UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

DISPOSITIVO AVANZADO DE GENERACIÓN Y ALMACENAMIENTO DE CLAVES

Proyecto Final de Carrera

Ingeniero Técnico en Informática de Sistemas

Autor: Luis M. Mahiques Carrasco

Director: José Ismael Ripoll Ripoll

Septiembre de 2013

Sudor, cansancio y noches en vela ...
No los que he sufrido yo para estar hoy aquí,
sino los que tu has pasado esperándome llegar.

Resumen

En el presente proyecto abordaremos el diseño y construcción de un dispositivo de gestión de claves.

La principal ventaja de emplear este dispositivo consiste en que el algoritmo que genera las claves produce siempre claves de longitud y complejidad superior a la media.

El dispositivo esta formado por una pantalla táctil que recibe entrada del usuario, que a su vez está conectada a un Arduino y tras haber generado la clave la envía por USB emulando un teclado estándar.

Para la generación de las claves, el dispositivo hace uso del algoritmo de cifrado simétrico *AES* y del algoritmo hash *SHA-256*

El dispositivo es capaz de emular a un teclado convencional ya que hace uso del protocolo HID.

Palabras clave: claves, Arduino^{TM}, seguridad, sha, aes, entrada táctil, usb

Índice general

Capítulo 1

Introducción

1.1. Motivación

Si entendemos la información como el conjunto de datos procesados que constituyen mensajes, pronto surge la necesidad de conseguir que esos datos deban y puedan ser protegidos.

Información comprende desde el estado de nuestras cuentas bancarias, hasta nuestras últimas fotos de un viaje con la familia, pasando por una libreta de notas donde guardamos la lista de la compra. En general, información posee varias de las siguientes características:

Significado
¿ Qué quiere decir ?

Importancia
¿ Trata sobre alguna cuestión relevante ?

Vigencia
¿ Es actual o desfasada ?

Validez
¿ El emisor es fiable o puede proporcionar información falsa ?

Valor
¿ Cómo de útil resulta para el destinatario ?

Todas estas características serán cuantificadas de un modo diferente para cada individuo en particular. Cuando alguna de estas características tiene un valor relativamente alto, el almacenamiento y posterior acceso a esa información juegan papeles decisivos. No querríamos que nadie pueda ver nuestra

correspondencia privada o colarse en nuestro ordenador y husmear en nuestros archivos.

Existen métodos y medios para proteger la información desde hace siglos. A día de hoy la forma más habitual de hacer esto es mediante códigos secretos: desde una simple llave que todos llevamos en el bolsillo, que no es más que una contraseña codificada como altura de pins hasta una firma de DNI electrónico.

Evidentemente, no toda la información se protege de la misma forma. En la actualidad, asociamos casi automáticamente, información con ordenadores y no sin razón, ya que estos centralizan la mayor cantidad de ésta.

Correo electrónico, banca online, redes sociales, acceso corporativo, ordenadores personales, móviles, servicios en la nube... son sólo algunos de los accesos que realizamos cada día y todos ellos están protegidos principalmente por una contraseña. Contraseñas que debemos escoger con ciertas condiciones para que sean seguras, idealmente distintas entre sí, ya que si se produjera la vulneración de una de ellas no afectara al resto de servicios.

Las estrategias clásicas de generación de las mismas de modo 'manual' son cada vez más difíciles de mantener ya que implican la memorización de hechos o gustos personales que son finitos, y si además tenemos en cuenta que todas deberían ser cambiadas con una periodicidad relativa, el asunto se complica aún más si cabe.

Para este problema planteado proponemos una solución mediante el presente proyecto. El proyecto consiste en un sistema que gestiona las claves que pueda tener un usuario, o conjunto de los mismos en un servicio o conjunto de éstos.

1.2. Objetivos

Entre los objetivos del siguiente proyecto podemos destacar:

- Aumentar la seguridad de los sistemas que protegen su acceso mediante contraseñas.
- Ayudar al usuario a establecer medios necesarios para que sus contraseñas sean fáciles de recordar y complejas a la vez.
- Practicar con el Entorno Arduino.
- Aportar un proyecto que pueda ser útil mejorando la seguridad.
- Investigar el estado actual de las pantallas táctiles para proyectos personales.
- Mejorar el ecosistema Arduino aportando soporte a idiomas.

1.3. Planificación

El resto de este documento se organiza como sigue: En el capítulo siguiente (2) estudiaremos las tecnologías en las que nos hemos basado, mientras que en los posteriores describiremos el proceso de diseño e implementación (3) así como la evaluación de resultados (4).

Capítulo 2

Tecnologías Empleadas

Dedicaremos el presente capítulo a exponer el estado del arte en los distintos elementos que finalmente formarán parte del dispositivo previamente descrito.

2.1. Algoritmo de resumen SHA

2.1.1. Algoritmos *hash* o resumen

El concepto de hash en el ámbito que nos ocupa consiste básicamente en la 'reducción' de una entrada de datos a otro conjunto de datos de longitud fija que identifica unívocamente al primero sin ser reversible. Dicho de un modo más correcto: una función hash H es una función computable mediante un algoritmo

$$H : E \to S$$
$$x \to h(x)$$

que tomando como entrada un conjunto de elementos lo convierte en un rango de salida finito, normalmente de longitud fija. Es decir, la función actúa como proyección del conjunto E sobre el conjunto S. Se debe cumplir además, preferentemente, que cada elemento del conjunto S tiene a lo sumo una antiimagen en E, o, lo que es lo mismo, en el conjunto E no puede haber dos o más elementos que tengan la misma imagen. Esto último impide las 'colisiones', que veremos más adelante.

Uno de los usos más extendidos de estas funciones es la comprobación de integridad de datos o la transmisión de conocimientos de claves sin transmitir la clave en sí. Por ejemplo, para que un emisor y receptor puedan asegurar que los datos que poseen son los mismos, se puede aplicar una función hash

sobre los dos conjuntos de datos, y si el resultado coincide, sabrán inequívocamente que, en un principio, no ha habido alteraciones accidentales o intencionadas. En el caso de la comprobación de claves, lo que se suele hacer, aprovechando el hecho de la función es inyectiva y por lo tanto no se puede obtener la clave de su resumen, es trabajar directamente con los resúmenes, el consignatario de las claves almacena únicamente el resumen de estas, y comprobando los resúmenes se puede garantizar que ambos la conocen. De este modo, además se logra una seguridad mayor, ya que si se viera comprometido el almacenamiento, no sería descubierta ninguna clave.

El empleo de funciones *hash* en el presente proyecto será justificado en la sección de Diseño

2.1.2. SHA-256

La familia SHA (Secure Hash Algorithm) aglutina un conjunto de funciones *hash* publicadas por el NIST [1] y relacionadas con la Agencia Nacional de Seguridad de Estados unidos. Existen ocho variantes dentro de la familia, ofreciendo cada una de ellas diferentes longitudes de salida, a saber: 160 (*SHA-0, SHA-1*), 224 (*SHA-224, SHA-512/224*), 256 (*SHA-256, SHA-512/256*), 384 (*SHA-384*) y 512 (*SHA-512*) bits respectivamente.

Esta familia se divide en cuatro grandes subconjuntos, agrupando los diferentes algoritmos por fecha y resistencia, aproximadamente. El primero de estos es *SHA-0*, nombre que se le ha dado posteriormente al *SHA* original para diferenciarlo de los demás. Después viene *SHA-1* que pese a tener el mismo tamaño de salida, incluye ciertas mejoras sobre el anterior. El siguiente subconjunto es el formado por todos los demás anteriormente nombrados y se denomina *SHA-2*. Queda únicamente por nombrar *SHA-3* que a fecha de septiembre de 2013 aun no ha sido completamente estandarizado ya que en el cuarto trimestre de 2014 será sometido a comentarios públicos y completado hacia el segundo trimestre de 2014.

A día de hoy se han encontrado colisiones[2] en *SHA-0* y el NIST desaconseja utilizar *SHA-1* en favor de las versiones superiores pese a no haber encontrado aún ninguna debilidad confirmada en este. Por lo tanto, para la realización del presente proyecto en los lugares donde necesitemos de *hash* emplearemos *SHA-256* por ser miembro de *SHA-2* y ser el que mejor rendimiento ofrece en el sistema que vamos a desarrollar, sin que ello afecte a la seguridad del mismo.

[1] National Institute of Standards and Technology

[2] Una colisión es un par (x, y) tal que $h(x) = h(y)$ siendo $x \neq y$

2.2. Algoritmo de cifrado AES

AES (Advanced Encryption Standard), también conocido como Rijndael es un esquema de cifrado por bloques desarrollado por los belgas Joan Daemen y Vincent Rijmen [1].

Fue presentado en 2001 como FIPS PUB 197 [2] tras un proceso de estandarización que duró 5 años. Desde 2006 es uno de los algoritmos más populares utilizados en criptografía simétrica.

2.2.1. Historia de AES

En 1997, el NIST decidió realizar un concurso para poder proteger información sensible durante el siglo XXI mediante un nuevo algoritmo de cifrado.

Tras una convocatoria formal quedaron admitidos 15 algoritmos que debían cumplir las siguientes condiciones:

- Ser un algoritmo de cifrado simétrico y soportar bloques de un tamaño como mínimo de 128 bits.
- Las claves podían ser de 128, 192 y 256 bits.
- Ser un algoritmo disponible para todo el mundo públicamente.
- Ser implementable tanto en software como en hardware.

En 1999, tras someter todos estos algoritmos a exhaustivas pruebas la lista de algoritmos se redujo a sólo 5 finalistas:

- MARS
- RC6
- RIJNDAEL
- SERPENT
- TWOFISH

Éstos fueron sometidos a pruebas hasta mayo de 2000. Concluidas estas pruebas, se produjo la votación y Rijndael ganó con 86 votos, su siguiente rival en la clasificación obtuvo 59 y se trató de Serpent.

2.2.2. Desarrollo

Rijndael es una red de sustitución-permutación. Es rápido tanto en hardware como en software y es fácil de implementar.

AES opera sobre una matriz de 4x4 bytes, llamada *state*.

Las fases del cifrado son las siguientes: En primer lugar se expande la clave. A continuación se produce la etapa inicial compuesta por **AddRoundKey** en la que cada byte del *state* se combina mediante operaciones **xor** con la subclave, que procede de la clave.

A continuación vienen la rondas, que constan de los siguientes pasos: En primer lugar se realiza el paso **SubBytes** dónde se realiza una sustitución no lineal de cada byte en base a una tabla de búsqueda. Tras esto viene el paso **ShiftRows** en el que las filas del *state* son transpuestas mediante rotación. Le sigue **MixColumns**, aplicando transformaciones lineales sobre cada columna combinando los 4 bytes de cada una de ellas. Por último **AddRoundKey** combinando cada byte del *state* con claves intermedias *round* derivadas de la original mediante iteración de clave.

La etapa final se compone de **SubBytes**, **ShiftRows** y **AddRoundKey**.

El número de rondas a aplicar en el cifrado o descifrado dependerá del tamaño de bloque del siguiente modo *a*) 10 para 128 bits *b*) 12 para 192 bits *c*) 14 para 256 bits

En el presente proyecto emplearemos *AES-256* por ser el de mayor seguridad dentro de la familia.

2.3. Interfaz HID

HID (Human Interface Device) es un conjunto de protocolos mediante los cuales se consigue unificar la forma de funcionamiento de diversos dispositivos que interactúan mediante USB.

HID permite innovar en lo que a periféricos se refiere, pues destierra el paradigma tradicional de los anteriores estándares estrictos. Los dispositivos que implementan esta interfaz la hace auto-descriptiva, siendo el sistema al que se conectan el encargado de elegir a cuales de sus funciones obedecen y en que grado, a diferencia de los drivers antiguos que o bien se 'hinchaban' para poder aglutinar familias más grandes de dispositivos o duplicaban código enormemente por ser más modulares

2.3.1. ¿Para qué se usa HID?

Entre sus principales usos, podemos encontrar teclados, ratones, joysticks, o tabletas gráficas. La gran mayoría de sistemas operativos implementan gran

soporte a este estándar y por lo general pueden reconocer dispositivos de este tipo al vuelo sin necesidad de reiniciar y en muchos casos sin drivers particulares. Este ultimo comportamiento se denomina plug-and-play.

HID distingue dos entidades: *dispositivo* y *host*. El primero es el que realiza la interacción con el humano mientras que el host gobierna a l dispositivo. Existen casos de inversión de autoridad, o más bien duplicación de la misma, pues existe un estándar USB denominado OTG que permite que dos dispositivos se conecten entre sí pudiendo gobernar ambos. Este es el caso de conectar un smartphone a un PC y el PC tendrá autoridad sobre el smartphone, mientras que si el smartphone se conecta a un ratón o teclado será el el smartphone quien ejercerá la autoridad.

Hay que agradecer a HID el gran impulso que tuvo USB ya que sin él probablemente no sería lo que conocemos hoy, unificando y homogeneizando conexiones. ¿O acaso imaginamos aún ordenadores con puerto serie, paralelo, pcmcia, y demás?

2.3.2. ¿Porqué HID?

Para la realización del presente proyecto se ha optado por utilizar esta característica ya que el dispositivo así obtenido es agnóstico de ningún sistema operativo, y puede funcionar indistintamente en todos ellos.[3]

Como opción inicial se planteó el desarrollo de un modulo PAM[4], pero esto conllevaba la recompilación del núcleo de Linux, haciendo que para emplear el presente dispositivo hiciese falta modificar aspectos esenciales del sistema que hubieran acarreado mantenimiento, versiones, y en definitiva una dificultad añadida totalmente innecesaria y fuera del ámbito del proyecto propiamente dicho. Esto sin contar las opciones para Windows, Mac, tablets, smartphones... donde el maremágnum habría sido tal que he proyecto habría desembocado en algo muy diferente.

2.4. Programa pwgen

Según el propio manual de la aplicación [4]:

> El programa `pwgen` genera contraseñas diseñadas para ser fácilmente recordadas por humanos, al mismo tiempo que son lo más seguras posible. Las contraseñas fáciles de recordar nunca serán tan seguras como las que son completamente aleatorias [...]

[3] Siempre que éstos dispongan de soporte USB

[4] Pluggable Autenticathion Method: Mecanismos flexible que permite abstaer las aplicaciones y otro software del proceso de identificación

> Por otra parte, las contraseñas completamente aleatorias tienden a ser anotadas, aumentando así sus posibilidades de ser expuestas.

Este programa genera contraseñas robustas en base a unas reglas de composición, pudiendo forzar que tengan diferentes tipos de símbolos como mayúsculas, números y puntuación.

Para realizar su trabajo obtiene entropía o bien de un modo puramente aleatorio, o bien tomando los valores del resumen *SHA-1* de un fichero a elegir más una semilla arbitraria. Es esta característica la que más nos interesa. En nuestro caso modificaremos profundamente el código convirtiéndolo en una librería para Arduino, donde además reemplazaremos *SHA-1* por *SHA-256* y añadiremos *AES-256* como justificaremos más adelante. Aquí el origen de la información será la entrada que realice el usuario sobre la pantalla táctil.

2.5. Ecosistema Arduino

Según la propia página del proyecto [3]:

> Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos.

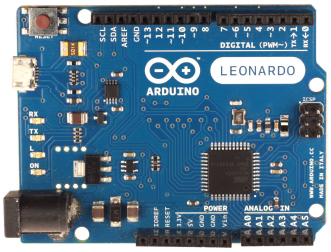


Figura 2.1: Imagen de un Arduino Leonardo como el empleado en el presente proyecto

Arduino es, por una parte una placa compuesta por un microcontrolador, basado generalmente en AVR (una familia RISC [5] fabricada por Atmel®[6])

[5] En arquitectura computacional, RISC (del inglés Reduced Instruction Set Computer, en español Computador con Conjunto de Instrucciones Reducidas) es un tipo de diseño de CPU generalmente utilizado en microprocesadores o microcontroladores.

[6] http://www.atmel.com/

junto con los componentes necesarios para que este funcione (cristales, condensadores, reguladores de voltaje ...) y los adecuados convertir a ésta en una plataforma "lista para usar" (conexión usb, adaptador de corriente, zócalos para más conexiones ...).

Por otra parte, y de un modo complementario, Arduino es un IDE[7] junto a una colección de librerías que facilitan el uso de determinados elementos del mismo.

Existen diferentes versiones de Arduino actualmente en el mercado, desde las que utilizan un chip FTDI para convertir las señales procedentes de un puerto USB al voltaje adecuado para transmitir por la interfaz serie de éste, hasta las que proveen de las capacidades de emular otros periféricos haciendo uso de HID. Todas estas versiones se dividen en dos grandes grupos atendiendo a su *core*. Por una parte están los que poseen uno *AVR* y por la otra los que lo tienen *ARM*. La principal diferencia radica en el ancho de palabra de los registros y CPU, los primeros tienen 8 bits, mientras que los segundos poseen palabras de 32 bits. Las diferencias aquí señaladas no hacen de ellos mejores o peores, sino simplemente mejor adaptados a unas funciones o a otras.

Independientemente de cual sea el procesador, todas las versiones actuales de Arduino compilan y ejecutan código escrito en un subconjunto de C/C++, consiguiendo de este modo que la curva de aprendizaje, una vez conocido este lenguaje sea muy suave.

Los procesadores del tipo *AVR* se caracterizan por un consumo más modesto de energía, además, su *pipeline* tiene dos etapas, cargar y ejecutar, lo que hace que sean de los más rápidos en sus clases. Aunque el conjunto de instrucciones sea común a todos ellos, no sucede lo mismo con los periféricos y conectividad de que disponen. Los hay con diferentes tamaños de RAM, ROM, EEPROM o lista de periféricos.

2.5.1. ¿Por qué Arduino?

Como hemos comentado, Arduino no sólo es la placa y el programa que lo acompaña, sino que existe gran cantidad de librerías (oficiales, o de terceros) que simplifican enormemente el trabajo a la hora de conseguir que las cosas funcionen, centrándonos simplemente en el problema a abordar.

Más concretamente, para el presente proyecto se ha optado por esta plataforma debido a:

[7]del inglés Integrated Development Environment: un Entorno de desarrollo integrado compuesto por una aplicación sobre la que escribir el código que será cargado en la placa, compilarlo, y transferirlo a la placa para que se comience a ejecutar.

- La facilidad de hacer que se comporte como un teclado USB gracias a HID.
- La componente Open Source del proyecto que nos ocupa.
- La gran comunidad de usuarios que actualmente contribuyen a mejorar Arduino día a día.
- La gran cantidad de librerías disponibles contribuidas por la comunidad que facilitan construir sobre él.

Como hemos comentado anteriormente, existen diversos modelos de Arduino, cada uno con sus peculiaridades. Para la elaboración del presente proyecto hemos utilizado básicamente dos de ellos: los modelos *Leonardo* y *Due*. Cada uno de ellos ha realizado una función bien distinta. Por una parte el *Leonardo* es el "alma" del proyecto, ya que ejecuta la mayoría del código escrito, y por la otra tenemos el *Due* que hace las funciones de "comprobador" del primero, realizando tests de estrés y de integración de un modo que veremos más adelante.

2.5.2. Mejoras realizadas a Arduino

Para llevar a cabo la elaboración de este proyecto hemos tenido que superar algunos inconvenientes, entre ellos la internacionalización del componente HID de Arduino.

Éste, y tal como viene distribuido, cuando emula teclados, únicamente conoce la disposición ASCII[8]. Su modo de hacerlo es mediante una tabla que contiene cada uno de los símbolos reconocidos por la librería y el conjunto de teclas al que va asociado dicho símbolo, por ejemplo en español sería:

$$\acute{a} = \acute{} + a$$

el Arduino emitiría el codigo asociado a la tecla [¨] y a continuación el correspondiente a la [A].

Lo mismo ocurre en el caso de mayúsculas ya que emite la pulsación de la tecla 'shift' y sin emitir su liberación emite la pulsación de la letra y por último la liberación de 'shift'.

Dado que las teclas de símbolos en español además no coinciden con las americanas/británicas hemos creado un nuevo mecanismo mediante el cual Arduino puede a partir de ahora emular pulsaciones de teclado en varios idiomas, reconociendo modificadores hasta ahora no implementados (Alt Gr).

[8]Acrónimo inglés de American Standard Code for Information Interchange – Código Estándar Estadounidense para el Intercambio de Información

Se ha añadido además soporte para 'deadkeys' o teclas muertas, que son aquellas que necesitan de la pulsación de una segunda tecla para hacerse visibles. Son ejemplo de estas en castellano, la tilde [´] o el acento circunflejo [^] aunque a no se utilice en nuestro idioma.

La librería `Keyboard`, encargada de estas labores cuenta ahora con una nueva función que permite escoger la disposición del teclado a emular. El soporte no es completo del todo, falta la 'ñ' y los caracteres acentuados, pero admite los 94 caracteres empleados por nuestro proyecto[9]

```
 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW
XYZ!"#$%&'()*+,-./:;<=>?@[\]^_{|}‘
```

2.6. Pantalla SMARTGPU

La pantalla escogida para realizar la interfaz entre Arduino™y el usuario ha sido la SMARTGPU, producida y distribuida por vizic technologies, una empresa mexicana que dispone de varios productos similares, todos ellos enfocados a ser periféricos de placas programables por el usuario como pueden ser Arduino, Netduino o mBed. El modelo escogido se caracteriza por tener una pantalla resistiva de 240x320px con un procesador Atxmega32A4 del mismo fabricante que el chip principal del Arduino, Atmel. Este procesador es similar en características al Atmega32u4 del *Leonardo* siendo sólo destacables diferencias internas de direccionamiento y mayores capacidades para criptografía, que en concreto, no se utilizan para nada.

La interfaz del módulo pantalla con Arduino es un simple puerto serie, por lo que además de las tres plataformas anteriormente mencionadas, puede funcionar sin problemas con las siguientes: 8051, PIC, ATMEL, FREESCALE, STMICRO, ARM, CORTEX, NXP, Fez Panda, FPGA [10]. La principal ventaja de esta pantalla es que al ser controlada por serie, dispone de cierta lógica interna que simplifica el uso de primitivas gráficas como círculos, rectángulos, y precarga de imágenes. Dispone además de una ranura para tarjeta microSD que permite acceder a los ficheros contenidos en la misma, así como crear y modificarlos. Como primera opción, se iba a utilizar ese almacenamiento para el procesado de los datos introducidos por pantalla pero ya que suponía un aumento considerable de la latencia ha sido eliminado del diseño final. Si que se utiliza no obstante para que el usuario cargue en la tarjeta las imágenes que desee para 'firmar' sobre ellas.

[9] Tal vez el lector eche de menos el símbolo '∼' pero este ha sido eliminado a propósito de la lista por ser tratado de modo diferente entre *GNU/Linux* y *Microsoft Windows*®

[10] Según la propia web del fabricante

Posee conjunto de instrucciones formado por tuplas de dos caracteres que le instruyen sobre la acción a realizar, bien sea en el sentido *pantalla* → *Arduino* o viceversa. Esta característica beneficia en gran medida respecto a otras opciones similares pues libera gran cantidad de pins en la placa principal que pueden ser gastados para otros menesteres. Las otras pantallas probadas tenían interfaces paralelas de 8 bits, imposibilitando la comunicación del sistema, hecho que condicionó finalmente para la elección de esta pantalla.

Cabe también mencionar, que pese a tener una *API* bastante bien definida con sus librerías el software interno del módulo no es gpl ni tampoco se puede reemplazar de un modo simple (hecho que se intentó tras contactar con el fabricante para añadir funcionalidades). La librería principal de la pantalla ha sido modificada para añadir funciones de lectura de posición del *stylus* más rápidas que las originales. y se ha corregido desde la librería intermedia una pequeño desfase que tenía la pantalla al leer estas posiciones, pues la pantalla incluye como se ve en la figura 2.2 botones que quedan dentro del área táctil pero no son leídos como parte de esta.

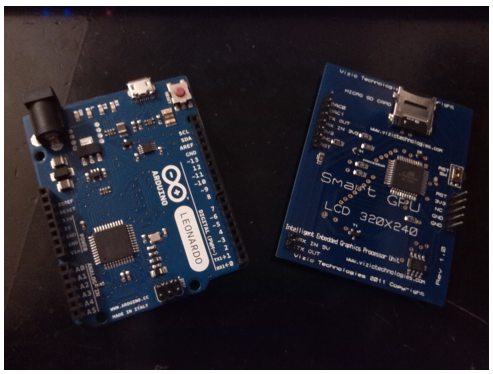


Figura 2.2: Imagen del Arduino Leonardo del proyecto junto a la SMARTGPU

Capítulo 3

Diseño e implementación

En este capítulo procederemos a explicar el problema detectado en la sección de análisis, que resolveremos implementando el diseño.

3.1. Análisis

Las técnicas habituales para generar manualmente claves seguras y diferentes poseen dos grandes debilidades: o son difíciles de recordar, o no son lo suficientemente seguras. Este hecho se agrava a medida que la cantidad de las mismas a memorizar comienza a aumentar. Los recuerdos singulares del usuario, también lo pueden ser de las personas cercanas a él, como colegas, familiares o amigos. Por lo tanto necesitamos un sistema que ayude al usuario a gestionar sus claves. Gestionar contraseñas comprende tanto su generación, como su almacenamiento. Queremos proveer al usuario de un dispositivo que aúne ambas cosas. Que además sea rápido y genere un gran numero de claves diferentes.

3.2. Diseño

Se ha optado por una pantalla táctil a color y un puerto USB, entre otras cosas, por ser dos de los elementos más cotidianos: el primero por simplicidad de uso y el segundo por simplicidad de conexión.

Una vez elegido esto, el paso siguiente fue decidir como se uniría la pantalla al conector, y tras analizar las diferentes opciones disponibles se optó por Arduino: simplicidad de programación, facilidad de interconexión, además del potencial para albergar las características necesarias.

El diseño del presente proyecto comprende por una parte la selección de librerías empleadas en Arduino. Éstas han sido:

- ARDUINOAES256 https://github.com/qistoph/ArduinoAES256
  Implementación de *AES-256* para avr;
- SHA https://github.com/Cathedrow/Cryptosuite
  Incluye tanto *SHA-1* como *SHA-256*
- SMARTGPU http://vizictechnologies.com/#/software-demos-sg/4554679040
  Librería propia de la pantalla táctil desarrollada por el fabricante de la pantalla.

Por otra parte se han elaborado librerías propias que serán compartidas con la comunidad Arduino:

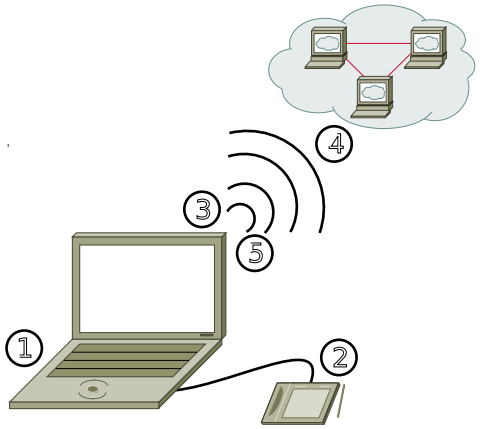
- GRID Es una abstracción sobre SMARTGPU que le añade funcionalidades de cuadrícula y botones.
- PGEN Comprende el núcleo del trabajo realizado en el proyecto, genera y almacena las claves.

PGEN comparte con pwgen la idea básica de la función generadora de claves en su versión *SHA-1*, pero mejorada/adaptada para emplear *AES-256* como seguidamente justificaremos. Es importante destacar que gracias a esta librería se cumplen los requisitos de longitud de la contraseña así como de complejidad. Al igual que en el pwgen original podemos elegir que subconjuntos de caracteres aparecerán en las claves generadas. Para este proyecto hemos elegido el tipo de complejidad más alto, con al menos una mayúscula, al menos un dígito y al menos un símbolo, todo ello con claves de una longitud de 30 caracteres.

El caso básico de uso aquí expuesto se resume en la figura 3.1:

- Un usuario necesita una clave nueva en un sistema
- El usuario conecta el gestor de claves al sistema
- El generador de claves muestra una imagen almacenada en el gestor mediante la pantalla
- El usuario marca puntos sobre la imagen de la pantalla
- El dispositivo genera una clave inicial en base a los puntos marcados mediante *SHA-256*
- El dispositivo muestra una segunda imagen

- El usuario marca puntos sobre la imagen de la pantalla
- El dispositivo genera un flujo de datos en base a los puntos marcados
- El flujo de datos es codificado en con la clave inicial mediante *AES-256*
- El flujo de datos codificados es utilizado como fuente para la generación de una clave dentro del espacio $\mathbb{N}^{94}$
- El dispositivo envía mediante la interfaz HID la clave recién generada.
- La clave es borrada de la memoria.



1. El usuario accede a la página del servicio, ésta le solicita la clave.
2. El usuario marca sobre el gestor de claves determinados puntos. Estos generan una clave que es enviada al ordenador.
3. El ordenador envía la clave al servicio, como hace habitualmente (resumida o recodificada).
4. El servidor autoriza el acceso.
5. El usuario obtiene el acceso que deseaba.

Figura 3.1: Ejemplo de acceso a servicio web

Vamos a analizar ahora la cantidad de claves que pueden ser generadas por nuestro sistema. En caso de no haber restricciones de caracteres con un cálculo rápido vemos que 30 *símbolos* provenientes de un alfabeto de 93 *símbolos* nos da una cardinalidad de

$$93^{30} = 113367470506300106210331419378950643945928686999446579392249$$

La cual es una cantidad nada despreciable de posibles contraseñas que sin duda deben protegernos un poco mejor frente a intentos de intrusión por fuerza bruta. Este cálculo no contempla las restricciones de los subconjuntos de símbolos. Procedemos ahora con nuestro sistema, estos son los subconjuntos:

```
Nombre       Elementos                          Cardinalidad
pw_digits = "0123456789";                             10
pw_uppers = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";             26
pw_lowers = "abcdefghijklmnopqrstuvwxyz";             26
pw_symbols = "!"#$%&'()*+,-./:;<=>?@[\]^_{|}‘";       31
```

Dado que con nuestra restricción debe haber un mínimo de un dígito, una mayúscula y un símbolo, la fórmula a calcular sería

$$N = 10^1 \times 26^1 \times 31^1 \times 93^{27}$$

$$N = 113599037775611930530258484751713752749610585500659462142 0$$

Que sigue sin ser una cantidad nada despreciable.

3.3. Programación

En el desarrollo del presente proyecto hemos tenido que programar en la creación/adaptación de varias librerías cuya documentación no era todo lo buena que debería ser. Este hecho ha causado alguna que otra demora por producir algún que otro desbordamiento, Aquí no hay ‘violaciones de segmento’, directamente deja de funcionar

Es curioso ver que pese a que nuestros instintos y patrones adquiridos nos llevan a una estructuración de código más bien modular, con un acoplamiento normal y una cohesión funcional al tratarse de sistemas empotrados hay que dejar un poco de lado estos grados. No quiere decir que hayamos programado en un único fichero monolítico y con variables globales, pero sí que hemos tenido que renunciar a algunos lujos. Cuando se dispone de 28KB de memoria flash,cada byte cuenta y una declaración de función ‘pasarela’[1] consume una media de 36 bytes, hay que estructurar muy bien el código para no quedarnos sin espacio.

El grueso de la programación se ha realizado en **C++** para las librerías mientras que el desarrollo directo sobre Arduino se ha realizado en un subconjunto de **C++**. Casi todas las librerías manipuladas cumplen con el patrón de diseño `Singleton` (Keyboard,Sha256,Grid,Pgen y SMARTGPU) muy comunes cuando las clases controlan elementos físicos. No han de causar ningún problema pues no tenemos nada de código multi-hilo. La única que no lo tiene es AES-256.

En el anexo Código fuente podremos encontrar el código más relevante del proyecto así como en

[1]Entendiedo función pasarela como la que llama a otras funciones ya agrupadas sin más

- https://github.com/chapuzzo/betterHID
- https://github.com/chapuzzo/Sha
- https://github.com/chapuzzo/grid
- https://github.com/chapuzzo/pgen
- https://github.com/chapuzzo/SMARTGPU

3.4. Construcción

Una vez hemos tenido todos los componentes tanto hardware como software implicados en el proyecto la construcción ha sido bastante sencilla. En la parte física no ha habido ningún inconveniente, ya que todas las piezas encajaban entre sí como un puzzle. No obstante en la parte lógica, queremos recordar que muchas librerías no están pensadas para trabajar entre sí y por eso se han realizado las adaptaciones/modificaciones ya explicadas.

En las siguientes figuras podemos ver fotografías de los elementos que componen el sistema, así como el sistema completo en funcionamiento.

Tal vez se eche de menos una carcasa que cubra todo el montaje, pero al tratarse de una pantalla ya ensamblada, con los componentes ya distribuidos es difícil diseñar una que los cubra todo y a la vez deje acceso a los conectores interiores, así como a la tarjeta microSD que hemos reemplazado varias veces para hacer diferentes pruebas.

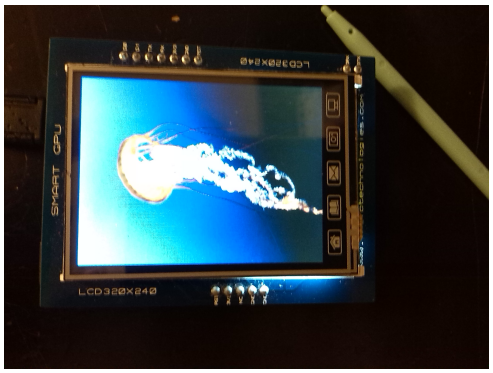


Figura 3.2: Imagen del sistema en funcionamiento

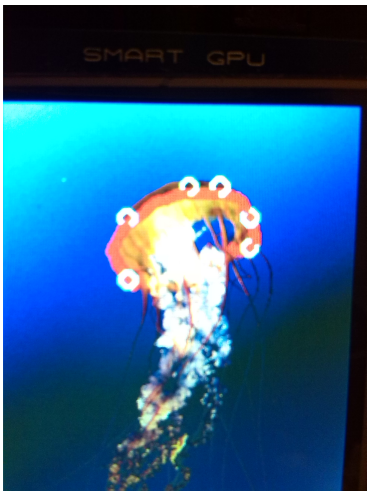


Figura 3.3: Detalle de los puntos marcados en rojo sobre la imagen de fondo

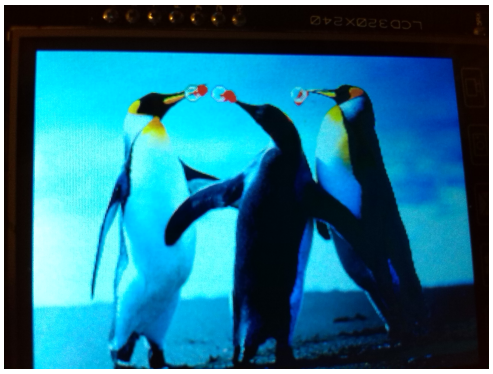


Figura 3.4: Otro detalle de los puntos marcados en rojo sobre la imagen de fondo

Capítulo 4

Evaluación de resultados

Este capítulo comprenderá tanto la verificación del diseño como la validación del resultado obtenido para solucionar el problema analizado mediante la solución propuesta.

4.1. Verificación

Para verificar nuestro diseño hemos abierto un fichero en blanco en un editor de textos y hemos deslizado el *stylus* aleatoriamente por la pantalla mientras veíamos como el fichero se llenaba de diferentes contraseñas que cumplían los parámetros especificados de longitud complejidad y tiempo de generación. Se han realizado además pruebas con un segundo arduino *DUE* conectado por el puerto serie al principal que disponía de claves precomputadas y comprobaba que el desempeño del *Leonardo* fuera el esperado. De estos datos desprendemos que sí que cumple con la verificación.

4.2. Validación

La validación implica que el sistema en conjunto soluciona el problema analizado. Es difícil cuantificar este parámetro, pero el tiempo que tardamos en dibujar sobre la dispositivo y que esta envíe la clave 'tecleada' al Host es despreciable comparado con el tiempo que tardamos en recordar y teclear una clave de similares características. Además al ser el propio usuario el que elige las fotografías que aparecen en la pantalla favorece a su uso.

Podemos pues concluir que sí es válido.

Capítulo 5

Conclusiones

Tras haber finalizado el proyecto podemos valorar aspectos que antes de empezar no teníamos previsto evaluar. Si bien es muy gratificante construir un proyecto tangible y que se puede utilizar en la vida cotidiana de muchas personas, también lo es que este tipo de proyectos tienen muchas las dependencias de las previstas. En este caso, a modo de curiosidad diré que llevo comprados 5 arduinos diferentes y pese a que sólo dos ha intervenido en este proyecto he aprendido un poco de cada uno de ellos.

Al principio, iba a ser algo totalmente distinto, pero como se ha comentado anteriormente, por utilizar un tipo diferente de pantalla con una interfaz mucho más compleja tardó varios meses en despegar. Por coincidencias del destino, en un accidente inesperado aquella pantalla cayó al suelo y su parte resistiva quedó dañada sirviendo únicamente como pantalla pero ya no adecuada para esta empresa. Fue entonces cuando viendo que todas las demás pantallas eran del estilo de aquella encontré esta y pese a valer casi el doble que la primera ha desempeñado su función mejor de lo que se esperaba.

También iba a a ser distinto en el sentido de que utilizaba un Arduino UNO, más limitado en características HID, pero al tener que añadir el soporte de idiomas el espacio se quedaba pequeño. Ahí fue cuando salió a la venta Arduino Leonardo casi de un modo providencial pudiendo hacer que el proyecto fuera sobre ruedas en la parte que le implica.

Concretamente, este proyecto comenzó a realizarse como un trabajo personal fuera del ámbito académico, pero pronto se pensó en convertirlo en algo que pudieran disfrutar los demás y si de paso se puede contribuir con el ecosistema Arduino que tanto aporta, pues mejor.

Como propuesta de futuras/posibles ampliaciones quedaría añadir un menú para seleccionar los parámetros de generación tanto en longitud como en complejidad, así como algún tipo de utilización de la tarjeta de memoria integrada como por ejemplo almacenar contraseñas que no se pueden modificar

y poder tenerlas más a mano dentro del dispositivo.

Respecto a Arduino, actualmente están apareciendo nuevos modelos y es un ambiente en el que me siento muy cómodo trabajando, como ya he dicho, sobre todo por lo tangible y la interacción con el medio.

En lo personal quisiera agradecer una vez más al director del proyecto, por la libertad que me ha concedido desde el primer día y por hacerme ver la luz cuando estaba perdido.

Capítulo 6

Anexos

6.1. Mejoras HID

Aquí tenemos el código modificado en el *core* de arduino en formato diff, para que pueda ser 'parcheado' en futuras versiones del mismo.Además está publicado bajo licencia GPL en https://github.com/chapuzzo/betterHID.git donde probablemente siga recibiendo actualizaciones.

Listing 6.1: HID.cpp.patch

```
diff --git a/HID.cpp b/HID.cpp
index ac63608..985617d 100644
--- a/HID.cpp
+++ b/HID.cpp
@@ -267,152 +267,166 @@ Keyboard_::Keyboard_(void)

 void Keyboard_::begin(void)
 {
+    setKmap(DEFAULT_KMAP);
+}
+
+void Keyboard_::begin(uint8_t kmap)
+{
+    setKmap(kmap);
 }

 void Keyboard_::end(void)
 {
 }

+void Keyboard_::setKmap(uint8_t lang)
+{
+    _kmap = lang;
```

```
+}
+
 void Keyboard_::sendReport(KeyReport* keys)
 {
    HID_SendReport(2,keys,sizeof(KeyReport));
 }

 extern
-const uint8_t _asciimap[128] PROGMEM;
+const uint16_t _asciimap[256] PROGMEM;

-#define SHIFT 0x80
-const uint8_t _asciimap[128] =
+#define SHIFT      0x0100
+#define ALTGR      0x0200
+#define DK         0x0400
+
+const uint16_t _asciimap[256] =
 {
-   0x00,             // NUL
-   0x00,             // SOH
-   0x00,             // STX
-   0x00,             // ETX
-   0x00,             // EOT
-   0x00,             // ENQ
-   0x00,             // ACK
-   0x00,             // BEL
-   0x2a,             // BS     Backspace
-   0x2b,             // TAB    Tab
-   0x28,             // LF     Enter
-   0x00,             // VT
-   0x00,             // FF
-   0x00,             // CR
-   0x00,             // SO
-   0x00,             // SI
-   0x00,             // DEL
-   0x00,             // DC1
-   0x00,             // DC2
-   0x00,             // DC3
-   0x00,             // DC4
-   0x00,             // NAK
-   0x00,             // SYN
-   0x00,             // ETB
-   0x00,             // CAN
-   0x00,             // EM
-   0x00,             // SUB
-   0x00,             // ESC
-   0x00,             // FS
-   0x00,             // GS
```

```
-   0x00,               // RS
-   0x00,               // US
-
-   0x2c,              //  ' '
-   0x1e|SHIFT,        // !
-   0x34|SHIFT,        // "
-   0x20|SHIFT,        // #
-   0x21|SHIFT,        // $
-   0x22|SHIFT,        // %
-   0x24|SHIFT,        // &
-   0x34,              // '
-   0x26|SHIFT,        // (
-   0x27|SHIFT,        // )
-   0x25|SHIFT,        // *
-   0x2e|SHIFT,        // +
-   0x36,              // ,
-   0x2d,              // -
-   0x37,              // .
-   0x38,              // /
-   0x27,              // 0
-   0x1e,              // 1
-   0x1f,              // 2
-   0x20,              // 3
-   0x21,              // 4
-   0x22,              // 5
-   0x23,              // 6
-   0x24,              // 7
-   0x25,              // 8
-   0x26,              // 9
-   0x33|SHIFT,        // :
-   0x33,              // ;
-   0x36|SHIFT,        // <
-   0x2e,              // =
-   0x37|SHIFT,        // >
-   0x38|SHIFT,        // ?
-   0x1f|SHIFT,        // @
-   0x04|SHIFT,        // A
-   0x05|SHIFT,        // B
-   0x06|SHIFT,        // C
-   0x07|SHIFT,        // D
-   0x08|SHIFT,        // E
-   0x09|SHIFT,        // F
-   0x0a|SHIFT,        // G
-   0x0b|SHIFT,        // H
-   0x0c|SHIFT,        // I
-   0x0d|SHIFT,        // J
-   0x0e|SHIFT,        // K
-   0x0f|SHIFT,        // L
-   0x10|SHIFT,        // M
```

```
-   0x11|SHIFT,       // N
-   0x12|SHIFT,       // O
-   0x13|SHIFT,       // P
-   0x14|SHIFT,       // Q
-   0x15|SHIFT,       // R
-   0x16|SHIFT,       // S
-   0x17|SHIFT,       // T
-   0x18|SHIFT,       // U
-   0x19|SHIFT,       // V
-   0x1a|SHIFT,       // W
-   0x1b|SHIFT,       // X
-   0x1c|SHIFT,       // Y
-   0x1d|SHIFT,       // Z
-   0x2f,             // [
-   0x31,             // bslash
-   0x30,             // ]
-   0x23|SHIFT,       // ^
-   0x2d|SHIFT,       // _
-   0x35,             // `
-   0x04,             // a
-   0x05,             // b
-   0x06,             // c
-   0x07,             // d
-   0x08,             // e
-   0x09,             // f
-   0x0a,             // g
-   0x0b,             // h
-   0x0c,             // i
-   0x0d,             // j
-   0x0e,             // k
-   0x0f,             // l
-   0x10,             // m
-   0x11,             // n
-   0x12,             // o
-   0x13,             // p
-   0x14,             // q
-   0x15,             // r
-   0x16,             // s
-   0x17,             // t
-   0x18,             // u
-   0x19,             // v
-   0x1a,             // w
-   0x1b,             // x
-   0x1c,             // y
-   0x1d,             // z
-   0x2f|SHIFT,       // {
-   0x31|SHIFT,       // |
-   0x30|SHIFT,       // }
-   0x35|SHIFT,       // ~
```

```
-   0                 // DEL
+/* 000 0x000 */     0x00,           0x00,              // NUL
+/* 001 0x001 */     0x00,           0x00,              // SOH
+/* 002 0x002 */     0x00,           0x00,              // STX
+/* 003 0x003 */     0x00,           0x00,              // ETX
+/* 004 0x004 */     0x00,           0x00,              // EOT
+/* 005 0x005 */     0x00,           0x00,              // ENQ
+/* 006 0x006 */     0x00,           0x00,              // ACK
+/* 007 0x007 */     0x00,           0x00,              // BEL
+/* 008 0x008 */     0x2a,           0x2a,              // BS
   Backspace
+/* 009 0x009 */     0x2b,           0x2b,              // TAB
   Tab
+/* 010 0x00A */     0x28,           0x28,              // LF
   Enter
+/* 011 0x00B */     0x00,           0x00,              // VT
+/* 012 0x00C */     0x00,           0x00,              // FF
+/* 013 0x00D */     0x00,           0x00,              // CR
+/* 014 0x00E */     0x00,           0x00,              // SO
+/* 015 0x00F */     0x00,           0x00,              // SI
+/* 016 0x010 */     0x00,           0x00,              // DEL
+/* 017 0x011 */     0x00,           0x00,              // DC1
+/* 018 0x012 */     0x00,           0x00,              // DC2
+/* 019 0x013 */     0x00,           0x00,              // DC3
+/* 020 0x014 */     0x00,           0x00,              // DC4
+/* 021 0x015 */     0x00,           0x00,              // NAK
+/* 022 0x016 */     0x00,           0x00,              // SYN
+/* 023 0x017 */     0x00,           0x00,              // ETB
+/* 024 0x018 */     0x00,           0x00,              // CAN
+/* 025 0x019 */     0x00,           0x00,              // EM
+/* 026 0x01A */     0x00,           0x00,              // SUB
+/* 027 0x01B */     0x00,           0x00,              // ESC
+/* 028 0x01C */     0x00,           0x00,              // FS
+/* 029 0x01D */     0x00,           0x00,              // GS
+/* 030 0x01E */     0x00,           0x00,              // RS
+/* 031 0x01F */     0x00,           0x00,              // US
+
+/* 032 0x020 */     0x2c,           0x2c,             // ' '
+/* 033 0x021 */     0x1e|SHIFT,     0x1e|SHIFT,       // !
+/* 034 0x022 */     0x34|SHIFT,     0x1f|SHIFT,       // "
+/* 035 0x023 */     0x20|SHIFT,     0x20|ALTGR,       // #
+/* 036 0x024 */     0x21|SHIFT,     0x21|SHIFT,       // $
+/* 037 0x025 */     0x22|SHIFT,     0x22|SHIFT,       // %
+/* 038 0x026 */     0x24|SHIFT,     0x23|SHIFT,       // &
+/* 039 0x027 */     0x34,           0x2d,             // '
+/* 040 0x028 */     0x26|SHIFT,     0x25|SHIFT,       // (
+/* 041 0x029 */     0x27|SHIFT,     0x26|SHIFT,       // )
+/* 042 0x02A */     0x25|SHIFT,     0x30|SHIFT,       // *
+/* 043 0x02B */     0x2e|SHIFT,     0x30,             // +
```

```
+/* 044 0x02C */    0x36,          0x36,          // ,
+/* 045 0x02D */    0x2d,          0x38,          // -
+/* 046 0x02E */    0x37,          0x37,          // .
+/* 047 0x02F */    0x38,          0x24|SHIFT,    // /
+/* 048 0x030 */    0x27,          0x27,          // 0
+/* 049 0x031 */    0x1e,          0x1e,          // 1
+/* 050 0x032 */    0x1f,          0x1f,          // 2
+/* 051 0x033 */    0x20,          0x20,          // 3
+/* 052 0x034 */    0x21,          0x21,          // 4
+/* 053 0x035 */    0x22,          0x22,          // 5
+/* 054 0x036 */    0x23,          0x23,          // 6
+/* 055 0x037 */    0x24,          0x24,          // 7
+/* 056 0x038 */    0x25,          0x25,          // 8
+/* 057 0x039 */    0x26,          0x26,          // 9
+/* 058 0x03A */    0x33|SHIFT,    0x37|SHIFT,    // :
+/* 059 0x03B */    0x33,          0x36|SHIFT,    // ;
+/* 060 0x03C */    0x36|SHIFT,    0x64,          // <
+/* 061 0x03D */    0x2e,          0x27|SHIFT,    // =
+/* 062 0x03E */    0x37|SHIFT,    0x64|SHIFT,    // >
+/* 063 0x03F */    0x38|SHIFT,    0x2d|SHIFT,    // ?
+/* 064 0x040 */    0x1f|SHIFT,    0x1f|ALTGR,    // @
+/* 065 0x041 */    0x04|SHIFT,    0x04|SHIFT,    // A
+/* 066 0x042 */    0x05|SHIFT,    0x05|SHIFT,    // B
+/* 067 0x043 */    0x06|SHIFT,    0x06|SHIFT,    // C
+/* 068 0x044 */    0x07|SHIFT,    0x07|SHIFT,    // D
+/* 069 0x045 */    0x08|SHIFT,    0x08|SHIFT,    // E
+/* 070 0x046 */    0x09|SHIFT,    0x09|SHIFT,    // F
+/* 071 0x047 */    0x0a|SHIFT,    0x0a|SHIFT,    // G
+/* 072 0x048 */    0x0b|SHIFT,    0x0b|SHIFT,    // H
+/* 073 0x049 */    0x0c|SHIFT,    0x0c|SHIFT,    // I
+/* 074 0x04A */    0x0d|SHIFT,    0x0d|SHIFT,    // J
+/* 075 0x04B */    0x0e|SHIFT,    0x0e|SHIFT,    // K
+/* 076 0x04C */    0x0f|SHIFT,    0x0f|SHIFT,    // L
+/* 077 0x04D */    0x10|SHIFT,    0x10|SHIFT,    // M
+/* 078 0x04E */    0x11|SHIFT,    0x11|SHIFT,    // N
+/* 079 0x04F */    0x12|SHIFT,    0x12|SHIFT,    // O
+/* 080 0x050 */    0x13|SHIFT,    0x13|SHIFT,    // P
+/* 081 0x051 */    0x14|SHIFT,    0x14|SHIFT,    // Q
+/* 082 0x052 */    0x15|SHIFT,    0x15|SHIFT,    // R
+/* 083 0x053 */    0x16|SHIFT,    0x16|SHIFT,    // S
+/* 084 0x054 */    0x17|SHIFT,    0x17|SHIFT,    // T
+/* 085 0x055 */    0x18|SHIFT,    0x18|SHIFT,    // U
+/* 086 0x056 */    0x19|SHIFT,    0x19|SHIFT,    // V
+/* 087 0x057 */    0x1a|SHIFT,    0x1a|SHIFT,    // W
+/* 088 0x058 */    0x1b|SHIFT,    0x1b|SHIFT,    // X
+/* 089 0x059 */    0x1c|SHIFT,    0x1c|SHIFT,    // Y
+/* 090 0x05A */    0x1d|SHIFT,    0x1d|SHIFT,    // Z
+/* 091 0x05B */    0x2f,          0x2f|ALTGR,    // [
+/* 092 0x05C */    0x31,          0x35|ALTGR,    // bslash
```

```
+/* 093 0x05D */    0x30,          0x30|ALTGR,      // ]
+/* 094 0x05E */    0x23|SHIFT,    0x2f|SHIFT|DK,   // ^
+/* 095 0x05F */    0x2d|SHIFT,    0x38|SHIFT,      // _
+/* 096 0x060 */    0x35,          0x2f|DK,         // `
+/* 097 0x061 */    0x04,          0x04,            // a
+/* 098 0x062 */    0x05,          0x05,            // b
+/* 099 0x063 */    0x06,          0x06,            // c
+/* 100 0x064 */    0x07,          0x07,            // d
+/* 101 0x065 */    0x08,          0x08,            // e
+/* 102 0x066 */    0x09,          0x09,            // f
+/* 103 0x067 */    0x0a,          0x0a,            // g
+/* 104 0x068 */    0x0b,          0x0b,            // h
+/* 105 0x069 */    0x0c,          0x0c,            // i
+/* 106 0x06A */    0x0d,          0x0d,            // j
+/* 107 0x06B */    0x0e,          0x0e,            // k
+/* 108 0x06C */    0x0f,          0x0f,            // l
+/* 109 0x06D */    0x10,          0x10,            // m
+/* 110 0x06E */    0x11,          0x11,            // n
+/* 111 0x06F */    0x12,          0x12,            // o
+/* 112 0x070 */    0x13,          0x13,            // p
+/* 113 0x071 */    0x14,          0x14,            // q
+/* 114 0x072 */    0x15,          0x15,            // r
+/* 115 0x073 */    0x16,          0x16,            // s
+/* 116 0x074 */    0x17,          0x17,            // t
+/* 117 0x075 */    0x18,          0x18,            // u
+/* 118 0x076 */    0x19,          0x19,            // v
+/* 119 0x077 */    0x1a,          0x1a,            // w
+/* 120 0x078 */    0x1b,          0x1b,            // x
+/* 121 0x079 */    0x1c,          0x1c,            // y
+/* 122 0x07A */    0x1d,          0x1d,            // z
+/* 123 0x07B */    0x2f|SHIFT,    0x34|ALTGR,      // { ****
+/* 124 0x07C */    0x31|SHIFT,    0x1e|ALTGR,      // |
+/* 125 0x07D */    0x30|SHIFT,    0x31|ALTGR,      // }
+/* 126 0x07E */    0x35|SHIFT,    0x21|ALTGR,      // ~
+/* 127 0x07F */    0,             0                // DEL
 };

 uint8_t USBPutChar(uint8_t c);
@@ -421,29 +435,50 @@ uint8_t USBPutChar(uint8_t c);
 // to the persistent key report and sends the report.
    Because of the way
 // USB HID works, the host acts like the key remains pressed
     until we
 // call release(), releaseAll(), or otherwise clear the
    report and resend.
-size_t Keyboard_::press(uint8_t k)
+size_t Keyboard_::press(uint8_t in)
 {
-   uint8_t i;
```

```
-    if (k >= 136) {    // it's a non-printing key (not a
    modifier)
-        k = k - 136;
-    } else if (k >= 128) {    // it's a modifier key
-        _keyReport.modifiers |= (1<<(k-128));
+    uint16_t k;
+    uint8_t wasDeadKey = 0;
+
+    if (in >= 136) {    // it's a non-printing key (not a
    modifier)
+        k = in - 136;
+    } else if (in >= 128) {    // it's a modifier key
+        _keyReport.modifiers |= (1<<(in-128));
         k = 0;
     } else {    // it's a printing key
-        k = pgm_read_byte(_asciimap + k);
+        k = pgm_read_word(_asciimap + in * 2 + _kmap);
         if (!k) {
             setWriteError();
             return 0;
         }
-        if (k & 0x80) {    // it's a character reached with
    shift
-            _keyReport.modifiers |= 0x02;    // the left
    shift modifier
-            k &= 0x7F;
+        if (k & SHIFT) {
+            _keyReport.modifiers |= KEY_MODIFIER_LEFT_SHIFT;
+            k &= ~SHIFT;
+        }
+        if (k & ALTGR) {
+            _keyReport.modifiers |= KEY_MODIFIER_RIGHT_ALT;
+            k &= ~ALTGR;
+        }
+        if (k & DK) {
+            k &= ~DK;
+            wasDeadKey = 1;
         }
     }
-
+    if (!addToReport(k))
+        return 0;
+    sendReport(&_keyReport);
+    if (wasDeadKey){
+        write(' ');
+    }
+    return 1;
+}
+
```

```
+uint8_t Keyboard_::addToReport(uint8_t k)
+{
+   uint8_t i;
    // Add k to the key report only if it's not already
       present
    // and if there is an empty slot.
-   if (_keyReport.keys[0] != k && _keyReport.keys[1] != k &&
+   if (_keyReport.keys[0] != k && _keyReport.keys[1] != k &&
        _keyReport.keys[2] != k && _keyReport.keys[3] != k &&
        _keyReport.keys[4] != k && _keyReport.keys[5] != k) {

@@ -458,41 +493,52 @@ size_t Keyboard_::press(uint8_t k)
            return 0;
        }
    }
-   sendReport(&_keyReport);
    return 1;
 }

+
 // release() takes the specified key out of the persistent
    key report and
 // sends the report.  This tells the OS the key is no longer
     pressed and that
 // it shouldn't be repeated any more.
-size_t Keyboard_::release(uint8_t k)
+size_t Keyboard_::release(uint8_t in)
 {
-   uint8_t i;
-   if (k >= 136) {    // it's a non-printing key (not a
   modifier)
-       k = k - 136;
-   } else if (k >= 128) {  // it's a modifier key
-       _keyReport.modifiers &= ~(1<<(k-128));
+   uint16_t k;
+   if (in >= 136) {    // it's a non-printing key (not a
   modifier)
+       k = in - 136;
+   } else if (in >= 128) { // it's a modifier key
+       _keyReport.modifiers &= ~(1<<(in-128));
        k = 0;
    } else {    // it's a printing key
-       k = pgm_read_byte(_asciimap + k);
+       k = pgm_read_word(_asciimap + in * 2 + _kmap);
        if (!k) {
            return 0;
        }
-       if (k & 0x80) {    // it's a character reached with
   shift
```

```
-            _keyReport.modifiers &= ~(0x02);   // the left
   shift modifier
-            k &= 0x7F;
+        if (k & SHIFT) {
+            _keyReport.modifiers &= ~KEY_MODIFIER_LEFT_SHIFT;
+            k &= ~SHIFT;
+        }
+        if (k & ALTGR) {
+            _keyReport.modifiers &= ~KEY_MODIFIER_RIGHT_ALT;
+            k &= ~ALTGR;
+        }
+        if (k & DK) {
+            k &= ~DK;
         }
     }
-
+    removeFromReport(k);
+    sendReport(&_keyReport);
+    return 1;
+}
+
+uint8_t Keyboard_::removeFromReport(uint8_t k){
+
     // Test the key report to see if k is present.  Clear it
        if it exists.
     // Check all positions in case the key is present more
        than once (which it shouldn't be)
-    for (i=0; i<6; i++) {
+    for (int i=0; i<6; i++) {
         if (0 != k && _keyReport.keys[i] == k) {
             _keyReport.keys[i] = 0x00;
         }
     }
-
-    sendReport(&_keyReport);
     return 1;
 }

@@ -511,10 +557,11 @@ void Keyboard_::releaseAll(void)
 size_t Keyboard_::write(uint8_t c)
 {
     uint8_t p = press(c);       // Keydown
-    uint8_t r = release(c);     // Keyup
+    release(c);     // Keyup
     return (p);     // just return the result of press() since
         release() almost always returns 1
 }

 #endif
```

Listing 6.2: USBAPI.h.patch

```
diff --git a/USBAPI.h b/USBAPI.h
index eb2e593..a01157d 100644
--- a/USBAPI.h
+++ b/USBAPI.h
@@ -111,6 +111,21 @@ extern Mouse_ Mouse;
 #define KEY_F11                 0xCC
 #define KEY_F12                 0xCD

+
+#define KEY_MODIFIER_LEFT_CTRL     0x01
+#define KEY_MODIFIER_LEFT_SHIFT    0x02
+#define KEY_MODIFIER_LEFT_ALT      0x04
+#define KEY_MODIFIER_LEFT_GUI      0x08
+#define KEY_MODIFIER_RIGHT_CTRL    0x010
+#define KEY_MODIFIER_RIGHT_SHIFT   0x020
+#define KEY_MODIFIER_RIGHT_ALT     0x040
+#define KEY_MODIFIER_RIGHT_GUI     0x080
+
+/*kmap options*/
+#define EN 0x00
+#define ES 0x01
+#define DEFAULT_KMAP ES
+
 // Low level key report: up to 6 keys and shift, ctrl etc at
     once
 typedef struct
 {
@@ -123,15 +138,20 @@ class Keyboard_ : public Print
 {
 private:
    KeyReport _keyReport;
+   uint8_t _kmap;
    void sendReport(KeyReport* keys);
+   uint8_t addToReport(uint8_t k);
+   uint8_t removeFromReport(uint8_t k);
 public:
    Keyboard_(void);
    void begin(void);
+   void begin(uint8_t kmap);
    void end(void);
    virtual size_t write(uint8_t k);
    virtual size_t press(uint8_t k);
    virtual size_t release(uint8_t k);
    virtual void releaseAll(void);
+   virtual void setKmap(uint8_t kmap);
 };
```

```
extern Keyboard_ Keyboard;
```

6.2. Código fuente

Listing 6.3: pgen.h

```
#ifndef _PGEN_H_
#define _PGEN_H_

#include "Arduino.h"
#include "aes256.h"
#include "sha256.h"

/*
 * Flags for the pwgen function
 */
#define PW_WEAK      0x0000  /* All lower */
#define PW_DIGITS    0x0001  /* At least one digit */
#define PW_UPPERS    0x0002  /* At least one upper letter */
#define PW_SYMBOLS   0x0004
#define PW_STRONG   (PW_DIGITS|PW_UPPERS|PW_SYMBOLS)

class pgen_
{
private:
  aes256_context ctxt;
  int shasum_idx;
  uint8_t my_key[32];
  uint8_t my_data[32];
  uint8_t pw_sha_number(uint8_t max_num);
public:
  pgen_(void);
  void init(uint8_t *key);
  void feed(uint8_t *data);
  char generate(char *buf, int size, int pw_flags);
  void clear(void);
};
extern pgen_ pgen;

#endif
```

Listing 6.4: pgen.cpp

```
#include "Arduino.h"
#include "pgen.h"
```

```
const char *pw_digits = "0123456789";
const char *pw_uppers = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
const char *pw_lowers = "abcdefghijklmnopqrstuvwxyz";
const char *pw_symbols = "!\"#$%&'()*+,-./:;<=>?@[\\]^_{|}`";

/*void printHash(uint8_t* hash) {
  int i;
  for (i=0; i<32; i++) {
    Serial.print("0123456789abcdef"[hash[i]>>4]);
    Serial.print("0123456789abcdef"[hash[i]&0xf]);
  }
  Serial.println();
}*/

pgen_::pgen_(void){
    shasum_idx=32;
}

void pgen_::init(uint8_t *key){
    Sha256.init();
    Sha256.print((char*)key);
    memcpy(my_key, Sha256.result(), 32);
    //aes256_init(&ctxt,key);
}

void pgen_::feed(uint8_t *data){
    Sha256.init();
    Sha256.print((char*)data);
    memcpy(my_data, Sha256.result(), 16);
    aes256_init(&ctxt,my_key);
    aes256_encrypt_ecb(&ctxt, my_data);
    aes256_done(&ctxt);
    Sha256.init();
    Sha256.print((char*)my_data);
}


void pgen_::clear(void){
    shasum_idx=32;
}

const char *sha_magic="minervaguapa";

uint8_t pgen_::pw_sha_number(uint8_t max_num){
    uint8_t val=0;
    if (shasum_idx>31) {
        shasum_idx = 0;
        Sha256.print(sha_magic);
```

```
    }
    //val = (int) (Sha256.result()[shasum_idx++] / ((float)
       256) * max_num);
    val = (Sha256.result()[shasum_idx++])%max_num;
    return val;
}


char pgen_::generate(char *buf, int size, int pw_flags)
{
    char    ch, *chars, *wchars, feature_flags;
    int     i, len;
    shasum_idx = 31;
    len = 0;
    if (pw_flags & PW_DIGITS) {
        len += strlen(pw_digits);
    }
    if (pw_flags & PW_UPPERS) {
        len += strlen(pw_uppers);
    }
    len += strlen(pw_lowers);
    if (pw_flags & PW_SYMBOLS) {
        len += strlen(pw_symbols);
    }
        chars = (char*) calloc(len+1,sizeof(char));
        if (!chars) {
            return 0;
        }
    wchars = chars;
    if (pw_flags & PW_DIGITS) {
        strcpy(wchars, pw_digits);
        wchars += strlen(pw_digits);
    }
    if (pw_flags & PW_UPPERS) {
        strcpy(wchars, pw_uppers);
        wchars += strlen(pw_uppers);
    }
    strcpy(wchars, pw_lowers);
    wchars += strlen(pw_lowers);
    if (pw_flags & PW_SYMBOLS) {
        strcpy(wchars, pw_symbols);
    }
try_again:
    len = strlen(chars);
    feature_flags = pw_flags;
    i = 0;
    while (i < size) {
        ch = chars[pw_sha_number(len)];
        buf[i++] = ch;
```

```
        if (strchr(pw_digits, ch))
            feature_flags &= ~PW_DIGITS;
        if (strchr(pw_uppers, ch))
            feature_flags &= ~PW_UPPERS;
        if (strchr(pw_symbols, ch))
            feature_flags &= ~PW_SYMBOLS;
    }
    if (feature_flags & (PW_UPPERS | PW_DIGITS | PW_SYMBOLS))
        goto try_again;
    buf[size] = 0;
    free(chars);
    return 1;
}
pgen_ pgen;
```

Listing 6.5: grid.h

```
#ifndef _SGLEO_H_GRID_
#define _SGLEO_H_GRID_

#include "Arduino.h"

class grid_
{
private:
  uint8_t *_grid;
  uint16_t _n;
  uint16_t _x,_y;
public:
  grid_(uint16_t x, uint16_t y);
  ~grid_(void);
  void marca(uint16_t x, uint16_t y);
  uint8_t* res(void);
  void borra(void);
};
extern grid_ grid;

#endif
```

Listing 6.6: grid.cpp

```
#include "Arduino.h"
#include "grid.h"

grid_::grid_(uint16_t x, uint16_t y){
  _grid = (uint8_t*)calloc(x*y+1,sizeof(uint8_t));
  _x = x;
  _y = y;
```

```
  for (uint16_t i=0; i<_x*_y; i++)
    _grid[i] = 1;
  _n = 2;
}

void grid_::marca(uint16_t x, uint16_t y){
  if (_grid[x*_y+y] == 1)
    _grid[x*_y+y] = _n++;
}

uint8_t* grid_::res(void){
  return _grid;
}

grid_::~grid_(void){
    free(_grid);
}

void grid_::borra(void){
    for (uint16_t i=0; i<_x*_y; i++)
    _grid[i] = 1;
  _n = 2;
}
//grid_ grid;
```

Listing 6.7: pfc_sketch.h

```
#ifndef _PFC_H_
#define _PFC_H_

#include "Arduino.h"

#ifdef __cplusplus
extern "C" {
#endif

  typedef struct{
    uint16_t x;
    uint16_t y;
  } pos_;

  union u{
    pos_ p;
    uint16_t xy[2];
  } touch;

#ifdef __cplusplus
}
```

```
#endif

#endif
```

Listing 6.8: pfc.ino

```
#include "SMARTGPU.h"
#include "pfc_sketch.h"
#include "sha256.h"
#include "aes256.h"
#include "grid.h"
#include "pgen.h"

SMARTGPU lcd;

#define CHUNK_SIZE 30
#define CIRC_RATIO 2/6

int screen = 1;
char password[35];

grid_ grid(WIDTH/CHUNK_SIZE,HEIGHT/CHUNK_SIZE);

void printHash(uint8_t* hash) {
  int i;
  for (i=0; i<(WIDTH/CHUNK_SIZE)*(HEIGHT/CHUNK_SIZE); i++) {
    Serial.print("0123456789abcdef"[hash[i]>>4]);
    Serial.print("0123456789abcdef"[hash[i]&0xf]);
  }
  Serial.println();
}

void setup() {
  Serial.begin(9600);
  Keyboard.begin(ES);
  startScreen();
}

void loop(){
  dispatchInput();
}

void startScreen(){
  lcd.start();
  lcd.baudChange(1000000);
  firstScreen();
}
```

```
void firstScreen(){
  screen = 1;
  pgen.clear();
  lcd.orientation(LANDSCAPEL);
  lcd.imageSD(0,0,(char*)"prime");
}

void secondScreen(){
  screen = 2;
  lcd.imageSD(0,0,(char*)"segun");
}

void dispatchInput(){
  uint8_t buff[5];

lcd.touchRaw(buff);
  if (buff[0] == 0x4E){
    //la mayoria del tiempo esta aqui: sin pulsaciones.
  }
  else if ((buff[0]<<8 | buff[1] ) < 0x0200){
    // inspiracion aja de las 2:40 de la madrugada!!
    touch.p.x=map(buff[0]<<8|buff[1],0,320,0,330);
    touch.p.y=buff[2]<<8|buff[3];

    // descartamos mediciones invalidas
    if (touch.p.x > WIDTH  || touch.p.x < 0) return;
    if (touch.p.y > HEIGHT || touch.p.y < 0) return;

    // procesamos input
    marca(touch.p);
  }
  else {
    Keyboard.begin();
    if (!strcmp((char*)buff,"PCMO")){
      //Keyboard.print(alfabeto);
    }
    if (!strcmp((char*)buff,"CAME")){
      grid.borra();
      printHash(grid.res());
      firstScreen();
    }
    if (!strcmp((char*)buff,"BOOK")){
      Keyboard.print("BOOK");
    }
    if (!strcmp((char*)buff,"HOME")){
      Serial.println("home");
      if (screen == 1){
        Serial.println("1");
        pgen.init(grid.res());
```

```
        lcd.bright(0x00);
        grid.borra();
        secondScreen();
        Serial.println("home");
      }
      else{
        Serial.println("2");
        pgen.feed(grid.res());
        pgen.generate(password,30,PW_STRONG);
        Keyboard.println(password);
        lcd.bright(0x00);
        grid.borra();
        secondScreen();
      }
    }
    if (!strcmp((char*)buff,"MESG")){
      Keyboard.print("MESG");
    }
    Keyboard.end();
    delay(700);
    lcd.bright(0x33);
  }
}

void marca(pos_ p){
  const uint8_t chunks= CHUNK_SIZE;
  const uint8_t radius= chunks*CIRC_RATIO;

  uint16_t newX,newY;
  uint8_t chunkX,chunkY;

  chunkX = p.x/chunks;
  chunkY = p.y/chunks;

  newX= constrain(chunkX*chunks+chunks/2,0,WIDTH-1);
  newY= constrain(chunkY*chunks+chunks/2,0,HEIGHT-1);

  lcd.drawCircle(p.x, p.y, 3, RED, FILL);
  if (isInCircle(p,chunks,radius)){
    lcd.drawCircle(newX, newY, radius, random(), UNFILL);
    lcd.drawCircle(newX, newY, radius-1, WHITE, UNFILL);
    lcd.drawCircle(newX, newY, radius-2, WHITE, UNFILL);
    grid.marca(chunkX,chunkY);
  }
}

// (x-a)**2 + (y-b)**2 = r**2;
uint8_t isInCircle(pos_ p, uint8_t gridsize, uint8_t radius){
  uint8_t relX = p.x%gridsize,
```

```
  relY = p.y%gridsize,
  center = gridsize/2;
  if ((relX-center)*(relX-center) + (relY-center)*(relY-
     center) <= radius*radius)
    return 1;
  return 0;
}
```

Bibliografía

[1] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. In *First Advanced Encryption Standard (AES) Conference*, 1998.

[2] NIST. Advanced encryption standard (AES). *Federal Information Processing Standards Publication, FIPS PUB*, 197, 2001.

[3] Arduino Team. Arduino - Home Page.

[4] Theodore Ts'o. *pwgen*, chapter 3. Linux Manpages, 2006.