Bitcoin Hardware Wallet

EI1062 – Diseño de Sistemas Empotrados y de Tiempo Real

Diego Lacomba Fañanás

2020/2021

Índice

1 Introducción

El proyecto propuesto a realizar consiste en el diseño y desarrollo de una billetera fría de Bitcoin, es decir, un dispositivo hardware que gestiona de forma segura las claves privadas de tus Bitcoin.

Para ello debe ser capaz de obtener una entropía suficientemente aleatoria para generar y derivar las claves del usuario de forma segura y eficaz. Estas claves se emplearán para obtener las direcciones asociadas y mostrarlas al usuario.

Para el uso del dispositivo es necesario un software externo como puede ser Bitcoin-Core, al que pasaremos la master public key derivada. Mediante el software externo recibiremos y enviaremos Bitcoin, al tener únicamente la master public key, por sus propios medios únicamente podrá gestionar los recibos que hagan al usuario, mientras que para hacer envíos, como es necesario firmar las transacciones y para esto se emplean las claves privadas, nuestro dispositivo es esencial.

De esta forma tenemos un control completo de nuestros Bitcoin sin tener que compartir las claves privadas con nadie.

2 Requisitos

En este apartado se comentará por puntos y de una forma genérica los requisitos y aspectos básicos del dispositivo a desarrollar.

Nombre: Bitcoin Hardware Wallet

Propósito: Dispositivo hardware para la gestión de las claves privadas asociadas a los Bitcoin del usuario.

Entradas: Entrada USB y tres botones.

Salidas: Pantalla 23 mm * 40 mm y tarjeta SD.

Funciones:

- Capaz de generar la frase de recuperación y derivar a partir de esta las claves privadas y públicas asociadas correctamente.
- Proporcionar al usuario las direcciones para que le hagan envíos.
- Firmar las transacciones Bitcoin proporcionadas por una plataforma online (Bitcoin-Core).
- Almacenar la frase de recuperación en tarjeta SD y restaurar el sistema a partir de una frase de recuperación.

Prestaciones:

- Posibilidad de generar aleatoriedad (entropía) de calidad, para así generar una frase mnemónica de recuperación y su master private key de forma segura.
- Derivación eficiente y eficaz de las claves.
- Garantizar la confidencialidad de la master private key, aunque tengan ataques físicos o remotos.

Coste de fabricación: < 40€

Consumo energético y alimentación:

- El consumo del dispositivo debe ser suficiente para garantizar un uso excelente siendo alimentado vía USB por un PC o un dispositivo móvil.
- Alimentación de 5V vía USB.

Dimensiones físicas y peso:

- Tamaño máximo de 100 mm x 60 mm x 20 mm.
- Peso máximo de 150 g.

3 Especificaciones

En este apartado se explicará de una manera más extensa y precisa los requisitos del dispositivo así como el proceso de uso. (Si la explicación no varía se omite el punto)

Entradas

- Entrada USB: Su función es alimentar al dispositivo.

- Tres botones: Habilitan la navegación por el menú.

Salidas

- Pantalla 23 mm * 40 mm aprox. : Display del menú y todas las acciones posibles a realizar con el dispositivo. Debe ser un menú sencillo e intuitivo para el usuario.

- Tarjeta SD: La función de la tarjeta SD es almacenar la frase de recuperación a partir de la cual se obtiene la semilla o master private key mediante un algortimo de derivación de clave (PBKDF2) el cual emplea un salteado solicitado al usuario como una “password” el cual no se almacena en ningún lugar (excepto en la memoria del usuario). Por lo tanto, esta frase se puede almacenar en texto plano tanto en el dispositivo como en la tarjeta SD ya que sin el salteado la derivación proporcionará otras claves y además no es necesario el uso de un PIN de desbloqueo ya que el

dispositivo no almacena información peligrosa. El usuario podrá almacenar la frase en la tarjeta SD o recuperar el dispositivo a partir de una.

Funciones:

- Capaz de generar la frase de recuperación de 12 palabras y derivar a partir de esta las claves privadas y públicas asociadas correctamente. Para ello el dispositivo obtendrá entropía de diversas fuentes como puede ser el TRNG del propio dispositivo o el ruido de los pines ADC o de un micrófono por ejemplo. Esta entropía se empleará para obtener la frase mnemonica y esta se derivará con el proceso explicado anteriormente para obtener la semilla o master public key. El proceso de derivación posterior se explicará más detalladamente en el apartado de implementación.
- Proporcionar al usuario las direcciones para que le hagan envíos. Este proceso también será explicado de forma precisa en el apartado de implementación.
- Firmar las transacciones Bitcoin proporcionadas por una plataforma online (Bitcoin-Core). Para ello el dispositivo obtendrá de el software externo la transacción sin firmar y hará las modificaciones necesarias para poder pasar al software la transacción firmada y este se encargará de finalizar el proceso. Además el dispositivo mostrará al usuario los datos pertinentes de la transacción.
- Almacenar la frase de recuperación en tarjeta SD y restaurar el sistema a partir de una frase de recuperación.

Proceso de uso

El primer paso para empezar a emplear el dispositivo es generar una frase mnemónica o introducir una existente. Una vez almacenada la frase en el dispositivo podremos comenzar a realizar acciones.

Cada vez que el usuario quiera realizar una acción que requiera la derivación de la mnemónica, el dispositivo solicitará al usuario la password o salteado y la red (mainet/testnet) para derivar las claves y direcciones correspondientes. Entre las acciones que requieren este proceso encontramos: mostrar la xpub o master public key, mostrar direcciones y firmar transacciones.

Otras opciones que encontramos y no requieren de el proceso anterior son: generar la frase mnemónica, mostrar la frase mnemónica, introducir otra frase mnemónica, almacenar la frase en la SD, obtener la frase de la SD y apagar el dispositivo.

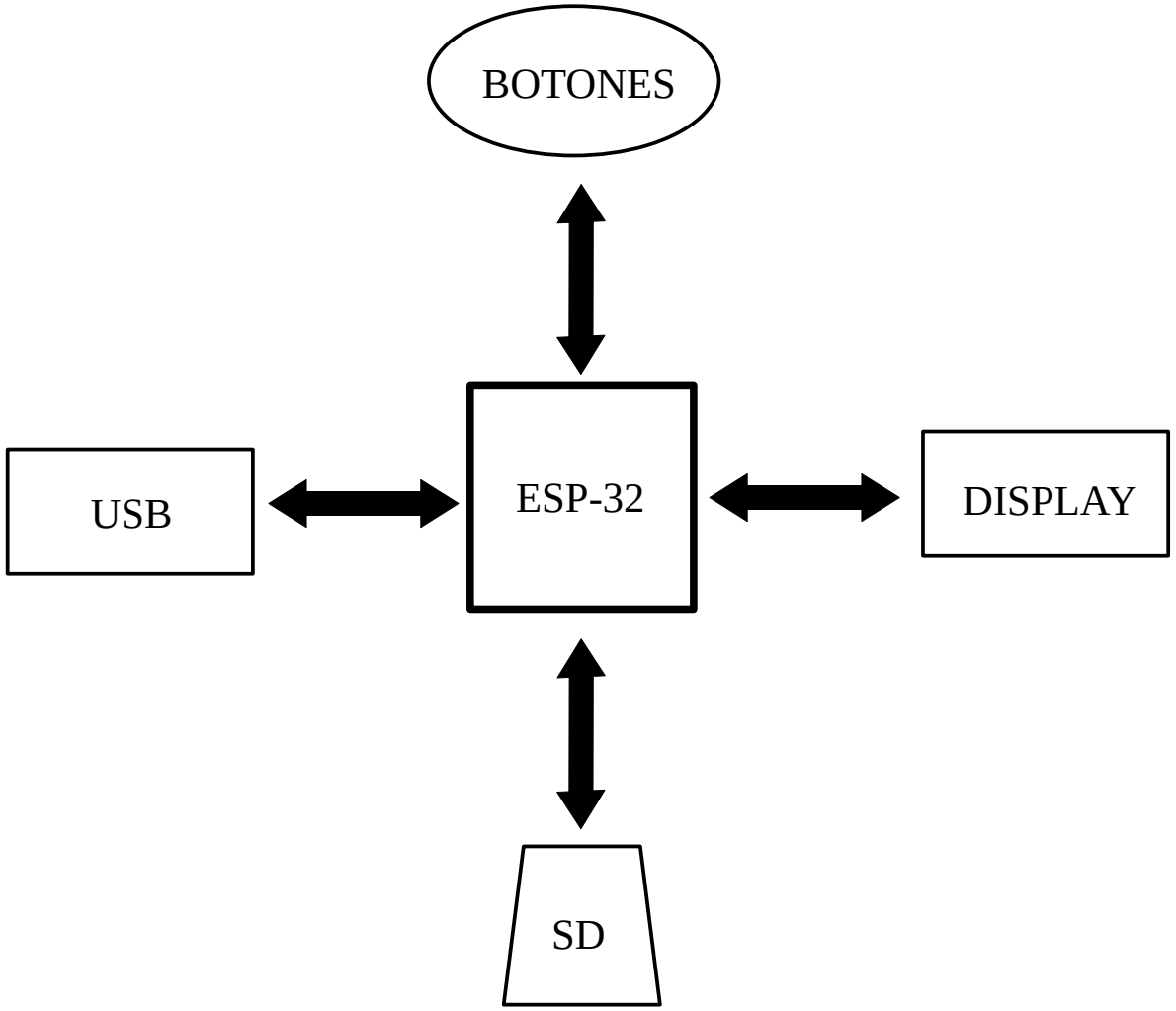
Mediante dos botones navegaremos por un sencillo menú que se desplaza de forma vertical y mediante un tercero seleccionaremos la acción deseada.

4 Arquitectura

En este apartado representaremos mediante dos esquemas la estructura global del sistema. EL primer esquema representa la estructura del hardware mientras que el segundo la del software.

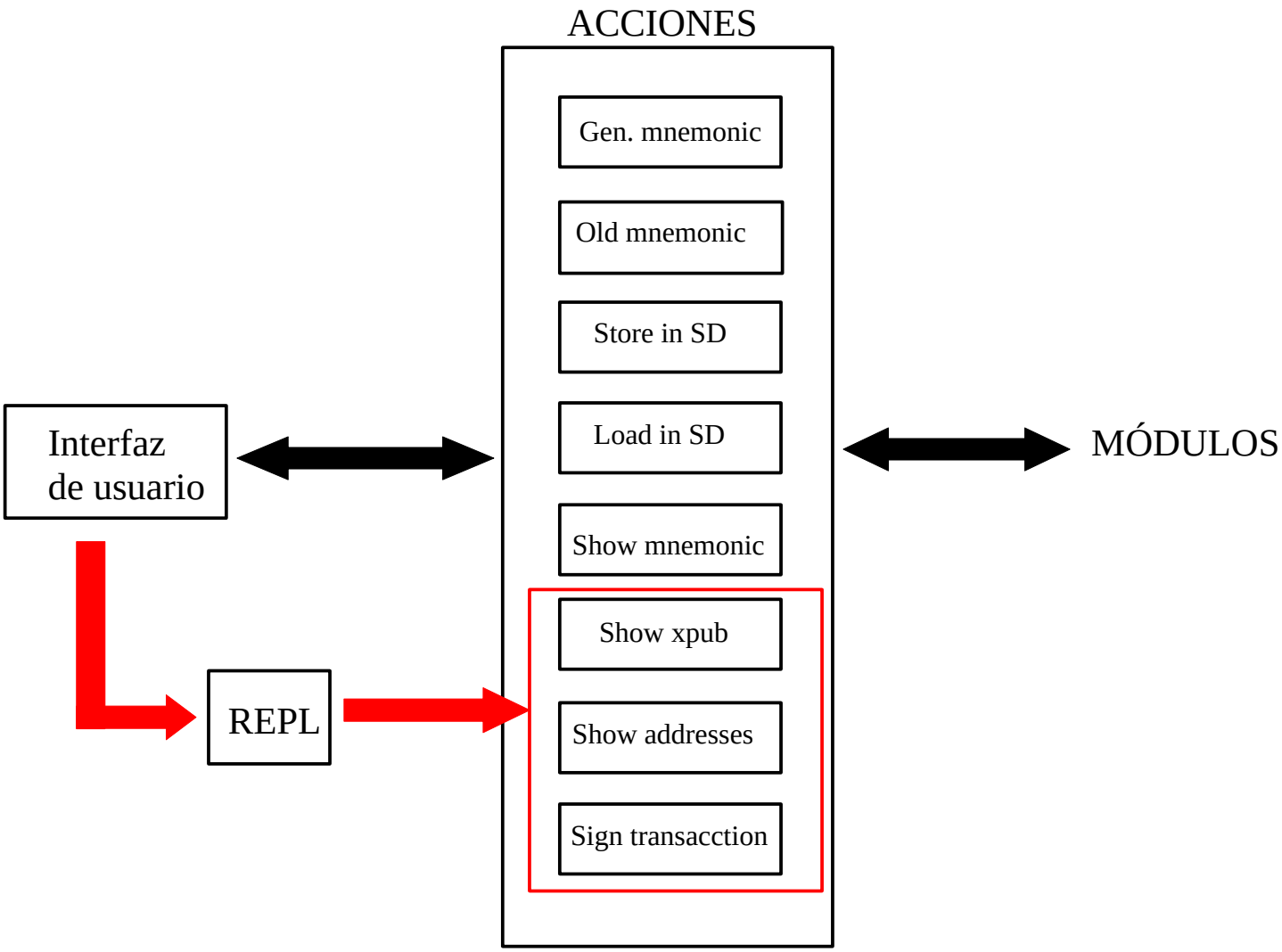
4.1 Diseño hardware

Como se puede apreciar en el siguiente esquema, el microcontrolador interacciona con sus componentes de forma directa.. El USB y la pantalla están incorporados al microcontrolador de fábrica.



4.2 Diseño software

Como se mencionó en apartados anteriores, cuando se requiere derivar la frase mnemónica para obtener claves o direcciones, es necesario solicitar al usuario el salteado o password de derivación. Esto se puede ver reflejado en el esquema inferior, se puede apreciar que para las acciones: “mostrar xpub” (master public key), “mostrar direcciones” y “firmar transacción” se requiere pasar por el REPL del microcontrolador e introducir manualmente la password. Los módulos son las librerías introducidas en el firmware del microcontrolador, contienen las clases y métodos que se encargan de derivar las claves y todo lo relacionado.

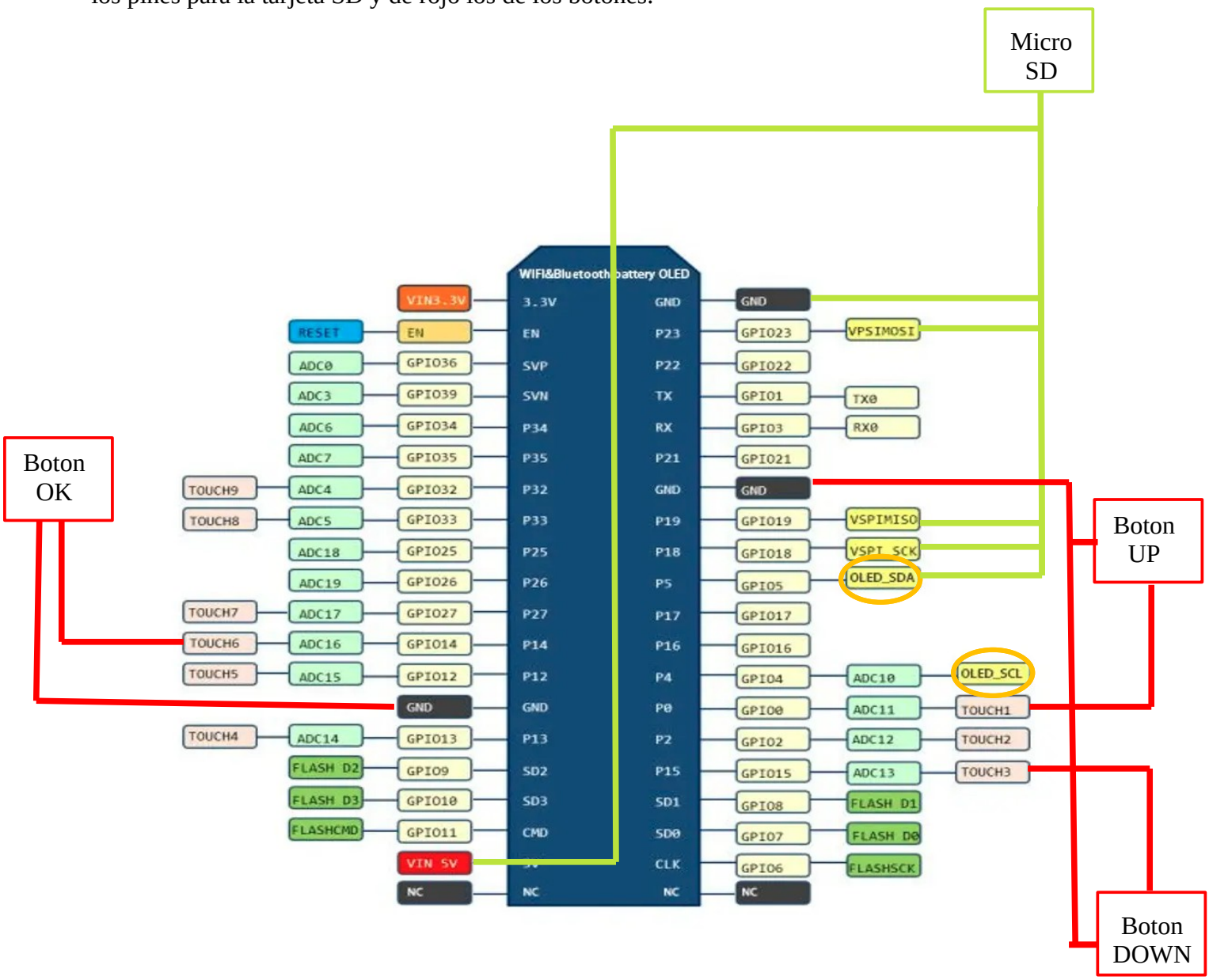


5 Implementación

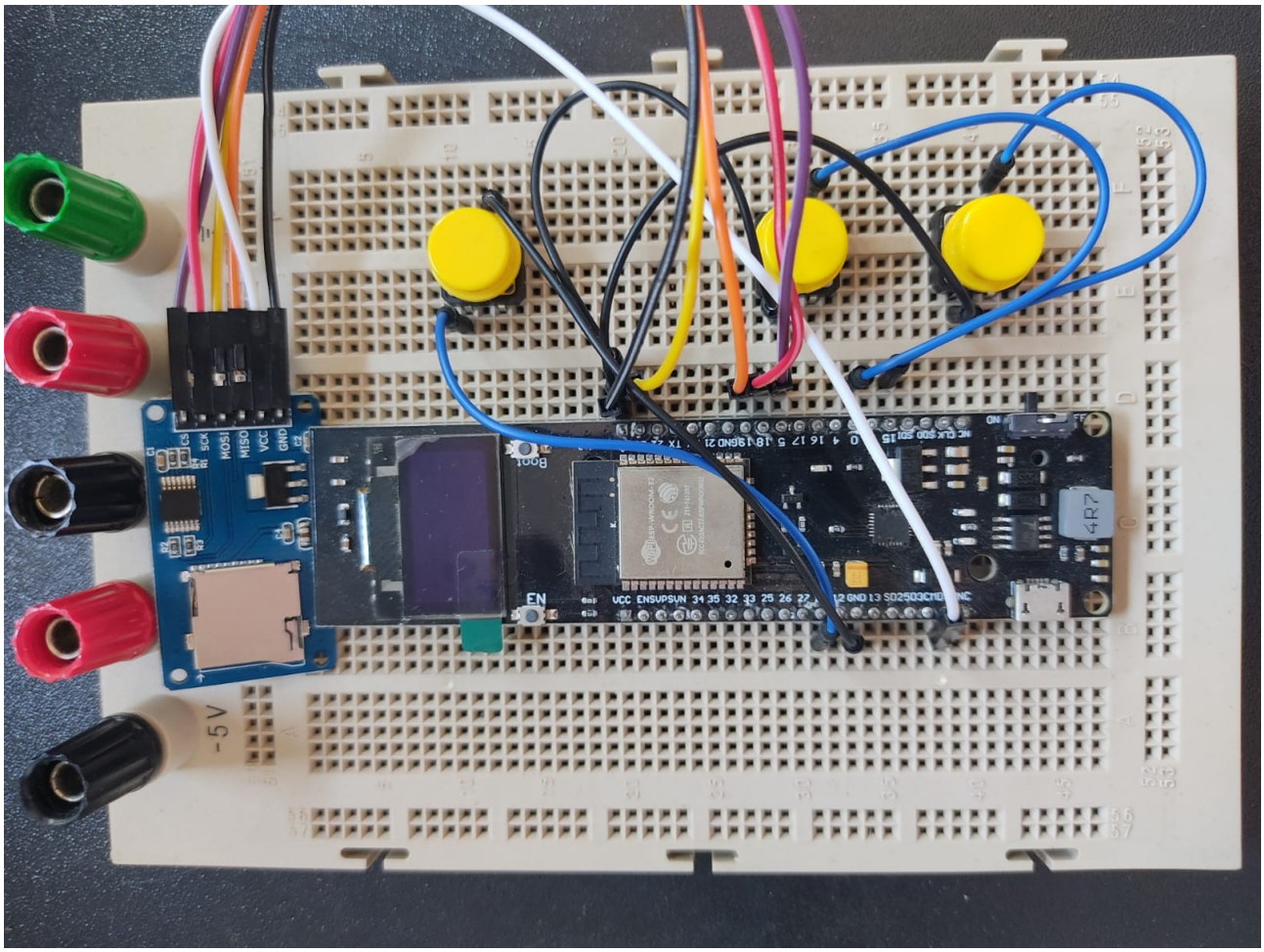
En este apartado explicaremos el esquema del circuito con sus componentes y el código desarrollado.

5.1 Esquema del circuito

En primer lugar, tenemos que destacar que la pantalla OLED viene incorporada, en el esquema podemos ver rodeados de color naranja los pines empleados para esta. De color lima podemos ver los pines para la tarjeta SD y de rojo los de los botones.



5.2 Circuito implementado



5.3 Código desarrollado

Para las explicaciones, he omitido bastantes detalles respecto a la derivación y otros aspectos como la firma. Esto es debido a que quedaría una explicación demasiado extensa y fuera de las competencias de esta asignatura.

5.3.1 main.py

5.3.1.1 Código

```
from machine import Pin, I2C
from time import sleep
import ssd1306, recovery_phrase, show_xpub, psbt, show_addresses


pinesI2C = I2C(1, scl=Pin(4), sda=Pin(5))   # SCL = GPIO4, SDA = GPIO5
oled = ssd1306.SSD1306_I2C(128, 64, pinesI2C)

botonUP = Pin(15, Pin.IN)
botonDOWN = Pin(0, Pin.IN)
botonOK = Pin(14, Pin.IN)

next = 1
shutdown = True

# M E N U
def menu(option1, option2, option3, prev_value, next_value, up_ok, down_ok):
    global next
    next = next_value

    mostrar_menu(option1, option2, option3)

    while next == next_value:
        sleep(0.1)
        if botonUP.value() == 0 and up_ok:
            sleep(0.35)
            next = next_value-1

        if botonDOWN.value() == 0 and down_ok:
            sleep(0.35)
            next = next_value+1
```

```
        if botonOK.value() == 0:
            if prev_value != 0:
                sleep(0.35)
                next = prev_value
            else:
                sleep(0.35)
                next = 0




def mostrar_menu(option1, option2, option3):
    oled.fill(0)
    oled.text(option1, 22, 0)
    oled.text('/', 42, 10)
    oled.text('\\', 47, 10)
    oled.text('|', 45, 13)
    oled.text('|', 45, 16)
    oled.text(option2, 22, 28)
    oled.text('|', 45, 39)
    oled.text('|', 45, 42)
    oled.text('\\', 42, 45)
    oled.text('/', 47, 45)
    oled.text(option3, 22, 55)
    oled.show()

def disconnect():
    global next
    next == 71
    oled.fill(0)
    oled.text('Press OK to', 0, 0)
    oled.text('DISCONNECT', 0, 10)
    oled.text('Press UP/DOWN to', 0, 30)
    oled.text('CANCEL', 0, 40)
    oled.show()

    while next == 71:
        sleep(0.1)

        if botonUP.value() == 0:
            sleep(0.35)
            next = 7

        if botonDOWN.value() == 0:
            sleep(0.35)
```

```
            next = 7

        if botonOK.value() == 0:
            sleep(0.35)
            next = 0

def main():
    print('empiezo')
    global shutdown
    global next
    oled.fill(0)
    oled.text('WELCOME', 28, 28)
    oled.show()
    sleep(3)

    while shutdown:
        while next == 1:
            menu('-------', 'New Recovery', 'Show xpub', 0, next, False, True)

            if next == 0:
                next = 11
                #Recovery menu
                while next > 1:
                    while next == 11:
                        menu('-------', 'Generate', '  Back', 0, next, False, True)

                        if next == 0:
                            next = 111

                            f = open("mnemonic.txt", "r")
                            mnemonic = f.read()
                            f.close()
                            if len(mnemonic) > 0:
                                print("WARNING: this will delete the previous mnemonic")
                                des = input("Are you sure do you want to continue? [y/n]")

                                while des != "y" and des != "n":
                                    network = input("Invalid input, [continue->\"y\"] or [cancel->\"n\"]: ")

                                if des == "y":
                                    recovery_phrase.new_phrase()
                                else:
                                    next = 1
                            else:
                                recovery_phrase.new_phrase()
```

```
                while next == 111:
                    menu('-------', 'Succesfull', '-OK to exit-', 1, next, False, False)

            while next == 12:
                menu('Generate', '  Back', '-------', 1, next, True, False)

    while next == 2:
        menu('New Recovery', 'Show xpub', 'Sign transaction', 0, next, True, True)

        if next == 0:
            next = 21
            #Show xpub menu
            while next > 2:
                while next == 21:
                    menu('-------', 'OK to show', '  Back', 0, next, False, True)

                    if next == 0:
                        next = 211

                        f = open("mnemonic.txt", "r")
                        mnemonic = f.read()
                        f.close()
                        if len(mnemonic) > 0:
                            show_xpub.show_xpub()
                        else:
                            print("There is not a mnemonic in the device")
                            next = 2

                        while next == 211:
                            menu('-------', 'Succesfull', '-OK to exit-', 2, next, False, False)

                while next == 22:
                    menu('OK to show', '  Back', '-------', 2, next, True, False)

    while next == 3:
        menu('Show xpub', 'Sign transaction', 'Old recovery', 0, next, True, True)

        if next == 0:
            next = 31
            #Sign transaction menu
            while next > 3:
                while next == 31:
                    menu('-------', 'OK to sign', '  Back', 0, next, False, True)

                    if next == 0:
                        next = 311
```

```
                f = open("mnemonic.txt", "r")
                mnemonic = f.read()
                f.close()
                if len(mnemonic) > 0:
                    psbt.sign_tx()
                else:
                    print("There is not a mnemonic in the device")
                    next = 3

                while next == 311:
                    menu('-------', 'Succesfull', '-OK to exit-', 3, next, False, False)

            while next == 32:
                menu('OK to sign', '  Back', '-------', 3, next, True, False)

    while next == 4:
        menu('Sign transaction', 'Old recovery', 'Show addresses', 0, next, True, True)

        if next == 0:
            next = 41
            #Old recovery menu
            while next > 4:
                while next == 41:
                    menu('-------', 'Ok to restore', '  Back', 0, next, False, True)

                    if next == 0:
                        next = 411

                        f = open("mnemonic.txt", "r")
                        mnemonic = f.read()
                        f.close()
                        if len(mnemonic) > 0:
                            print("WARNING: this will delete the previous mnemonic")
                            des = input("Are you sure do you want to continue? [y/n]")

                            while des != "y" and des != "n":
                                network = input("Invalid input, [continue->\"y\"] or [cancel->\"n\"]: ")

                            if des == "y":
                                recovery_phrase.recover()
                            else:
                                next = 4
                        else:
                            recovery_phrase.recover()
```

```
                while next == 411:
                    menu('-------', 'Succesfull', '-OK to exit-', 4, next, False, False)

            while next == 42:
                menu('Ok to restore', '  Back', '-------', 4, next, True, False)

    while next == 5:
        menu('Old recovery', 'Show addresses', 'Show mnemonic', 0, next, True, True)

        if next == 0:
            next = 51
            #Show address menu
            while next > 5:
                while next == 51:
                    menu('-------', 'OK to show', '  Back', 0, next, False, True)

                    if next == 0:
                        next = 511

                        f = open("mnemonic.txt", "r")
                        mnemonic = f.read()
                        f.close()
                        if len(mnemonic) > 0:
                            show_addresses.show_addresses()
                        else:
                            print("There is not a mnemonic in the device")
                            next = 5

                        while next == 511:
                            menu('-------', 'Succesfull', '-OK to exit-', 5, next, False, False)

                while next == 52:
                    menu('OK to show', '  Back', '-------', 5, next, True, False)

    while next == 6:
        menu('Show addresses', 'Show mnemonic', 'Disconnect', 0, next, True, True)

        if next == 0:
            next = 61
            #Show mnemonic menu
            while next > 6:
                while next == 61:
                    menu('-------', 'OK to show', '  Back', 0, next, False, True)

                    if next == 0:
                        next = 611
```

```
                    f = open("mnemonic.txt", "r")
                    mnemonic = f.read()
                    f.close()
                    if len(mnemonic) > 0:
                        print(mnemonic)
                    else:
                        print("There is not a mnemonic in the device")
                        next = 6

                    while next == 611:
                        menu('-------', 'Succesfull', '-OK to exit-', 6, next, False, False)

                while next == 62:
                    menu('OK to show', '  Back', '-------', 6, next, True, False)


            while next == 7:
                menu('Show mnemonic', 'Disconnect', '-------', 0, next, True, False)

                if next == 0:
                    next = 71
                    disconnect()
                    if next == 0:
                        oled.fill(0)
                        oled.text('Disconnecting...', 0, 28)
                        oled.show()
                        sleep(2)
                        oled.fill(0)
                        oled.text('BYE BYE', 20, 28)
                        oled.show()
                        sleep(1)
                        oled.fill(0)
                        oled.show()
                        shutdown = False

        print('Fin programa')

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

5.3.1.2 Comentarios main.py

El fichero main.py contiene lo relativo al menú. El menú se desplaza en función de una variable global llamada **next**. La función real de los botones es incrementar, decrementar o igualar a 0 la variable **next**, como se puede apreciar en el método menu(). En caso de igualar a 0 se interpreta como OK y se selecciona la opción en la que nos encontramos, en caso de incrementar o decrementar, el menú sube o baja a la opción siguiente o anterior (entrar en el siguiente while o en el anteior).

Como hemos comentado, cada opción del menú corresponde a un while al que se entra cuando **next** corresponde a su dígito de condición, cada opción contiene un submenú que funciona de la misma manera pero con dos digitos, y así sucesivamente. Dentro de cada opción se hacen las comprobaciones pertinentes, como por ejemplo, comprobar que haya una frase mnemónica almacenada ya que en caso de no haber una, por ejemplo no se podría mostrar la master public key.

Al método menu() le pasamos como parámetros las opciones a mostrar (anterior, actual y siguiente, en ese orden), el valor anterior de next para saber si no podemos proseguir o si (prev_value = 0), el actual valor de next para incrementarlo, decrementarlo o igualarlo a 0, y dos variales booleanas que se emplean para saber si estamos al principio del menú y no podemos subir o al fnal y no podemos bajar. El método mostrar_menu() simplemente se encarga de modificar el display y el método disconnect() funciona de manera similar a menu().

5.3.2 recovery_phrase.py

5.3.2.1 Código

```
import machine, os, hashlib
from binascii import hexlify
from bitcoin import bip39, bip32, script
from bitcoin.networks import NETWORKS
from time import sleep

#RECOVERY/MNEMONIC PHRASE
def new_phrase():
    trng_entropy = os.urandom(256)

    #Entropy from ADC pins
    adc1 = machine.ADC(machine.Pin(35))
    adc2 = machine.ADC(machine.Pin(34))
    adc3 = machine.ADC(machine.Pin(32))
    adc4 = machine.ADC(machine.Pin(33))

    #Entropy from ADC pin to bytes
```

```
    adc_entropy = bytes([(adc1.read()+adc2.read()+adc3.read()+adc4.read())%256 for i in
          range(2048)])

    #16 bytes from sha256( total entropy )
    final_entropy = hashlib.sha256(trng_entropy+adc_entropy).digest()[:16]

    #MNEMONIC PHRASE: Convert entropy to binary, add checksum from sha256, split into parts
                  of 11 bits, each part is one word
    phrase = bip39.mnemonic_from_bytes(final_entropy)
    print(phrase)

    f = open('mnemonic.txt', 'w')
    f.write("")
    f.close()
    f = open('mnemonic.txt', 'w')
    f.write(phrase)
    f.close()

def recover():
    print("WARNING: Make sure the mnemonic is correct")
    mnemonic = input("Type the mnemonic: ")

    f = open('mnemonic.txt', 'w')
    f.write("")
    f.close()
    f = open('mnemonic.txt', 'w')
    f.write(mnemonic)
    f.close()
```

5.3.2.2 Comentarios recovery_phrase.py

Este fichero contiene lo relativo a generar y cargar una frase mnemónica en el dispositivo. El método new_phrase() es el que se encarga de generar la nueva frase mnemónica. Para ello, obtiene un valor de 256 bytes del TRNG del dispositivo y lo combina con 4 valores obtenidos de los pines de ADC, los cuales tienen “ruido”. Después se aplica el algoritmo SHA256 que devuelve una cadena única de 256 bytes que representa en su totalidad a la entropía obtenida, finalmente se obtienen los últmos 16 bytes como la entropía final.

Para obtener la frase empleamos el módulo **bitcoin** cargado en el firmware del dispositivo, este módulo contiene la clase **bip39** que es el protocolo de Bitcoin que se encarga de obtener una semilla y su master private key a partir de entropía. Para obtener la frase empleamos el método mnemonic_from_bytes(). Este método convierte a binario la entropía, le añade un checksum al final para que la longitud total sea múltiplo de 11, se divide en trozos de 11, y cada trozo equivale a una palabra de un diccionario que contiene $2^{11} = 2048$ palabras.

El método recover() simplemente solicita al usuario que introduzca la frase de forma manual por el REPL.

5.3.3 show_xpub.py

5.3.3.1 Código

```
from bitcoin import bip32, bip39
from bitcoin.networks import NETWORKS
from ubinascii import hexlify

def show_xpub():
    f = open('mnemonic.txt', 'r')
    mnemonic = f.read()
    f.close()

    #Mnemonic to seed [64 bytes]: PBKDF2 key derivation
    passwd = input("Introduce password for key derivation: ")

    seed = bip39.mnemonic_to_seed(mnemonic, password=passwd)
    network = input("Choose network [mainet->1] or [testnet->2]: ")

    while network != "1" and network != "2":
        network = input("Invalid input, [mainet->1] or [testnet->2]: ")

    if network == "1":
        root = bip32.HDKey.from_seed(seed, version=NETWORKS['main']['xprv'])
        hardened_derivation = "m/84h/0h/0h"
        account = root.derive(hardened_derivation)
        #MAINET - NATIVE SEGWIT
        account.version = NETWORKS['main']['zprv']
    else:
        root = bip32.HDKey.from_seed(seed, version=NETWORKS['test']['xprv'])
        hardened_derivation = "m/84h/1h/0h"
        account = root.derive(hardened_derivation)
        #TESTNET - NATIVE SEGWIT
        account.version = NETWORKS['test']['zprv']

    # [fingerprint/derivation]xpub to import to Bitcoin Core with descriptors
    # root fingerprint from any child of the root key
    fingerprint = root.child(0).fingerprint
    xpub = account.to_public()
    print("XPUB:")
    print("[%s%s]%s" % (
            hexlify(fingerprint).decode('utf-8'),
            hardened_derivation[1:],
            xpub.to_base58())
        )
```

5.3.3.2 Comentarios show_xpub.py

Este fichero tiene una única función, mostrar la master public key. Para ello, se carga la frase mnemónica y se solicita al usuario el password para proceder con la derivación y obtener la semilla como se explicó en apartados anteriores (PBKDF2) empleando la clase **bip-39** y el método mnemonic_to_seed(). A continuación se solicita al usuario que escoja entre mainet o testnet y se deriva la root key o master private key a partir de la cual se derivan todas las claves mediante la clase **bip-32** y la subclase **HDKey** y el método **from_seed()**. La derivación puede variar en función de el tipo de direcciones que vayamos a emplear. En nuestro caso únicamente emplearemos “native segwit” (BIP-84), mainnet (0h) o testnet (1h) y el numero de cuenta (account) que siempre será 0 (es lo más recomendable). Esta derivación se puede ver reflejada en el código en la variable hardened_derivation. Finalmente obtenemos la clave pública asociada mediante account.to_public().

5.3.4 psbt.py

5.3.4.1 Código

```
from bitcoin import script
from bitcoin import bip32
from bitcoin import bip39
from bitcoin.networks import NETWORKS
from bitcoin import psbt
from ubinascii import unhexlify, hexlify
from ubinascii import a2b_base64, b2a_base64

def sign_tx():
    f = open("mnemonic.txt", "r")
    mnemonic = f.read()
    f.close()

    passwd = input("Introduce password for key derivation: ")
    seed = bip39.mnemonic_to_seed(mnemonic, password=passwd)

    network = input("Choose network [mainet->1] or [testnet->2]: ")

    while network != "1" and network != "2":
        network = input("Invalid input, [mainet->1] or [testnet->2]: ")

    if network == "1":
        network = "main"
    else:
        network = "test"
```

```
root = bip32.HDKey.from_seed(seed, version=NETWORKS[network]["xprv"])

fingerprint = root.child(0).fingerprint

# parse psbt transaction
b64_psbt = input("Paste here psbt transaction from Bitcoin Core: ")
# convert it to binary
raw = a2b_base64(b64_psbt)
# parse
tx = psbt.PSBT.parse(raw)

# print how much we are spending and where
total_in = 0
for inp in tx.inputs:
    total_in += inp.witness_utxo.value

print("Inputs:", total_in, "satoshi")

change_out = 0 # btc that goes back to us
send_outputs = []

for i, out in enumerate(tx.outputs):
    # check if it is a change or not:
    change = False

    # should be one or zero for single-key addresses
    for pub in out.bip32_derivations:
        # check if it is our key
        if out.bip32_derivations[pub].fingerprint == fingerprint:
            hdkey = root.derive(out.bip32_derivations[pub].derivation)
            mypub = hdkey.key.get_public_key()
            #or
            mypub = hdkey.to_public().key
            if mypub != pub:
                raise ValueError("Derivation path doesn't look right")
            # now check if provided scriptpubkey matches
            sc = script.p2wpkh(mypub)
            if sc == tx.tx.vout[i].script_pubkey:
                change = True
                continue

    if change:
        change_out += tx.tx.vout[i].value
        print("Change output:
        ",tx.tx.vout[i].value,"to",tx.tx.vout[i].script_pubkey.address(NETWORKS["test"]))
    else:
```

```
        send_outputs.append(tx.tx.vout[i])

fee = total_in-change_out

for out in send_outputs:
    fee -= out.value
    print("Send output: ",out.value,"to",out.script_pubkey.address(NETWORKS["test"]))

print("------------")
print("Sending: ", total_in)
print("Spending", total_in-change_out, "satoshi")
print("Fee:",fee,"satoshi")
print("Change:",change_out,"satoshi")

# sign the transaction - include into the psbt the partial signatures
tx.sign_with(root)
raw = tx.serialize()

# convert to base64
b64_psbt = b2a_base64(raw)

# b2a ends with \n
if b64_psbt[-1:] == b"\n":
    b64_psbt = b64_psbt[:-1]

print("\nSigned transaction:")
print(b64_psbt.decode('utf-8'))
# now transaction is ready to be finalized and broadcasted it can be done with Bitcoin Core
        bitcoin-cli [-testnet/-regtest/...] finalizepsbt [b64_psbt]
```

5.3.4.2 Comentarios psbt.py

Este fichero también tiene una única función, firmar una transacción. Para ello comienza con el proceso de derivación explicado anteriormente, hasta obtener la root key y su huella digital, la cual se emplea para comprobar que outputs de la transacción son de cambio. De manera resumida, el método solicita al usuario que pegue en el REPL la transacción psbt proporcionada por Bitcoin-Core, luego determina los datos importantes a mostrar al usuario, como puede ser la cantidad de BTC que envía en la transacción, la cantidad de BTC que está pagando a alguien, el cambio y la comisión, y luego los muestra. Posteriormente introduce en cada input de la transacción el scriptsig correspondiente para validar la transacción y muestra esta por el REPL para que el usuario la finalice mediante Bitcoin-Core.

5.3.5 show_addresses.py

5.3.5.1 Código

```
from binascii import hexlify, unhexlify
from bitcoin import bip32, bip39, script
from bitcoin.networks import NETWORKS

def show_addresses():
    f = open('mnemonic.txt', 'r')
    mnemonic = f.read()
    f.close()

    #Mnemonic to seed [64 bytes]: PBKDF2 key derivation
    passwd = input("Introduce password for key derivation: ")

    seed = bip39.mnemonic_to_seed(mnemonic, password=passwd)

    net = input("Choose network [mainet->1] or [testnet->2]: ")

    while net != "1" and net != "2":
        net = input("Invalid input, [mainet->1] or [testnet->2]: ")

    if net == "1":
        net = "main"
        hardened_derivation = "m/84h/0h/0h"
    else:
        net = "test"
        hardened_derivation = "m/84h/1h/0h"

    root = bip32.HDKey.from_seed(seed, version=NETWORKS[net]['xprv'])
    account = root.derive(hardened_derivation)
    #MAINET - NATIVE SEGWIT
    account.version = NETWORKS[net]['zprv']

    xpub = account.to_public()

    n = int(input("Introduce a number of addresses: "))

    for i in range(n):
        derivation = "m/0/"+str(i)
        pubkey=xpub.derive(derivation).key
        sc = script.p2wpkh(pubkey)
        addr = sc.address(network = NETWORKS[net])
        print("Address nº%s :" % i, addr)
```

5.3.5.2 Comentarios show_addresses.py

Este fichero contiene el código que se encarga de obtener la cantidad deseado por el usuario de direcciones. Comienza con un proceso explicado anteriormente, obtener la root key a partir de la frase mnemónica y luego la master public key, solicitando al usuario password y red. Luego solicita la cantidad de direcciones a mostrar, y construye los scripts de donde obtendremos las direcciones para luego mostrarlas.

6 Mejoras

Este proyecto tiene mucho margen de desarrollo y mejora, se pueden aplicar infinitud de nuevas funcionalidades y mencionaré algunas.

- Tipos de direcciones (BIP): Este proyecto se ha desarrollado para derivar direcciones native sewgit, pero también están las legacy, las nested segwit o las multisig.
- Lector QR: La gran mayoría de aplicaciones o webs en la red relacionadas con el mundo de las criptodivisas facilitan la lectura de direcciones y claves con códigos QR.
- Pantalla táctil: Con este accesorio se podría desarrollar un dispositivo mucho más atractivo e intuitivo para los usuarios.
- Otras criptodivisas: Se podría incluir en el dispositivo la gestión de claves de otras criptovidisas como puede ser Ethereum.

7 Conclusiones

Las expectativas del resultado final del proyecto eran mucho más altas, pero las circunstancias y la cantidad invertida en investigar debido al desconocimiento en el sector, tanto en el tema del hardware como el en de Bitcoin, han dejado un peor sabor de boca del esperado, dejando de lado funcionalidades como el uso de microSD. Aún así, estoy satisfecho de haber conseguido desarrollar un dispositivo funcional. Con más tiempo y dedicación, se puede pulir el proyecto hasta dejar un dispositivo realmente interesante.

8 Bibliografía

- Micropython: https://micropython.org/download/esp32/
- ESP-32: https://www.espressif.com/en/products/socs/esp32
- Bitcoin Core: https://bitcoin.org/en/bitcoin-core/
- Módulos Bitcoin (ESP32 Firmware) : https://github.com/stepansnigirev/esp32_embit
- Librería Display: https://github.com/adafruit/Adafruit_SSD1306
- Bitcoin Whitepaper: https://bitcoin.org/bitcoin.pdf