dependiendo de la función de la caja fuerte (uso doméstico o 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, explosivos)...

Los materiales pueden ir desde el plástico endurecido pasando por el acero, hasta 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 [3], la norma UNE EN-1143/1 o la norma EN-1300 sobre CAS(cerraduras de alta seguridad).

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

Cerradura

Una cerradura es un mecanismo mecánico o eléctrico que permite el acceso a lo que guarda a través de la apertura del mecanismo cuando se proporciona algún tipo de información secreta (una llave, una combinación, la biometría del usuario, token...) conocida por ambas partes [4].

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

Para abrir ésta cerradura, es necesario tener una llave, un trozo de algún material duradero cuya forma ha sido alterada para seguir un patrón que encaje con el interior de la cerradura. Este interior consiste de una combinación de varios pines apoyados sobre un muelle, todos a distintas alturas. En su estado habitual, prohíben el movimiento del cilindro interior, hasta que se introduce el perfil correcto, que hace que cada pin se levante a la

altura adecuada para poder mover el cilindro, y poder abrir o cerrar la cerradura.

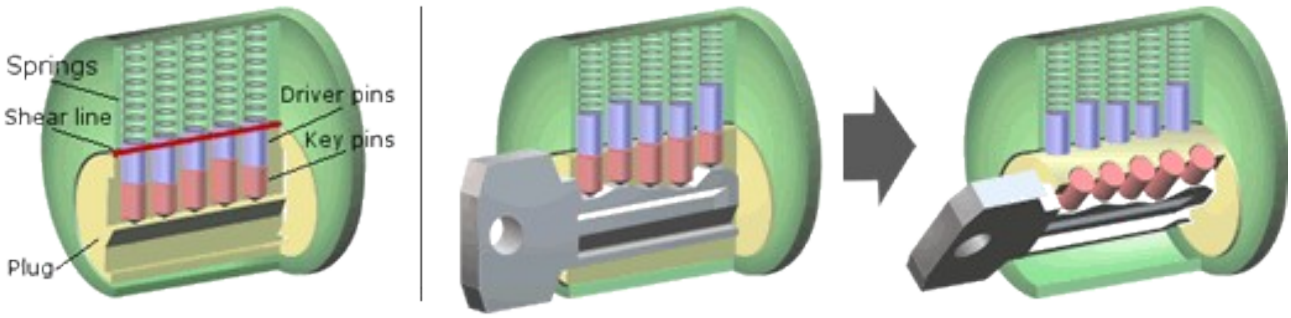


Ilustración 1: Fuente: www.safe.co.uk

Este método da lugar a varias posibles amenazas. Puede suceder que estas llaves sean sustraídas debido al factor físico que presentan, no poseen ningún tipo de secreto adicional; la apertura posterior se producirá sin depender de quién introduzca estas llaves.

Un método que no necesita de ningún token físico es la cerradura de combinación, que consiste en un dial externo que se conecta a unos discos interiores (tantos como "números" haya que introducir). Cada disco, a su vez, tiene un agujero. La alineación de todos los agujeros en una posición concreta hace que, por presión, una pieza en el interior "salte" y el mecanismo se abra. Cada disco también posee una conexión al siguiente disco, y un "diente" en cada lado.

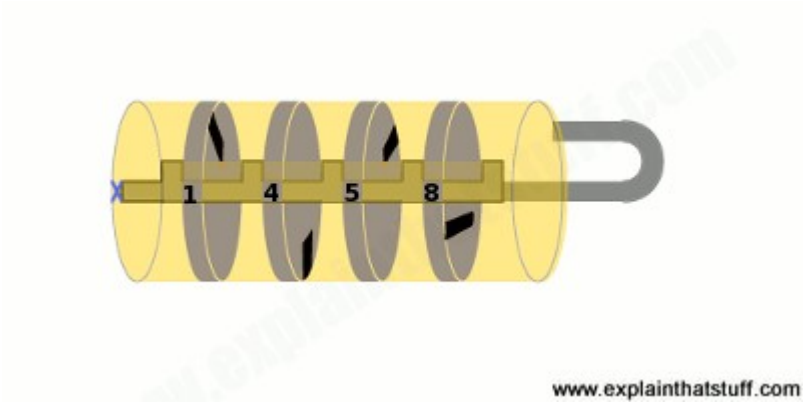


Ilustración 2: Fuente: https://www.explainthatstuff.com/yalelock.html

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

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

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

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

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

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

Mecanismo de apertura

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

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

Conclusión

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

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 tradicionales que componen una cerradura tradicional; no se puede diseñar un sistema de ésta forma.

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

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

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

Al ser necesario un token físico para el acceso a la caja fuerte, es recomendable que se haga inventario regular de quiénes tienen permisos para acceder (si es un entorno bancario o profesional) y disponer de algún componente adicional que pruebe que el portador es quien dice ser. Obtener el sistema token portátil, de otro modo, garantiza el acceso a la caja fuerte; sólo garantiza que ojos indiscretos no puedan reintroducir la clave que vieron para acceder.

Diseño del sistema

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

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

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

La parte criptográfica del sistema se describirá en detalle a continuación.

Criptografía

En el mundo de la seguridad tradicional, se denomina criptografía simétrica a aquellos métodos que emplean la misma clave tanto para cifrar como para descifrar los mensajes intercambiados. Uno de los principales problemas que introduce este tipo de criptografía es el intercambio inicial de claves, y cómo llegar a un acuerdo entre las partes interesadas para acordar una clave en común, y, además, que no se entere nadie que pueda estar escuchando el intercambio por el medio usado.

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

Sin embargo, ya existen métodos para hacer intercambio en un medio público partiendo de unos parámetros secretos iniciales de manera sencilla.

Por ejemplo, Diffie–Hellman [5] [6] (muy común y extendido en el campo desde su nacimiento a finales de los años 70) puede añadir aún más seguridad al sistema y generar claves en un medio público sin que nadie más se entere del resultado final.

Diffie-Hellman

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

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

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

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

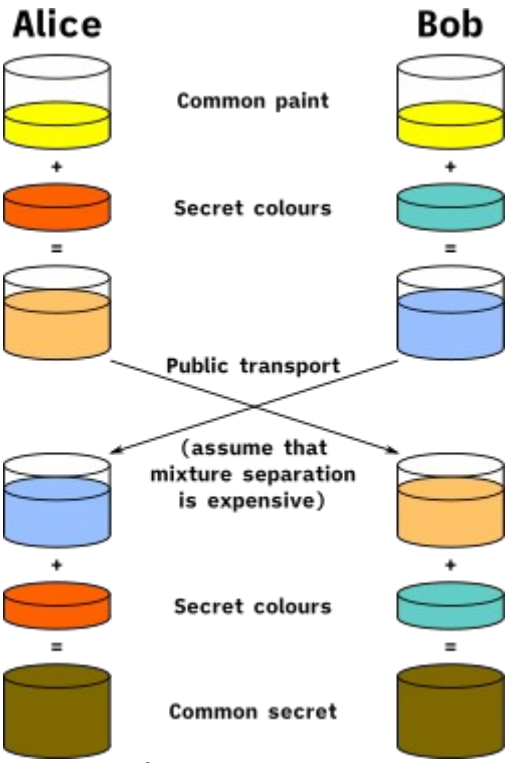


Ilustración 3: Esquema simplificado de cómo funciona Diffie-Hellman

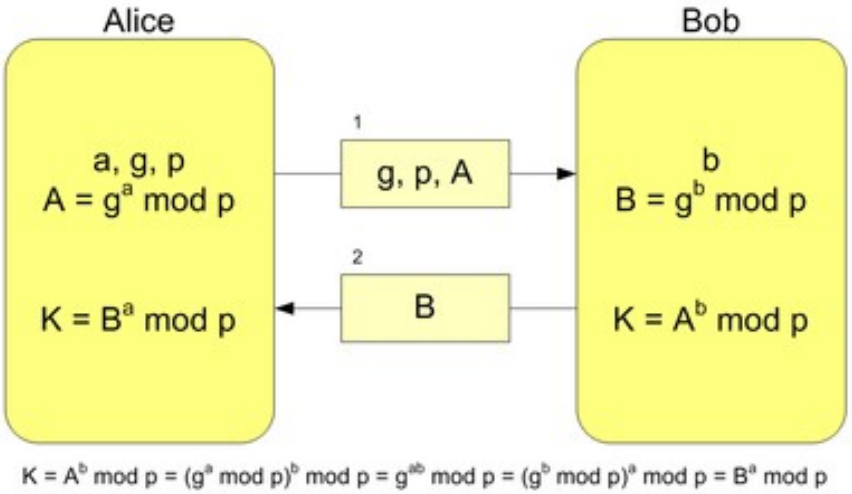
En detalle, los números públicos son un primo *p* y un generador de números *g* (enteros menores que *p* que son primos relativos de *p*). Son conocidos incluso por posibles adversarios.

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

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

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

El tamaño de *p* suele ser de 2000 a 4000 bits, siendo 4000 el valor recomendado.



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.

HMAC

Las especificaciones del HMAC se han sacado del RFC (Request For Comments) 2104 [7], que especifica cómo y con qué parámetros se debe implementar para su uso de forma segura.

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

Los componentes necesarios para construir el HMAC son los siguientes:

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

La clave *K* puede tener hasta un tamaño *B*, longitud del bloque de la función Hash. La longitud mínima recomendada es del tamaño de *L*, el output.

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

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

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

Los pasos en detalle son los siguientes:

1. Insertar ceros al final de *K* para crear un string de *B* bytes (Si *K* tiene un tamaño de 20 Bytes y *B*=64, se insertan 44 Bytes de 0x00).
2. XOR del paso 1 con el ipad.

3. "Append"/inserta al string de texto junto al resultado del paso 2.
4. Aplicar *H* a la salida del paso 3.
5. XOR del paso 1 con el *opad*.
6. "Append"/inserta la salida *H* del paso 4 a la salida del paso 5.
7. Aplicar *H* a la salida anterior y devolver el resultado.

La clave usada en HMAC puede ser de cualquier tamaño, pero se desaconseja que sea menor de *L*, ya que podría disminuir la seguridad de la función *H*. Las claves usadas, a su vez, tienen que ser elegidas al azar, de forma pseudo-aleatoria, con algún tipo de generador, y tienen que ser cambiadas cada cierto tiempo.

Toda la seguridad del método recae sobre las propiedades de la función hash *H* (tamaño de bloques y output).

HOTP

Sin embargo, el RFC de HMAC con su documentación es del año 1997, por lo que también se ha estudiado el funcionamiento de HOTP (HMAC-based One-time Password algorithm), sacado del RFC 4226 [8] del año 2005, algo más reciente.

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

Este algoritmo presenta los siguientes requisitos:

1. Debe estar basado en un contador o una secuencia numérica creciente que conozcan el sistema llave T y el sistema de caja fuerte S, o que sea fácil de sincronizar.
2. El algoritmo debe ser económico a la hora de disponer de hardware, como batería y su consumo, coste computacional...
3. Debe de poder trabajar sin ningún tipo de input, pero en dispositivos más avanzados puede usar teclados de tipo PIN.
4. Si se hace display de la contraseña, debe de ser sencilla de leer por un usuario. Esto es, debe de tener un tamaño adecuado, y mayor a 6 caracteres.
5. Debe incluir mecanismos sencillos en caso de querer resincronizar el contador.
6. El secreto debe de ser de un tamaño **MÍNIMO** de de al menos 128 bits, siendo el **RECOMENDADO** de 160 bits.

Símbolos

- **C**, contador de 8 bytes. Debe estar sincronizado entre cliente y servidor para operar de la misma manera.
- **K**, secreto compartido entre cliente y servidor. Cada cliente posee un secreto único.

- **T**, número de conexiones fallidas tras el cual el servidor rechaza las conexiones del cliente.
- **s**, parámetro de resincronización, el servidor intentará verificar una conexión recibida con s valores de contador seguidos.

El algoritmo funciona gracias a una clave simétrica sólo conocida por el servicio de validación y el usuario, y un contador que se incrementa.

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

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

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

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

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

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

```
SHA-1 HMAC Bytes (Example)

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



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

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

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

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

Si el valor calculado por el usuario no es igual al del sistema, puede haber un error en la sincronización entre los dos. Esto se debe a que el sistema sólo aumenta el contador cuando hay un acierto al calcular la contraseña HOTP, mientras que el usuario incrementa su propio contador cada vez que lo pide. Esto puede dar lugar a una diferencia entre ambos valores.

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

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

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

```
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.

Función Hash SHA-512

SHA-512 se encuentra dentro del conjunto de funciones criptográficas SHA-2. Como todas las funciones hash, crea una salida de longitud fija a partir de un conjunto de datos variable, como un mensaje o un documento de texto.

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

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

Además de ser un error, puede ser aprovechado por atacantes para realizar cambios en ficheros y, sabiendo las debilidades de la función hash, ocultar estos cambios bajo el mismo hash. Es interesante comentar que no es imposible no encontrar colisiones con una función resistente a colisiones, sólo

quiere decir que es muy difícil encontrar por medios tradiciones dos valores H(a) y H(b) que sean iguales.

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

Acceso al Sistema

Los pasos para pedir acceso al sistema son los siguientes:

En el primer acceso al sistema

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

Siguientes accesos al sistema

- El usuario se conecta al sistema desde su el dispositivo eléctrico llave (que más tarde se diseñará) con el ID de su dispositivo.
- Se comprueba que el ID en cuestión tiene los permisos necesarios para acceder al sistema.
- El sistema genera un desafío **R**, un texto de longitud segura y que nunca se vaya a volver a generar. El sistema pasa a esperar una respuesta del usuario al desafío, una respuesta que puede ser calcular el HOTP del mensaje, con la fórmula que se mencionó anteriormente.
- El usuario responde al desafío. Si es igual a la respuesta del sistema central, da acceso al código de acceso/combinación de la caja.
- Después del paso anterior, se sobreescribe la combinación de la caja por una nueva aleatoria, y también se cambia la clave asociada al usuario por otra al azar, y se pasa al dispositivo del usuario para que se actualice en el siguiente uso. Estas claves no deberían nunca de reutilizarse, pero sí que se pueden almacenar para a) comprobar que no se generan de nuevo (aunque puede escalar de manera negativa tras muchos usos) y b) añadir ruido a las claves que se generen para que no sea determinista para un atacante.
- Finalmente, la caja/cámara puede ser abierta tras meter la combinación calculada en pasos anteriores.

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

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

Componentes del sistema

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

Para la construcción, se han seguido las especificaciones de la página oficial de Arduino (varios modelos comparten las mismas especificaciones excepto por algunas diferencias en sus pines y su tamaño)[9].

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 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[10].

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

Sistema Llave

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

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

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

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

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

Sistema Caja Fuerte

Para el Arduino de la caja fuerte, se necesitaría:

- 1 x Arduino Uno, Nano...
- 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 [11].
- 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:

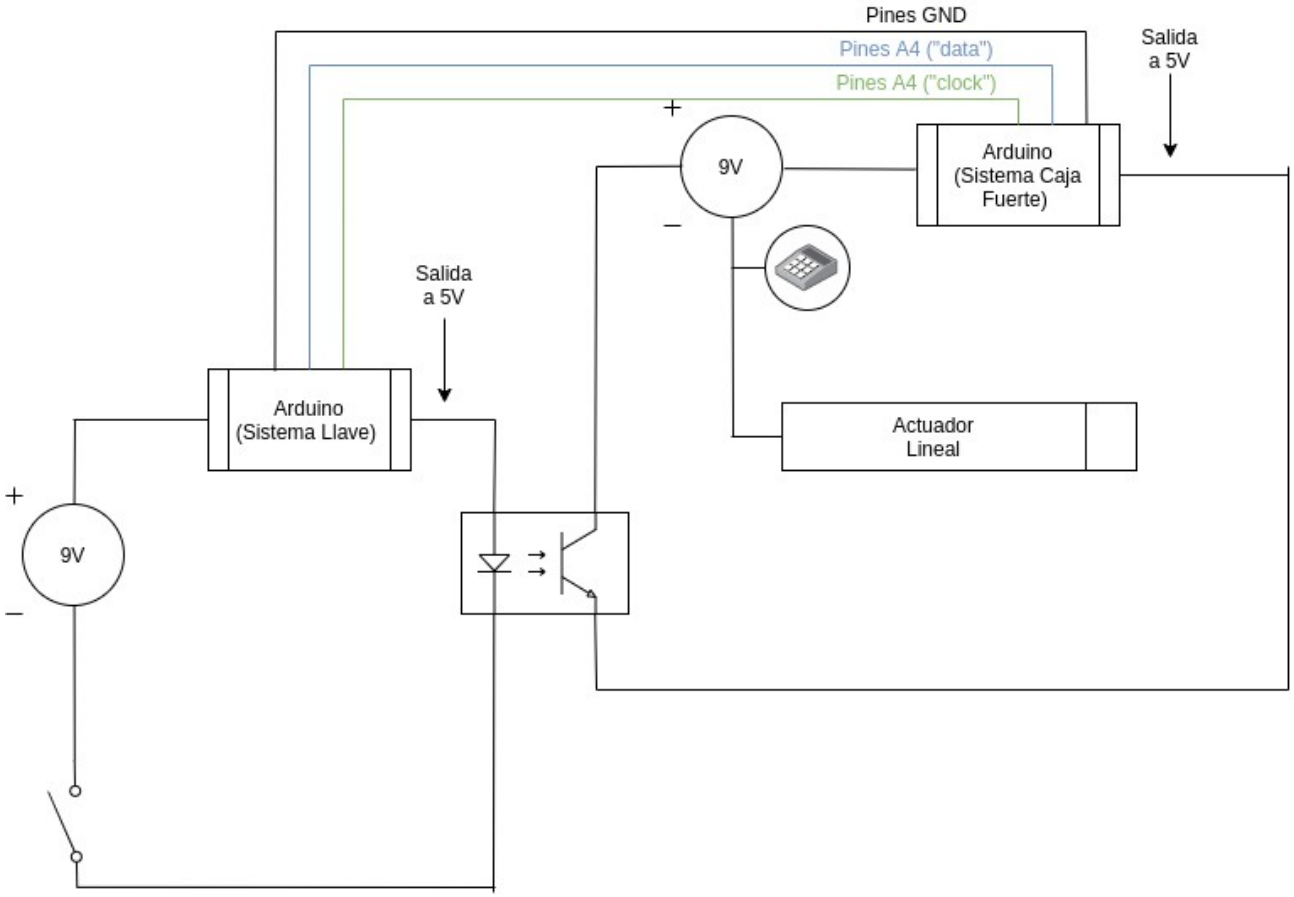


Ilustración 4: Esquema del circuito

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:

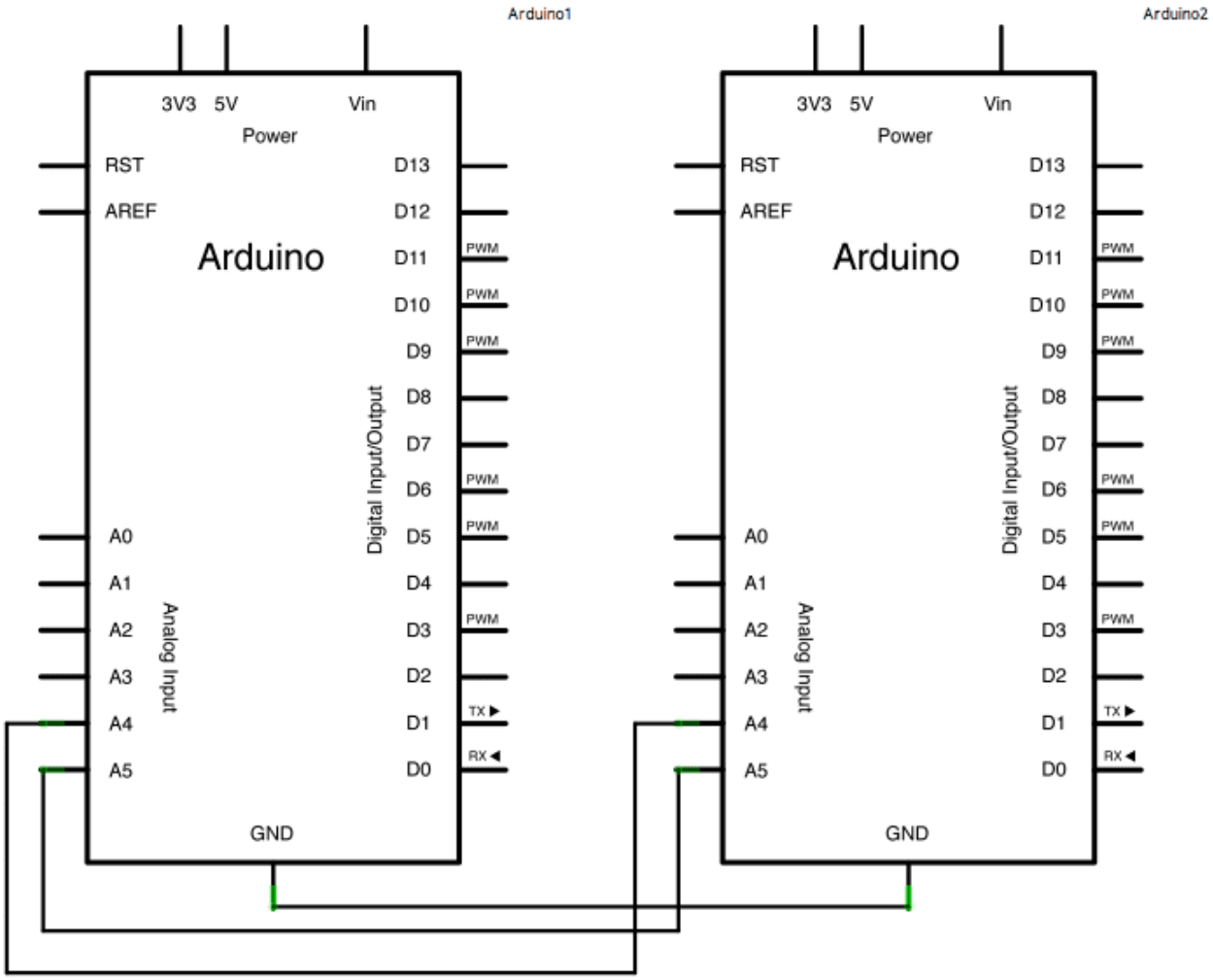


Ilustración 5: Detalle de la conexión. Fuente: https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter

Seguridad

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.

Implementación en código

Para la implementación inicial, el código se ha dividido en 2 archivos de python, *diffieHellman.py* y *hash.py*.

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).

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.

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

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.

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.

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 256 grupos, pero nos pasamos.

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

Al ser un valor aleatorio, podemos obtener un valor de 0 (cogemos desde el grupo 0 al 3), 1 (grupo 1 a 4), 2 (2 a 5)… El último valor válido es 60, que nos da desde el grupo 60 al 63, el último. Si se obtiene alguno de los valores sobrantes de 2^5 (61, 62 ó 63), se descarta y se pone el valor a 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.

Bibliografía

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

[2] https://dle.rae.es/caja#2s7HuNa

[3] https://www.boe.es/buscar/act.php?id=BOE-A-2011-3171

[4] https://www.locks.ru/germ/informat/schlagehistory.htm

[5] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[6] https://tools.ietf.org/html/rfc2631

[7] https://tools.ietf.org/html/rfc2104

[8] https://tools.ietf.org/html/rfc4226

[9] https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD

[10] https://www.arduino.cc/en/Tutorial/LibraryExamples/MasterWriter

[11] https://www.firgelliauto.com/blogs/tutorials/how-do-you-control-a-linear-actuator-with-an-arduino

Anexo A: Componentes

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].