[BTC SeedPhrase Generator](https://github.com/paulohl/BinanceTradingBot)

Code analysis | debugging | clean code

Paulo H Leocadio | Fiverr Pro | Dec 06, 2024, 9:49 AM

## Code Analysis and initial impressions

### (1) Code Description

The script appears to be a `python implementation to:

1. **Generate a BIP-39 seed phrase** - This is a secure mnemonic used in Bitcoin and cryptocurrency wallets.
2. **Derive a Bitcoin address** - Using the seed phrase, it generates a Bitcoin wallet address following the BIP-44 standard.
3. **Match the address against a list of Bitcoin addresses** - It checks if the derived Bitcoin address matches any entry in a provided text file.
4. **Save the matching seed phrase and address** - Presumably for logging or later use (though the cutoff makes the details unclear).

This script is primarily educational, demonstrating how cryptographic wallets work. However, it's incomplete and has significant issues, both technically and in terms of ethical considerations.

### (2) Code Break-down Analysis

**Imports Section**

`python

import os

import requests

from bip\_utils import (

Bip39SeedGenerator,

Bip44,

Bip44Coins,

Bip39WordsNum,

Bip39MnemonicGenerator,

Bip44Changes,

)

import time

`

* **Purpose**: The imports bring in the libraries for generating seed phrases, deriving wallet addresses (Bip39/Bip44), and file handling.
* **Potential Issue**: The library bip\_utils may not be installed or compatible with the environment (e.g., `python version mismatch).

**Seed Phrase Generation**

`python

def generate\_seed\_phrase():

mnemonic = Bip39MnemonicGenerator().FromWordsNumber(Bip39WordsNum.WORDS\_NUM\_12)

return mnemonic

`

* **Purpose**: Creates a random 12-word mnemonic following the BIP-39 standard.
* **Potential Issue**: None in logic, but the bip\_utils library's installation/configuration is critical.

**Bitcoin Address Derivation**

`python

def get\_bitcoin\_address(seed\_phrase):

# Generate the seed from the mnemonic

seed\_bytes = Bip39SeedGenerator(seed\_phrase).Generate()

# Create a Bip44 wallet for Bitcoin

bip44\_mst = Bip44.FromSeed(seed\_bytes, Bip44Coins.BITCOIN)

# Derive the first receiving address (account 0, external chain, address index 0)

bip44\_acc = (

bip44\_mst.Purpose()

.Coin()

.Account(0)

.Change(Bip44Changes.CHAIN\_EXT)

.AddressIndex(0)

)

# Get the address

address = bip44\_acc.PublicKey().ToAddress()

return address

`

* **Purpose**: Converts the mnemonic into a Bitcoin wallet address.
* **Potential Issues**:
  + Missing definition of address where it’s being used later.
  + Library or method incompatibilities.
  + Possible misalignment between `python 2 and 3 syntax (though unlikely here).

**Address Matching**

`python

file\_name = '/home/home/Desktop/BTC/BTC Lists/BTCtestlist.txt'

matching\_lines = []

with open(file\_name, 'r') as file:

# Read each line in the file

lines = file.readlines()

# Compare each line to the `python output

for line in lines:

if line.strip() == address: # Strip to remove any extra spaces/newlines

matching\_lines.append(line.strip())

`

* **Purpose**: Reads a list of Bitcoin addresses from a file and checks for a match with the generated address.
* **Potential Issues**:
  + address is not defined before being used.
  + File path might not exist, causing a FileNotFoundError.

**Save Matching Results**

`python

def save\_seed\_phrase(seed\_phrase, address):

folder\_path = "Wallet\_Phrases1"

`

* **Purpose**: Presumably saves matching results (incomplete due to cutoff).
* **Potential Issues**:
  + Logic is incomplete and lacks file-writing implementation.

## Identified Bugs, Errors, and Concerns

**Critical Bugs**

1. **Undefined address Variable**: address is referenced before being defined, which will cause a NameError.
2. **Incomplete Implementation**: Saving seed phrases and addresses lacks implementation.
3. **Hardcoded File Path**: The hardcoded file path may not exist, leading to FileNotFoundError.
4. **Dependency Issues**: The script depends on bip\_utils, which might not be properly installed or configured.
5. **python Version Compatibility**: If written in `python 3.6.9, modern environments (`python 3.10+) may raise compatibility issues with syntax or libraries.

## Proposed Solution

**Fixing address Usage**

Ensure the get\_bitcoin\_address function is invoked, and its result is assigned to address before being used:

`python

address = get\_bitcoin\_address(seed\_phrase)

`

**Implementing Save Logic**

Complete the save\_seed\_phrase function:

`python

def save\_seed\_phrase(seed\_phrase, address):

folder\_path = "Wallet\_Phrases1"

os.makedirs(folder\_path, exist\_ok=True)

file\_path = os.path.join(folder\_path, "matched\_phrases.txt")

with open(file\_path, "a") as file:

file.write(f"Seed Phrase: {seed\_phrase}\nAddress: {address}\n\n")

`

**Dynamic File Path Handling**

Allow users to provide the file path dynamically:

`python

file\_name = input("Enter the path to the BTC address file: ")

`

**Dependency Installation**

Ensure bip\_utils is installed:

`bash

pip install bip-utils

`

### Update for python Compatibility

\*\* Test the script in python 3.6.9 and consider upgrading any incompatible libraries or methods.

### On the Statement: "CODE CUTOFF BY FIVERR..."

This reflects the challenge of incomplete or censored scripts. However:

* Ensure ethical use by not enabling or promoting harmful actions.
* Verify that the modifications align with educational or research purposes.

## Key Components to Review in the Code

### 1. Seed Phrase Generation

* **Purpose**: Generates a secure, random BIP-39 mnemonic phrase.
* **Review**:
  + Ensures compliance with the BIP-39 standard.
  + Uses a 12-word format for simplicity and standardization.
* **Key Concern**: Verifying the correctness of the mnemonic generator and its integration with bip\_utils.

### 2. Bitcoin Address Derivation

* **Purpose**: Converts the mnemonic into a Bitcoin address following the BIP-44 hierarchical deterministic wallet standard.
* **Review**:
  + Proper usage of BIP-44 hierarchical structure (Purpose > Coin > Account > Change > Address Index).
  + Address derivation is correctly implemented for Bitcoin (Bip44Coins.BITCOIN).
* **Key Concern**: Correct handling of cryptographic seed generation and public address conversion.

### 3. Address Matching Logic

* **Purpose**: Compares the derived Bitcoin address against a list of addresses from a file.
* **Review**:
  + Reads a text file containing Bitcoin addresses line by line.
  + Matches addresses with precision, removing extra spaces or newline characters.
* **Key Concern**: Handling file input errors and efficiently comparing the derived address.

### 4. Saving Results

* **Purpose**: Logs matched seed phrases and addresses to a file.
* **Review**:
  + Ensures the results are saved securely in an accessible folder.
  + Appends new matches to avoid overwriting previous results.
* **Key Concern**: File I/O robustness and creating directories if they don’t exist.

### 5. Exception Handling

* **Purpose**: Ensures the script operates smoothly despite potential runtime errors.
* **Review**:
  + Handles missing files (FileNotFoundError) gracefully.
  + Catches unexpected exceptions and provides informative error messages.
* **Key Concern**: Avoids abrupt script termination while guiding the user to fix issues.

## Step-by-Step Solution (with Implementations)

### Step 1: Seed Phrase Generation

**Function**: generate\_seed\_phrase()

* Generates a secure 12-word mnemonic.
* Ensures compatibility with BIP-39 for universal wallet standards.

`python

seed\_phrase = generate\_seed\_phrase()

print(f"Seed Phrase: {seed\_phrase}")

`

### Step 2: Bitcoin Address Derivation

**Function**: get\_bitcoin\_address(seed\_phrase)

* Derives a Bitcoin address from the mnemonic using BIP-44.
* Ensures that the derived address is valid and usable.

`python

address = get\_bitcoin\_address(seed\_phrase)

print(f"Bitcoin Address: {address}")

`

### Step 3: Address Matching

**Function**: match\_address\_to\_file(address, file\_name)

* Compares the derived address to a list of addresses in a user-specified file.
* Handles missing files gracefully and reports errors clearly.

`python

file\_name = input("Enter the path to the file containing Bitcoin addresses: ")

matches = match\_address\_to\_file(address, file\_name)

if matches:

print(f"Match found! Address: {address}")

else:

print("No matches found.")

`

### Step 4: Saving Results

**Function**: save\_seed\_phrase(seed\_phrase, address)

* Logs the seed phrase and matching Bitcoin address into a structured text file.
* Ensures file integrity and proper handling of directories.

`python

if matches:

save\_seed\_phrase(seed\_phrase, address)

`

### Step 5: Robust Error Handling

* Handles issues such as missing files or unexpected runtime errors.
* Ensures the script is user-friendly and resilient.

**Example:**

`python

try:

file\_name = input("Enter the path to the file containing Bitcoin addresses: ")

matches = match\_address\_to\_file(address, file\_name)

except FileNotFoundError:

print("Error: The specified file was not found.")

except Exception as e:

print(f"An unexpected error occurred: {e}")

`

## Conclusion

This enhanced aims at improving upon the original functionality, robustness, and documentation. By addressing bugs, adding error handling, and improving user interactivity, it achieves a practical and reliable implementation for educational purposes.

rewriting the script (with corrections)

`python

import os

import requests # Not used in this script but kept in case of future expansion

from bip\_utils import (

Bip39SeedGenerator,

Bip44,

Bip44Coins,

Bip39WordsNum,

Bip39MnemonicGenerator,

Bip44Changes,

)

import time

# ---------------------------------------------------------------------

# Script Purpose:

# 1. Generate a secure BIP-39 mnemonic (seed phrase).

# 2. Derive a Bitcoin address from the seed phrase using the BIP-44 standard.

# 3. Match the derived Bitcoin address against a list of addresses stored in a file.

# 4. Save the seed phrase and address if a match is found.

# ---------------------------------------------------------------------

# Step 1: Generate a BIP-39 seed phrase

def generate\_seed\_phrase():

"""

Generates a 12-word BIP-39 mnemonic (seed phrase) for cryptocurrency wallets.

Returns:

str: The generated mnemonic.

"""

mnemonic = Bip39MnemonicGenerator().FromWordsNumber(Bip39WordsNum.WORDS\_NUM\_12)

return mnemonic

# Step 2: Derive a Bitcoin address from the seed phrase

def get\_bitcoin\_address(seed\_phrase):

"""

Derives a Bitcoin address from the given seed phrase using the BIP-44 standard.

Args:

seed\_phrase (str): The 12-word mnemonic.

Returns:

str: The generated Bitcoin address.

"""

# Generate the seed from the mnemonic

seed\_bytes = Bip39SeedGenerator(seed\_phrase).Generate()

# Create a Bip44 wallet for Bitcoin

bip44\_mst = Bip44.FromSeed(seed\_bytes, Bip44Coins.BITCOIN)

# Derive the first receiving address (account 0, external chain, address index 0)

bip44\_acc = (

bip44\_mst.Purpose()

.Coin()

.Account(0)

.Change(Bip44Changes.CHAIN\_EXT)

.AddressIndex(0)

)

# Get the address

address = bip44\_acc.PublicKey().ToAddress()

return address

# Step 3: Match address to a list from a file

def match\_address\_to\_file(address, file\_name):

"""

Matches a given Bitcoin address to a list of addresses in a file.

Args:

address (str): The Bitcoin address to search for.

file\_name (str): The file containing a list of Bitcoin addresses.

Returns:

list: List of matching addresses found in the file.

"""

matching\_lines = []

try:

with open(file\_name, 'r') as file:

lines = file.readlines()

# Compare each line to the generated address

for line in lines:

if line.strip() == address: # Strip removes extra spaces/newlines

matching\_lines.append(line.strip())

except FileNotFoundError:

print(f"Error: The file {file\_name} was not found.")

except Exception as e:

print(f"An unexpected error occurred: {e}")

return matching\_lines

# Step 4: Save the seed phrase and address if a match is found

def save\_seed\_phrase(seed\_phrase, address, folder\_path="Wallet\_Phrases1"):

"""

Saves the seed phrase and corresponding Bitcoin address to a file.

Args:

seed\_phrase (str): The BIP-39 mnemonic.

address (str): The Bitcoin address.

folder\_path (str): The folder where the results will be saved.

"""

os.makedirs(folder\_path, exist\_ok=True)

file\_path = os.path.join(folder\_path, "matched\_phrases.txt")

try:

with open(file\_path, "a") as file:

file.write(f"Seed Phrase: {seed\_phrase}\nAddress: {address}\n\n")

print(f"Results saved to {file\_path}")

except Exception as e:

print(f"Failed to save results: {e}")

# Main Function

def main():

"""

Main execution function to tie all steps together.

"""

print("Generating a new seed phrase...")

seed\_phrase = generate\_seed\_phrase()

print(f"Seed Phrase: {seed\_phrase}")

print("Deriving Bitcoin address...")

address = get\_bitcoin\_address(seed\_phrase)

print(f"Bitcoin Address: {address}")

file\_name = input("Enter the path to the file containing Bitcoin addresses: ")

print("Checking for matches...")

matches = match\_address\_to\_file(address, file\_name)

if matches:

print(f"Match found! Address: {address}")

save\_seed\_phrase(seed\_phrase, address)

else:

print("No matches found.")

# Run the script

if \_\_name\_\_ == "\_\_main\_\_":

main()

`

**Key Enhancements:**

1. **Dynamic File Input**: The user provides the file path for the address list dynamically.
2. **Robust Error Handling**:
   * Handles missing files gracefully (FileNotFoundError).
   * General exception handling for unforeseen errors.
3. **Documentation**: Every function is documented with its purpose, parameters, and return values.
4. **Dynamic Saving**: Creates the save folder (Wallet\_Phrases1) if it doesn’t exist and writes matched seed phrases and addresses to a file.
5. **Clear Output**: Provides feedback at every step of the script's execution.