6.115 Final Project: Password Keeper

Ben Kettle

May 12, 2020

Contents

1 System Overview

Far too many people use the same password for all their accounts, making small adjustments when the sites force them to change it. Software password managers are a good solution, but restrictive work computers may not allow installation of extra software. Even users who are able to install a password manager are often overwhelmed by the wide variety of options, and many are rightfully hesitant to allow an external entity access to the passwords to their bank accounts, email, and more. To try to help solve this problem, encourage password storage via offline media, and learn about USB, security, and encryption, I built a prototype hardware password keeper for my final project.

With this motivation, I started this project with a few goals in mind.

- The device should have an intuitive, user-friendly interface.
- The device should allow transport of passwords across computers without need for cloud storage or USB backups.
- The device should be secure and disconnected from the internet.
- The core functionality should be usable on any computer without extra software.

To address these goals, I decided on a few key characteristics. The central idea of the device is password entry by HID keyboard emulation. The user will select a set of credentials, referred to as ”records”, via an onboard OLED screen and a rotary encoder. After selecting a record, they will be able to instruct the device to type out either the username or password, or view the credential on the screen—for example, to allow use with a smartphone.

Many security-conscious users are also likely to use a second-factor authorization (2FA) for many websites. Though it would be unwise to have these codes in the same location as the website’s passwords, this is important functionality to many users. To this end, I planned to allow records to include the keys to generate these one-time authorization codes based on the current time by implementing the industry-standard TOTP algorithm.

Of course, users also need to be able to manage which records are stored on the device. I planned to implement this through emulating a virtual COM port and providing a terminal-based interface with options to add and modify records. The device would present a series of menus that the user could interact with by entering simple commands or arguments, much like the configuration scripts of many command-line programs. Though this solution is not as intuitive as a dedicated piece of software, it requires at most the installation of a driver and one of many terminal programs. No dedicated software will take up space on the user’s computer, and all processing is kept contained to the PSoC stick that works as the main controller.

To simplify and add functionality to this process, I planned to implement a password generation feature as part of this terminal-based interface. Rather than entering a password manually,

the user should be able to instruct the PSoC to generate a password within its own hardware.

Another important element is the storage of records themselves on the device. Clearly, the records contain sensitive information, so having them stored in a way that anyone could read would not be acceptable. I initially planned to store the records in flash on the PSoC through use of the EmEEPROM component, but soon realized that this would not be the best solution. Not only would storage space be limited due to the relatively large size of each record, but the limited read-write cycles of the flash memory would be best to avoid in extended use.

I decided instead on storing the records in external memory, specifically an SD card. SD cards are obviously easy to remove and read externally, so records will be encrypted before they are loaded onto the SD card. The only time the passwords are stored in plaintext will be in RAM on the PSoC, after the user has entered their PIN.

While each of these elements operates somewhat independently, the device-side user interface ties all aspects together, requesting information from the other modules as requested by the user. Together, they will form a prototype of a cohesive system that offers a solution to several security issues in the general public.

2 Hardware Requirements

In order to enable each of the functions listed above, several hardware components were needed. The specific way they were wired in my project can be seen in Figure 5, and I will go into more detail about each one in this section.

1. to serve as the main controller for the modules, a **PSoC 5LP CY8CKIT-059**.
2. to provide feedback to the user, an **OLED Display** based on the SSD1306 driver.
3. to allow user input, a **rotary encoder with push switch**.
4. to hold the current time through power losses, a **DS3231 RTC** with battery backup.
5. to store records and other information, a **MicroSD reader** supporting 5v-3.3v conversion.

2.1 PSOC 5LP

As the main microcontroller for the project, I used the PSoC 5LP CY8CKIT-059. Its capabilities will be described in the next several sections. Any microcontroller supporting I2C, SPI, and that has several additional inputs could be used, but the PSoC's clear datasheets for its components as well as flexible UDB system allowing for implementation of extra hardware components help smooth the prototyping process.

2.2 OLED Display

Requirements for the user interface were fairly simple—I planned to have the only input be the rotary encoder, so scrolling in a single direction was the only mechanism for making inputs on the device. Because of this, feedback to the user would be fairly simple, and I chose to embrace this simplicity with a small OLED display.

The SSD1306 controller embedded in the module I chose supports several modes of operation, including 8000-style parallel, 3- and 4-wire SPI, as well as I2C. However, only the I2C pins

are broken out on the model I purchased, so this is what I used. As described in the SSD1306 datasheet,[1] the controller responds to an initial transmission called a ”start condition,” where SDA pulled from HIGH to LOW while SCL is HIGH. This is followed by a 1-byte transmission of the display's I2C address (`0x3C` or `0x3D`) along with a read/write bit. For our usage, we never need to read from the display, so this bit is always LOW for write. The device responds with an acknowledgement bit (ACK) by pulling SDA LOW.

After this, the controller is in write mode and data can be sent. It accepts either a ”control” or ”data” byte, where a control byte defines whether the next byte is a command or data. Commands allow for setup of the controller, and importantly allow setting the start address in the device's internal RAM for the next data that will be sent. The internal RAM has one bit for each onscreen bit, arranged into 128 columns and 64 rows to mimic the arrangement of the pixels themselves. Because of this, setting the start address is fairly easy, and corresponds with exactly which part of the internal buffer needs to be updated. Data bytes, on the other hand, are simply stored into internal RAM from wherever the current start location is. The controller continues to accept data until it recieves a ”stop condition”, defined as pulling SDA from LOW to HIGH while SCL is HIGH.

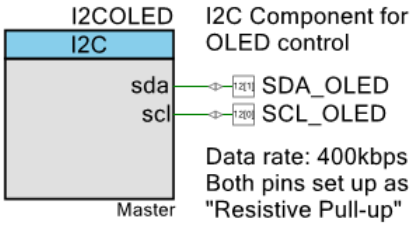


Figure 1: I2C component setup

Fortunately, however, this format is for the most part the I2C standard, and Cypress provides a very nice I2C Master component that we can use with the PSoC. As can be seen in Figure 1, setup for this component is very simple, and the only configuration to be made is to make sure it is set to operate as an I2C master and change the pin mappings in Design Wide Resources to match the hardware connections to the OLED. Any pins can be used for this, as the PSoC will be an I2C master and wakeup from sleep is not required. I chose to implement this I2C component using the Fixed Function option, but it could be implemented in UDB as well with no difference.

Additionally, the component's datasheet tells us that it requires pullup resistors on SDA and SCL. The datasheet recommends external pullups of specific values for certain speeds of operation, but I found that for the rate of 400 kbps that I was using, the internal pin pullups worked just fine. Both SDA and SCL digital output pins should be configured as ”Resistive Pull-up”.

This component's APIs provide much of the functionality we need, including the initial sending of the start condition, address, and R/W bit as well as handling the ACK signals and sending the stop condition. However, there is still a good bit of work to do: sending setup commands, moving the start position, and then of course setting the individual pixels in a meaningful way. Luckily, Adafruit has built a library that supports all of this as well as several graphics functions such as drawing text and rectangles for the Arduino, and I was lucky enough to find that someone had already ported the library to support the PSoC's I2C APIs.

In essence, the modified library stores a local copy of the display's buffer which can be modified locally through the several graphics commands, and then this buffer can be send to the OLED all at once by calling the `display_update()` function. Because it pushes the full array, this command first sets the RAM start location to be the pixel (0,0), then sends each pixel sequentially. This results in very fast updating of the screen, and I was pleasantly surprised to find that I could redraw the full

screen every loop without much of a slowdown nor noticable delay in user interaction.

The PSoC I2C API combined with this library provided an abstraction for the OLED display that proved very useful when working on the device-side UI code.

2.3 Rotary Encoder

Once again, I wanted the user interaction to be very simple—only a dial and a button to scroll and select. To do this, I decided to use a rotary encoder for the sole means of user input. Internally, the rotary encoder functions as three switches, one for output A, one for output B, and a third for the push-button switch when the dial is pushed. The sequence that outputs A and B turn on and off indicates the direction of rotation. These switches each have two pins that will be connected when that output is active, with outputs A and B sharing a common pin. I chose to set the common pin for A and B to ground, and so used pull-up resistors on the pins in the PSoC configuration to ensure that the pin voltages read HIGH when disconnected within the encoder. This setup, along with a representation of the internal configuration of the encoder, can be seen in the full schematic in Figure 5 as well.

The rotary encoder I used outputs in two-bit gray code, with output A going from ACTIVE to INACTIVE while output B is ACTIVE when turning clockwise, and output A going INACTIVE while output B is INACTIVE when turning counterclockwise. Similarly, output B goes ACTIVE while output A is ACTIVE when turning clockwise, and when turning counterclockwise output B goes ACTIVE while output A is INACTIVE. In my configuration with ground as the common pin and pullup resistors, active corresponds to LOW and INACTIVE to HIGH. To code this manually would require storing the previous state, but luckily this is a common enough application that Cypress has created a quadrature decoder component to take care of that for us, as seen in Figure 2.

In addition to the obvious A and B inputs, the QuadDec component also requires a clock input that is used for glitch filtering—as mentioned in the datasheet, the component will not register a change until the same reading is read for 3 consecutive clock cycles. The datasheet recommends that the clock input be at least 10 times the maximum input frequency, and I chose 12MHz for the clock, though I'm sure it could go as low as 5KHz without affecting the readings while still allowing for fast dial rotations by the user.

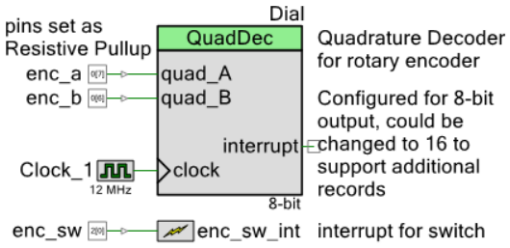


Figure 2: QuadDec for dial decoding

The QuadDec component includes another feature called indexing, which allows "zeroing" the encoder's position based on some external input. For my application, I wanted a continuous scroll and the encoder had no hard stop, so I disabled this feature. I also set the component's counter resolution to 8-bit, which allows for the user to scroll through at most 128 options. This could be increased to 16-bit to allow 32,000 options without any other changes.

Once the QuadDec component is configured, it provides a simple API with two primary commands: `Dial_GetCounter()` and `Dial_SetCounter(x)`, allowing reading and setting of the current counter value. This can then be used directly in software without needing an additional counter variable.

Figure 2 also includes an input for `enc_sw`, which is ACTIVE when the rotary encoder's dial is pushed down, and INACTIVE otherwise. Once again, I chose to configure this as active LOW, connecting one of the pins on the encoder to ground and the other to the PSoC, using internal pullup resistors to cause the input to go HIGH when inactive. I connected this input to a simple interrupt configured with a DERIVED interrupt type, which will trigger on the rising edge of the input—when the switch is released. In software, this allows me to use an interrupt service routine to set a flag when the button is pushed, which can then be handled by the UI state machine, as will be described in the software section.

When configured, the QuadDec component along with the interrupt for the push-button switch provide a simple interface to use the rotary encoder as the primary means of user input.

2.4 DS3231 Real Time Clock

In order to generate Time-based One Time Passwords (TOTPs), the system needed to have access to the current time. While the PSoC provides a real time clock component, this relies on the standard USB power, and will not provide accurate time and date information through a power cycle. For this reason, I decided to use an assembled RTC module with an onboard battery backup, the DS3231.

Referencing its datashseet, we see that the DS3231 communicates with the microcontroller over I2C, like the SSD1306. The DS3231 itself supports several features that I won't need, including square wave output and alarm interrupt outputs. However, the module I purchased from MPJA once again only breaks out the bare minimum needed to communicate with the RTC, which is the I2C interface pins (SDA and SCL, labelled "D" and "C" on the module) and the power pins.

I connected the DS3231 as shown in the full schematic in Figure 5. Once again, the pins themselves were chosen only for convenience in wiring—any of the PSoC's pins can be used.

The DS3231 follows the I2C standard for the initial connection and ending a transmission, as described in detail in the OLED section. Once again, we use the Cypress-provided I2C component to simplify the handshaking and sending of individual bytes. The I2C slave addresses for the OLED and for the RTC are different, so both should be able to be controlled using the same I2C master component. However, the PSoC's resources allow for a second component without even approaching resource limits, so I chose to create a second component in order to guarantee there would be no interference and to help distinguish the API calls. I set up this second component much the same way as the component for the OLED, as shown in Figure 3. Once again, using internal pullup resistors on the SDA and SCL pins worked well, and I left the data rate at the default 400kbps.

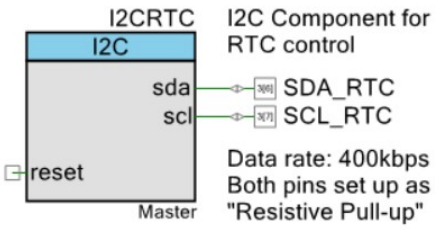


Figure 3: RTC I2C Component setup

Like the SSD1306, a popular Arduino library exists for the DS3231. However, I was unable to find anyone that had ported this library to support the PSoC and its APIs, which meant I had to do it myself. After converting the C++ class structure they had used to a more C-friendly one using a `DateTime` struct I created, all that was left was to translate the API calls for the functions I needed: setting and reading the time.

Referring to the DS3231 datasheet,[2] I found that the module internally has a sequence of registers that store seconds, minutes, hour, day of week, date, month, and year from 2000, each in binary coded decimal (BCD). There are several additional registers used for setting the alarms, but reading from and writing to these were all that were needed for my fairly simple application.

To set the time, then, we need to write these registers. First issuing the byte with the slave address and a 0 in the write bit to enter write mode, the DS3231 expects the next byte to be the address of the first register to write. We want to write the first 7 registers, so we first send a 0. Then, the DS3231 expects all following bytes to be data to write to its registers, so we write the BCD-encoded current time sequentially, as stored in a DateTime struct, followed by a stop condition. The PSoC's `I2C_MasterWriteBuf` sends the initial start condition and slave address followed by any bytes stored in the buffer passed to it, so we can first construct the buffer and then write all 8 bytes and the stop condition to the DS3231 with this single API call.

Reading the current time from the DS3231 is a similar process. We must first set the register address to begin reading from, which we do by writing a single 0 to the DS3231. Now, when the DS3231 sees its slave address along with a READ bit on the I2C bus, it will begin to transmit the contents of each of its registers sequentially, starting with the register we just set as the start register. Because the current time is stored in the first 7 bytes, we set this start register to zero, wait for the ACK from the DS3231, and send its slave address and a READ, saving the first 7 bytes it transmits to a buffer. Finally, we can construct a DateTime struct by converting each of these values from BCD to standard binary.

For the specific implementation of these functions, as well as the code for converting between BCD and binary, see the Appendix. Through the use of this ported and simplified library and the PSoC's I2C master component, I was able to construct a very simple abstraction for the DS3231 RTC that will be used for both TOTP generation and seeding of the pseudorandom sequence generator for password creation.

2.5 MicroSD Interface

Of course, the password keeper also needed some way of keeping the passwords long-term. While the PSoC's flash memory can be used for non-volatile storage, read/write cycles as well as space are limited. I chose to use an external microSD card instead. SD cards by default communicate over a proprietary protocol, but also SPI by nature, so a simple breakout board with no extra logic would be enough to interface with one. However, SD cards work on 3.3V while the PSoC CY8CKIT-059 I used for this project runs on 5V for all VDD and logic pins when powered by USB, as it would be in this project. For this reason, level shifters for the logic pins and a 3.3V regulator for VCC were required to interface with a microSD card. Fortunately, these are readily available online, so I bought a ready-made module that includes everything needed to interface with an SD card from a 5V system.

We could use the SD card over simple SPI, but this would only allow very primitive access to the data on the SD card—we would need to read and write 512-bit data blocks, and would have to issue a series of power-up and other commands to initially interface with the card. Luckily, the PSoC ships with an EmFile component and companion library by Segger that makes this process much simpler. The component handles the SPI aspect and generates an API that can be used by the Segger library.

The EmFile library handles all the reading and writing to the SD card, allowing us to interface with the SD card as if it were a file system. It provides functions to open files, create folders, format the SD card, and more. Much of this is more than is required for simply holding password records, but nonetheless greatly simplifies the process of using an SD card to store information. All that is required for configuration is to set the PSoC pins used for MOSI, MISO, SCK, and CS on the SD card, which can be any pins.

Referring to the user manual,[3] we can see that the EmFile library provides `FS_FOpen`, `FS_Write`, and `FS_Read` functions, which allow us to read and write from a certain file. Together with the commands to format the SD card, these allow us to construct a database file and interact with it in a straightforward way.

2.6 USB Full-Speed

In order to enable the device to communicate with the user's PC through both HID keyboard emulation and CDC virtual COM port emulation, the Cypress-provided USBFS component proved very useful. Each of these features is fairly simple to implement on their own, and Cypress provides very helpful code examples for each.

In order to set up the device descriptors as an HID keyboard, Cypress provides a template configuration for a standard HID keyboard, and describes the process in their AN57473.[4] To set up the keyboard piece of the USBFS component for this application, the template for an HID keyboard can be imported using the green "import" arrow in the HID Descriptors tab of the USBFS component and selecting the "Keyboard" option. Then, in the Device Descriptor tab, values must be updated as instructed in the Cypress video.

To set up the virtual COM port, the USBFS component must be modified to also act as a CDC USB device. Cypress provides the USBUART component to fill this need, and to add it to the USBFS component we must simply implement the correct descriptors, which can be done by clicking the green arrow "import" button in the CDC descriptor tab, then selecting "USBUART Single COM Port."

Finally, the device must be set up to allow it to enumerate on the host PC as both an HID keyboard and a CDC USB device. To do this, an Interface Association Descriptor must be added in the Device Descriptor tab, with the interface count set to 2. This allows the computer to recognize both interfaces simultaneously.

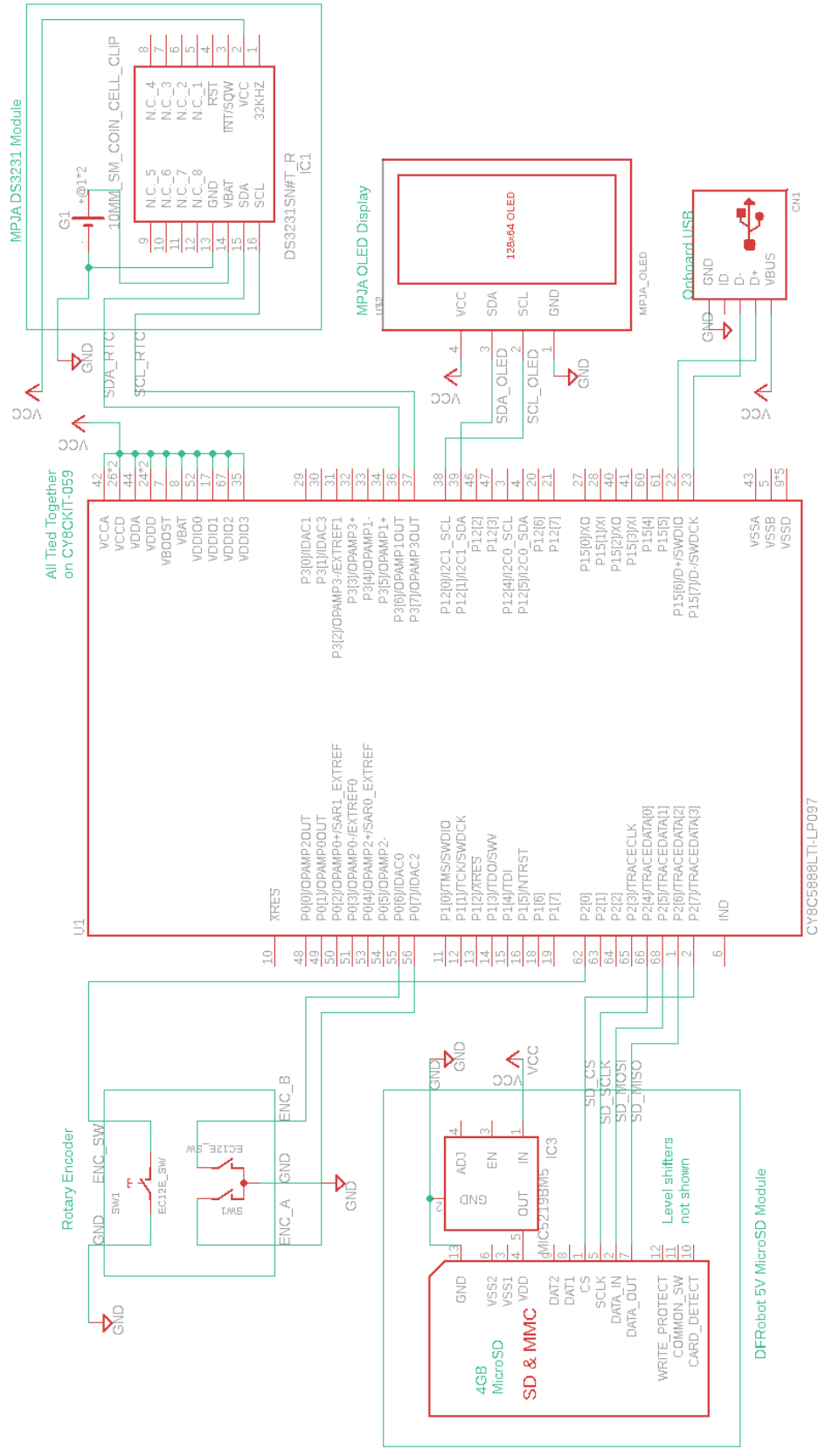


Figure 4: The connections required to wire all hardware components of the password keeper.

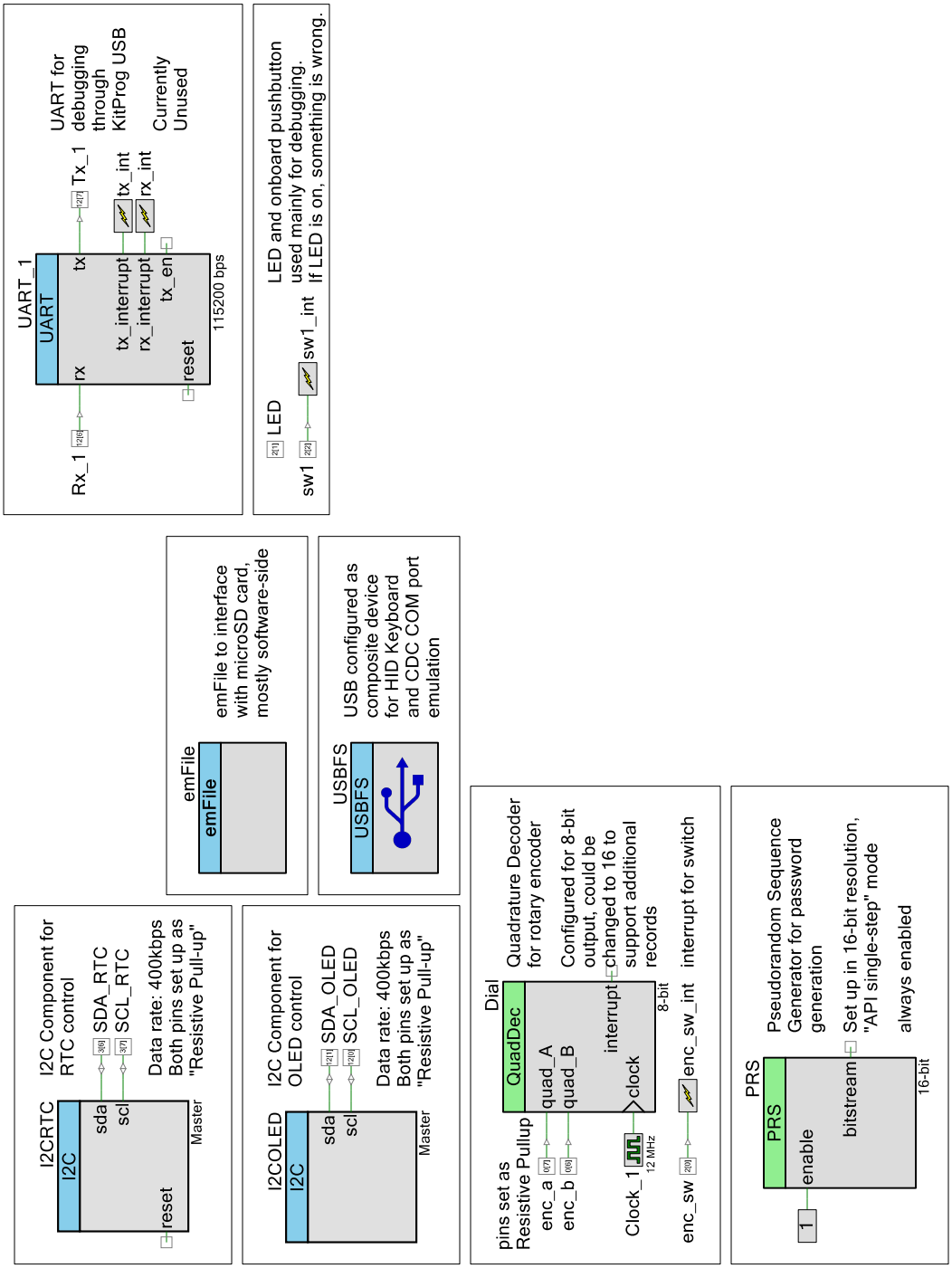


Figure 5: The TopDesign sheet for the PSoC creator project.

3 Software

The ultimate goal of the software for this system is to provide a cohesive user interface. To achieve that goal, however, there are several distinct areas that should be abstracted for simplicity, from communicating with hardware to displaying graphics on the OLED. In developing the software that runs on the PSoC, I worked to modularize it into several components of various complexity:

1. Interface with the SSD1306 and basic graphics functions
2. Interface with DS3231 and `DateTime` struct
3. Creation of a universal `Record` type that would be used across other modules
4. Implementation of the Time-based One Time Password (TOTP) algorithm and underlying HOTP and HMAC-SHA1
5. Record encryption and decryption, as well as verification of the initial user PIN.
6. Functions to create, read from, and write to the SD-based Record database
7. Functions to interface with the user's PC through keyboard and virtual COM port emulation
8. Support functions to display items and menus over the console-based interface, as well as the finite state machine (FSM) powering the console UI and allowing users to add records.
9. Support functions to draw items and menus on the OLED display; FSM for the OLED UI.

In each of these modules, I wrapped the relatively primitive API calls into more applicable and involved functions that could be called for a specific purpose. As can be seen in Figure 6, they serve as building blocks towards the user-facing modules, the two UI finite state machines. In this section, I will describe many of the choices I made in implementing these modules, but for details on each function see the code listings in the appendix.

3.1 SSD1306 Support

As described in the hardware section, I used an existing port[5] of the Adafruit SSD1306 library that was initially designed for the PSoC Pioneer, but after testing I found that it worked without modification on the PSoC 5LP I was using. The functions provided by this library are used extensively in the OLED UI section of the software.

3.2 DS3231 Support

In order to interface with the DS3231 module, I ported the relevant parts of Adafruit's RTClib[6] for the Arduino. The original library was written in C++, so this involved moving away from the class-based system they had used for DateTime objects and the various RTC functions. Because I only needed to support the DS3231, rather than the 3 RTCs their library supported, this was simplified a little. The best way to do this seemed to be creating a DateTime type that could be used to represent the current time. Functions would then translate the RTC output, given as a DateTime type by the `RTC_now()` function, into other formats such as Unix time (which is what this project makes use of). The function to write to the RTC, though rarely used, also takes an object of the DateTime type as an argument, which can be generated from a number of other formats. While all I really needed for this project was the ability to set the correct time and read the current Unix time, porting this library allowed for other uses as well, and provides a basis for possible future expansions.

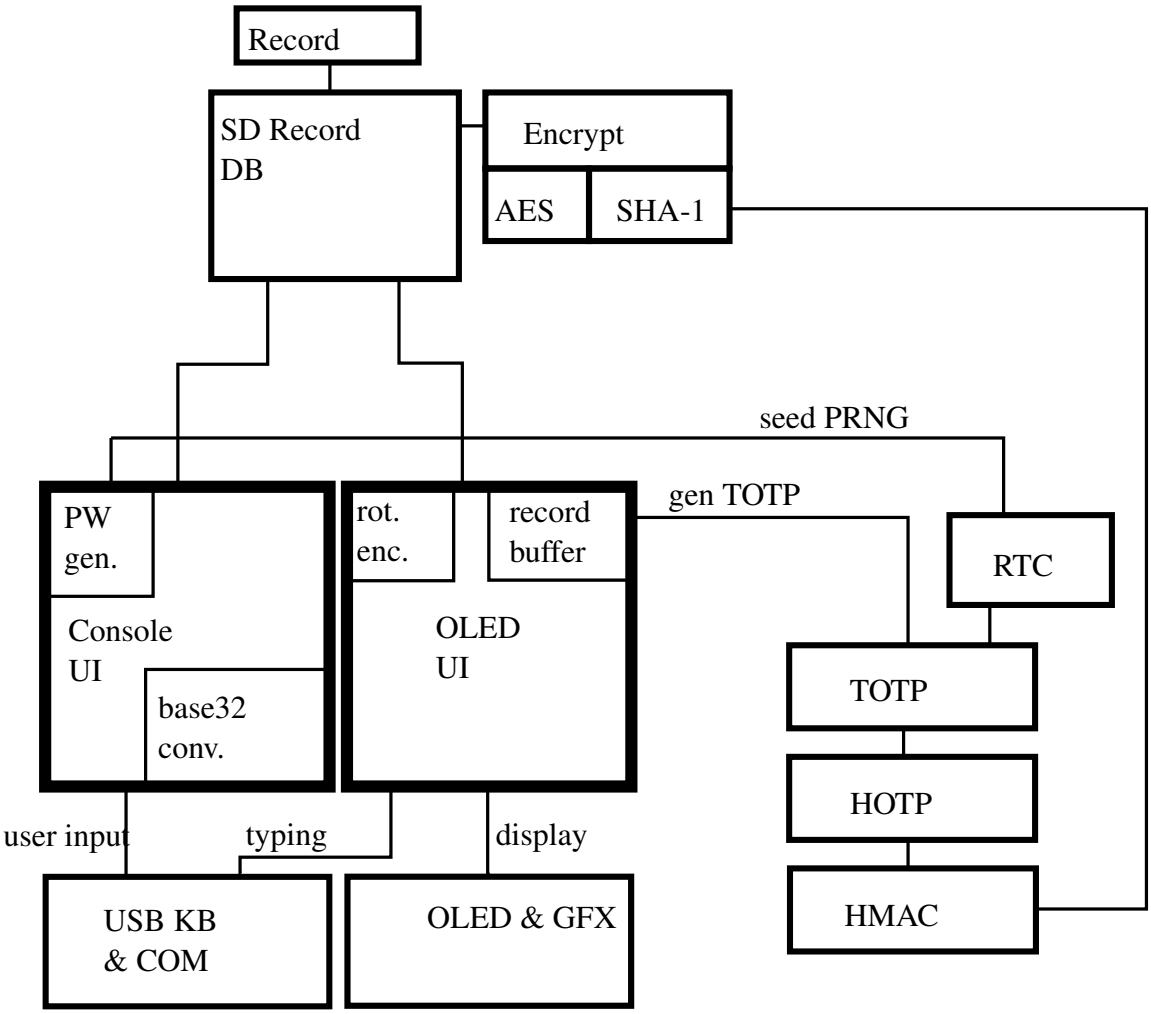


Figure 6: A simplified representation of the reliances between each software module.

3.3 Record Type

As the core of this device was in password management, I though it would be crucial to have a simple and expandable way of storing passwords. To do this, I created a simple `Record` type that is shared across most of the modules, consisting of a record name, username, password, TOTP key, and indicator for whether TOTP is enabled for each record.

3.4 TOTP, HOTP and HMAC Implementation

The generation of Time-based One Time Passwords (TOTPs) relies on quite a bit of cryptography as well, which all build on each other. TOTP is a very simple application[7] of HOTP, which uses a counter as well as a secret key to generate an OTP—TOTP simply uses the current time to derive the value of this counter. HOTP relies on HMAC, which relies on the SHA-1 hashing algorithm. I was able to find implementations of SHA-1[8] and HMAC[9] that worked without modification on the PSoC, and only had to modify the HMAC implementation slightly to match the API of the SHA-1 code I used. Once I had a working HMAC implementation, I chose to implement my own version of the HOTP algorithm based on the description outlined by the IETF.[10]

After implementing HOTP and TOTP, I had a very simple function `calc_totp` that could be called from the OLED UI in order to generate the TOTP for the record's key and the current time according to the DS3231 RTC.

3.5 Record Encryption/Decryption and PIN Checking

I also needed to provide a mechanism for encrypting and decrypting the records, as well as for enabling the initial login. I chose to use an existing implementation of AES[11] to encrypt and decrypt, which uses a 128-bit key. However, some way of logging in without storing the correct password or encryption key was necessary. To do this, the key is given by the SHA-1 hash of the PIN, and only the SHA-1 hash of that key is stored permanently (on the SD card in this case). Once the correct PIN has been entered, and the derived key is stored in RAM to decrypt and encrypt records. Encrypting and decrypting records is done by an additional function that is called when records are loaded into the internal buffer.

3.6 SD Card Record Database Interface

The database of records is stored on the SD card in a single file as simply sequential bytes. The first 20 bytes store the hash of the correct encryption key, and all later bytes store records. To avoid needing to open the file, move the cursor to the right place, and read the correct number of bytes each time, I abstracted this into several functions that can be used to read from, write to, and format the SD card. As the OLED UI holds a buffer of multiple records to allow smooth scrolling, I chose to allow the read function to read any number of records.

3.7 Keyboard Emulation and Virtual COM port

The USBFS API for the HID keyboard was very barebones, and required sending an array of modifier keys and keystrokes that matched the report structure declared in the HID Descriptor,[13] with keycodes as given in the USB HID standard.[12] To simplify this process, I created a set of functions that allow any character array to be ”typed” with ease.

For the communication over the virtual COM port, a bit of work needed to be done to handle inputs. The API simply presented data as ”ready” whenever a single letter is typed, and these letters needed to end up as a command that could be used in the console UI. To do this, the USB module has its own input buffer, and whenever it recieves an enter character, it moves this buffer's contents into the Console UI's serial command buffer.

3.8 Console UI Finite State Machine

One of the primary methods of user interaction is through the console-based user interface, where users can add records, view all their records, and reset the device. To implement this, several types of support functions were used, from drawing menus on the screen to generating a password for the user. With these functions implemented, the user interface itself is a simple finite state machine, the basis of which can be seen in Figure 7.

3.9 OLED UI Finite State Machine

For everyday use, however, the user will not need to use the console-based UI, and will instead be able to control the device using the onboard rotary encoder and OLED. This was implemented in a similar way to the console-based UI, with a range of support functions that simplified the overall FSM. A notable challenge when implementing this was ensuring that the screen was responsive when scrolling through the list of records. In order to avoid loading a potentially very large number of records into RAM and to limit the number of SD read requests, I implemented a local buffer of records that is updated when the user nears the end of the loaded records. A simplified version of the state machine that operates the UI can be seen in Figure 8.

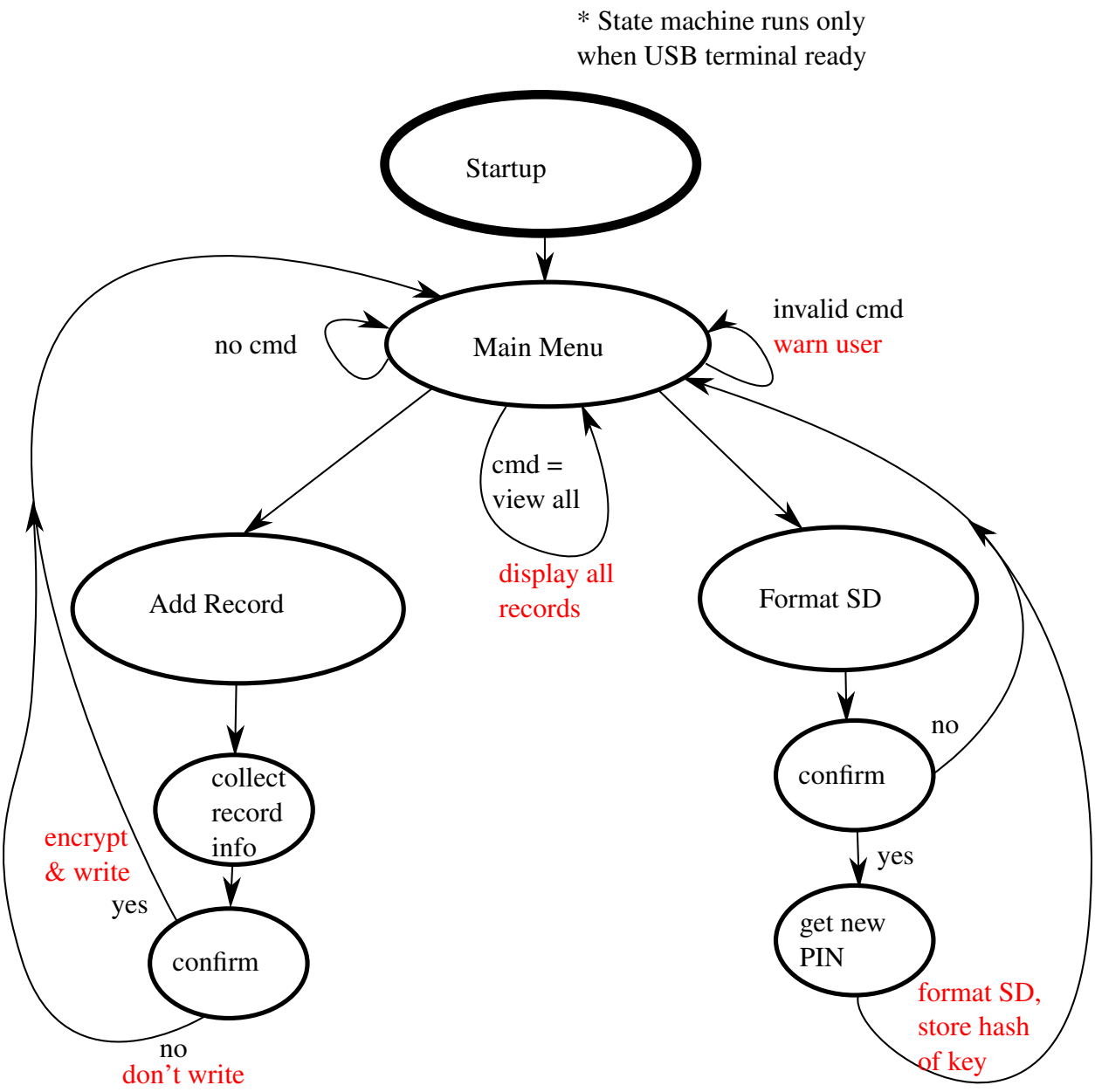


Figure 7: State diagram showing the simplified structure of the console-based user interface.

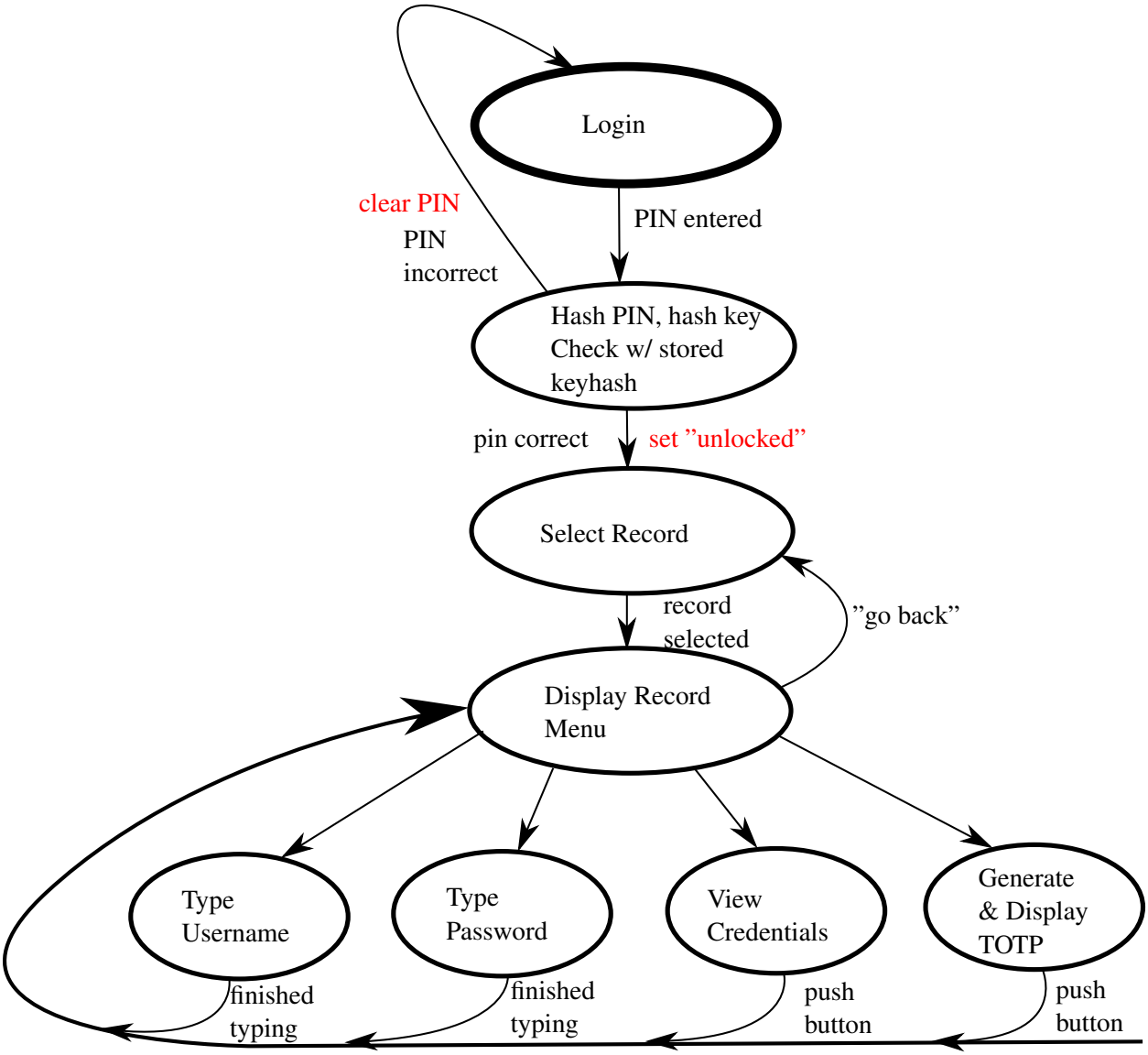


Figure 8: State diagram showing the simplified structure of the OLED-based UI.

4 Conclusions

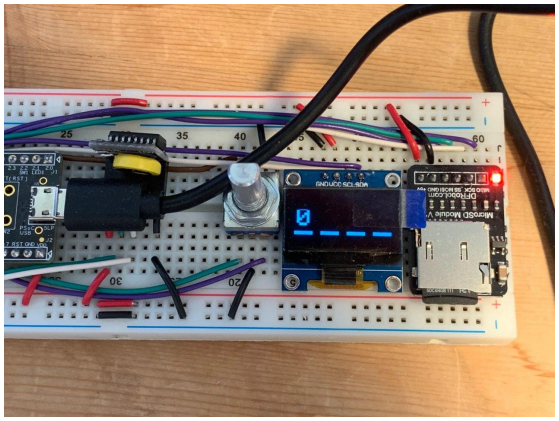
In designing and building this project, I learned much more about communication protocols and encryption than I knew previously, and I think I created a cohesive project that met the goals I had set out. Were I to move forward with this project, there are several improvements that could be made.

1. Deriving the AES key from the PIN using only a simple hash is not terribly secure, and could be brute-forced fairly quickly assuming the perpetrator knew how the key was derived—they would need to try only the 10,000 possible pins. Security by obscurity is never good, and it would be much better to replace this hash function with a more secure key derivation functino such as PBKDF2, which uses significant processing time to generate a key from a password, slowing down a brute-force attack. Rate-limiting could also be added to prevent brute-forcing of PINs on the device.
2. While the system I have works, there is limited error-checking when making requests to the SD card and other components. Were this product to go to market, error checking and user feedback would be a necessary addition.
3. Usability could be significantly improved with the addition of a PC-side software app that would automatically select the record for the user based on their browsing activity, and even automatically enter passwords. Of course, this strays from the goal of no external software dependencies, but now that that goal is achieved, the same interface could be used with a more convenient software wrapper.

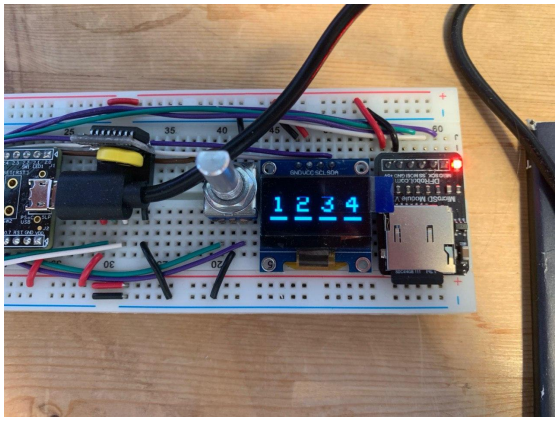
Despite these possible improvements, I believe that I built a fairly user-friendly device that achieves the basic functionality that I initially set out to achieve: save and retrieve passwords with a simple device that requires limited extra software.

5 Demo

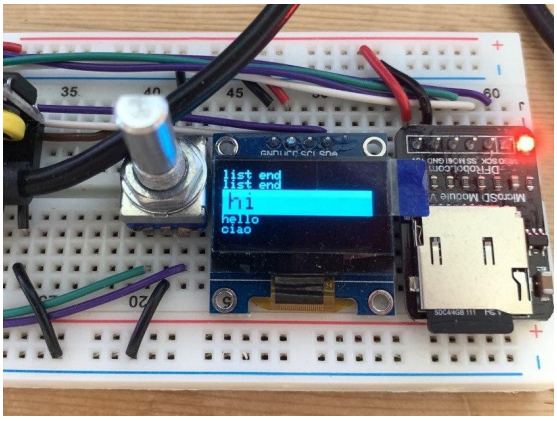
To demonstrate the functionality of the device, let's begin with the most common use case: accessing records that are already stored. First, the user must log in to the device. When plugging the device into their computer, they are presented with the following screen, prompting them to enter their PIN:



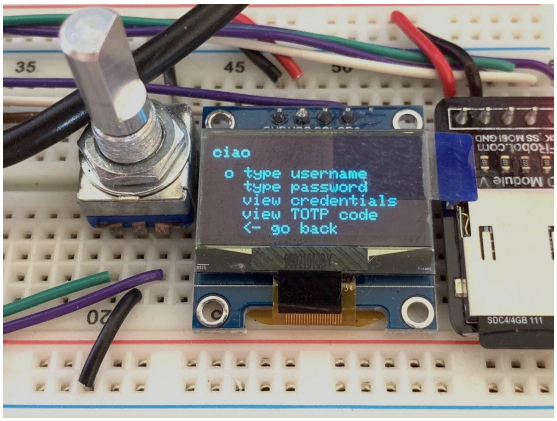
As the user enters their pin by turning the dial and pushing to lock in each digit, the confirmed digits will show up on the screen.



After entering all four digits, the software will check the PIN that the user entered. If the PIN is incorrect, the entered PIN is cleared and the user is given another chance to enter the PIN. If it is correct, and thus the key derived from it can be used to decrypt records, the device will load the first several records from the SD card into the RAM buffer and display these on the screen. In this case, these records are titled "hi", "hello", and "ciao".

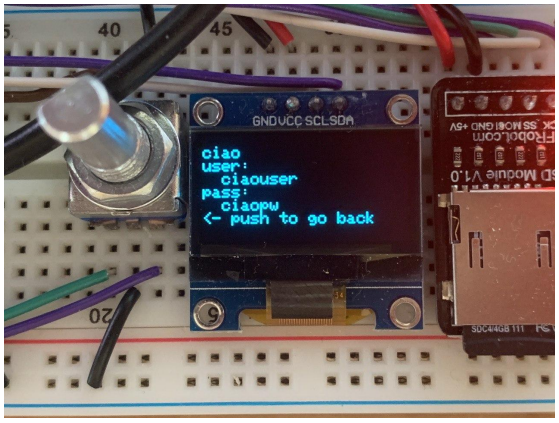


As there are no records that can be accessed before the first record, "list end" is displayed in the place of a record name, and scrolling upwards is prevented. The user is then able to use the rotary encoder to scroll through the full set of records on the SD card. As they approach the end of the buffer of records that is currently loaded, a new set of records is loaded in the buffer that is centered on the record they currently have selected. After choosing the record they would like by pushing on the encoder, the user is presented with another screen presenting several options.

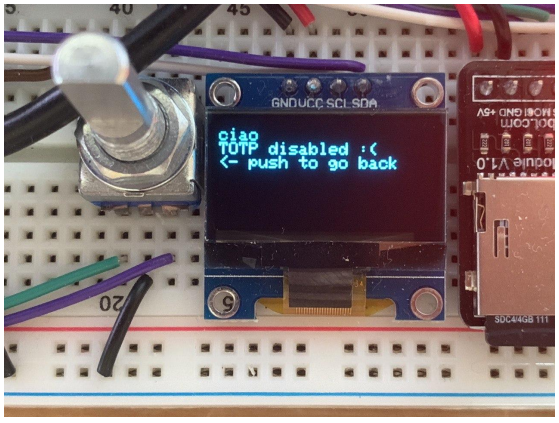


Again, the user chooses which option they would like by scrolling with the rotary encoder

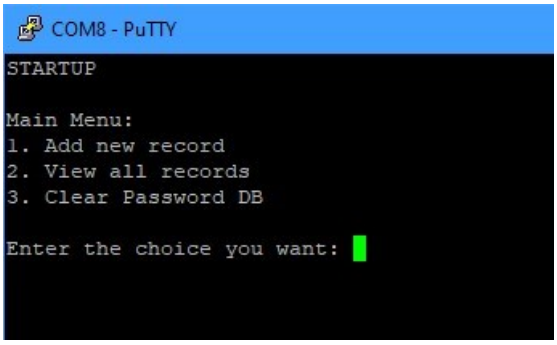
and selecting by pushing. Selecting ”type username” or ”type password” types in the username or password for the currently selected record to the connected computer. Choosing ”view credentials”, as expected, will display the record’s username and password on the screen.



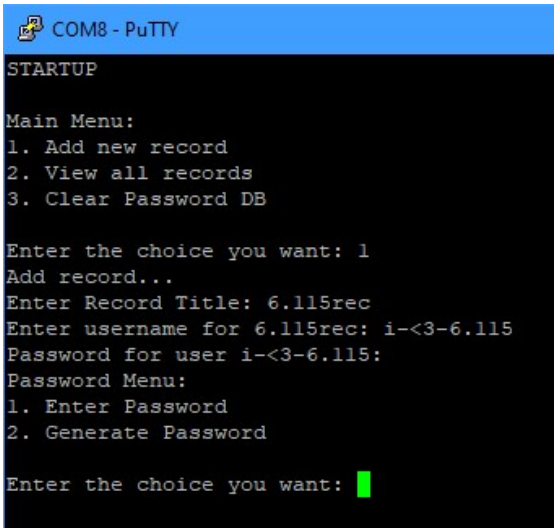
From here, the user can push to go back. If they select ”view TOTP code”, the device will check whether TOTP is enabled for the current record. If so, it will display the TOTP code based on the current time, and if not (as is the case for this record), it will display a message saying so.



This functionality exists for every record in the system, and can be accessed solely using the controls on the device. To add additional records, however, entering usernames, passwords, and TOTP keys on the device would be infuriating. For this, the console-based interface is used. When opening communication with the virtual COM port created by the device through a program like PuTTY, a menu is displayed prompting the user with several options.



To add a record, the user should type ”1” followed by enter. When they do so, they will be asked to give a title for their record, which should be the name of the website or something similar. After entering this, they will be asked for a username. Each of these are entered simply by typing them then pressing enter. Once the username has been entered, a password can be entered manually or generated directly on the device. To choose which of these options the user would like, a menu is presented.



If they choose to enter a password, the same procedure as entering a title or username applies. If they choose to generate a password, one is generated for them and displayed. Here, we chose to have the device generate a password.

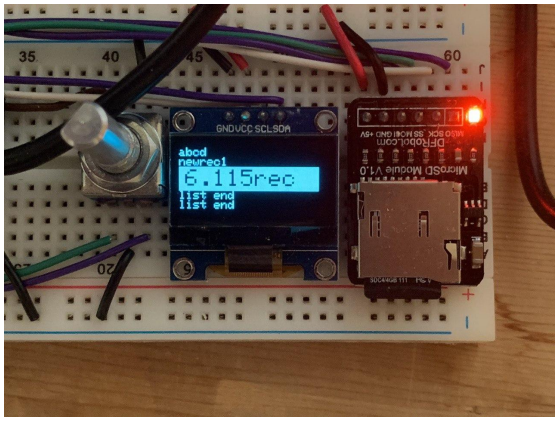
In either case, they are then asked whether they would like to add a TOTP key.

```
Enter username for 6.115rec: i-<3-6.115
Password for user i-<3-6.115:
Password Menu:
1. Enter Password
2. Generate Password

Enter the choice you want: 2
Generating password!
Generated Password for user i-<3-6.115: o{vjSN;w}g?~,Yg>
 Would you like to enter a TOTP key? y/n:
```

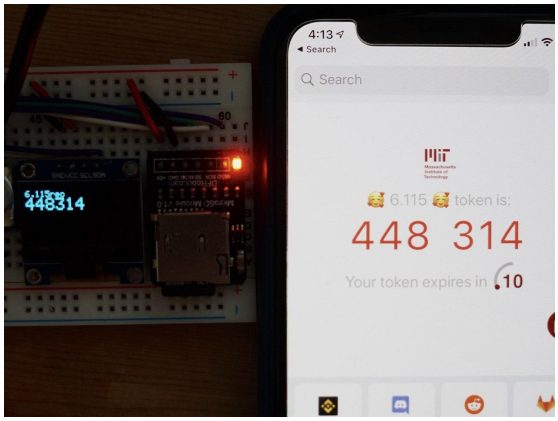
If they choose to add one, they are instructed to do so in base32, which is how the keys are almost always presented, and with a 20-byte length, which is the standard for TOTP keys. Once the key has been entered, the user is asked to confirm that they would like to add the record to the database. If the user chose to skip inputting a TOTP key, they are brought to the same confirmation stage directly. From here, they can choose to either save the record or discard it.

```
Enter TOTP Key (20 bytes, base32 encoded)
Note: Only 6-digit TOTP generation currently supported
> ZAZNEAAWH5PE5OWUW6JCWFJ7CBCPTWCI
Recieved TOTP Key: ZAZNEAAWH5PE5OWUW6JCWFJ7CBCPTWCI
Full Record to Add:

   ****************************
   * Title: 6.115rec
   * Username: i-<3-6.115
   * Password: o{vjSN;w}g?~,Yg>
   * TOTP: Enabled
   ****************************

Enter 'y' to confirm, anything else to cancel.
```

After entering their choice, the user is given a confirmation message and brought back to the main menu, where they can add another record, view all records, or format the SD card. Here, we chose to add the record, and indeed the newly added record immediately shows up at the bottom of the record list:



Navigating to that record and selecting to view the current TOTP record, we can see that the code generated matches the code generated by adding the same secret key to a recognized 2FA app (Authy in this case).



From the main serial menu, the user also has the option to format the SD card. If they do so, they are asked to type exactly the phrase ”yes, delete all records” followed by enter to confirm that they want to format the SD card.

```
Main Menu:
1. Add new record
2. View all records
3. Clear Password DB

Enter the choice you want: 3
WARNING: FORMATTING SD CARD
Do you really want to format the SD card?
To cancel, type n and press enter
Type "yes, delete all records" to continue:
```

Here, we chose not to do so. If the user had chosen to format the SD card, they would have been asked to enter a new PIN to use to encrypt the records, and the SD card would be formatted. After this, they would be sent back to the main menu.

References

[1] Solomon Systech: SSD1306 Datasheet, https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[2] Maxim Integrated: DS3231 Datasheet, https://datasheets.maximintegrated.com/en/ds/DS3231.pdf

[3] Segger: EmFile User Manual, https://www.segger.com/downloads/emfile/UM02001

[4] Cypress: AN57473 - HID Basics with PSoC, https://www.cypress.com/documentation/application-notes/an57473-usb-hid-basics-psoc-3-and-psoc-5lp

[5] Derk Steggewentz: PSoC OLED Library: github.com/derkst/Cypress-PSOC-OLED

[6] Adafruit: RTClib, https://github.com/adafruit/RTClib

[7] IETF: Time-Based One-Time Password Algorithm, https://tools.ietf.org/html/rfc6238

[8] clibs: SHA-1 Hash Algorithm, https://github.com/clibs/sha1

[9] Akagi201: hmac-sha1, https://github.com/Akagi201/hmac-sha1

[10] IETF: HMAC-based One-Time Password Algorithm, https://github.com/Akagi201/hmac-sha1

[11] kokke: Tiny AES in C, https://github.com/kokke/tiny-AES-c/

[12] USB-IF: HID Usage Tables, https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf

[13] USB-IF: HID Class Definitions, https://www.usb.org/sites/default/files/documents/hid1_11.pdf

A Code Listings: Original Implementations

A.1 Main

```
/* 6.115 Project: Password Keeper
 * Ben Kettle bkettle@mit.edu
 *
 * This file contains the main
 * code that is run, which interfaces
 * between several other modules */

#include "project.h"
#include <FS.h>
#include <string.h>
#include <Global.h>
#include <stdbool.h>

#include <ds3231.h>
#include <ssd1306.h>

#include <pk_record.h>
#include <pk_sdrecorddb.h>
#include <pk_serialfuncs.h>
#include <pk_ui.h>

// flag for whether the dial's switch was pushed
bool enc_sw_flag;
bool sw1_flag; // flag for the onboard switch

CY_ISR(ENC_SW_INT) {
    // set the enc_sw_flag to indicate
    // that the rot. enc. switch has been pushed
    enc_sw_flag = true;
}

CY_ISR(SW1_INT) {
    // toggle flag for onboard switch
    sw1_flag = !sw1_flag;
}

int main() {
    CyGlobalIntEnable; /* Enable global interrupts. */
    enc_sw_int_StartEx(ENC_SW_INT); // start encoder switch interrupt
    sw1_int_StartEx(SW1_INT); // start encoder switch interrupt

    // start UART for debugging
    UART_1_Start();

    /* Initialize file system */
    FS_Init();

    Dial_Start(); // start rotary encoder decoder
    s_main_state = S_STARTUP; // initialize serial fsm state

    // to store the currently selected record
    // in the OLED UI
    Record curr_record;

    // start the rtc communication over i2c
    I2CRTC_Start();
    I2CRTC_EnableInt();

    // start i2c connection with the oled
    I2COLED_Start();
    I2COLED_EnableInt();
    display_init(DISPLAY_ADDRESS);
```

```
63
64      // get the current time
65      DateTime now = RTC_now();
66
67      // start generating random numbers
68      // using current time as seed
69      PRS_Start();
70      PRS_WriteSeed(dt_getUnixtime(now));
71
72      // initialize the serial fsm
73      serial_menu_init();
74
75      // show startup message on display
76      gfx_setCursor(10,10);
77      gfx_setTextSize(2);
78      gfx_setTextColor(WHITE);
79      gfx_println("STARTUP");
80      display_update();
81
82
83      // wait for USB to enumerate on PC
84      USB_Initialize();
85
86      USB_print("STARTUP\n\r");
87
88      for(;;) {
89          // update the UI record buffer if the serial FSM
90          // has just added a record to the SD card
91          if (serial_status == S_STATUS_REQUIRE_RELOAD) {
92              serial_status = S_STATUS_GOOD;  // reset the status
93              ui_forceBufferUpdate = true;    // set the buffer update flag for OLED fsm
94          }
95
96          // CALL FSMs
97          // only do the serial things if serial is connected
98          // i.e. the user has putty open,
99          // AND the device has been unlocked
100         if (USB_isReady() && unlocked) serial_fsm(&curr_record);
101
102         // always call the OLED UI FSM, pass it the address
103         // of the enc_sw_flag so it can modify it
104         ui_fsm(&enc_sw_flag);
105
106         // ui fsm handled any encoder input, so reset it
107         enc_sw_flag = false;
108     }
109 }
110
111 /* [] END OF FILE */
```

A.2 Terminal UI

Listing 1: pk_serialfuncs.h

```
1  /* 6.115 Final Project: Password Generator
2     Ben Kettle bkettle@mit.edu
3
4      defines functions
5      for interfacing with the user
6      over serial.
7  */
8
9  #ifndef PK_SERIALFUNCS
10 #define PK_SERIALFUNCS
11 #include <pk_record.h>
12 #include <project.h>
13 #include <ds3231.h>
```

```
14 #include <stdio.h>
15 #include <stdlib.h>
16
17 // Serial Status
18 // for communication with other parts of the system
19 int serial_status;
20 #define S_STATUS_GOOD 0 // all is good, self-contained
21 #define S_STATUS_REQUIRE_RELOAD 1 // after modifying record db
22 #define S_STATUS_FORMATTING_SD 2 // when formatting sd card
23
24 // this type is used for anytime the user  needs to
25 // make a decision. Supports up to 10 20-character
26 // options.
27 typedef struct {
28     int size;
29     char title[20];
30     char items[10][20];
31 } SerialMenu;
32
33 // MENU/Serial State DEFINITIONS
34 #define S_WAIT -1
35 #define S_STARTUP 0
36 #define S_MAIN_MENU 1
37 #define S_ADD_RECORD 2
38 #define S_FORMAT_SD_PROMPT 3
39 #define S_FORMAT_SD 4
40
41 /* send a representation of the given
42     menu over USB, and ask the user
43     to input their choice. Options are
44     numbered sequentially.
45
46     input:
47     menu: a SerialMenu object of the
48     menu that should be printed
49
50     example output:
51         Main Menu:
52         1. Add new record
53         2. View all records
54         3. Clear Password DB
55
56         Enter the choice you want:
57 */
58 void serial_print_menu(SerialMenu menu);
59
60 /* send a pretty represenntation of the
61     given record over USB. Includes
62     name, username, password, and
63     TOTP status, but not TOTP key.
64
65     inputs:
66     record: the record to be printed
67
68     example output:
69         ****************************
70         * Title: 6.115rec
71         * Username: i-<3-6.115
72         * Password: o{vjSN;w}g?~,Yg>
73         * TOTP: Enabled
74         ****************************
75 */
76
77 void serial_print_record(Record record);
78
79 /* notify the user that their command
80     was invalid
81 */
```

```
82  void serial_inv_cmd();
83
84  /* generate a password using the
85      psuedo RNG, which should be seeded
86      at boot time.
87
88      inputs:
89      out: the start of a 16-byte char
90          array to hold the generated pw
91  */
92  void generate_password(char* out);
93
94  /* for RTC debugging, print a nice
95      representation of the given DateTime
96      to the USB output.
97
98      input:
99      dt: the DateTime struct to print
100 */
101 void serial_print_dt(DateTime dt);
102
103 /* print the binary contents of a given
104     array in hex to the USB output.
105
106     input:
107     arr: a pointer to a byte array to print
108     size: the number of bytes to print
109 */
110 void serial_printHex(unsigned char * arr, size_t size);
111
112 // Serial state variables
113 int s_main_state; // main state
114 int s_sec_state; // secondary state
115
116 /* The main user interface state machine.
117     should be called every loop, USB input
118     and all other functions needed are handled
119     internally.
120 */
121 void serial_fsm(); // the main FSM to be called from main()
122
123 void serial_menu_init(); // set the serial state to STARTUP
124
125
126 #endif
127
128 /* [] END OF FILE */
```

Listing 2: pk_serialfuncs.c

```
1   /* 6.115 Final Project: Password Generator
2      Ben Kettle bkettle@mit.edu
3
4       defines functions
5       for interfacing with the user
6       over serial.
7   */
8
9   #include <pk_serialfuncs.h>
10  #include <pk_sdrecorddb.h>
11  #include <pk_usb.h>
12
13  #include "base32.h" // for decoding/encoding base32
14
15  /*****************************
16  *       UART STUFF             *
17  *****************************/
18
```

```
19
20  // last serial command holds the previous command
21  // while rx_buffer holds the one currently being
22  // typed. last serial command is updated on enter
23  char rx_buffer[40];
24  int rx_buff_length;
25  char last_serial_cmd[40];
26  int last_serial_cmd_length = 40;
27
28  void clr_last_serial_cmd(){
29      memset(last_serial_cmd, 0, last_serial_cmd_length);
30  }
31
32  Record new_record = {"","","",false,""};
33
34  // Serial Main Menu
35  SerialMenu main_menu = {
36      3,
37      "Main Menu",
38      {
39          "Add new record",
40          "View all records",
41          "Clear Password DB"
42      }
43  };
44
45  // password creation menu
46  SerialMenu pass_menu = {
47      2,
48      "Password Menu",
49      {
50          "Enter Password",
51          "Generate Password"
52      }
53  };
54
55  // FOR SERIAL MENUS
56  // main fsm state definitions in .h
57
58  // definitions for commands for serial main menu
59  #define INVALID_CMD 0
60  #define S_MAIN_ADD 1
61  #define S_MAIN_VIEW 2
62  #define S_MAIN_FORMAT 3
63
64  // states for adding a record
65  #define ADD_REC_START 0
66  #define ADD_REC_TITLE 1
67  #define ADD_REC_UNAME 2
68  #define ADD_REC_PASS_MENU 3
69  #define ADD_REC_PASS_ENTRY 4
70  #define ADD_REC_PROMPT_TOTP 5
71  #define ADD_REC_TOTP_ENTRY 6
72  #define ADD_REC_CONF_PROMPT 7
73  #define ADD_REC_CONFIRM 8
74
75  // command definitions for password menu
76  #define PASS_MENU_ENTER 1
77  #define PASS_MENU_GEN 2
78
79  // HOLDS THE RECORD BEING CREATED
80  Record new_record;
81
82  void update_rx_buffer(char to_add) {
83      char str_to_add[2] = {to_add, 0};
84      strcat(rx_buffer, str_to_add);
85  }
86
```

```
87  void serial_print_menu(SerialMenu menu) {
88      USB_print("\n\r");
89      USB_print(menu.title);
90      USB_print(":\n\r");
91      for (int i = 0; i<menu.size; i++) {
92          // print item number (max of 9 items)
93          // just does it by ascii conversion
94          char itemnum[2] = {(char) (i + 0x31), 0};
95          USB_print(itemnum);
96          USB_print(". ");
97          USB_print(menu.items[i]);
98          USB_print("\n\r");
99      }
100     USB_print("\n\rEnter the choice you want: ");
101 }
102
103 void serial_print_record(Record record) {
104     // prints a record's info to the serial monitor in a nice way
105     USB_print("\n\r\n\r   ");
106     char star_line[30] = {0};
107     for (int i=0; i<29; i++) {
108         star_line[i] = '*';
109     }
110     star_line[29] = 0;
111     USB_print(star_line);
112     USB_print("\n\r   * Title: ");
113     USB_print(record.title);
114     USB_print("\n\r   * Username: ");
115     USB_print(record.username);
116     USB_print("\n\r   * Password: ");
117     USB_print(record.password);
118     USB_print("\n\r   * TOTP: ");
119     USB_print(record.totp_enabled ? "Enabled" : "Disabled");
120     USB_print("\n\r   ");
121     USB_print(star_line);
122 }
123
124 void serial_printAllRecords() {
125     // prints all records on the sd card
126     int sd_size = SD_getNumRecords();
127     Record curr; // holds the current record to be printed
128     for (int i=0; i<sd_size; i++) {
129         SD_readRecords(i, 1, &curr);
130         serial_print_record(curr);
131     }
132 }
133
134 void serial_inv_cmd() {
135     USB_print("\n\rInvalid Command!");
136 }
137
138 // print hex of a char array
139 void serial_printHex(unsigned char * arr, size_t size) {
140     char tp[3]; //holds things to be printed
141     USB_print("0x");
142     for (uint i=0; i<size; i++) {
143         sprintf(tp, "%02X", arr[i]);
144         USB_print(tp);
145     }
146     USB_print(" ");
147 }
148
149 // generate a password
150 void generate_password(char* out) {
151     /* generate a 16-character ascii
152         password using the PRS component
153     */
154     char current_char = 0;
```

```
155     for (int i = 0; i<16; i++) {
156         current_char = (char) PRS_Read();
157         while ((int) current_char < 33 || (int) current_char > 126) {
158             // if outside of valid range of characters, generate new character
159             // spaces are not allowed, to include them change 33 to 32
160             PRS_Step();
161             current_char = (char) PRS_Read();
162         }
163         out[i] = current_char;
164         // move to next PRS value
165         PRS_Step();
166     }
167     out[16] = 0; // null terminate password
168 }
169 
170 // for the rtc, just debugging stuff
171 void serial_print_dt(DateTime dt) {
172     USB_print("\n\rdatetime: ");
173     char to_send[40];
174     sprintf(to_send, "%d-%d-%d %d:%d:%2.0d unix: %d", dt_getYear(dt), \
175         dt_getMonth(dt), dt_getDay(dt), dt_getHour(dt), \
176         dt_getMinute(dt), dt_getSecond(dt), dt_getUnixtime(dt));
177     USB_print(to_send);
178 }
179 
180 /*************************************
181 *         SERIAL MENU FSM              *
182 *************************************/
183 
184 // Serial state variables
185 int s_main_state = S_MAIN_MENU;
186 int s_sec_state = 0; // secondary state
187 int serial_status = S_STATUS_GOOD;
188 
189 void serial_menu_init() {
190     s_main_state = S_STARTUP;
191 }
192 
193 void serial_fsm() {
194     // step to the next random value
195     PRS_Step();
196     // update last_serial_cmd based on input
197     USB_handleInput(last_serial_cmd);
198 
199     LED_Write(last_serial_cmd[0] != 0);
200     switch (s_main_state) {
201         case S_WAIT:
202         break;
203         case S_STARTUP:
204             serial_print_menu(main_menu);
205             s_main_state = S_MAIN_MENU;
206         break;
207         case S_MAIN_MENU:
208             if (last_serial_cmd[0] != 0) {
209                 int cmd = atoi(last_serial_cmd);
210                 switch(cmd) {
211                     // first case is if atoi returned 0
212                     // ie letters, etc
213                     case INVALID_CMD:
214                         serial_inv_cmd();
215                         serial_print_menu(main_menu);
216                     break;
217                     case S_MAIN_ADD:
218                         s_main_state = S_ADD_RECORD;
219                         s_sec_state = ADD_REC_START;
220                         USB_print("\n\rAdd record...");
221                     break;
222                     case S_MAIN_VIEW:
```

```
223                         s_main_state = S_STARTUP;
224                         USB_print("\n\rView All Records...");
225                         serial_printAllRecords();
226                     break;
227                     case S_MAIN_FORMAT:
228                         USB_print("\n\rWARNING: FORMATTING SD CARD");
229                         USB_print("\n\rDo you really want to format the SD card?");
230                         USB_print("\n\rTo cancel, type n and press enter");
231                         USB_print("\n\rType \"yes, delete all records\" to continue: ");
232                         s_main_state = S_FORMAT_SD_PROMPT;
233                     break;
234                     default:
235                         {
236                         USB_print("\n\rGot Command: ");
237                         char to_send[5];
238                         sprintf(to_send, "%d %s", cmd, last_serial_cmd);
239                         USB_print(to_send);
240                         serial_print_menu(main_menu);
241                         }
242                     break;
243                 }
244                 // always reset last_serial_cmd
245                 // after handling an inputted command
246                 //last_serial_cmd[0] = 0;
247                 clr_last_serial_cmd();
248             };
249         break;
250         case S_ADD_RECORD:
251             switch(s_sec_state) {
252                 case ADD_REC_START:
253                     // clear the current record
254                     new_record = (Record){"","","",false,""};
255                     USB_print("\n\rEnter Record Title: ");
256                     s_sec_state = ADD_REC_TITLE;
257                 break;
258                 case ADD_REC_TITLE:
259                     // once something has been entered
260                     if (last_serial_cmd[0] != 0) {
261                         // update curr record with title
262                         strcpy(new_record.title, last_serial_cmd);
263                         clr_last_serial_cmd();
264                         // set up for username entry
265                         USB_print("\n\rEnter username for ");
266                         USB_print(new_record.title);
267                         USB_print(": ");
268                         s_sec_state = ADD_REC_UNAME;
269                     }
270                 break;
271                 case ADD_REC_UNAME:
272                     // once a title has been entered
273                     if (last_serial_cmd[0] != 0) {
274                         // update curr record with username
275                         strcpy(new_record.username, last_serial_cmd);
276                         clr_last_serial_cmd();
277                         // set up for pw entry
278                         USB_print("\n\rPassword for user ");
279                         USB_print(new_record.username);
280                         USB_print(": ");
281                         serial_print_menu(pass_menu);
282                         s_sec_state = ADD_REC_PASS_MENU;
283                     }
284                 break;
285                 case ADD_REC_PASS_MENU:
286                     if (last_serial_cmd[0] != 0) {
287                         // listen for menu entry
288                         int cmd = atoi(last_serial_cmd);
289                         switch(cmd) {
290                             // first case is if atoi returned 0
```

```
                        // ie letters, etc
                        case INVALID_CMD:
                            serial_inv_cmd();
                            serial_print_menu(pass_menu);
                        break;
                        case PASS_MENU_ENTER:
                            USB_print("\n\rEnter Password for user ");
                            USB_print(new_record.username);
                            USB_print(": ");
                            s_sec_state = ADD_REC_PASS_ENTRY;
                        break;
                        case PASS_MENU_GEN:
                            {
                            USB_print("\n\rGenerating password! ");

                            // generate password
                            char pass[17];
                            generate_password(pass);
                            strcpy(new_record.password, pass);
                            USB_print("\n\rGenerated Password for user ");
                            USB_print(new_record.username);
                            USB_print(": ");
                            USB_print(new_record.password);
                            // ask user if they want a TOTP code to be added
                            USB_print("\n\r Would you like to enter a TOTP key? y/n: ");
                            s_sec_state = ADD_REC_PROMPT_TOTP;
                            }
                        break;
                    }
                    clr_last_serial_cmd();
                }
            break;
            case ADD_REC_PASS_ENTRY:
                // password entry option, password entered
                if (last_serial_cmd[0] != 0) {
                    // update curr record with username
                    strcpy(new_record.password, last_serial_cmd);
                    clr_last_serial_cmd();
                    // show password to user
                    USB_print("\n\rSet password for user ");
                    USB_print(new_record.username);
                    USB_print(": ");
                    USB_print(new_record.password);
                    // ask user if they want a TOTP code to be added
                    USB_print("\n\r Would you like to enter a TOTP key? y/n: ");
                    s_sec_state = ADD_REC_PROMPT_TOTP;
                }
            break;
            case ADD_REC_PROMPT_TOTP:
                // check whether the user wants to input totp key
                if (last_serial_cmd[0] != 0) {
                    if (last_serial_cmd[0] == 'y') {
                        // ask user to enter their totp key
                        USB_print("\n\rEnter TOTP Key (20 bytes, base32 encoded)");
                        USB_print("\n\rNote: Only 6-digit TOTP generation currently supported
                            ");
                        USB_print("\n\r> ");
                        s_sec_state = ADD_REC_TOTP_ENTRY;
                    } else {
                        USB_print("\n\rSkipping TOTP Entry");
                        new_record.totp_enabled = false;
                        s_sec_state = ADD_REC_CONF_PROMPT;
                    }
                    clr_last_serial_cmd(); // clear command buffer
                }
            break;
            case ADD_REC_TOTP_ENTRY:
                // wait for the user to finish entering their totp code
```

```
358                     // and save it in the new_record
359                     if (last_serial_cmd[0] != 0) {
360                         // convert base32 ascii input to byte array
361                         // we use google authenticator base32 code for this
362                         // and store the key directly into the new record
363                         base32_decode((uint8_t*) last_serial_cmd, new_record.totp_key, 20);
364                         new_record.totp_enabled = true; // enable totp for this record
365                         // re encode to confirm correctness and echo to user
366                         // we just reuse last_serial_cmd as a buffer to avoid allocating 40 bytes
367                         base32_encode(new_record.totp_key, 20, last_serial_cmd,
                                last_serial_cmd_length);
368                         USB_print("\n\rRecieved TOTP Key: ");
369                         USB_print(last_serial_cmd);
370                         s_sec_state = ADD_REC_CONF_PROMPT; // next: ask user to confirm
371                         clr_last_serial_cmd();
372                     }
373                 break;
374                 case ADD_REC_CONF_PROMPT:
375                     // echo full record and prompt user to confirm
376                     USB_print("\n\rFull Record to Add:");
377                     serial_print_record(new_record);
378                     USB_print("\n\r\n\rEnter 'y' to confirm, anything else to cancel.");
379                     s_sec_state = ADD_REC_CONFIRM;
380                 break;
381                 case ADD_REC_CONFIRM:
382                     // handle user confirmation input
383                     if (last_serial_cmd[0] != 0) {
384                         if (last_serial_cmd[0] == 'y') {
385                             // new_record should be complete
386                             // save the new record to sd
387                             SD_addRecord(&new_record);
388                             // tell ui to reload records
389                             serial_status = S_STATUS_REQUIRE_RELOAD;
390                         } else {
391                             USB_print("\n\rCancelling!");
392                         }
393                         clr_last_serial_cmd();
394                         s_main_state = S_MAIN_MENU;
395                         serial_print_menu(main_menu);
396                     }
397                 break;
398             }
399         break;
400         case S_FORMAT_SD_PROMPT:
401             // check if user inputted "yes, delete all records"
402             if (last_serial_cmd[0] != 0) {
403                 if (!strcmp(last_serial_cmd, "yes, delete all records")) {
404                     USB_print("\n\rPlease enter a 4-digit pin: ");
405                     s_main_state = S_FORMAT_SD;
406                 } else {
407                     // that command was not entered, cancel
408                     USB_print("\n\rCancelling, not modifying records");
409                     s_main_state = S_STARTUP;
410                 }
411                 clr_last_serial_cmd();
412             }
413         break;
414         case S_FORMAT_SD:
415             // collect the pin they entered
416             if (last_serial_cmd[0] != 0) {
417                 if (strlen(last_serial_cmd) == 4) {
418                     // this will format the SD card and recreate the db file
419                     USB_print("\n\rFormatting SD... ");
420                     // pass pin into create db
421                     SD_createDB(last_serial_cmd);
422                     USB_print("SD card formatted successfully.");
423                     s_main_state = S_STARTUP;
424                 } else {
```

```
                    USB_print("\n\rNot the right length :( must be 4");
                    s_main_state = S_STARTUP;
                }
                clr_last_serial_cmd();
            }
        break;
    }
}


/* [] END OF FILE */
```

A.3 OLED UI

Listing 3: pk_ui.h

```
/* 6.115 Final Project: Password Keeper
    Ben Kettle bkettle@mit.edu

    holds functions
    for the OLED UI
*/

#ifndef PK_UI
#define PK_UI

#include <project.h>
#include <ssd1306.h>
#include <pk_record.h>
#include <pk_usb.h>
#include <pk_sdrecorddb.h>
#include <pk_serialfuncs.h>
#include <stdbool.h>
#include <stdio.h>

// address for the oled
#define DISPLAY_ADDRESS 0x3C // 011110+SA0+RW - 0x3C or 0x3D

// for input from main
bool ui_forceBufferUpdate;

// current login pin
char login_pin[5];

// dislay states
#define STARTUP 0
#define LOGIN 1
#define RECORD_SELECT 2
#define RECORD_DETAIL 3
#define TYPE_CREDENTIAL 7
#define DISPLAY_CREDENTIALS 4
#define DISPLAY_TOTP 5
#define MANAGE_RECORDS 6 // over serial
uint8 display_state;


/************************************************
*       VALUES FOR RECORD DETAIL MENU           *
************************************************/
// cursor values for detail menu
#define RECORD_TYPE_USER 0
#define RECORD_TYPE_PW 1
#define RECORD_VIEW_CREDENTIALS 2
#define RECORD_VIEW_TOTP 3
#define RECORD_BACK 4

// menu items for record detail menu
```

```
52 int detail_menu_size;
53 char detail_menu_items[5][20];
54 // menu coordinates
55 int detail_menu_top;
56 int detail_menu_left;
57
58 // hold current records for menu
59 int buff_offset; // sd card location of first record
60 int buff_size; // length of the buffer
61 int num_loaded; // number of records currently loaded in buff
62 Record record_buffer[12];
63
64 // to store the currently selected record
65 Record curr_record;
66
67 // store the previous index to calculate if a change occurred
68 int prev_index;
69
70 // UI Methods
71
72 /* displays the currently selected record
73     prominently, as well as the two records
74     behind and in front of it, or "list end"
75     if these records don't exist. Also centers
76     the buffer if approaching the end of the record
77     buffer by calling ui_updateBuffer
78
79     inputs:
80     buff: pointer to the start of the record buffer
81         to draw records from
82     curr_index: the absolute index of the current record
83 */
84 void ui_draw_recordList(Record * buff, int curr_index);
85
86 /* update the position of the
87     currently selected pointer
88     in the record detail menu.
89     Also restricts dial range to
90     that of the menu size
91
92     input:
93     cursor: the index of the currently
94     selected menu option
95 */
96 void ui_recordDetailUpdate(int cursor);
97
98 /* draws the record detail
99     menu for the given record
100
101     input:
102     record: the currently selected record
103     whose title will be shown
104 */
105 void ui_drawRecordDetail(Record record);
106
107 /* the main user interface
108     finite state machine,
109     to be called every loop
110
111     input:
112     enc_sw_flag: the pointer to
113     the flag for the encoder switch,
114     so that the FSM can disable the
115     flag after handling it.
116 */
117 void ui_fsm(bool * enc_sw_flag);
118
119
```

```
120
121 #endif
122
123 /* [] END OF FILE */
```

Listing 4: pk_ui.c

```
1  /* 6.115 Final Project: Password Keeper
2      Ben Kettle bkettle@mit.edu
3
4      holds functions
5      for the OLED UI
6  */
7
8  #include <pk_ui.h>
9  #include "otp.h" // one time password generation
10 #include "ds3231.h" // current time for totp
11
12 uint8 display_state = STARTUP;
13
14 // for external input
15 bool ui_forceBufferUpdate;
16
17
18 /**************
19 * FOR LOGIN   *
20 **************/
21 char login_pin[5] = {0};
22
23 /*************************************************
24 *       VALUES FOR RECORD DETAIL MENU             *
25 *************************************************/
26 // menu items for record detail menu
27 int detail_menu_size = 5;
28 char detail_menu_items[5][20] = {
29     "type username",
30     "type password",
31     "view credentials",
32     "view TOTP code",
33     "<- go back"
34 };
35 // menu coordinates
36 int detail_menu_top = 20;
37 int detail_menu_left = 20;
38
39 // load first items from memory
40 int buff_offset = 0; // sd card index of first record in buffer
41 int buff_size = 12; // max number of records in buffer
42 int num_loaded = 0; // the number currently in buffer
43 Record record_buffer[12]; // the record buffer itself
44
45 // to store the currently selected record
46 Record curr_record;
47
48 // store the previous index to calculate if a change occurred
49 int prev_index = 0;
50
51 /* debug: print the current record
52     buffer to UART
53 */
54 void debug_printBuff() {
55     UART_1_PutString("\n\r------------");
56     char to_print[10];
57     sprintf(to_print, "%d", SD_getNumRecords());
58     UART_1_PutString(to_print);
59     for (int i=0; i<num_loaded; i++) {
60         serial_print_record(record_buffer[i]);
61     }
```

```
62  }
63
64  /* update the buffer by loading
65      new records from the sd card. Loads 5 records
66      behind the current index, then the record at the
67      current index, then 6 records after the curreent index.
68
69      input:
70      curr_index: the absolute index (on the sd card)
71      of the currently selected record
72  */
73  void ui_updateBuffer(int curr_index) {
74      // determine where the new buffer should start
75      // assume it starts a 0 first
76      buff_offset = 0;
77      // check if there are >5 previous records to load
78      // if so, we won't load all records from start
79      if (curr_index - 5 > 0)
80          // set the start of the buffer
81          // to 5 behind the current
82          buff_offset = curr_index-5;
83
84      // find how many records we should load
85      // based on the number in the SD card
86      int sd_size = SD_getNumRecords();
87
88      // first assume we hold all records after starting record in buff
89      num_loaded = sd_size - buff_offset;
90      // if there are more total records than
91      // will fit in the buffer, update this
92      if (buff_offset + buff_size < num_loaded)
93          num_loaded = buff_size;
94
95      // read the correct set of records into the buffer
96      SD_readRecords(buff_offset, num_loaded, record_buffer);
97  }
98
99
100 /*********************
101  * UI METHODS ********
102  *******************/
103
104 /* displays the currently selected record
105     prominently, as well as the two records
106     behind and in front of it, or "list end"
107     if these records don't exist. Also centers
108     the buffer if approaching the end of the record
109     buffer by calling ui_updateBuffer
110
111     inputs:
112     buff: pointer to the start of the record buffer
113         to draw records from
114     curr_index: the absolute index of the current record
115 */
116 void ui_drawRecordList(Record * buff, int curr_index) {
117     // calculate where in the buffer our current index is
118     int buff_index = curr_index - buff_offset;
119     // handle overflow beyond bounds
120     if (buff_index < 0) {
121         Dial_SetCounter(0);
122         curr_index = 0; // don't count as scroll - don't update buffer
123         buff_index = 0;
124     } else if (buff_index > num_loaded -1) {
125         // reached end of records
126         // loaded in the buffer
127         // don't move further obvs
128         // if we're here, there must not be any more
129         // records in the sd card
```

```
130         // because that should be taken care of at the end
131         // of this function (centering the current rec)
132         // thus, set to total number on sd card - 1
133         // there is an edge case at the bottom, need to
134         // prevent index from going negative
135         curr_index = buff_index + buff_offset -1; // don't count as scroll
136         // make sure we don't make curr index negative
137         if (curr_index < 0) curr_index = 0;
138         // prevent dial from moving beyond end of list
139         Dial_SetCounter(curr_index);
140         buff_index = num_loaded - 1;
141         // make sure we don't make buff index negative
142         if (buff_index < 0) buff_index = 0;
143     }
144
145     // update buffer if required by external input
146     if (ui_forceBufferUpdate) {
147         ui_updateBuffer(curr_index);
148         ui_forceBufferUpdate = false; // handled it
149     }
150
151     // draw relevant 5 records on screen
152     display_clear();
153     gfx_fillRect(0, 22, 120, 20, WHITE);
154     gfx_setTextSize(1);
155     gfx_setCursor(0,6);
156     gfx_setTextColor(WHITE);
157     gfx_setTextSize(1);
158     // only print these if we're not near the bottom end, otherwise "list end"
159     // buffer is always loaded from the bottom, so we can compare to 1,2 directly
160     gfx_println(buff_index >= 2 ? buff[buff_index-2].title : "list end"); // -2 line
161     gfx_println(buff_index >= 1 ? buff[buff_index-1].title : "list end"); // -1 line
162     gfx_setTextSize(2);
163     gfx_setTextColor(BLACK);
164     gfx_setCursor(5,24);
165     // if there are no records loaded, print db empty, otherwise curr title
166     // current record is larger, with white background
167     gfx_println(num_loaded > 0 ? buff[buff_index].title : "db empty"); // 0 line
168     gfx_setTextSize(1);
169     gfx_setTextColor(WHITE);
170     gfx_setCursor(0,42);
171     // only print these if we're not near the top end, otherwise "list end"
172     gfx_println(buff_index <= num_loaded-2 ? buff[buff_index+1].title : "list end"); // +1 line
173     gfx_println(buff_index <= num_loaded-3 ? buff[buff_index+2].title : "list end"); // +2 line
174     display_update();
175
176     // BUFFER CENTERING
177     // update record buffer if approaching the end of the buffer
178     // and we have a different cursor value than last time
179     if (curr_index != prev_index) { // check if scrolled
180         if (curr_index - buff_offset < 3) {
181             // currently highlighted is the third from the bottom
182             // we want this to be middle-ish, so update
183             ui_updateBuffer(curr_index);
184         } else if (curr_index - buff_offset > buff_size - 4) {
185             // 3rd from top is currently hightlighted
186             // so center it by updating buffer
187             ui_updateBuffer(curr_index);
188         }
189     }
190 }
191
192 /* update the position of the
193     currently selected pointer
194     in the record detail menu.
195     Also restricts dial range to
196     that of the menu size
197
```

```
198     input:
199     cursor: the index of the currently
200     selected menu option
201 */
202 void ui_recordDetailUpdate(int cursor) {
203     // force dial position into bounds
204     if (cursor < 0) {
205         // too low
206         Dial_SetCounter(0);
207         cursor = 0;
208     } else if (cursor > detail_menu_size -1) {
209         // too high
210         Dial_SetCounter(detail_menu_size - 1);
211         cursor = detail_menu_size - 1;
212     }
213     // this uses the coordinates of the menu from draw_record_detail
214     // assumes 4 entries for the height (44px rectangle height)
215     gfx_fillRect(0,detail_menu_top,detail_menu_left,44,BLACK);
216     // draw the cursor at the current position
217     gfx_drawCircle(10,24+9*cursor,2,WHITE);
218     // update the display
219     display_update();
220 }
221
222 /* draws the record detail
223     menu for the given record
224
225     input:
226     record: the currently selected record
227     whose title will be shown
228 */
229 void ui_drawRecordDetail(Record record) {
230     display_clear();
231     gfx_setCursor(0,6);
232     gfx_setTextSize(1);
233     gfx_setTextColor(WHITE);
234     gfx_println(record.title); // curr record title
235     // print each of the menu options
236     for (int i = 0; i<detail_menu_size; i++) {
237         gfx_setCursor(detail_menu_left, detail_menu_top + 9*i);
238         gfx_println(detail_menu_items[i]);
239     }
240     display_update();
241     Dial_SetCounter(0); // reset dial to start at top item
242     ui_recordDetailUpdate(Dial_GetCounter()); // draw the cursor
243 }
244
245 /* draw the username and password
246     on the oled for the given record
247
248     input:
249     record: the record to print the
250     credentials of
251 */
252 void ui_drawRecordCredentials(Record record) {
253     display_clear();
254     gfx_setCursor(0,6);
255     gfx_setTextSize(1);
256     gfx_setTextColor(WHITE);
257     gfx_println(record.title);
258     gfx_print("user:\n\r  "); gfx_println(record.username);
259     gfx_print("pass:\n\r  "); gfx_println(record.password);
260     gfx_println("<- push to go back");
261     display_update();
262 }
263
264 /* generate and draw the current TOTP
265     code for the given record
```

```
*/
void ui_drawTOTP(Record record) {
    display_clear();
    gfx_setCursor(0,6);
    gfx_setTextSize(1);
    gfx_setTextColor(WHITE);
    gfx_println(record.title); // print record title
    if (record.totp_enabled) {
        // only print the totp code if enabled
        gfx_setTextSize(2);
        char totp_str[10]; // holds ascii representation of totp
        int unixtime = dt_getUnixtime(RTC_now()); // get current unix time
        // calculate totp value and print to screen
        sprintf(totp_str, "%06d", calc_totp(unixtime, record.totp_key, 20, 30, 0));
        gfx_println(totp_str);
    } else {
        // no totp code for this record
        gfx_println("TOTP disabled :(");
        gfx_setTextSize(1);
        gfx_println("<- push to go back");
    }
    display_update();
}

/* draw the login screen based on
    the current login_pin and the
    currently selected digit

    input:
    curr_dig: the ascii value of
    the currently selected digit
*/
void ui_drawLogin(int curr_dig) {
    display_clear();
    // draw four underlines
    gfx_fillRect(5, 48, 22, 4, WHITE);
    gfx_fillRect(37, 48, 22, 4, WHITE);
    gfx_fillRect(69, 48, 22, 4, WHITE);
    gfx_fillRect(101, 48, 22, 4, WHITE);

    // draw numbers
    gfx_setTextSize(3);
    int pin_len = strlen(login_pin);
    char curr_char[2] = {0};
    for (int i=0; i<=pin_len; i++) {
        gfx_setCursor(8+32*i, 20);
        // previous digits
        curr_char[0] = login_pin[i];
        // current digit
        if (i == pin_len) {
            curr_char[0] = curr_dig;
        }
        gfx_print(curr_char);
    }
    display_update();
}

void ui_fsm(bool * enc_sw_flag) {
    // handle display updates
    int curr_index = Dial_GetCounter();
    switch (display_state) {
        case STARTUP:
            display_clear();
            gfx_setCursor(10,10);
            gfx_setTextSize(2);
            gfx_setTextColor(WHITE);
            gfx_println("LOGIN");
            display_update();
```

```
            display_state = LOGIN;
            // clear the login pin
            memset(login_pin, 0, sizeof(login_pin));
            Dial_SetCounter('0'); // start at ascii number 0
            CyDelay(100);
        break;
        case LOGIN:
        {
            // limit dial range to ascii 0-10 for this
            uint8_t curr_dig = Dial_GetCounter();
            if (curr_dig < '0') {
                curr_dig = '0';
                Dial_SetCounter('0');
            } else if (curr_dig > '9') {
                curr_dig = '9';
                Dial_SetCounter('9');
            }

            // draw the login screen based on bounded dial val
            // also uses login_pin to find previous vals
            ui_drawLogin(curr_dig);

            // handle button press
            // either move on to next digit or try to login
            if (*enc_sw_flag == true) {
                int pin_len = strlen(login_pin);
                if (pin_len < 3) {
                    // move on to next digit
                    // lock in currently selected digit
                    login_pin[pin_len] = curr_dig;
                    Dial_SetCounter('0'); // reset for next digit
                } else if (pin_len == 3) {
                    // assign final digit
                    login_pin[pin_len] = curr_dig;
                    // done, try to login with this pin
                    if (enc_checkPin(login_pin)) {
                        // success! PIN Correct
                        // key should be saved,
                        // so we can proceed to record select
                        display_state = RECORD_SELECT;
                        Dial_SetCounter(0);
                        curr_index = 0;
                        // set the initial buffer up
                        ui_updateBuffer(0);
                        ui_drawRecordList(record_buffer, curr_index);
                    } else {
                        // login failed, try again
                        display_state = STARTUP;
                    }
                }
            }
        }
        break;
        case RECORD_SELECT:
            // handle any screen updates due to scrolling
            // this check is being glitchy so skipping it
            //if (Dial_GetCounter() != prev_dial) {
            if (true) {
                // could load new values from SD card if nearing buffer ends
                // if close to end of sd card, don't update--last item in buff
                // should be last item on SD card.
                ui_drawRecordList(record_buffer, curr_index);
                curr_record = record_buffer[curr_index - buff_offset];
            };
            if (*enc_sw_flag) {
                display_state = RECORD_DETAIL;
                ui_drawRecordDetail(curr_record);
            }
```

```
402         break;
403         case RECORD_DETAIL:
404             ui_recordDetailUpdate(curr_index);
405             if (*enc_sw_flag) {
406                 switch (curr_index) {
407                     case RECORD_TYPE_USER:
408                     {
409                         KB_TypeString(curr_record.username, true);
410                         // typestring writes to screen, so redraw.
411                         ui_drawRecordDetail(curr_record);
412                         // can set counter to 1, then update
413                         // to have it go to password next
414                         Dial_SetCounter(1);
415                         ui_recordDetailUpdate(1);
416                     }
417                     break;
418                     case RECORD_TYPE_PW:
419                     {
420                         KB_TypeString(curr_record.password, true);
421                         // typestring writes to screen, so redraw.
422                         ui_drawRecordDetail(curr_record);
423                     }
424                     break;
425                     case RECORD_VIEW_CREDENTIALS:
426                     {
427                         ui_drawRecordCredentials(curr_record);
428                         display_state = DISPLAY_CREDENTIALS;
429                     }
430                     break;
431                     case RECORD_VIEW_TOTP:
432                     {
433                         ui_drawTOTP(curr_record);
434                         display_state = DISPLAY_TOTP;
435 
436                     }
437                     break;
438                     case RECORD_BACK:
439                         // just go back to the record menu
440                         display_state = RECORD_SELECT;
441                         Dial_SetCounter(0);
442                         ui_updateBuffer(0);
443                         ui_drawRecordList(record_buffer, curr_index);
444                     break;
445                 }
446             }
447         break;
448         case DISPLAY_CREDENTIALS:
449             // basically just wait for button push
450             if (*enc_sw_flag) {
451                 // return to main record menu on button push
452                 display_state = RECORD_DETAIL;
453                 Dial_SetCounter(0);
454                 ui_updateBuffer(0);
455                 ui_drawRecordDetail(curr_record);
456             }
457         break;
458         case DISPLAY_TOTP:
459             // refresh if enough time has passed?
460             // not super necessary feature but would be dope
461             // return to menu on button press
462             if (*enc_sw_flag) {
463                 // return to main record menu on button push
464                 display_state = RECORD_DETAIL;
465                 Dial_SetCounter(0);
466                 ui_updateBuffer(0);
467                 ui_drawRecordDetail(curr_record);
468             }
469         break;
```

```
470         case MANAGE_RECORDS:
471         break;
472     }
473     prev_index = curr_index;
474 }
475
476 /* [] END OF FILE */
```

A.4 Record Type

Listing 5: pk_record.h

```
1  /*
2      6.115 Final Project: Password Keeper
3      Ben Kettle bkettle@mit.edu
4
5      Defines the Record datatype
6      for use holding username,
7      password, title data.
8  */
9
10 #ifndef PK_RECORD
11 #define PK_RECORD
12
13 #include <stdbool.h> // for totp_enabled
14
15 // struct to hold each item, when decrypted they should fit this format
16 // 96 bytes total
17 typedef struct {
18     char title[10];
19     char username[39];
20     char password[26];
21     bool totp_enabled; // whether or not totp is enabled for this record
22     unsigned char totp_key[20];
23 } Record;
24 int record_size; // record size in bytes
25
26 #endif
27
28 /* [] END OF FILE */
```

Listing 6: pk_record.c

```
1  /*
2      6.115 Final Project: Password Keeper
3      Ben Kettle bkettle@mit.edu
4
5      Defines the Record datatype
6      for use holding username,
7      password, title data.
8  */
9
10 int record_size = 96; // record size in bytes
11
12 /* [] END OF FILE */
```

A.5 SD Card Record Database

Listing 7: pk_sdrecorddb.h

```
1  /*
2      6.115 Final Project: Password Keeper
3      Ben Kettle bkettle@mit.edu
4
5      Defines functions for
```

```
6       interfacing with the
7       sd-card database
8   */
9
10  #ifndef SD_RECORDDB
11  #define SD_RECORDDB
12
13  #include <FS.h> // for SD interface
14  #include <stdbool.h> // bool type
15  #include <pk_record.h> // record type
16  #include "aes.h" // encryption
17  #include <project.h> // USB/UART output
18
19  #include <stdio.h>
20
21  char sdFile[9];
22  char sdVolName[10];
23  FS_FILE * pFile;  // hold current file pointer
24
25  bool unlocked; // stores whether the device is currently unlocked
26  bool enc_checkPin(char * pin);
27
28  /* formats the SD card, creating a new database
29      file and initializing it with the key-hash of the
30      given pin.
31
32      inputs:
33      pin_str: a pointer to the 4-byte char array of
34      ascii-encoded PIN digits that will be used to
35      encrypt records in this database
36  */
37  void SD_createDB();
38
39  /* read the saved key hash from the SD card
40      for use when checking whether a given PIN
41      is correct
42
43      inputs:
44      hash_out: pointer to a 20-byte char array where
45      the correct hash will be stored.
46  */
47  void SD_readKeyHash(char* hash_out);
48
49  /* read a set of records from the SD card,
50      and decrypt them using the global saved key
51
52      inputs:
53      rec_index: the integer record index that should
54      be the first record to be fetched
55
56      num_records: the number of records to fetch sequentially
57      from rec_index
58
59      out: a pointer to the first location in memory
60      that the records should be stored
61  */
62  bool SD_readRecords(int rec_index, int num_records, Record * out);
63
64  /* encrypt and write a specified record
65      to the SD card, placing it at the end of the
66      database.
67
68      inputs:
69      to_add: pointer to the record that should be
70      added to the SD card database
71
72      returns: bool, true if successful, false otherwise
73  */
```

```
74 bool SD_addRecord(Record * to_add);
75
76 /* read the number of records currently stored
77     on the SD card
78
79     returns: integer number of records on SD
80 */
81 int SD_getNumRecords();
82
83 #endif
84
85 /* [] END OF FILE */
```

Listing 8: pk_sdrecorddb.c

```
1  /*
2      6.115 Final Project: Password Keeper
3      Ben Kettle bkettle@mit.edu
4
5      Defines functions for
6      interfacing with the
7      sd-card database
8  */
9
10 #include <pk_sdrecorddb.h>
11
12 #include "sha1.h" // for key generation and hashing
13 #include "pk_usb.h" // just for debug
14 #include "pk_serialfuncs.h" // for debug
15
16 //Record example_record = {"example", "record", "this is an example"};
17
18 char sdFile[9] = "pwdb"; // filename of database
19 /* for now, key hash and records are stored in the same file.
20     it would be a bit cleaner to store them in two files,
21     but all in one contributes to security by obscurity a bit
22     which seems like it can't hurt
23     also just seems best to avoid complicatedness
24     with opening multiple files
25 */
26
27 int key_hash_len = 20; // length of db header that holds the hash of the enc key
28
29 // used to decrypt and encrypt records once unlocked
30 uint8_t unlocked_key[16] = {0}; // will be filled in with real correct key when unlocked
31 bool unlocked = false; // whether the correct pin has been entered
32
33 // ENCRYPTION FUNCTIONS
34 struct AES_ctx ctx; // required for the AES library
35
36 /* encrypts the record starting at the given
37     pointer using the global stored key
38
39     inputs:
40     to_encrypt: pointer to the record
41     to encrypt
42 */
43 void enc_encryptRecord(Record * to_encrypt) {
44     // encrypts the record
45     // record size must be a mutliple of 16
46     AES_init_ctx(&ctx, unlocked_key); // initialize AES functions with key and ctx
47     for (int i=0; i<(record_size/16); i++) {
48         // encrypt each 16-bit block and replace the data
49         // in its original memory location with the encrypted version
50         AES_ECB_encrypt(&ctx, (uint8_t *) to_encrypt + i*16);
51         // we use ECB because not much data will be repeated and
52         // for simplicity when decrypting unknown chunks
53     }
```

```
54  }
55
56  /* decrypts the record starting at the given
57      pointer using the global stored key
58
59      inputs:
60      to_decrypt: pointer to the record
61      to decrypt
62  */
63  void enc_decryptRecord(Record * to_decrypt) {
64      // decrypts the record
65      AES_init_ctx(&ctx, unlocked_key); // initialize AES functions with key and ctx
66      for (int i=0; i<(record_size/16); i++) {
67          // replaces each 16-bit block of data with the decrypted version
68          AES_ECB_decrypt(&ctx, (uint8_t *) to_decrypt + i*16);
69      }
70  }
71
72  /* Derives an encryption key from the given pin.
73      the same pin will always give the same key.
74
75      inputs:
76      pin_str: pointer to a 4-byte array of the pin
77      encoded as ascii characters.
78
79      key_out: pointer to a 16-byte char array
80      where the derived key will be output to
81  */
82  void enc_pinToKey(char* pin_str, char* key_out) {
83      char hash_out[20];
84      SHA1(hash_out, pin_str, 4); // find the SHA-1 hash of the PIN
85      strncpy(key_out, hash_out, 16); // copy first 16 bytes as key
86  }
87
88  /* hashes a given key to find the key-hash
89
90      inputs:
91      key: pointer to a 16-byte array holding the
92      key bytes
93
94      hash_out: pointer to a 20-byte array that
95      will hold the hash of the key
96  */
97  void enc_getKeyHash(char * key, char * hash_out) {
98      // hashes a 16-byte key using SHA1 to get a 20-byte
99      // hash_out value
100     SHA1(hash_out, key, 16);
101 }
102
103 /* checks whether a given PIN string is correct
104     by comparing the corresponding keyhash to the
105     correct one stored on the SD card. If correct,
106     sets `unlocked` to true.
107
108     inputs:
109     pin: pointer to a 4-byte array of the pin
110     encoded as ascii characters.
111
112     returns:
113     bool: true if correct,
114         false if incorrect.
115 */
116 bool enc_checkPin(char * pin) {
117     char key[16];
118     enc_pinToKey(pin, key); // find the key for this pin
119     char key_hash[20];
120     enc_getKeyHash(key, key_hash); // hash the key for keyhash
121     char correct_hash[20];
```

```
122     SD_readKeyHash(correct_hash); // get the correct keyhash
123     // compare the two and return true if there is no diff
124     if (strncmp(key_hash, correct_hash, key_hash_len) == 0) {
125         // no difference, key is correct
126         // copy key to unlocked_key for later use
127         strncpy((char*) unlocked_key, key, 16);
128         unlocked = true; // allow device to open
129         return true;
130     } else {
131         // there is a difference in the hashes, incorrect
132         return false;
133     }
134 }
135 // END ENCRYPTION
136 
137 /*************************************
138  * descriptions of the following
139  * functions given in the header file
140  *************************************/
141 
142 void SD_createDB(char* pin_str) {
143     UART_1_PutString("\n\r SD card format");
144 
145     char sdVolName[10];     /* Buffer that will hold SD card Volume name */
146 
147     /* Get volume name of SD card #0 */
148     if(0 != FS_GetVolumeName(0u, &sdVolName[0], 9u))
149     {
150         // display volume name
151         UART_1_PutString("\n\r SD card name:");
152         UART_1_PutString(sdVolName);
153     }
154     else
155     {
156         // getting volume name failed
157         UART_1_PutString("\n\r Failed to get ");
158         UART_1_PutString("SD card name");
159     }
160 
161     if(0 == FS_FormatSD(sdVolName)) {
162         UART_1_PutString(" Succeeded");
163     }
164     else {
165         UART_1_PutString(" Failed");
166     }
167 
168     /* Create specific file for modification */
169     pFile = FS_FOpen(sdFile, "w");
170 
171     /* Check if file was created */
172     if(pFile) {
173 
174         // generate key hash
175         char key[16];
176         enc_pinToKey(pin_str, key);
177         char key_hash[20];
178         enc_getKeyHash(key, key_hash);
179         // debug statements
180         USB_print("\n\rGot key "); serial_printHex(key, 16);
181         USB_print("\n\rfrom pin "); serial_printHex(pin_str, 4);
182         USB_print("\n\rand hash "); serial_printHex(key_hash, 20);
183 
184         // write key hash to the file
185         FS_Write(pFile, key_hash, key_hash_len);
186 
187         if(0 == FS_FClose(pFile)) {
188             UART_1_PutString("\n\r File was closed");
189         }
```

```
190         else {
191             UART_1_PutString("\n\r Failed to close");
192             UART_1_PutString("file");
193         }
194     }
195     else {
196         UART_1_PutString("\n\r Failed to create");
197         UART_1_PutString(" file");
198     }
199
200     UART_1_PutString("\n\r Format complete ");
201 }
202
203 void SD_readKeyHash(char* hash_out) {
204     // reads the stored hash of the encryption
205     // key and copies it into hash_out
206     // -> hash_out must be 20 bytes long (hash length for SHA1)
207     pFile = FS_FOpen(sdFile, "r");
208     if (pFile) {
209         // check that the database actually is long enough
210         // to hold a hash of the key
211         int db_size = FS_GetFileSize(pFile);
212         if (db_size < 20) {
213             // not enough records in db, abort
214             UART_1_PutString("\n\r !! BAD NEWS! SDDB is too short!");
215         }
216
217         // cursor should already be at the start
218         FS_Read(pFile, hash_out, key_hash_len); // read first (hash length) bytes from db
219
220         // hash is stored in plaintext (obviously, or else we couldn't
221         // use it to check if the key is correct) so we're done
222         // close the file
223         FS_FClose(pFile);
224     }
225 }
226
227 bool SD_readRecords(int rec_index, int num_records, Record * out) {
228     /* records are offset by key_hash_len
229         to make space for key hash at header
230     */
231     // open the db file
232     pFile = FS_FOpen(sdFile, "r");
233
234     if (pFile) {
235         // should first check that the file is long enough
236         // to hold all the records the person asked for
237         // and the key hash in the header
238         int db_size = FS_GetFileSize(pFile);
239         if (db_size < key_hash_len + (rec_index + num_records)*record_size) {
240             // not enough records in db, abort
241             UART_1_PutString("\n\r !! Requested records outside range!");
242             return false;
243         }
244
245         // move the cursor to the start of the record we want
246         // this record should be located at rec_index * record_size + key_hash_len
247         if (0 != FS_FSeek(pFile, key_hash_len + rec_index*record_size, FS_SEEK_CUR)) {
248             // seek failed, abort
249             UART_1_PutString("\n\r !! Failed to move cursor!");
250             return false;
251         }
252
253         FS_Read(pFile, out, record_size*num_records);
254
255         // decrypt the records
256         for (int i = 0; i<num_records; i++) {
257             // find the address of each record and decrypt
```

```
258             enc_decryptRecord(&out[i]);
259         }
260 
261         if (0 != FS_FClose(pFile)) {
262             // failed to close file
263             UART_1_PutString("\n\r !! Failed to close file!");
264             return false;
265         }
266     }
267 
268     return false;
269 }
270 
271 bool SD_addRecord(Record * to_add) {
272     // open the db file for APPENDING (a)
273     pFile = FS_FOpen(sdFile, "a");
274 
275     if (pFile) {
276         // encrypt record
277         enc_encryptRecord(to_add);
278 
279         // cursor should be at the end of the file, bc appending
280         // write the record in to_add to the file at the cursor pos
281         // should be record_size long (I hope)
282         FS_Write(pFile, to_add, record_size);
283 
284         if (0 != FS_FClose(pFile)) {
285             // failed to close file
286             UART_1_PutString("\n\r !! Failed to close file!");
287             return false;
288         }
289         UART_1_PutString("\n\r Record added successfully!");
290     }
291 
292     return false;
293 }
294 
295 
296 int SD_getNumRecords() {
297     pFile = FS_FOpen(sdFile, "r");
298 
299     int db_size = 0;
300 
301     if (pFile) {
302         // should first check that the file is long enough
303         // to hold all the records the person asked for
304         db_size = FS_GetFileSize(pFile);
305     }
306 
307     if (0 != FS_FClose(pFile)) {
308         // failed to close file
309         UART_1_PutString("\n\r !! Failed to close file!");
310     }
311 
312     char to_print[10];
313     sprintf(to_print, "%d", db_size);
314     UART_1_PutString(to_print);
315     // need to subtract size of key has from db_size
316     // to get the size taken up by records.
317     return ((db_size - key_hash_len)/record_size);
318 }
319 
320 /* [] END OF FILE */
```

A.6 USB Interface

Listing 9: pk_usb.h

```
/*
    6.115 Final Project: Password Keeper
    Ben Kettle bkettle@mit.edu

    functions to abstract the use of the
    keyboard and COM port emulation
*/

#ifndef PK_USB
#define PK_USB

#include <project.h>  // USBFS APIs
#include <stdbool.h>  // bool type
#include <ssd1306.h>  // OLED

/*****************************
 * USB COM Port Emulation   *
 *****************************/

/* handles input from the PC
 * should be called before interpreting
 * serial commands.
 *
 * input:
 *  cmd_output: a pointer to a 40-byte char array
 *  to hold the next complete command
 */
void USB_handleInput(char * cmd_output);

/* wait for the host PC to be ready
 * and print the given string over USB-UART
 *
 * input:
 *  str_to_print: a pointer to a null-terminated
 *  char array that will be printed to the user
 */
void USB_print(char * str_to_print);

/* returns whether the USB host is
 * ready to recieve a transmission
 *
 * returns:
 *  bool, true if ready, false otherwise
 */
bool USB_isReady();

/*******************************************************
*     KEYBOARD EMULATION STUFF                         *
*******************************************************/

// holds the endpoint numbers for the KB emulation
enum
{
    EP_KEYBOARD_IN = 4,
    EP_KEYBOARD_OUT = 5
};

bool KB_Initialized; // whether the keyboard has been enumerated

// buffer of data to be sent to the host endpoint
int8 KB_Data[8]; // [0]: Modifiers [1]: reserved [2:7]: Key codes
typedef struct {
    int uid;
    bool uppercase;
} hid_usage; // holds usage IDs as well as the state of the shift modifier

```

```
67  // this will start at ascii 32 to get to keyboard values
68  // converts ascii values to hid_usage types
69  hid_usage ascii_to_hid[96];
70
71  /* types a single character over the HID keyboard
72   * protocol, blocking until finished. Requires inputting
73   * the HID Usage Code, for sending ascii use TypeString
74   *
75   * inputs:
76   *  hid_id: an hid_usage type specifying usage codes
77   *  and shift status of the char to be typed.
78   */
79  void KB_TypeChar(hid_usage hid_id); // types a single character
80
81  /* tests the keyboard connection
82   * by tying every ASCII character from 32 upwards
83   * sequentially (all those that can be typed)
84   */
85  void ascii_test();
86
87  /* initializes the USB connection with the PC,
88   * allowing the PSoC to enumerate. Blocks until
89   * the device enumerates, and draws a message on
90   * the screen.
91   */
92  void USB_Initialize();
93
94  /* Types a full string over the HID keyboard protocol
95   *      note: this function blocks
96   *      for a fraction of a second
97   *
98   * inputs:
99   *   str_to_type: a pointer to a
100  *      null-terminated character array that
101  *      will be typed char-by-char
102  *  draw: whether or not to update the
103  *      OLED with the typing status
104  */
105 void KB_TypeString(char* str_to_type, bool draw);
106
107 #endif
108
109 /* [] END OF FILE */
```

Listing 10: pk_usb.c

```
1   /*
2       6.115 Final Project: Password Keeper
3       Ben Kettle bkettle@mit.edu
4
5       functions to abstract the use of the
6       keyboard and COM port emulation
7   */
8
9   #include <pk_usb.h>
10
11  // holds command in progress
12  char rx_buffer[40] = {0};
13  int rx_buff_length = 40;
14
15  // hold current USB input
16  // has to be long to support copy/paste
17  char inputbuffer[40];
18
19  void clr_rx_buffer(){
20      memset(rx_buffer, 0, 40);
21      rx_buff_length = 0;
22  };
```

```
23
24  /*****************************
25   * Virtual COM Port Functions  *
26   *****************************/
27  void USB_handleInput(char * cmd_output) {
28      // updates the buffer given based on the
29      // input recieved
30      if (USBFS_DataIsReady()) {
31          while (USBFS_DataIsReady()) {
32              // this only works as intended for 1 character at a time,
33              // could add a loop over inputbuffer to fix this
34              int count = USBFS_GetAll((uint8_t*) inputbuffer);
35              inputbuffer[count] = 0; // null terminate the input
36              if (inputbuffer[0] == 0xD) {
37                  // move buffer into cmd_output on enter
38                  strcpy(cmd_output, rx_buffer);
39                  clr_rx_buffer();
40              } else if (inputbuffer[0] == 0x7F) {
41                  // handle backspaces
42                  int rx_len = strlen(rx_buffer);
43                  if (rx_len > 0) {
44                      // if there is something in rx_buffer
45                      // then replace the last thing with \0
46                      rx_buffer[rx_len - 1] = 0;
47                  }
48                  // echo the backspace to the terminal
49                  USBFS_PutChar(0x7F);
50              } else {
51                  USB_print(inputbuffer);    // echo recieved character
52                  // add non-enter characters to rx buffer
53                  strcat(rx_buffer, inputbuffer);
54              }
55          }
56      }
57  }
58
59  void USB_print(char * str_to_print) {
60      // wait until computer is ready
61      // then send.
62      while (!USBFS_CDCIsReady());
63      USBFS_PutString(str_to_print);
64  }
65
66  bool USB_isReady() {
67      return USBFS_CDCIsReady();
68  }
69
70
71
72  /**********************
73   * KEYBOARD FUNCTIONS *
74   **********************/
75
76  // initialize keyboard variables
77  bool KB_Initialized = false;
78  int8 KB_Data[8] = {0, 0, 0, 0, 0, 0, 0, 0}; // [0]: Modifiers [1]: reserved [2:7]: Key codes
79
80  // this will start at ascii 32 to get to keyboard values
81  hid_usage ascii_to_hid[] = {
82      {0x2c, 0}, {0x1e, 1}, {0x34, 1}, {0x20, 1}, {0x21, 1}, {0x22, 1}, {0x24, 1}, {0x34, 0}, {0x26
            , 1}, {0x27, 1}, {0x25, 1}, {0x2e, 1}, {0x36, 0}, {0x2d, 0}, {0x37, 0}, {0x38, 0},
83      {0x27, 0}, {0x1e, 0}, {0x1f, 0}, {0x20, 0}, {0x21, 0}, {0x22, 0}, {0x23, 0}, {0x24, 0}, {0x25
            , 0}, {0x26, 0}, {0x33, 1}, {0x33, 0}, {0x36, 1}, {0x2e, 0}, {0x37, 1}, {0x38, 1},
84      {0x1f, 1}, {0x04, 1}, {0x05, 1}, {0x06, 1}, {0x07, 1}, {0x08, 1}, {0x09, 1}, {0x0a, 1}, {0x0b
            , 1}, {0x0c, 1}, {0x0d, 1}, {0x0e, 1}, {0x0f, 1}, {0x10, 1}, {0x11, 1}, {0x12, 1},
85      {0x13, 1}, {0x14, 1}, {0x15, 1}, {0x16, 1}, {0x17, 1}, {0x18, 1}, {0x19, 1}, {0x1a, 1}, {0x1b
            , 1}, {0x1c, 1}, {0x1d, 1}, {0x2f, 0}, {0x31, 0}, {0x30, 0}, {0x23, 1}, {0x2d, 1},
86      {0x35, 0}, {0x04, 0}, {0x05, 0}, {0x06, 0}, {0x07, 0}, {0x08, 0}, {0x09, 0}, {0x0a, 0}, {0x0b
```

```
        , 0}, {0x0c, 0}, {0x0d, 0}, {0x0e, 0}, {0x0f, 0}, {0x10, 0}, {0x11, 0}, {0x12, 0},
87      {0x13, 0}, {0x14, 0}, {0x15, 0}, {0x16, 0}, {0x17, 0}, {0x18, 0}, {0x19, 0}, {0x1a, 0}, {0x1b
        , 0}, {0x1c, 0}, {0x1d, 0}, {0x2f, 1}, {0x31, 1}, {0x30, 1}, {0x35, 1}, {0x2a, 0},
88  };
89
90  void KB_TypeChar(hid_usage hid_id) {
91      // send the current char
92      while(!USBFS_GetEPAckState(EP_KEYBOARD_IN));                        /* Wait for ACK before
            loading data */
93      KB_Data[2] = hid_id.uid;
94      KB_Data[0] = 2*(int)hid_id.uppercase;
95      USBFS_LoadInEP(EP_KEYBOARD_IN, (uint8 *)KB_Data, 8);        /* Load latest mouse data into
            EP1 and send to PC */
96
97      // send a blank keystroke to clear the state of shift
98      while(!USBFS_GetEPAckState(EP_KEYBOARD_IN));                        /* Wait for ACK before
            loading data */
99      KB_Data[2] = 0;
100     KB_Data[0] = 0;
101     USBFS_LoadInEP(EP_KEYBOARD_IN, (uint8 *)KB_Data, 8);        /* Load latest mouse data into
            EP1 and send to PC */
102 }
103
104 void ascii_test() {
105     for (unsigned int i=0;i<95;i++){
106         LED_Write(0);
107         KB_TypeChar(ascii_to_hid[i]);
108         LED_Write(1);
109     }
110 }
111
112 void USB_Initialize() {
113     // usb keyboard setup
114     // exit if we already intialized the keyboard
115     if (KB_Initialized) return;
116     display_clear();
117     gfx_setCursor(20,20);
118     gfx_println("initializing usb...");
119     gfx_println("must be connected to micro-usb port");
120     display_update();
121     CyGlobalIntEnable; /* Enable global interrupts. */
122     USBFS_Start(0, USBFS_DWR_VDDD_OPERATION);   /* Start USBFS Operation/device 0 and with 5V
            operation */
123     LED_Write(1);
124         while(!USBFS_bGetConfiguration()) {                     /* Wait for Device to enumerate
                */
125         // should add a timeout here
126     }
127     LED_Write(0);
128     USBFS_LoadInEP(EP_KEYBOARD_IN, (uint8 *)KB_Data, 8);        /* Loads an inital value into EP1
             and sends it out to the PC */
129     USBFS_CDC_Init(); // initialize cdc comms
130     KB_Initialized = true;
131 }
132
133 void KB_TypeString(char* str_to_type, bool draw) {
134     // enters the given string through keyboard emulation
135     // takes a li bit of time
136     USB_Initialize();
137     if (KB_Initialized) {
138         if (draw) {
139             display_clear();
140             gfx_setCursor(20,40);
141             gfx_println("typing...");
142             display_update();
143         }
144         for (unsigned int i = 0; i<strlen(str_to_type); i++) {
145             // the ascii to hid table starts at ascii 32
```

```
146             int shifted_ascii = (int) str_to_type[i] - 32;
147             // type the current character
148             KB_TypeChar(ascii_to_hid[shifted_ascii]);
149         }
150         //USB_print(str_to_type);
151         if (draw) {
152             gfx_setCursor(20,48);
153             gfx_println("done!");
154             display_update();
155             CyDelay(400);
156         }
157     }
158     else {
159         display_clear();
160         gfx_setCursor(20,40);
161         gfx_println("failed, not connected");
162         display_update();
163         CyDelay(600);
164     }
165 }
166
167 /* [] END OF FILE */
```

A.7 TOTP from HMAC

Listing 11: otp.h

```
1  // OTP implementation using HMAC
2  // for a little endian system
3
4  #ifndef OTP_H
5  #define OTP_H
6
7  /* calculates the 6-digit HOTP value
8      based on the given counter and key.
9
10     inputs:
11     counter: integer to be used as the counter
12         in the HOTP algorithm
13     key: pointer to the start of the key for the HOTP algorithm
14     key_len: length of the key
15
16     returns:
17     integer: the HOTP value (when represented in base 10)
18 */
19 int calc_hotp(int counter, char * key, int key_len);
20
21 /* calculates the 6-digit HOTP value
22     based on the given key and time,
23     as well as timestep information.
24
25     inputs:
26     unixtime: the unix time the TOTP should be calculated for
27     key: pointer to the start of the key
28     key_len: length of the key
29     timestep: the time period to split into. By default in the
30         totp standard this is 30 seconds.
31     T0: the start offset for the time. By default in the
32         totp standard this is 0.
33
34     returns:
35     integer: the HOTP value (when represented in base 10)
36 */
37 int calc_totp(int unixtime, char* key, int key_len, int timestep, int T0);
38
39 #endif
```

Listing 12: otp.c

```c
// OTP implementation using HMAC
// for a little endian system

#include "otp.h"
#include "project.h"
#include "hmac.h"
#include <stdio.h>

// counter can be any integer, will be converted to 8 bytes
int calc_hotp(int counter, char * key, int key_len) {
    // convert counter to byte array
    uint64_t c_int = (uint64_t) counter; // convert counter to 8 Bytes
    // this function compiles to a single instruction
    c_int = __builtin_bswap64(c_int); // convert counter to big-endian
    unsigned char * c = (unsigned char *) &c_int; // get pointer to char array for thats

    unsigned char result[20];
    uint rlen = 20;
    // generate the HMAC-SHA1 digest of the key and counter
    hmac_sha1((uint8_t *) key, key_len, c, 8, result, &rlen);

    // Calculate offset from last nibble of digest
    // just take lower 4 bits of the last byte
    uint8_t offset = (uint8_t) result[19] & 0xF;

    // TRUNCATE
    // take all but msb of result[offset]...result[offset+3]
    // aka we want a substring of 4 bytes starting at offset
    char trunc[4];
    strncpy(trunc, (char*) result+offset, 4);
    // delete most significant bit of trunc
    trunc[0] = trunc[0] & 0x7F;

    // compute hotp value
    // convert these 4 bytes into a single int
    uint32_t t_num = trunc[0] << 24 | trunc[1] << 16 | trunc[2] << 8 | trunc[3];

    // mod with 10^6 for a 6-digit number
    // that's our answer!
    uint32_t hotp_val = t_num % 1000000;

    return hotp_val;
}

int calc_totp(int unixtime, char* key, int key_len, int timestep, int T0) {
    // by default, timestep is 30 seconds and t0 is 0.
    // calculate number of time-periods
    int T = (unixtime - T0) / timestep;
    return calc_hotp(T, key, key_len);
}
```

A.8 DS3231 Interface

Listing 13: ds3231.h

```c
/* Very simple library for DS3231
    on PSoC, modified from Adafruit's
    RTClib. Only crucial functionality
    is ported over.

    Original code released under the MIT
    licence by Adafruit:

    MIT License

    Copyright (c) 2019 Adafruit Industries
```

```
12
13      Permission is hereby granted, free of charge, to any person obtaining a copy
14      of this software and associated documentation files (the "Software"), to deal
15      in the Software without restriction, including without limitation the rights
16      to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
17      copies of the Software, and to permit persons to whom the Software is
18      furnished to do so, subject to the following conditions:
19
20      The above copyright notice and this permission notice shall be included in all
21      copies or substantial portions of the Software.
22
23      THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
24      IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
25      FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
26      AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
27      LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
28      OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
29      SOFTWARE.
30  */
31
32  #ifndef DS3231_H
33  #define DS3231_H
34
35  #include <project.h>
36  #include <stdbool.h>
37
38
39  #define DS3231_ADDRESS 0x68   ///< I2C address for DS3231
40  #define DS3231_TIME 0x00      ///< Time register
41  #define DS3231_ALARM1 0x07    ///< Alarm 1 register
42  #define DS3231_ALARM2 0x0B    ///< Alarm 2 register
43  #define DS3231_CONTROL 0x0E   ///< Control register
44  #define DS3231_STATUSREG 0x0F ///< Status register
45  #define DS3231_TEMPERATUREREG                                                  \
46    0x11 ///< Temperature register (high byte - low byte is at 0x12), 10-bit
47         ///< temperature value
48
49  /** Constants */
50  #define SECONDS_PER_DAY 86400L ///< 60 * 60 * 24
51  #define SECONDS_FROM_1970_TO_2000                                              \
52    946684800 ///< Unixtime for 2000-01-01 00:00:00, useful for initialization
53
54
55  // we emulate Adafruit's DateTime class using a struct
56  typedef struct {
57      uint8_t yOff; // Year offset from 2000
58      uint8_t m;    // Month 1-12
59      uint8_t d;    // Day 1-31
60      uint8_t hh;   // Hours 0-23
61      uint8_t mm;   // Minutes 0-59
62      uint8_t ss;   // Seconds 0-59
63  } DateTime;
64
65  // return the year from a DateTime struct
66  uint16_t dt_getYear(DateTime dt);
67  // return the month from a DateTime struct
68  uint8_t dt_getMonth(DateTime dt);
69  // return the day from a DateTime struct
70  uint8_t dt_getDay(DateTime dt);
71  // return the hour from a DateTime struct
72  uint8_t dt_getHour(DateTime dt);
73  // return the minute from a DateTime struct
74  uint8_t dt_getMinute(DateTime dt);
75  // return the second from a DateTime struct
76  uint8_t dt_getSecond(DateTime dt);
77  // return whether the time is PM from a DateTime struct
78  uint8_t dt_isPM(DateTime dt);
79
```

```
/* 32-bit times as seconds since 2000-01-01. */
uint32_t dt_getSecondstime(DateTime dt);
/* 32-bit times as seconds since 1970-01-01. */
uint32_t dt_getUnixtime(DateTime dt);
uint8_t dt_twelveHour(DateTime dt);

/*
    set the current time on the connected RTC module

    input:
    dt: DateTime struct containing the time and date
    that should be written to the RTC
*/
void RTC_setTime(DateTime dt);

/*
    read the current time from the connected RTC

    returns:
    DateTime, the current time according to the
    connected RTC
*/
DateTime RTC_now();


#endif
/* [] END OF FILE */
```

Listing 14: ds3231.c

```
/* Very simple library for DS3231
    on PSoC, modified from Adafruit's
    RTClib. Only crucial functionality
    is ported over.

    See .h file for license information
*/

#include <project.h>
#include <stdlib.h>
#include <stdbool.h>

#include <ds3231.h>

#define pgm_read_byte(addr) (*(const unsigned char *)(addr))

// this stuff should go in the header but that's not working
uint8 RTC_read_buff[10];

// return the year from a DateTime struct
uint16_t dt_getYear(DateTime dt) { return 2000 + dt.yOff; }
// return the month from a DateTime struct
uint8_t dt_getMonth(DateTime dt) { return dt.m; }
// return the day from a DateTime struct
uint8_t dt_getDay(DateTime dt) { return dt.d; }
// return the hour from a DateTime struct
uint8_t dt_getHour(DateTime dt) { return dt.hh; }
// return the minute from a DateTime struct
uint8_t dt_getMinute(DateTime dt) { return dt.mm; }
// return the second from a DateTime struct
uint8_t dt_getSecond(DateTime dt) { return dt.ss; }
// return whether the time is PM from a DateTime struct
uint8_t dt_isPM(DateTime dt) { return dt.hh >= 12; }

/**************************************************************************/
/*
    Read a byte from an I2C register

```

```
39      addr I2C address
40      reg Register address
41      returns:
42        Register value
43  */
44  /**************************************************************************/
45  uint8_t read_i2c_register(uint8_t addr, uint8_t reg) {
46      I2CRTC_MasterSendStart(addr,0); // send write command to addr
47      I2CRTC_MasterWriteByte(reg);
48      I2CRTC_MasterSendStop();
49
50      I2CRTC_MasterReadBuf(addr, RTC_read_buff, 1, I2CRTC_MODE_COMPLETE_XFER);
51      return RTC_read_buff[0];
52  }
53
54  /**************************************************************************/
55  /*!
56      @brief  Write a byte to an I2C register
57      @param addr I2C address
58      @param reg Register address
59      @param val Value to write
60  */
61  /**************************************************************************/
62  void write_i2c_register(uint8_t addr, uint8_t reg, uint8_t val) {
63    I2CRTC_MasterSendStart(addr,0); // send write command to addr
64    I2CRTC_MasterWriteByte(reg);
65    I2CRTC_MasterWriteByte(val);
66    I2CRTC_MasterSendStop();
67  }
68
69
70
71  /**************************************************************************/
72  // utility code, some of this could be exposed in the DateTime API if needed
73  /**************************************************************************/
74
75  /**
76    Number of days in each month, from January to November. December is not
77    needed. Omitting it avoids an incompatibility with Paul Stoffregen's Time
78    library. C.f. https://github.com/adafruit/RTClib/issues/114
79  */
80  const uint8_t daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30};
81
82  /**************************************************************************/
83  /*!
84      @brief  Given a date, return number of days since 2000/01/01,
85              valid for 2000--2099
86      @param y Year
87      @param m Month
88      @param d Day
89      @return Number of days
90  */
91  /**************************************************************************/
92  uint16_t date2days(uint16_t y, uint8_t m, uint8_t d) {
93    if (y >= 2000)
94      y -= 2000;
95    uint16_t days = d;
96    for (uint8_t i = 1; i < m; ++i)
97      days += pgm_read_byte(daysInMonth + i - 1);
98    if (m > 2 && y % 4 == 0)
99      ++days;
100   return days + 365 * y + (y + 3) / 4 - 1;
101 }
102
103 /**************************************************************************/
104 /*!
105     @brief  Given a number of days, hours, minutes, and seconds, return the
106    total seconds
```

```
107       @param days Days
108       @param h Hours
109       @param m Minutes
110       @param s Seconds
111       @return Number of seconds total
112   */
113   /*****************************************************************************/
114   uint32_t time2ulong(uint16_t days, uint8_t h, uint8_t m, uint8_t s) {
115     return ((days * 24UL + h) * 60 + m) * 60 + s;
116   }
117
118   // these were adapted from DateTime classes to build DateTime structs
119   /*****************************************************************************/
120   /*!
121       Generates a DateTime struct from the given unix time.
122   */
123   /*****************************************************************************/
124   DateTime dt_fromUnix(uint32_t t) {
125     t -= SECONDS_FROM_1970_TO_2000; // bring to 2000 timestamp from 1970
126
127     DateTime dt = {};
128     dt.ss = t % 60;
129     t /= 60;
130     dt.mm = t % 60;
131     t /= 60;
132     dt.hh = t % 24;
133     uint16_t days = t / 24;
134     uint8_t leap;
135     for (dt.yOff = 0;; ++dt.yOff) {
136       leap = dt.yOff % 4 == 0;
137       if (days < 365U + leap)
138         break;
139       days -= 365 + leap;
140     }
141     for (dt.m = 1; dt.m < 12; ++dt.m) {
142       uint8_t daysPerMonth = pgm_read_byte(daysInMonth + dt.m - 1);
143       if (leap && dt.m == 2)
144         ++daysPerMonth;
145       if (days < daysPerMonth)
146         break;
147       days -= daysPerMonth;
148     }
149     dt.d = days + 1;
150
151     return dt;
152   }
153
154   /*!
155       Generates a DateTime struct from the given date and time
156   */
157   DateTime dt_fromYMD(uint16_t year, uint8_t month, uint8_t day, uint8_t hour,
158                       uint8_t min, uint8_t sec) {
159     DateTime dt = {};
160     if (year >= 2000)
161       year -= 2000;
162     dt.yOff = year;
163     dt.m = month;
164     dt.d = day;
165     dt.hh = hour;
166     dt.mm = min;
167     dt.ss = sec;
168     return dt;
169   }
170
171   /*!
172       Returns the unix time that corresponds with the given datetime struct
173   */
174   uint32_t dt_getUnixtime(DateTime dt) {
```

```
175   uint32_t t;
176   uint16_t days = date2days(dt.yOff, dt.m, dt.d);
177   t = time2ulong(days, dt.hh, dt.mm, dt.ss);
178   t += SECONDS_FROM_1970_TO_2000; // seconds from 1970 to 2000
179
180   return t;
181 }
182
183 /*!
184     @brief  Convert the DateTime to seconds since 1 Jan 2000
185     The result can be converted back to a DateTime with:
186     ```cpp
187     DateTime(SECONDS_FROM_1970_TO_2000 + value)
188     ```
189     @return Number of seconds since 2000-01-01 00:00:00.
190 */
191 uint32_t dt_getSecondstime(DateTime dt) {
192   uint32_t t;
193   uint16_t days = date2days(dt.yOff, dt.m, dt.d);
194   t = time2ulong(days, dt.hh, dt.mm, dt.ss);
195   return t;
196 }
197
198 /******************************************************************************/
199 /*!
200     @brief  Convert a binary coded decimal value to binary. RTC stores time/date
201    values as BCD.
202     @param val BCD value
203     @return Binary value
204 */
205 /******************************************************************************/
206 uint8_t bcd2bin(uint8_t val) { return val - 6 * (val >> 4); }
207
208 /******************************************************************************/
209 /*!
210     @brief  Convert a binary value to BCD format for the RTC registers
211     @param val Binary value
212     @return BCD value
213 */
214 /******************************************************************************/
215 uint8_t bin2bcd(uint8_t val) { return val + 6 * (val / 10); }
216
217
218 /*
219     set the current time on the connected RTC module
220
221     input:
222     dt: DateTime struct containing the time and date
223     that should be written to the RTC
224 */
225 void RTC_setTime(DateTime dt) {
226     //I2CRTC_MasterSendStart(DS3231_ADDRESS,0); // send write command to addr
227     //I2CRTC_MasterWriteByte(DS3231_TIME);// start at location 0
228     uint8 tx_buff[8] = {
229         0,                          // start writing at register 0
230         bin2bcd(dt_getSecond(dt)),  // write BCD second in reg 1
231         bin2bcd(dt_getMinute(dt)),  // write BCD minute in reg 2
232         bin2bcd(dt_getHour(dt)),    // write BCD hour in reg 3
233         0,                          // day of week is unecessary
234         bin2bcd(dt_getDay(dt)),     // current date is in reg 5
235         bin2bcd(dt_getMonth(dt)),   // month in reg 6
236         bin2bcd(dt_getYear(dt)-2000), // year offset in reg 7
237     };
238     I2CRTC_MasterWriteBuf(DS3231_ADDRESS,tx_buff,8,I2CRTC_MODE_COMPLETE_XFER );   // write the tx
          buffer
239     while(0u == (I2CRTC_MasterStatus() & I2CRTC_MSTAT_WR_CMPLT));      // Waiting for writing to
          finish
240     I2CRTC_MasterClearStatus(); // clear the status
```

```
}

/*
    read the current time from the connected RTC

    returns:
    DateTime, the current time according to the
    connected RTC
*/
DateTime RTC_now() {
    // first, tell the RTC which register we would like to start reading from
    uint8 tx_buff[1] = {0}; // we want register 0 in this case, see datasheet
    I2CRTC_MasterWriteBuf(DS3231_ADDRESS,tx_buff,1,I2CRTC_MODE_COMPLETE_XFER ); // Send buffer
    while(0u == (I2CRTC_MasterStatus() & I2CRTC_MSTAT_WR_CMPLT));      // Waiting for write to
        finish
    I2CRTC_MasterClearStatus();

    // the time and date information is stored in the first 7 registers
    I2CRTC_MasterReadBuf(DS3231_ADDRESS, RTC_read_buff, 7, I2CRTC_MODE_COMPLETE_XFER); // read
        the first 7 registers
    while(0u == (I2CRTC_MasterStatus() & I2CRTC_MSTAT_RD_CMPLT)); // Wait for read to finish
    I2CRTC_MasterClearStatus();

    uint8_t ss = bcd2bin(RTC_read_buff[0] & 0x7F); // register 0 holds seconds
    uint8_t mm = bcd2bin(RTC_read_buff[1]); // minutes
    uint8_t hh = bcd2bin(RTC_read_buff[2]); // hours
                                            // day of week is in register 3, unneeded
    uint8_t d = bcd2bin(RTC_read_buff[4]); // day
    uint8_t m = bcd2bin(RTC_read_buff[5]); // month
    uint16_t yOff = bcd2bin(RTC_read_buff[6]); // year
    I2CRTC_MasterClearReadBuf(); // clear the read buffer

    DateTime dt = {yOff, m, d, hh, mm, ss}; // construct a DateTime struct from the above
    return dt;
}

// sqw, alarm and temperature functionality not ported yet, see adafruit RTClib

/* [] END OF FILE */
```

B Code Listings: External Sources

Note: all code after this point is from outside sources, and is cited in my references page.

B.1 SSD1306 Interface

Listing 15: ssd1306.h

```
/* Graphics Library and I2C wrapper
 * for the SSD1306 OLED driver and the
 * Cypress PSoC.
 *
 * Original code is from
 * Derk Steggewentz, 3/2015
 * github.com/derkst/Cypress-PSOC-OLED
 * with no license restrictions.
 *
 * The graphics portions of this code
 * (gfx_ functions) are modified from
 * the Adafruit graphics library, which is released
 * under the following BSD license:
 *
 * Software License Agreement (BSD License)
 *
```

```
17  * Copyright (c) 2012 Adafruit Industries. All rights reserved.
18  *
19  * Redistribution and use in source and binary forms, with or without
20  * modification, are permitted provided that the following conditions are met:
21  * - Redistributions of source code must retain the above copyright notice,
22  * this list of conditions and the following disclaimer.
23  * - Redistributions in binary form must reproduce the above copyright notice,
24  * this list of conditions and the following disclaimer in the documentation
25  * and/or other materials provided with the distribution.
26  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
27  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
28  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
29  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
30  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
31  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
32  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
33  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
34  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
35  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
36  * POSSIBILITY OF SUCH DAMAGE.
37  *
38  */
39
40
41
42  #ifndef _SSD1306_H
43  #define _SSD1306_H
44
45  #define BLACK 0
46  #define WHITE 1
47  #define INVERSE 2
48
49  typedef enum{
50      SCROLL_RIGHT = 0x26,
51      SCROLL_LEFT = 0x2A
52  }SCROLL_DIR;
53
54  typedef enum{
55      SCROLL_SPEED_0 = 0x03,  // slowest
56      SCROLL_SPEED_1 = 0x02,
57      SCROLL_SPEED_2 = 0x01,
58      SCROLL_SPEED_3 = 0x06,
59      SCROLL_SPEED_4 = 0x00,
60      SCROLL_SPEED_5 = 0x05,
61      SCROLL_SPEED_6 = 0x04,
62      SCROLL_SPEED_7 = 0x07   // fastest
63  }SCROLL_SPEED;
64
65  typedef enum{
66      SCROLL_PAGE_0 = 0,
67      SCROLL_PAGE_1,
68      SCROLL_PAGE_2,
69      SCROLL_PAGE_3,
70      SCROLL_PAGE_4,
71      SCROLL_PAGE_5,
72      SCROLL_PAGE_6,
73      SCROLL_PAGE_7
74  }SCROLL_AREA;
75
76  void display_init( uint8 i2caddr );
77  void display_update(void);
78  void display_clear(void);
79  void display_stopscroll(void);
80  void display_scroll( SCROLL_AREA start, SCROLL_AREA end, SCROLL_DIR dir, SCROLL_SPEED speed );
81  void display_contrast( uint8_t contrast );
82  void display_invert( uint8_t invert );
83
84
```

```
85
86
87  void gfx_drawPixel(int16_t x, int16_t y, uint16_t color);
88  void gfx_drawLine( int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color );
89  void gfx_setCursor( int16_t x, int16_t y );
90  void gfx_setTextSize( uint8_t size );
91  void gfx_setTextColor( uint16_t color );
92  void gfx_setTextBg( uint16_t background );
93  void gfx_write( uint8_t ch );
94  int16_t gfx_width(void);
95  int16_t gfx_height(void);
96  void gfx_print( const char* s );
97  void gfx_println( const char* s );
98  void gfx_drawRect( int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color );
99  void gfx_fillRect( int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color );
100 void gfx_drawCircle( int16_t x0, int16_t y0, int16_t r,uint16_t color );
101 void gfx_drawTriangle( int16_t x0, int16_t y0,int16_t x1, int16_t y1, int16_t x2, int16_t y2,
        uint16_t color );
102 void gfx_setRotation( uint8_t x );
103
104 #endif  // _SSD1306_H
105
106
107
108 /* [] END OF FILE */
```

Listing 16: ssd1306.c

```
1   /* Graphics Library and I2C wrapper
2    * for the SSD1306 OLED driver and the
3    * Cypress PSoC.
4    *
5    * see .h file for license information
6    * and original code source
7    */
8
9
10  #include <project.h>
11  #include <stdlib.h>
12
13  #include "font.h"
14  #include "ssd1306.h"
15
16
17  #define DISPLAYWIDTH 128
18  #define DISPLAYHEIGHT 64
19
20  #define SSD1306_SETCONTRAST 0x81
21  #define SSD1306_DISPLAYALLON_RESUME 0xA4
22  #define SSD1306_DISPLAYALLON 0xA5
23  #define SSD1306_NORMALDISPLAY 0xA6
24  #define SSD1306_INVERTDISPLAY 0xA7
25  #define SSD1306_DISPLAYOFF 0xAE
26  #define SSD1306_DISPLAYON 0xAF
27  #define SSD1306_SETDISPLAYOFFSET 0xD3
28  #define SSD1306_SETCOMPINS 0xDA
29  #define SSD1306_SETVCOMDETECT 0xDB
30  #define SSD1306_SETDISPLAYCLOCKDIV 0xD5
31  #define SSD1306_SETPRECHARGE 0xD9
32  #define SSD1306_SETMULTIPLEX 0xA8
33  #define SSD1306_SETLOWCOLUMN 0x00
34  #define SSD1306_SETHIGHCOLUMN 0x10
35  #define SSD1306_SETSTARTLINE 0x40
36  #define SSD1306_MEMORYMODE 0x20
37  #define SSD1306_COLUMNADDR 0x21
38  #define SSD1306_PAGEADDR 0x22
39  #define SSD1306_COMSCANINC 0xC0
40  #define SSD1306_COMSCANDEC 0xC8
```

```
41  #define SSD1306_SEGREMAP 0xA0
42  #define SSD1306_CHARGEPUMP 0x8D
43  #define SSD1306_EXTERNALVCC 0x1
44  #define SSD1306_SWITCHCAPVCC 0x2
45  #define SSD1306_ACTIVATE_SCROLL 0x2F
46  #define SSD1306_DEACTIVATE_SCROLL 0x2E
47
48
49  // I2C result status
50  #define TRANSFER_CMPLT    (0x00u)
51  #define TRANSFER_ERROR    (0xFFu)
52
53
54  static uint32 display_write_buf( uint8* buf, uint16_t size );
55
56  void gfx_init( int16_t width, int16_t height );
57
58  static uint8 _i2caddr;
59
60  // display memory buffer ( === MUST INCLUDE === the preceding I2C 0x40 control byte for the
        display)
61  static uint8_t SSD1306_buffer[DISPLAYHEIGHT * DISPLAYWIDTH / 8 + 1] = { 0x40 };
62  // pointer to actual display memory buffer
63  static uint8_t* _displaybuf = SSD1306_buffer+1;
64  static uint16_t _displaybuf_size = sizeof(SSD1306_buffer) - 1;
65
66  // see data sheet page 25 for Graphic Display Data RAM organization
67  // 8 pages, each page a row of DISPLAYWIDTH bytes
68  // start address of of row: y/8*DISPLAYWIDTH
69  // x pos in row: == x
70  #define GDDRAM_ADDRESS(X,Y) ((_displaybuf)+((Y)/8)*(DISPLAYWIDTH)+(X))
71
72  // lower 3 bit of y determine vertical pixel position (pos 0...7) in GDDRAM byte
73  // (y&0x07) == position of pixel on row (page). LSB is top, MSB bottom
74  #define GDDRAM_PIXMASK(Y) (1 << ((Y)&0x07))
75
76  #define PIXEL_ON(X,Y) (*GDDRAM_ADDRESS(x,y) |= GDDRAM_PIXMASK(y))
77  #define PIXEL_OFF(X,Y) (*GDDRAM_ADDRESS(x,y) &= ~GDDRAM_PIXMASK(y))
78  #define PIXEL_TOGGLE(X,Y) (*GDDRAM_ADDRESS(x,y) ^= GDDRAM_PIXMASK(y))
79
80
81
82  // call before first use of other functions
83  void display_init( uint8 i2caddr ){
84
85      _i2caddr = i2caddr;
86      gfx_init( DISPLAYWIDTH, DISPLAYHEIGHT );
87
88      uint8 cmdbuf[] = {
89          0x00,
90          SSD1306_DISPLAYOFF,
91          SSD1306_SETDISPLAYCLOCKDIV,
92          0x80,
93          SSD1306_SETMULTIPLEX,
94          0x3f,
95          SSD1306_SETDISPLAYOFFSET,
96          0x00,
97          SSD1306_SETSTARTLINE | 0x0,
98          SSD1306_CHARGEPUMP,
99          0x14,
100         SSD1306_MEMORYMODE,
101         0x00,
102         SSD1306_SEGREMAP | 0x1,
103         SSD1306_COMSCANDEC,
104         SSD1306_SETCOMPINS,
105         0x12,
106         SSD1306_SETCONTRAST,
107         0xcf,
```

```
108         SSD1306_SETPRECHARGE,
109         0xf1,
110         SSD1306_SETVCOMDETECT,
111         0x40,
112         SSD1306_DISPLAYALLON_RESUME,
113         SSD1306_NORMALDISPLAY,
114         SSD1306_DISPLAYON
115     };
116
117     display_write_buf( cmdbuf, sizeof(cmdbuf) );
118 }
119
120
121 // for submitting command sequences:
122 //  buf[0] must be 0x00
123 // for submitting bulk data (writing to display RAM):
124 //  buf[0] must be 0x40
125 static uint32 display_write_buf( uint8* buf, uint16_t size ){
126
127     uint32 status = TRANSFER_ERROR;
128     int i;
129     I2COLED_MasterSendStart(_i2caddr,I2COLED_WRITE_XFER_MODE);
130
131     for ( i = 0; i < size; i++)
132     {
133         status = I2COLED_MasterWriteByte(buf[i]);
134     }
135
136     I2COLED_MasterSendStop();
137
138     return status;
139 }
140
141
142 // used by gfx_ functions. Needs to be implemented by display_
143 static void display_setPixel( int16_t x, int16_t y, uint16_t color ){
144
145     if( (x < 0) || (x >= DISPLAYWIDTH) || (y < 0) || (y >= DISPLAYHEIGHT) )
146         return;
147
148     switch( color ){
149         case WHITE:
150             PIXEL_ON(x,y);
151             break;
152         case BLACK:
153             PIXEL_OFF(x,y);
154             break;
155         case INVERSE:
156             PIXEL_TOGGLE(x,y);
157             break;
158     }
159 }
160
161
162 void display_clear(void){
163     memset( _displaybuf, 0, _displaybuf_size );
164     SSD1306_buffer[0] = 0x40; // to be sure its there
165 }
166
167
168 // contrast: 0 ...255
169 void display_contrast( uint8_t contrast ){
170
171     uint8 cmdbuf[] = {
172         0x00,
173         SSD1306_SETCONTRAST,
174         contrast
175     };
```

```
176     display_write_buf( cmdbuf, sizeof(cmdbuf) );
177 }
178 
179 
180 // invert <> 0 for inverse display, invert == 0 for normal display
181 void display_invert( uint8_t invert ){
182 
183     uint8 cmdbuf[] = {
184         0x00,
185         0
186     };
187     cmdbuf[1] = invert ? SSD1306_INVERTDISPLAY : SSD1306_NORMALDISPLAY;
188     display_write_buf( cmdbuf, sizeof(cmdbuf) );
189 }
190 
191 
192 void display_update(void) {
193 
194     uint8 cmdbuf[] = {
195         0x00,
196         SSD1306_COLUMNADDR,
197         0,                          // start
198         DISPLAYWIDTH-1, // end
199         SSD1306_PAGEADDR,
200         0,                          // start
201         7                           // end
202     };
203     display_write_buf( cmdbuf, sizeof(cmdbuf) );
204     display_write_buf( SSD1306_buffer, sizeof(SSD1306_buffer) );
205 }
206 
207 
208 // draws horizontal or vertical line
209 // Note: no check for valid coords, this needs to be done by caller
210 // should only be called from gfx_hvline which is doing all validity checking
211 static void display_line( int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color ){
212 
213     if( x1 == x2 ){
214         // vertical
215         uint8_t* pstart = GDDRAM_ADDRESS(x1,y1);
216         uint8_t* pend = GDDRAM_ADDRESS(x2,y2);
217         uint8_t* ptr = pstart;
218 
219         while( ptr <= pend ){
220 
221             uint8_t mask;
222             if( ptr == pstart ){
223                 // top
224                 uint8_t lbit = y1 % 8;
225                 // bottom (line can be very short, all inside this one byte)
226                 uint8_t ubit = lbit + y2 - y1;
227                 if( ubit >= 7 )
228                     ubit = 7;
229                 mask = ((1 << (ubit-lbit+1)) - 1) << lbit;
230             }else if( ptr == pend ){
231                 // top is always bit 0, that makes it easy
232                 // bottom
233                 mask = (1 << (y2 % 8)) - 1;
234             }
235 
236             if( ptr == pstart || ptr == pend ){
237                 switch( color ){
238                     case WHITE:     *ptr |= mask; break;
239                     case BLACK:     *ptr &= ~mask; break;
240                     case INVERSE:   *ptr ^= mask; break;
241                 };
242             }else{
243                 switch( color ){
```

```
                    case WHITE:     *ptr = 0xff; break;
                    case BLACK:     *ptr = 0x00; break;
                    case INVERSE:   *ptr ^= 0xff; break;
                };
            }

            ptr += DISPLAYWIDTH;
        }
    }else{
        // horizontal
        uint8_t* pstart = GDDRAM_ADDRESS(x1,y1);
        uint8_t* pend = pstart + x2 - x1;
        uint8_t pixmask = GDDRAM_PIXMASK(y1);

        uint8_t* ptr = pstart;
        while( ptr <= pend ){
            switch( color ){
                case WHITE:     *ptr |= pixmask; break;
                case BLACK:     *ptr &= ~pixmask; break;
                case INVERSE:   *ptr ^= pixmask; break;
            };
            ptr++;
        }
    }
}



void display_stopscroll(void){

    uint8 cmdbuf[] = {
        0x00,
        SSD1306_DEACTIVATE_SCROLL
    };
    display_write_buf( cmdbuf, sizeof(cmdbuf) );
}

void display_scroll( SCROLL_AREA start, SCROLL_AREA end, SCROLL_DIR dir, SCROLL_SPEED speed ){

    uint8 cmdbuf[] = {
        0x00,
        dir,                    // 0x26 or 0x2a
        0x00,                   // dummy byte
        start,                  // start page
        speed,                  // scroll step interval in terms of frame frequency
        end,                    // end page
        0x00,                   // dummy byte
        0xFF,                   // dummy byte
        SSD1306_ACTIVATE_SCROLL // 0x2F
    };
    display_write_buf( cmdbuf, sizeof(cmdbuf) );
}



// ============================================================
// graphics library stuff

int16_t WIDTH, HEIGHT; // This is the 'raw' display w/h - never changes
static int16_t _width, _height; // Display w/h as modified by current rotation
static int16_t cursor_x, cursor_y;
static uint16_t textcolor, textbgcolor;
static uint8_t textsize;
uint8_t rotation;
uint8_t wrap; // If set, 'wrap' text at right edge of display
```

```
312 void gfx_init( int16_t width, int16_t height ){
313     WIDTH = width;
314     HEIGHT = height;
315     _width = WIDTH;
316     _height = HEIGHT;
317 
318     rotation = 0;
319     cursor_y = cursor_x = 0;
320     textsize = 1;
321     textcolor = textbgcolor = 0xFFFF;
322     wrap = 1;
323 }
324 
325 // Return the size of the display (per current rotation)
326 int16_t gfx_width(void){
327     return _width;
328 }
329 
330 int16_t gfx_height(void){
331     return _height;
332 }
333 
334 uint8_t gfx_rotation(void){
335     return rotation;
336 }
337 
338 void gfx_setCursor( int16_t x, int16_t y ){
339     cursor_x = x;
340     cursor_y = y;
341 }
342 
343 void gfx_setTextSize( uint8_t size ){
344     textsize = (size > 0) ? size : 1;
345 }
346 
347 void gfx_setTextColor( uint16_t color ){
348     // For 'transparent' background, we'll set the bg
349     // to the same as fg instead of using a flag
350     textcolor = textbgcolor = color;
351 }
352 
353 void gfx_setTextBg( uint16_t color ){
354     textbgcolor = color;
355 }
356 
357 void gfx_setTextWrap( uint8 w ){
358     wrap = w;
359 }
360 
361 void gfx_setRotation( uint8_t x ){
362 
363     rotation = (x & 3);
364     switch( rotation ){
365         case 0:
366         case 2:
367             _width = WIDTH;
368             _height = HEIGHT;
369             break;
370         case 1:
371         case 3:
372             _width = HEIGHT;
373             _height = WIDTH;
374         break;
375     }
376 }
377 
378 static void gfx_rotation_adjust( int16_t* px, int16_t* py ){
379 
```

```
380     int16_t y0 = *py;
381
382     switch( rotation ){
383         case 1:
384             *py = *px;
385             *px = WIDTH - y0 - 1;
386             break;
387         case 2:
388             *px = WIDTH - *px - 1;
389             *py = HEIGHT - *py - 1;
390             break;
391         case 3:
392             *py = HEIGHT - *px - 1;
393             *px = y0;
394             break;
395     }
396 }
397
398 void gfx_drawPixel( int16_t x, int16_t y, uint16_t color ){
399
400     if( (x < 0) || (x >= _width) || (y < 0) || (y >= _height) )
401         return;
402
403     gfx_rotation_adjust( &x, &y );
404
405     display_setPixel(x,y,color);
406 }
407
408 // helper function for gfx_drawLine, handles special cases of horizontal and vertical lines
409 static void gfx_hvLine( int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t color ){
410
411     if( x1 != x2 && y1 != y2 ){
412         // neither vertical nor horizontal
413         return;
414     }
415
416     // bounds check
417     if( rotation == 1 || rotation == 3 ){
418         if( x1 < 0 || x1 >= HEIGHT || x2 < 0 || x2 >= HEIGHT )
419             return;
420         if( y1 < 0 || y1 >= WIDTH || y2 < 0 || y2 >= WIDTH )
421             return;
422     }else{
423         if( y1 < 0 || y1 >= HEIGHT || y2 < 0 || y2 >= HEIGHT )
424             return;
425         if( x1 < 0 || x1 >= WIDTH || x2 < 0 || x2 >= WIDTH )
426             return;
427     }
428
429     gfx_rotation_adjust( &x1, &y1 );
430     gfx_rotation_adjust( &x2, &y2 );
431
432     // ensure coords are from left to right and top to bottom
433     if( (x1 == x2 && y2 < y1) || (y1 == y2 && x2 < x1) ){
434         // swap as needed
435         int16_t t = x1; x1 = x2; x2 = t;
436         t = y1; y1 = y2; y2 = t;
437     }
438
439     display_line( x1, y1, x2, y2, color );
440 }
441
442 // always use this function for line drawing
443 void gfx_drawLine( int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color ){
444
445     if( x0 == x1 || y0 == y1 ){
446         // vertical and horizontal lines can be drawn faster
447         gfx_hvLine( x0, y0, x1, y1, color );
```

```
448         return;
449     }
450 
451     int16_t t;
452 
453     int16_t steep = abs(y1 - y0) > abs(x1 - x0);
454     if( steep ){
455         t = x0; x0 = y0; y0 = t;
456         t = x1; x1 = y1; y1 = t;
457     }
458     if( x0 > x1 ){
459         t = x0; x0 = x1; x1 = t;
460         t = y0; y0 = y1; y1 = t;
461     }
462     int16_t dx, dy;
463     dx = x1 - x0;
464     dy = abs(y1 - y0);
465     int16_t err = dx / 2;
466     int16_t ystep;
467     if( y0 < y1 ){
468         ystep = 1;
469     }else{
470         ystep = -1;
471     }
472     for( ; x0<=x1; x0++ ){
473         if( steep ){
474             gfx_drawPixel( y0, x0, color );
475         }else{
476             gfx_drawPixel( x0, y0, color );
477         }
478         err -= dy;
479         if( err < 0 ){
480             y0 += ystep;
481             err += dx;
482         }
483     }
484 }
485 
486 void gfx_drawRect( int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color ){
487 
488     gfx_drawLine( x, y, x+w-1, y, color );
489     gfx_drawLine( x, y+h-1, x+w-1, y+h-1, color );
490     gfx_drawLine( x, y, x, y+h-1, color );
491     gfx_drawLine( x+w-1, y, x+w-1, y+h-1, color );
492 }
493 
494 void gfx_fillRect( int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color ){
495     int16_t i = 0;
496     if( h > w ){
497         for( i = x ; i < x+w ; i++ ){
498             gfx_drawLine( i, y, i, y+h-1, color );
499         }
500     }else{
501         for( i = y ; i < y+h ; i++ ){
502             gfx_drawLine( x, i, x+w-1, i, color );
503         }
504     }
505 }
506 
507 
508 // circle outline
509 void gfx_drawCircle( int16_t x0, int16_t y0, int16_t r,uint16_t color ){
510 
511     int16_t f = 1 - r;
512     int16_t ddF_x = 1;
513     int16_t ddF_y = -2 * r;
514     int16_t x = 0;
515     int16_t y = r;
```

```
516     gfx_drawPixel( x0 , y0+r, color );
517     gfx_drawPixel( x0 , y0-r, color );
518     gfx_drawPixel( x0+r, y0 , color );
519     gfx_drawPixel( x0-r, y0 , color );
520     while( x < y ){
521         if( f >= 0 ){
522             y--;
523             ddF_y += 2;
524             f += ddF_y;
525         }
526         x++;
527         ddF_x += 2;
528         f += ddF_x;
529         gfx_drawPixel( x0 + x, y0 + y, color );
530         gfx_drawPixel( x0 - x, y0 + y, color );
531         gfx_drawPixel( x0 + x, y0 - y, color );
532         gfx_drawPixel( x0 - x, y0 - y, color );
533         gfx_drawPixel( x0 + y, y0 + x, color );
534         gfx_drawPixel( x0 - y, y0 + x, color );
535         gfx_drawPixel( x0 + y, y0 - x, color );
536         gfx_drawPixel( x0 - y, y0 - x, color );
537     }
538 }
539
540 void gfx_drawTriangle( int16_t x0, int16_t y0,int16_t x1, int16_t y1, int16_t x2, int16_t y2,
        uint16_t color ){
541
542     gfx_drawLine( x0, y0, x1, y1, color );
543     gfx_drawLine( x1, y1, x2, y2, color );
544     gfx_drawLine( x2, y2, x0, y0, color );
545 }
546
547
548 // Draw a character
549 void gfx_drawChar( int16_t x, int16_t y, unsigned char c,uint16_t color, uint16_t bg, uint8_t
        size) {
550     if( (x >= _width) || // Clip right
551         (y >= _height) || // Clip bottom
552         ((x + 6 * size - 1) < 0) || // Clip left
553         ((y + 8 * size - 1) < 0)) // Clip top
554         return;
555
556     int8_t i = 0;
557     for( i = 0 ; i < 6 ; i++ ){
558         uint8_t line;
559         if( i == 5 )
560             line = 0x0;
561         else
562            line = font[(c*5)+i];
563         int8_t j = 0;
564         for( j = 0; j < 8 ; j++ ){
565             if( line & 0x1 ){
566                 if( size == 1 ) // default size
567                     gfx_drawPixel( x+i, y+j, color );
568                 else { // big size
569                     gfx_fillRect( x+(i*size), y+(j*size), size, size, color );
570                 }
571             } else if( bg != color ){
572                 if( size == 1 ) // default size
573                     gfx_drawPixel( x+i, y+j, bg );
574                 else { // big size
575                     gfx_fillRect( x+i*size, y+j*size, size, size, bg );
576                 }
577             }
578             line >>= 1;
579         }
580     }
581 }
```

```
582
583 void gfx_write( uint8_t ch ){
584     if( ch == '\n' ){
585         cursor_y += textsize*8;
586         cursor_x = 0;
587     }else if( ch == '\r' ){
588         // skip em
589     }else{
590         gfx_drawChar(cursor_x, cursor_y, ch, textcolor, textbgcolor, textsize);
591         cursor_x += textsize*6;
592         if( wrap && (cursor_x > (_width - textsize*6)) ){
593             cursor_y += textsize*8;
594             cursor_x = 0;
595         }
596     }
597 }
598
599 void gfx_print( const char* s ){
600
601     unsigned int len = strlen( s );
602     unsigned int i = 0;
603     for( i = 0 ; i < len ; i++ ){
604         gfx_write( s[i] );
605     }
606 }
607
608 void gfx_println( const char* s ){
609     gfx_print( s );
610     gfx_write( '\n' );
611 }
612 /* [] END OF FILE */
```

B.2 Cryptography Algorithms

B.2.1 SHA-1

Listing 17: sha1.h

```
1  #ifndef SHA1_H
2  #define SHA1_H
3
4  /*
5     SHA-1 in C
6     By Steve Reid <steve@edmweb.com>
7     100% Public Domain
8
9     from here: https://github.com/clibs/sha1
10  */
11
12 #include "stdint.h"
13
14 /** SHA-1 Digest size in bytes */
15 #define SHA1_DIGEST_SIZE 20
16 /** SHA-1 Digest size in bytes (OpenSSL compat) */
17 #define SHA_DIGEST_LENGTH SHA1_DIGEST_SIZE
18
19 typedef struct
20 {
21     uint32_t state[5];
22     uint32_t count[2];
23     unsigned char buffer[64];
24 } SHA1_CTX;
25
26 void SHA1Transform(
27     uint32_t state[5],
28     const unsigned char buffer[64]
29     );
```

```
30
31  void SHA1Init(
32      SHA1_CTX * context
33      );
34
35  void SHA1Update(
36      SHA1_CTX * context,
37      const unsigned char *data,
38      uint32_t len
39      );
40
41  void SHA1Final(
42      unsigned char digest[20],
43      SHA1_CTX * context
44      );
45
46  void SHA1(
47      char *hash_out,
48      const char *str,
49      int len);
50
51  #endif /* SHA1_H */
```

Listing 18: sha1.c

```
1   /*
2   SHA-1 in C
3   By Steve Reid <steve@edmweb.com>
4   100% Public Domain
5
6   Test Vectors (from FIPS PUB 180-1)
7   "abc"
8     A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
9   "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"
10    84983E44 1C3BD26E BAAE4AA1 F95129E5 E54670F1
11  A million repetitions of "a"
12    34AA973C D4C4DAA4 F61EEB2B DBAD2731 6534016F
13  */
14
15  /* #define LITTLE_ENDIAN * This should be #define'd already, if true. */
16  /* #define SHA1HANDSOFF * Copies data before messing with it. */
17
18  #define SHA1HANDSOFF
19
20  #include <stdio.h>
21  #include <string.h>
22
23  /* for uint32_t */
24  #include <stdint.h>
25
26  #include "sha1.h"
27
28
29  #define rol(value, bits) (((value) << (bits)) | ((value) >> (32 - (bits))))
30
31  /* blk0() and blk() perform the initial expand. */
32  /* I got the idea of expanding during the round function from SSLeay */
33  #if BYTE_ORDER == LITTLE_ENDIAN
34  #define blk0(i) (block->l[i] = (rol(block->l[i],24)&0xFF00FF00) \
35      |(rol(block->l[i],8)&0x00FF00FF))
36  #elif BYTE_ORDER == BIG_ENDIAN
37  #define blk0(i) block->l[i]
38  #else
39  #error "Endianness not defined!"
40  #endif
41  #define blk(i) (block->l[i&15] = rol(block->l[(i+13)&15]^block->l[(i+8)&15] \
42      ^block->l[(i+2)&15]^block->l[i&15],1))
43
```

```
44  /* (R0+R1), R2, R3, R4 are the different operations used in SHA1 */
45  #define R0(v,w,x,y,z,i) z+=((w&(x^y))^y)+blk0(i)+0x5A827999+rol(v,5);w=rol(w,30);
46  #define R1(v,w,x,y,z,i) z+=((w&(x^y))^y)+blk(i)+0x5A827999+rol(v,5);w=rol(w,30);
47  #define R2(v,w,x,y,z,i) z+=(w^x^y)+blk(i)+0x6ED9EBA1+rol(v,5);w=rol(w,30);
48  #define R3(v,w,x,y,z,i) z+=(((w|x)&y)|(w&x))+blk(i)+0x8F1BBCDC+rol(v,5);w=rol(w,30);
49  #define R4(v,w,x,y,z,i) z+=(w^x^y)+blk(i)+0xCA62C1D6+rol(v,5);w=rol(w,30);
50
51
52  /* Hash a single 512-bit block. This is the core of the algorithm. */
53
54  void SHA1Transform(
55      uint32_t state[5],
56      const unsigned char buffer[64]
57  )
58  {
59      uint32_t a, b, c, d, e;
60
61      typedef union
62      {
63          unsigned char c[64];
64          uint32_t l[16];
65      } CHAR64LONG16;
66
67  #ifdef SHA1HANDSOFF
68      CHAR64LONG16 block[1];      /* use array to appear as a pointer */
69
70      memcpy(block, buffer, 64);
71  #else
72      /* The following had better never be used because it causes the
73       * pointer-to-const buffer to be cast into a pointer to non-const.
74       * And the result is written through.  I threw a "const" in, hoping
75       * this will cause a diagnostic.
76       */
77      CHAR64LONG16 *block = (const CHAR64LONG16 *) buffer;
78  #endif
79      /* Copy context->state[] to working vars */
80      a = state[0];
81      b = state[1];
82      c = state[2];
83      d = state[3];
84      e = state[4];
85      /* 4 rounds of 20 operations each. Loop unrolled. */
86      R0(a, b, c, d, e, 0);
87      R0(e, a, b, c, d, 1);
88      R0(d, e, a, b, c, 2);
89      R0(c, d, e, a, b, 3);
90      R0(b, c, d, e, a, 4);
91      R0(a, b, c, d, e, 5);
92      R0(e, a, b, c, d, 6);
93      R0(d, e, a, b, c, 7);
94      R0(c, d, e, a, b, 8);
95      R0(b, c, d, e, a, 9);
96      R0(a, b, c, d, e, 10);
97      R0(e, a, b, c, d, 11);
98      R0(d, e, a, b, c, 12);
99      R0(c, d, e, a, b, 13);
100     R0(b, c, d, e, a, 14);
101     R0(a, b, c, d, e, 15);
102     R1(e, a, b, c, d, 16);
103     R1(d, e, a, b, c, 17);
104     R1(c, d, e, a, b, 18);
105     R1(b, c, d, e, a, 19);
106     R2(a, b, c, d, e, 20);
107     R2(e, a, b, c, d, 21);
108     R2(d, e, a, b, c, 22);
109     R2(c, d, e, a, b, 23);
110     R2(b, c, d, e, a, 24);
111     R2(a, b, c, d, e, 25);
```

```
    R2(e, a, b, c, d, 26);
    R2(d, e, a, b, c, 27);
    R2(c, d, e, a, b, 28);
    R2(b, c, d, e, a, 29);
    R2(a, b, c, d, e, 30);
    R2(e, a, b, c, d, 31);
    R2(d, e, a, b, c, 32);
    R2(c, d, e, a, b, 33);
    R2(b, c, d, e, a, 34);
    R2(a, b, c, d, e, 35);
    R2(e, a, b, c, d, 36);
    R2(d, e, a, b, c, 37);
    R2(c, d, e, a, b, 38);
    R2(b, c, d, e, a, 39);
    R3(a, b, c, d, e, 40);
    R3(e, a, b, c, d, 41);
    R3(d, e, a, b, c, 42);
    R3(c, d, e, a, b, 43);
    R3(b, c, d, e, a, 44);
    R3(a, b, c, d, e, 45);
    R3(e, a, b, c, d, 46);
    R3(d, e, a, b, c, 47);
    R3(c, d, e, a, b, 48);
    R3(b, c, d, e, a, 49);
    R3(a, b, c, d, e, 50);
    R3(e, a, b, c, d, 51);
    R3(d, e, a, b, c, 52);
    R3(c, d, e, a, b, 53);
    R3(b, c, d, e, a, 54);
    R3(a, b, c, d, e, 55);
    R3(e, a, b, c, d, 56);
    R3(d, e, a, b, c, 57);
    R3(c, d, e, a, b, 58);
    R3(b, c, d, e, a, 59);
    R4(a, b, c, d, e, 60);
    R4(e, a, b, c, d, 61);
    R4(d, e, a, b, c, 62);
    R4(c, d, e, a, b, 63);
    R4(b, c, d, e, a, 64);
    R4(a, b, c, d, e, 65);
    R4(e, a, b, c, d, 66);
    R4(d, e, a, b, c, 67);
    R4(c, d, e, a, b, 68);
    R4(b, c, d, e, a, 69);
    R4(a, b, c, d, e, 70);
    R4(e, a, b, c, d, 71);
    R4(d, e, a, b, c, 72);
    R4(c, d, e, a, b, 73);
    R4(b, c, d, e, a, 74);
    R4(a, b, c, d, e, 75);
    R4(e, a, b, c, d, 76);
    R4(d, e, a, b, c, 77);
    R4(c, d, e, a, b, 78);
    R4(b, c, d, e, a, 79);
    /* Add the working vars back into context.state[] */
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    state[4] += e;
    /* Wipe variables */
    a = b = c = d = e = 0;
#ifdef SHA1HANDSOFF
    memset(block, '\0', sizeof(block));
#endif
}
```

```
180 /* SHA1Init - Initialize new context */
181
182 void SHA1Init(
183     SHA1_CTX * context
184 )
185 {
186     /* SHA1 initialization constants */
187     context->state[0] = 0x67452301;
188     context->state[1] = 0xEFCDAB89;
189     context->state[2] = 0x98BADCFE;
190     context->state[3] = 0x10325476;
191     context->state[4] = 0xC3D2E1F0;
192     context->count[0] = context->count[1] = 0;
193 }
194
195
196 /* Run your data through this. */
197
198 void SHA1Update(
199     SHA1_CTX * context,
200     const unsigned char *data,
201     uint32_t len
202 )
203 {
204     uint32_t i;
205
206     uint32_t j;
207
208     j = context->count[0];
209     if ((context->count[0] += len << 3) < j)
210         context->count[1]++;
211     context->count[1] += (len >> 29);
212     j = (j >> 3) & 63;
213     if ((j + len) > 63)
214     {
215         memcpy(&context->buffer[j], data, (i = 64 - j));
216         SHA1Transform(context->state, context->buffer);
217         for (; i + 63 < len; i += 64)
218         {
219             SHA1Transform(context->state, &data[i]);
220         }
221         j = 0;
222     }
223     else
224         i = 0;
225     memcpy(&context->buffer[j], &data[i], len - i);
226 }
227
228
229 /* Add padding and return the message digest. */
230
231 void SHA1Final(
232     unsigned char digest[20],
233     SHA1_CTX * context
234 )
235 {
236     unsigned i;
237
238     unsigned char finalcount[8];
239
240     unsigned char c;
241
242 #if 0    /* untested "improvement" by DHR */
243     /* Convert context->count to a sequence of bytes
244      * in finalcount.  Second element first, but
245      * big-endian order within element.
246      * But we do it all backwards.
247      */
```

```
248     unsigned char *fcp = &finalcount[8];
249
250     for (i = 0; i < 2; i++)
251     {
252         uint32_t t = context->count[i];
253
254         int j;
255
256         for (j = 0; j < 4; t >>= 8, j++)
257             *--fcp = (unsigned char) t}
258 #else
259     for (i = 0; i < 8; i++)
260     {
261         finalcount[i] = (unsigned char) ((context->count[(i >= 4 ? 0 : 1)] >> ((3 - (i & 3)) * 8)
                ) & 255);       /* Endian independent */
262     }
263 #endif
264     c = 0200;
265     SHA1Update(context, &c, 1);
266     while ((context->count[0] & 504) != 448)
267     {
268         c = 0000;
269         SHA1Update(context, &c, 1);
270     }
271     SHA1Update(context, finalcount, 8); /* Should cause a SHA1Transform() */
272     for (i = 0; i < 20; i++)
273     {
274         digest[i] = (unsigned char)
275             ((context->state[i >> 2] >> ((3 - (i & 3)) * 8)) & 255);
276     }
277     /* Wipe variables */
278     memset(context, '\0', sizeof(*context));
279     memset(&finalcount, '\0', sizeof(finalcount));
280 }
281
282 void SHA1(
283     char *hash_out,
284     const char *str,
285     int len)
286 {
287     SHA1_CTX ctx;
288     unsigned int ii;
289
290     SHA1Init(&ctx);
291     for (ii=0; ii<len; ii+=1)
292         SHA1Update(&ctx, (const unsigned char*)str + ii, 1);
293     SHA1Final((unsigned char *)hash_out, &ctx);
294     hash_out[20] = '\0';
295 }
```

B.2.2 AES

Listing 19: aes.h

```
1  /* This code is unmodified from the AES implementation provided
2   * at https://github.com/kokke/tiny-AES-c . The original code is
3   * released with the unlicense, included below.
4  /*
5
6  /*
7    The Unlicense:
8    This is free and unencumbered software released into the public domain.
9
10   Anyone is free to copy, modify, publish, use, compile, sell, or
11   distribute this software, either in source code form or as a compiled
12   binary, for any purpose, commercial or non-commercial, and by any
13   means.
```

```
14
15    In jurisdictions that recognize copyright laws, the author or authors
16    of this software dedicate any and all copyright interest in the
17    software to the public domain. We make this dedication for the benefit
18    of the public at large and to the detriment of our heirs and
19    successors. We intend this dedication to be an overt act of
20    relinquishment in perpetuity of all present and future rights to this
21    software under copyright law.
22
23    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
24    EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
25    MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
26    IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
27    OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
28    ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
29    OTHER DEALINGS IN THE SOFTWARE.
30
31    For more information, please refer to <http://unlicense.org/>
32  */
33
34  #ifndef _AES_H_
35  #define _AES_H_
36
37  #include <stdint.h>
38
39  // #define the macros below to 1/0 to enable/disable the mode of operation.
40  //
41  // CBC enables AES encryption in CBC-mode of operation.
42  // CTR enables encryption in counter-mode.
43  // ECB enables the basic ECB 16-byte block algorithm. All can be enabled simultaneously.
44
45  // The #ifndef-guard allows it to be configured before #include'ing or at compile time.
46  #ifndef CBC
47    #define CBC 1
48  #endif
49
50  #ifndef ECB
51    #define ECB 1
52  #endif
53
54  #ifndef CTR
55    #define CTR 1
56  #endif
57
58
59  #define AES128 1
60  //#define AES192 1
61  //#define AES256 1
62
63  #define AES_BLOCKLEN 16 // Block length in bytes - AES is 128b block only
64
65  #if defined(AES256) && (AES256 == 1)
66      #define AES_KEYLEN 32
67      #define AES_keyExpSize 240
68  #elif defined(AES192) && (AES192 == 1)
69      #define AES_KEYLEN 24
70      #define AES_keyExpSize 208
71  #else
72      #define AES_KEYLEN 16   // Key length in bytes
73      #define AES_keyExpSize 176
74  #endif
75
76  struct AES_ctx
77  {
78    uint8_t RoundKey[AES_keyExpSize];
79  #if (defined(CBC) && (CBC == 1)) || (defined(CTR) && (CTR == 1))
80    uint8_t Iv[AES_BLOCKLEN];
81  #endif
```

```
};

void AES_init_ctx(struct AES_ctx* ctx, const uint8_t* key);
#if (defined(CBC) && (CBC == 1)) || (defined(CTR) && (CTR == 1))
void AES_init_ctx_iv(struct AES_ctx* ctx, const uint8_t* key, const uint8_t* iv);
void AES_ctx_set_iv(struct AES_ctx* ctx, const uint8_t* iv);
#endif

#if defined(ECB) && (ECB == 1)
// buffer size is exactly AES_BLOCKLEN bytes;
// you need only AES_init_ctx as IV is not used in ECB
// NB: ECB is considered insecure for most uses
void AES_ECB_encrypt(const struct AES_ctx* ctx, uint8_t* buf);
void AES_ECB_decrypt(const struct AES_ctx* ctx, uint8_t* buf);

#endif // #if defined(ECB) && (ECB == !)


#if defined(CBC) && (CBC == 1)
// buffer size MUST be mutile of AES_BLOCKLEN;
// Suggest https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS7 for padding scheme
// NOTES: you need to set IV in ctx via AES_init_ctx_iv() or AES_ctx_set_iv()
//        no IV should ever be reused with the same key
void AES_CBC_encrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);
void AES_CBC_decrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);

#endif // #if defined(CBC) && (CBC == 1)


#if defined(CTR) && (CTR == 1)

// Same function for encrypting as for decrypting.
// IV is incremented for every block, and used after encryption as XOR-compliment for output
// Suggesting https://en.wikipedia.org/wiki/Padding_(cryptography)#PKCS7 for padding scheme
// NOTES: you need to set IV in ctx with AES_init_ctx_iv() or AES_ctx_set_iv()
//        no IV should ever be reused with the same key
void AES_CTR_xcrypt_buffer(struct AES_ctx* ctx, uint8_t* buf, uint32_t length);

#endif // #if defined(CTR) && (CTR == 1)


#endif // _AES_H_
```

Listing 20: aes.c

```
/* This code is unmodified from the AES implementation provided
 * at https://github.com/kokke/tiny-AES-c . The original code is
 * released with the unlicense, included in the .h file.
/*

/*
This is an implementation of the AES algorithm, specifically ECB, CTR and CBC mode.
Block size can be chosen in aes.h - available choices are AES128, AES192, AES256.

The implementation is verified against the test vectors in:
  National Institute of Standards and Technology Special Publication 800-38A 2001 ED

ECB-AES128
----------

  plain-text:
    6bc1bee22e409f96e93d7e117393172a
    ae2d8a571e03ac9c9eb76fac45af8e51
    30c81c46a35ce411e5fbc1191a0a52ef
    f69f2445df4f9b17ad2b417be66c3710

  key:
    2b7e151628aed2a6abf7158809cf4f3c
```

```
24
25    resulting cipher
26      3ad77bb40d7a3660a89ecaf32466ef97
27      f5d3d58503b9699de785895a96fdbaaf
28      43b1cd7f598ece23881b00e3ed030688
29      7b0c785e27e8ad3f8223207104725dd4
30
31
32  NOTE:   String length must be evenly divisible by 16byte (str_len % 16 == 0)
33          You should pad the end of the string with zeros if this is not the case.
34          For AES192/256 the key size is proportionally larger.
35
36  */
37
38
39  /*****************************************************************************/
40  /* Includes:                                                                 */
41  /*****************************************************************************/
42  #include <string.h> // CBC mode, for memset
43  #include "aes.h"
44
45  /*****************************************************************************/
46  /* Defines:                                                                  */
47  /*****************************************************************************/
48  // The number of columns comprising a state in AES. This is a constant in AES. Value=4
49  #define Nb 4
50
51  #if defined(AES256) && (AES256 == 1)
52      #define Nk 8
53      #define Nr 14
54  #elif defined(AES192) && (AES192 == 1)
55      #define Nk 6
56      #define Nr 12
57  #else
58      #define Nk 4        // The number of 32 bit words in a key.
59      #define Nr 10       // The number of rounds in AES Cipher.
60  #endif
61
62  // jcallan@github points out that declaring Multiply as a function
63  // reduces code size considerably with the Keil ARM compiler.
64  // See this link for more information: https://github.com/kokke/tiny-AES-C/pull/3
65  #ifndef MULTIPLY_AS_A_FUNCTION
66    #define MULTIPLY_AS_A_FUNCTION 0
67  #endif
68
69
70
71
72  /*****************************************************************************/
73  /* Private variables:                                                        */
74  /*****************************************************************************/
75  // state - array holding the intermediate results during decryption.
76  typedef uint8_t state_t[4][4];
77
78
79
80  // The lookup-tables are marked const so they can be placed in read-only storage instead of RAM
81  // The numbers below can be computed dynamically trading ROM for RAM -
82  // This can be useful in (embedded) bootloader applications, where ROM is often limited.
83  static const uint8_t sbox[256] = {
84    //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
85    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
86    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
87    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
88    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
89    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
90    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
91    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
```

```
92    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
93    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
94    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
95    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
96    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
97    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
98    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
99    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
100   0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
          };
```

the remainder of this file caused the report to be too large, but can be accessed at https://github.com/kokke/tiny-AES-c.

B.2.3 HMAC-SHA1

Listing 21: hmac.h

```
1  /**
2  * @file re_hmac.h  Interface to HMAC functions
3  *
4  * Copyright (C) 2010 Creytiv.com
5  */
6
7  #ifndef HMAC_H_
8  #define HMAC_H_ (1)
9
10 #include <stdint.h>
11
12 void hmac_sha1(const uint8_t *k,   /* secret key */
13         size_t lk,  /* length of the key in bytes */
14         const uint8_t *d,   /* data */
15         size_t ld,  /* length of data in bytes */
16         uint8_t *out, /* output buffer, at least "t" bytes */
17         size_t *t);
18
19 #endif // HMAC_H_
```

Listing 22: hmac_sha1.c

```
1  /**
2  * @file hmac_sha1.c  Implements HMAC-SHA1 as of RFC 2202
3  *
4  * Copyright (C) 2010 Creytiv.com
5  https://github.com/Akagi201/hmac-sha1
6
7  modified for use with another sha implementation
8
9  */
10 #include <string.h>
11 #include <stdint.h>
12
13 #include <sha1.h>
14 #include <hmac.h>
15
16
17 /** SHA-1 Block size */
18 #ifndef SHA_BLOCKSIZE
19 #define SHA_BLOCKSIZE   (64)
20 #endif
21
22
23 /**
24 * Function to compute the digest
25 *
26 * @param k   Secret key
27 * @param lk  Length of the key in bytes
```

```
28 * @param d   Data
29 * @param ld  Length of data in bytes
30 * @param out Digest output
31 * @param t   Size of digest output
32 */
33 void hmac_sha1(const uint8_t *k,  /* secret key */
34         size_t lk,       /* length of the key in bytes */
35         const uint8_t *d,  /* data */
36         size_t ld,       /* length of data in bytes */
37         uint8_t *out,      /* output buffer, at least "t" bytes */
38         size_t *t) {
39
40     SHA1_CTX ictx, octx;
41     uint8_t isha[SHA_DIGEST_LENGTH], osha[SHA_DIGEST_LENGTH];
42     uint8_t key[SHA_DIGEST_LENGTH];
43     uint8_t buf[SHA_BLOCKSIZE];
44     size_t i;
45
46     if (lk > SHA_BLOCKSIZE) {
47         SHA1_CTX tctx;
48
49         SHA1Init(&tctx);
50         SHA1Update(&tctx, k, lk);
51         SHA1Final(key, &tctx);
52
53         k = key;
54         lk = SHA_DIGEST_LENGTH;
55     }
56
57     /**** Inner Digest ****/
58
59     SHA1Init(&ictx);
60
61     /* Pad the key for inner digest */
62     for (i = 0; i < lk; ++i) {
63         buf[i] = k[i] ^ 0x36;
64     }
65     for (i = lk; i < SHA_BLOCKSIZE; ++i) {
66         buf[i] = 0x36;
67     }
68
69     SHA1Update(&ictx, buf, SHA_BLOCKSIZE);
70     SHA1Update(&ictx, d, ld);
71
72     SHA1Final(isha, &ictx);
73
74     /**** Outer Digest ****/
75
76     SHA1Init(&octx);
77
78     /* Pad the key for outter digest */
79
80     for (i = 0; i < lk; ++i) {
81         buf[i] = k[i] ^ 0x5c;
82     }
83     for (i = lk; i < SHA_BLOCKSIZE; ++i) {
84         buf[i] = 0x5c;
85     }
86
87     SHA1Update(&octx, buf, SHA_BLOCKSIZE);
88     SHA1Update(&octx, isha, SHA_DIGEST_LENGTH);
89
90     SHA1Final(osha, &octx);
91
92     /* truncate and print the results */
93     *t = *t > SHA_DIGEST_LENGTH ? SHA_DIGEST_LENGTH : *t;
94     memcpy(out, osha, *t);
95 }
```

B.3 Base32 Encoding and Decoding

Listing 23: base32.h

```
// Base32 implementation
//
// Copyright 2010 Google Inc.
// Author: Markus Gutschke
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//
// Encode and decode from base32 encoding using the following alphabet:
//   ABCDEFGHIJKLMNOPQRSTUVWXYZ234567
// This alphabet is documented in RFC 4648/3548
//
// We allow white-space and hyphens, but all other characters are considered
// invalid.
//
// All functions return the number of output bytes or -1 on error. If the
// output buffer is too small, the result will silently be truncated.

#ifndef _BASE32_H_
#define _BASE32_H_

#include <stdint.h>

int base32_decode(const uint8_t *encoded, uint8_t *result, int bufSize)
    __attribute__((visibility("hidden")));
int base32_encode(const uint8_t *data, int length, uint8_t *result,
                  int bufSize)
    __attribute__((visibility("hidden")));

#endif /* _BASE32_H_ */
```

Listing 24: base32.c

```
// Base32 implementation
//
// Copyright 2010 Google Inc.
// Author: Markus Gutschke
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

#include <string.h>

#include "base32.h"

```

```
22 int base32_decode(const uint8_t *encoded, uint8_t *result, int bufSize) {
23   int buffer = 0;
24   int bitsLeft = 0;
25   int count = 0;
26   for (const uint8_t *ptr = encoded; count < bufSize && *ptr; ++ptr) {
27     uint8_t ch = *ptr;
28     if (ch == ' ' || ch == '\t' || ch == '\r' || ch == '\n' || ch == '-') {
29       continue;
30     }
31     buffer <<= 5;
32
33     // Deal with commonly mistyped characters
34     if (ch == '0') {
35       ch = 'O';
36     } else if (ch == '1') {
37       ch = 'L';
38     } else if (ch == '8') {
39       ch = 'B';
40     }
41
42     // Look up one base32 digit
43     if ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z')) {
44       ch = (ch & 0x1F) - 1;
45     } else if (ch >= '2' && ch <= '7') {
46       ch -= '2' - 26;
47     } else {
48       return -1;
49     }
50
51     buffer |= ch;
52     bitsLeft += 5;
53     if (bitsLeft >= 8) {
54       result[count++] = buffer >> (bitsLeft - 8);
55       bitsLeft -= 8;
56     }
57   }
58   if (count < bufSize) {
59     result[count] = '\000';
60   }
61   return count;
62 }
63
64 int base32_encode(const uint8_t *data, int length, uint8_t *result,
65                   int bufSize) {
66   if (length < 0 || length > (1 << 28)) {
67     return -1;
68   }
69   int count = 0;
70   if (length > 0) {
71     int buffer = data[0];
72     int next = 1;
73     int bitsLeft = 8;
74     while (count < bufSize && (bitsLeft > 0 || next < length)) {
75       if (bitsLeft < 5) {
76         if (next < length) {
77           buffer <<= 8;
78           buffer |= data[next++] & 0xFF;
79           bitsLeft += 8;
80         } else {
81           int pad = 5 - bitsLeft;
82           buffer <<= pad;
83           bitsLeft += pad;
84         }
85       }
86       int index = 0x1F & (buffer >> (bitsLeft - 5));
87       bitsLeft -= 5;
88       result[count++] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ234567"[index];
89     }
```

```
90    }
91    if (count < bufSize) {
92      result[count] = '\000';
93    }
94    return count;
95  }
```