Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго этапа составляет 54 дня. Задачи требуют от участников наличие навыков программирования и помогают отработать те навыки, которые потребуются для решения командной задачи заключительного этапа.

Участники не были ограничены в выборе языка программирования для решения задач.

Объем и сложность задач этого этапа подобраны таким образом, чтобы решение всех задач одним человеком было маловероятно. Это призвано обеспечить включение командной работы и распределения обязанностей. Решение каждой задачи дает определенное количество баллов. Все задачи предусматривали возможность частичного решения. В данном этапе можно получить суммарно от 0 до 100 баллов.

Команды могут выполнять задачи в любом порядке. Задачи допускают неограниченное число попыток сдать решение.

Задачи второго этапа

Задачи по информатике

Задача II.1.1.1. Hierarchical Deterministic Wallet (10 баллов)

Иерархичные детерминированные кошельки (https://github.com/ethereumbook/ethereumbook/blob/develop/05wallets.asciidoc#hierarchical-deterministic-wallets-bip-32bip-44) широко используются в блокчейн платформах *Bitcoin* и *Ethereum*. Основное их удобство в том, что относительно небольшого набора данных достаточно, чтобы поставить в соответствие кошельку много приватных ключей, а, значит, много разных аккаунтов. Не нужно обеспечивать безопасное хранение всех приватных ключей иерархичного детерминированного кошелька – достаточно только хранить базовый набор данных. После восстановления кошелька после переустановки ПО или при переносе кошелька на другое устройств, все приватные ключи будут восстановлены благодаря свойству детерминированности.

Для простоты хранения и запоминания базовый набор данных представляют в виде 12 или 24 английских слов. Поэтому его еще называют посевной фразой (*seed phrase*).

В самом простом случае, иерархичность кошелька позволяет поставить в соответствие порядковый номер каждому приватному ключу, который может быть сгенерирован из *seed phrase*. В более сложных случаях, можно построить древовидную структуру, где приватные ключи могут быть объединены в группы. Тогда для генерации приватного ключа из *seed phrase* нужно знать путь в дереве до этого ключа. Этот путь еще называют *derivation path* (https://medium.com/myetherwallet/hd-wallets-and-derivation-paths-explained-865a643c7bf2).

Например, стандартный *derivation path* для *Ethereum* выглядит как набор байт `m/44'/60'/0'/0`.

В данной задаче вам нужно подготовить код, который бы по заданной *seed phrase* и *derivation path* мог получить аккаунт тестовой сети, построенной на технологии Ethereum, и узнать баланс данного аккаунта на указанный блок.

Формат входных данных

Первая строка входные данных определяет номер блока, для которого необходимо будет получать баланс аккаунтов. Далее будет следовать от от 10 до 20 строк, в каждой из которых будет указан *seed phrase* и *index* (последнее число в стандартном *Ethereum derivation path*).

Формат выходных данных

Одно число – максимальный баланс из тех аккаунтов, чьи приватные ключи могут быть получены из входных данных. Баланс определяется в тестовой сети, JSON RPC url для доступа к сети - `http://165.227.146.243:3000`.

Примеры

Пример №1

Стандартный ввод
590 enable clutch win expand zebra attend news plug age expose lamp same 273 amazing apology soccer maze delay below burger inmate major force rifle tunnel 237 code armed select price belt tonight stumble buddy orphan zero maid arch 576 odor dynamic maple solution hat pole luxury citizen devote sleep cart garage 97 powder horn catch forward scare sunny fortune flee picnic brown thunder romance 90 heavy uphold term arm rival afraid level abandon black few either found 39 quiz plunge way cradle truly cereal size omit mistake bargain wheat need 941 permit artist endless sleep junk undo enrich color sail excite candy lamp 65 recipe buyer sell hawk scan release chapter source robot faith boost female 914 base pelican civil lion warrior neither wood slim neutral cost razor frame 265

Стандартный вывод
853944

Решение

Задача сводится к перебору каждого кошелька:

1. Функцией `Web3.eth.account.from_mnemonic(words, account_path)` получаем аккаунт из фразы.
2. Функцией `get_balance(address, block_number)` получаем баланс текущего аккаунта, записываем этот баланс в массив: `balances.append(get_balance(address, block_number))`
3. Выводим максимум из этого массива: `max(balances)`

Пример программы-решения

Ниже представлено решение на языке Python 3

```
from web3 import Web3, HTTPProvider

node = Web3(HTTPProvider("http://165.227.146.243:3000"))

node.eth.account.enable_unaudited_hdwallet_features()

def get_balance(address, block_number=None):
    return node.eth.getBalance(address, block_number)

def solve(filename):
    file = open(filename, 'r')
    lines = file.read().split('\n')
    block = int(lines[0])
    balances = []
    for ln in lines[1:-1]:
        lns = ln.split()
        path = "m/44'/60'/0'/0/"+lns[-1]
        words = ' '.join(lns[:-1])
        acc = node.eth.account.from_mnemonic(words,account_path=path)
        balances.append(get_balance(acc.address, block_number=block))

    file.close()
    return max(balances)

```

Задача II.1.1.2. Тестовая сеть Ganache (5 баллов)

На ранних этапах разработки децентрализованных приложений для платформы *Ethereum* принято тестировать работу контрактов в тестовой блокчейн сети. По факту, это такая сеть в начале состоит даже всего лишь из одного узла. Тестовая сеть запускается, чаще всего, на той же самой системе, где и приложение, поэтому проверка кода на соответствие технической спецификации происходит существенно быстрее. А одно из правил разработки ПО – ошибка, выявленная на ранних стадиях разработки, обеспечивает экономию бюджета проекта на порядки.

Ниже перечислены несколько преимуществ тестовых локальных блокчейн сетей:

- Высокая скорость выпуска новых блоков (блок выпускается, как только появляется новая транзакция)
- Механизм выпуска новых блоков не оказывает существенной нагрузки на процессор, не требует серьезных вычислительных мощностей (*Proof-of-Authority* консенсус)
- Аккаунты, используемые для тестирования, могут иметь любое количество тестовых монет сразу при запуске сети
- Простота в настройке JSON RPC доступа

Несмотря на то, что наиболее популярное программное обеспечение для постройки блокчейн сетей на базе технологии *Ethereum* (например, *geth* (https://github.com/ethereum/go-ethereum) или *openethereum* (https://github.com/openethereum/openethereum)) поддерживает режим быстрой организации тестовой сети, их мало

используют для организации индивидуального рабочего окружения разработчика ПО (*SDE*, *software development environment*) именно в силу их универсальности.

Наоборот, проанализировав инструментарий разработчика можно увидеть, что широко применяются решения, специализирующиеся исключительно на тестовых сетях:

- ganache (https://github.com/trufflesuite/ganache-cli)
- hardhat network (https://github.com/nomiclabs/hardhat)

В этой задаче вам предстоит научиться запускать узел тестовой сети в *контейнеризированном* (https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) окружении *Docker* (https://docs.google.com/presentation/d/1JEHz3heYuyjLjjAnW_LJoN1MiNNtL2ZNWNmkSPi_onA/edit) так, чтобы к узлу был доступ через JSON RPC, используя который необходимо получить данные о балансах аккаунтов.

Формат входных данных

Строка – имя *Docker* образа, построенного на основе *ganache-cli*, с уже проинициализированной базой данных, в которой хранится информация о выпущенных блоках и о выполненных транзакциях. Количество выпущенных блоков и выполненных транзакций заранее неизвестно. Но известно, что все аккаунты, сгенерированные на момент инициализации блокчейн узла и имеющие на момент инициализации ненулевой баланс, принимали участие, как минимум, в одной транзакции.

Формат выходных данных

Одно число – число аккаунтов с ненулевым балансом, сформированным в результате инициализации базы данных блокчейн узла и выполнением транзакций из всей цепочки блоков, которые зафиксированы в образе, указанном во входных данных.

Примеры

Пример №1

Стандартный ввод
`dashvayet/fintech2020node11`

Стандартный вывод
`156`

Решение

Основные шаги решения

1. Запуск docker-контейнера с тестовой сетью ganache
2. Подключение к тестовой сети
3. Поулчение информации о тестовой сети
4. Проход по всем блокам задекларированных в базе данных тестовой сети

5. Проверка сохраненных адресов на наличие активов тестовой сети на их счету

Шаг 1: Запуск docker-контейнера с тестовой сетью ganache

Для решения задачи необходимо первоначально скачать выданный во входных данных docker-образ из сетевого хранилища docker hub, используя команду

```
docker pull dashvayet/fintech2020node
```

Шаг 2: Подключение к тестоввой сети

На основе выданного docker-образа необходимо запустить docker-контейнер с тестовой сетью использую команду:

```
docker run -p <физический порт>:8545 <имя docker-образа>
```

Затем необходимо выполнить подключение к тестовой сети. Автомтически это можно производить с помощью внешних библиотек для работы с блокчейн узлами. Ниже приведен пример на языке Python:

```
from web3 import Web3

web3 = Web3(Web3.HTTPProvider('http://127.0.0.1:8545'))
```

Шаг 3: Поулчение информации о тестовой сети

Для решения задачи необходимо знать количество блоков встроенных в цепь. Эту информцию можно получить из базы данных тестовой сети. Ниже приведен пример на языке Python:

```
blockNum = web3.eth.blockNumber
```

Шаг 4: Проход по всем блокам задекларированных в базе данных тестовой сети

Используя информацию полученную в шаге 3, необходимо пройти алгоритмом по каждому задекларированному блоку и сохранить все адреса, которые хотя бы раз участвовали в транзакциях в качестве отправителя или получателя. Пример получения информации о блоке из базы данных на языке Python:

```
block = web3.eth.getBlock(i)
transaction = web3.eth.getTransaction(block["transactions"][0].hex())
sender = transaction["from"]
receiver = transaction["to"]
```

Шаг 5: Проверка сохраненных адресов на наличие активов тестовой сети на их счету

Используя список адресов полученных в шаге 4, проверяем баланс каждого адреса на наличие средств. Количество адресов с ненулевым балансом – конечный ответ на задачу.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
import random
import json
from web3 import Web3
from eth_account import Account
web3 = Web3(Web3.HTTPProvider('http://0.0.0.0:8545'))
blockNum = web3.eth.blockNumber
senders = []
all = []
for i in range(blockNum+1):
    block = web3.eth.getBlock(i)
    try:
        transaction = web3.eth.getTransaction(block["transactions"][0].hex())
        sender = transaction["from"]
        receiver = transaction["to"]
        if sender not in senders:
            senders.append(receiver)
        if receiver not in all:
            all.append(receiver)
        if sender not in all:
            all.append(sender)
    except Exception as e:
        print(e)
answer = 0
print(all)
print(len(all))
for i in range(len(all)):
    if web3.eth.getBalance(all[i]) != 0:
        answer += 1
print(answer)
# solution code here
```

Задача II.1.1.3. Extract, Transform and Load (25 баллов)

В базах данных широко применяется такое понятие, как *ETL* (*Extract, Transform, Load*, https://ru.wikipedia.org/wiki/ETL) – специальный процесс в управлении хранилищами данных.

Иногда, похожий на *ETL* процесс используют для миграции данных из одной базы данных в другую.

Представим, что есть магазин (или сервис), который до сегодняшнего дня для своей бонусной программы использовал SQL базу данных. База данных была представлена всего двумя таблицами: таблица начисления бонусных баллов покупателям и таблица перевода бонусных баллов покупателями друг другу. Но, повинуясь трендам, руководство решило перевести хранение информации о бонусах в блокчейн, построенный по технологии Ethereum. Для этого, идеально подходит контракт токена, реализованный по стандарту *ERC20* (https://forklog.com/chto-takoe-tokeny-erc-20/). Данный стандарт позволит хранить состояние бонусного счета конкретного покупателя, а также передавать бонусы от одного покупателя к другому.

По умолчанию, стандарт токена не предусматривает наличие у пользователей начального ненулевого баланса после регистрации контракта токена в блокчейн сети.

Это препятствует использованию существующих (например, https://gist.github.com/bajcmartinez/c1cefbb3b1375d037f07b7b779568e42/) реализации стандарта. Следовательно, нужно изменить код контракта так, чтобы сразу после регистрации контракта пользователи могли получить информацию о состоянии своих бонусных счетов и передавать бонусы друг другу.

У вас есть база данных в виде файла в формате *SQLite* (https://en.wikipedia.org/wiki/SQLite). Вам необходимо подготовить скрипт на языке Python, который бы формировал байткод контракта, реализующий токен ERC20 стандарта.

Скрипт будет запускаться на системе, где нет ни компилятора *Solidity*, ни *Vyper*. Не гарантируется, что скрипту будет предоставлена возможность доставить необходимые библиотеки в систему. Иными словами, для решения задачи необходимо обойтись только стандартными библиотеками, *web3py* и библиотеками, которые устанавливаются по зависимостям от *web3py*.

Формат входных данных

Строка, содержащая путь до файла с *SQLite* базой данных. В базе данных – таблица начисления бонусных баллов пользователям, таблица переводов бонусных баллов пользователей друг другу, таблица соответствия пользователей аккаунтам блокчейн сети, построенной на технологии *Ethereum*.

Пример базы данных: https://drive.google.com/file/d/1clk392aLfKYJFAk6aJzfbGoLtY4KQ2view?usp=sharing

Формат выходных данных

Байт код контракта *ERC20* токена, представленный в виде последовательности шестнадцатиричных цифр (без префикса `0x`). Если данный контракт зарегистрировать (выполнить *deploy*) в блокчейн, то сразу после этой операции (нет никаких дополнительных транзакций к этому контракту) пользователи могут получать информацию о состоянии своих бонусных счетов и передавать бонусы друг другу.

Пример байткода, полученного для базы данных, представленной выше: https://drive.google.com/file/d/1TEjCjStOfx8gK-fOajbG3BVUAwjVUPnA/view?usp=sharing

Комментарии

Перед решением задачи рекомендуется ознакомиться с набором статей под общим заголовком "Разборка Solidity-контрактов" ("Deconstructing a Solidity Contract"), которая дает первоначальное понимание, как устроен байткод контракта, полученный после компиляции: https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-i-introduction-832efd2d7737/.

Для решения задачи также рекомендуется ознакомиться с::

- со справочниками команд виртуальной машины Ethereum: https://ethervm.io/ и https://github.com/crytic/evm-opcodes
- с инструментарием по декомпиляции Ethereum контрактов: https://ethervm.io/decompile и https://etherscan.io/opcode-tool

Решение

Первым делом необходимо получить данные о балансах пользователей и их адреса кошельков. Для получения адресов достаточно выполнить SQL запрос:

```
SELECT * FROM users
```

Это вернет таблицу, состоящюю из *id* и адресов пользователей. Далее необходимо посчитать количество начисленных средств, количество отправленных средств и количество принятых средств с помощию следующих SQL запросов соответсвенно:

```
SELECT SUM(_amount) FROM accruals  WHERE _to = user_id
SELECT SUM(_amount) FROM transfers WHERE _from = user_id
SELECT SUM(_amount) FROM accruals  WHERE _to = user_id
```

Далее необходимо сложить получившиеся три параметра чтобы получить текущий баланс пользователя. Для работы с *sqlite* базами данных в python3 есть встроенная библиотека `sqlite3`.

```
import sqlite3

def get_balances(db_path:str) -> dict:
    db = sqlite3.connect(db_path)

    balances = {}

    for (id, address) in db.execute('SELECT * FROM users').fetchall():
        sended   = db.execute(f'SELECT SUM(_amount) FROM transfers WHERE _from = {id}').fetchone()[0]
        received = db.execute(f'SELECT SUM(_amount) FROM transfers WHERE _to   = {id}').fetchone()[0]
        accruals = db.execute(f'SELECT SUM(_amount) FROM accruals WHERE _to   = {id}').fetchone()[0]

        balance = 0

        if accruals:
            balance += accruals

        if received:
            balance += received

        if sended:
            balance -= sended

        balances[f'0x{address.hex()}'] = balance

    return balances
```

Составление байткода

Для решения был взят изветсный токен DAI Token, исходный код котракта можно найти в браузере блоков *Etherscan*: https://etherscan.io/address/0x6b175474e89094c44da98b
code.

Байт код контракта можно условно разделить на три части: *deploy* код, *runtime* код и *metadata*. Для решения использования *metadata* неважно, им можно пренебречь. Чтобы получить *runtime* байткод достаточно скомпилировать исходный код в среде разработки *Remix* (https://remix.ethereum.org/) и после компиляции нажать кнопку "Details" и скопировать *runtime bytecode object*. Получить *deploy* код можно скопировав *bytecode object* из той же страницы и удалить *runtime bytecode* и всё, что после него (*metadata*) путём обыного поиска подстроки в строке. После этого мы получим две шестнадцетиричные строки с *deploy* и *runtime bytecode*.

Deploy bytecode состоит их из множества частей, но нам важны последние три, это тело конструктора, копирования *runtime* кода и код возврата. Если мы посмотрим на конец deploy *bytecode* в опкодах (опкоды можно получить путём декомпиляции байткода в https://etherscan.io/opcode-tool. Также список опкодов https://github.com/crytic/evm-opcodes), то мы увидим:

```
[399] SSTORE
[400] POP
[401] POP
[404] PUSH2 0x1ee0
[405] DUP1
[408] PUSH2 0x01a1
[410] PUSH1 0x00
[411] CODECOPY
[413] PUSH1 0x00
[414] RETURN
[415] 'fe'(Unknown Opcode)
```

Эквивалент в байткоде: `555050611ee0806101a16000396000f3fe`. Сдесь инструкции после `[411] CODECOPY` – код возварата а байты 401-411 (включительно) код копирования *runtime*. Следовательно выше идет конец тела конструктора. Если мы вставим в тело конструтора код, который задаст начальный баланс для одного адреса:

```
balanceOf[0xdCad3a6d3569DF655070DEd06cb7A1b2Ccd1D3AF] = 1234;
```

Мы увидим, что байткод изменился:

```
[398] SWAP1
[399] SSTORE
[400] POP
[403] PUSH2 0x04d2
[405] PUSH1 0x02
[407] PUSH1 0x00
[428] PUSH20 0xdcad3a6d3569df655070ded06cb7a1b2ccd1d3af
[449] PUSH20 0xffffffffffffffffffffffffffffffffffffffff
[450] AND
[471] PUSH20 0xffffffffffffffffffffffffffffffffffffffff
[472] AND
[473] DUP2
[474] MSTORE
[476] PUSH1 0x20
[477] ADD
[478] SWAP1
[479] DUP2
[480] MSTORE
```

```
[482] PUSH1 0x20
[483] ADD
[485] PUSH1 0x00
[486] SHA3
[487] DUP2
[488] SWAP1
[489] SSTORE
[490] POP
[491] POP
[494] PUSH2 0x1ee0
[495] DUP1
[498] PUSH2 0x01fb
[500] PUSH1 0x00
[501] CODECOPY
[503] PUSH1 0x00
[504] RETURN
[505] 'fe'(Unknown Opcode)
```

T.e. появился фрагмент:

```
[400] POP
[403] PUSH2 0x04d2
[405] PUSH1 0x02
[407] PUSH1 0x00
[428] PUSH20 0xdcad3a6d3569df655070ded06cb7a1b2ccd1d3af
[449] PUSH20 0xffffffffffffffffffffffffffffffffffffffff
[450] AND
[471] PUSH20 0xffffffffffffffffffffffffffffffffffffffff
[472] AND
[473] DUP2
[474] MSTORE
[476] PUSH1 0x20
[477] ADD
[478] SWAP1
[479] DUP2
[480] MSTORE
[482] PUSH1 0x20
[483] ADD
[485] PUSH1 0x00
[486] SHA3
[487] DUP2
[488] SWAP1
[489] SSTORE
[490] POP
```

В байткоде: `6104d26002600073dcad3a6d3569df655070ded06cb7a1b2ccd1d3af73ffffffffffffff`

Здесь первый `PUSH [403]` добавляет в стек баланс, `PUSH` на `[405]` указывает на индекс маппинга (маппинг *balanceOf* имеет идекс 2, если мы посмотрим на исходный код контракта) и `PUSH` на `[428]` указывает адресс. Следовательно, нам достаточно менять баланс и адресс в исходном байткоде и опционально индекс маппинга (для универсальности, чтобы легко менять исходный код контракта).

Тут стоит заметить, что в зависимости от размера числа, которое добавляется в стек также меняется команда `PUSH` - `PUSH1` добавляет один байт, `PUSH2` два байта и так далее до 32 байт. Мы можем написать функцию, которая будет расчитывать `PUSH` и возащать необходимы данные для форматирования строки:

```
1 def sizeof(value:int) -> int:
2     if value == 0:
```

```
        return 1

    # Amount of bytes to store the value.
    return math.floor(math.log(value, 256) + 1)

def push(value:int) -> tuple[int, int, int]:
    size = sizeof(value)

    # Computing PUSH opcode.
    p = 96 + (size - 1)

    return (p, size * 2, value)

s = '%x%.*x' % push(value)
```

Здесь строка s сначала подставляет необходимый PUSH, затем минимальный размер поля и само *value*. Все числа переводятся в шестнадцатиричную систему. В итоге, мы можем составить строку-шаблон, которая будет определять начальный баланс одного кошелька:

```
append = \

        ↪  '%x%.*x%x%.*x600073%s73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffff'
        ↪  + \
        'ffffffffffffffffffffffff1681526020019081526020016000 2081905550'
```

И далее ее использовать:

```
for (address, balance) in balances.items():
        bytecode += append % (*push(balance), *push(mapping_index), address[2:])
```

Вернемя к изначальному байткоду `555050611ee0806101a16000396000f3fe`.

```
[399] SSTORE
[400] POP
[401] POP
[404] PUSH2 0x1ee0
[405] DUP1
[408] PUSH2 0x01a1
[410] PUSH1 0x00
[411] CODECOPY
[413] PUSH1 0x00
[414] RETURN
[415] 'fe'(Unknown Opcode)
```

Далее нам итересен фрагмент `CODECOPY` 401-411 (включительно) , т.е. `50611ee0806101a1600039`. Первый `PUSH [404]` добавляет в стек размер кода для копирования, второй `PUSH [408]` указывает позицию, с которой копировать, и вызов `CODECOPY [411]` копирует заданные байты. Аналогично примеру выше, создаем фрагмент для `COPYCODE`.

```
copycode = '50%x%.*x80%x%.*x6000396000f3fe'

# In bytes.
runtime_size  = len(runtime) // 2

```

```
# In bytes.
bytecode_size = len(bytecode) // 2 + (len(copycode) - 8) // 2

size  = push(runtime_size)
from_ = push(bytecode_size + sizeof(runtime_size) + sizeof(bytecode_size))

bytecode += copycode % (*size, *from_)
```

Вот весь код (строки с байткодом сокращены для лучшего представления), который создаёт байткод в правильном порядке:

```
def build_contract(balances:dict) -> str:
    def sizeof(value:int) -> int:
        if value == 0:
            return 1

        # Amount of bytes to store the value.
        return math.floor(math.log(value, 256) + 1)

    def push(value:int) -> tuple[int, int, int]:
        size = sizeof(value)

        # Computing PUSH opcode.
        p = 96 + (size - 1)

        return (p, size * 2, value)


    constructor = 'hex_string'
    runtime = 'hex_string'

    mapping_index = 2

    append = \

        ↪  '%x%.*x%x%.*x600073%s73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffff'
        ↪  + \
        'ffffffffffffffffffffffff16815260200190815260200160002081905550'

    copycode = '50%x%.*x80%x%.*x6000396000f3fe'


    bytecode = ''

    bytecode += constructor

    for (address, balance) in balances.items():
        bytecode += append % (*push(balance), *push(mapping_index), address[2:])

    # In bytes.
    runtime_size  = len(runtime) // 2

    # In bytes.
    bytecode_size = len(bytecode) // 2 + (len(copycode) - 8) // 2

    size  = push(runtime_size)
    from_ = push(bytecode_size + sizeof(runtime_size) + sizeof(bytecode_size))

    bytecode += copycode % (*size, *from_)
```

```
47
48      bytecode += runtime
49
50      return bytecode
```

Пример программы-решения

Ниже представлено решение на языке Python 3

```
1   import sys
2   import math
3   import sqlite3
4
5   def get_balances(db_path:str) -> dict:
6       db = sqlite3.connect(db_path)
7
8       balances = {}
9
10      for (id, address) in db.execute('SELECT * FROM users').fetchall():
11          sended   = db.execute(f'SELECT SUM(_amount) FROM transfers WHERE _from =
            ↪  {id}').fetchone()[0]
12          received = db.execute(f'SELECT SUM(_amount) FROM transfers WHERE _to   =
            ↪  {id}').fetchone()[0]
13          accruals = db.execute(f'SELECT SUM(_amount) FROM accruals  WHERE _to   =
            ↪  {id}').fetchone()[0]
14
15          balance = 0
16
17          if accruals:
18              balance += accruals
19
20          if received:
21              balance += received
22
23          if sended:
24              balance -= sended
25
26          balances[f'0x{address.hex()}'] = balance
27
28      return balances
29
30  def build_contract(balances:dict) -> str:
31      def sizeof(value:int) -> int:
32          if value == 0:
33              return 1
34
35          # Amount of bytes to store the value.
36          return math.floor(math.log(value, 256) + 1)
37
38      def push(value:int) -> tuple[int, int, int]:
39          size = sizeof(value)
40
41          # Computing PUSH opcode.
42          p = 96 + (size - 1)
43
44          return (p, size * 2, value)
45
46      constructor = \
```

```
47
        ↪ '6080604052348015610010576000 80fd5b506040516121e13803806121e18339818101604052602 0'
        ↪ + \
48
        ↪ '8110156100335760008 0fd5b810190808051906020019092919050505060016000803373ffffffff'
        ↪ + \
49
        ↪ 'ffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681'
        ↪ + \
50
        ↪ '5260200190815260200160002081905550604051808061218f60529139605201905060405180910 3'
        ↪ + \
51
        ↪ '9020604051806040016040528060 0e81526020017f44616920537461626c65636f696e0000000000'
        ↪ + \
52
        ↪ '0000000000000000000000000815250805190602001206040518060400160405280600181526020'
        ↪ + \
53
        ↪ '017f31000000000000000000000000000000000000000000000000000000000815250805190'
        ↪ + \
54
        ↪ '6020012083306040516020018086815260200185815260200184815260200183815260200182 73ff'
        ↪ + \
55
        ↪ 'ffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffff'
        ↪ + \
56
        ↪ 'ff16815260200195505050505050604051602081830303815290604052805190602001206005 8190'
        ↪ + \
57      '5550'
58
59  runtime = \
60
        ↪ '6080604052348015610010576000 80fd5b5060043610610142576000356 0e01c80637ecebe001161'
        ↪ + \
61
        ↪ '00b8578063a9059cbb1161007c578063a9059cbb146106b4578063b753a98c1461071a578063bb35'
        ↪ + \
62
        ↪ '783b14610768578063bf353dbb146107d6578063dd62ed3e1461082e578063f2d5d56b146108a657'
        ↪ + \
63
        ↪ '610142565b80637ecebe00146104a15780638fcbaf0c146104f957806395d89b411461059f578063'
        ↪ + \
64
        ↪ '9c52a7f1146106225780639dc29fac14610666576101425 65b8063313ce5671161010a578063313c'
        ↪ + \
65
        ↪ 'e567146102f25780633644e5151461031657806340c10f191461033457806354fd4d501461038257'
        ↪ + \
66
        ↪ '806365fae35e1461040557806370a08231146104495761014256 5b806306fdde0314610147578063'
        ↪ + \
67
        ↪ '095ea7b3146101ca57806318160ddd1461023057806323b872dd1461024e57806330adf81f146102'
        ↪ + \
68
        ↪ 'd4575b600080fd5b61014f6108f4565b6040518080602001828103825283818151815260200191 50'
        ↪ + \
```

```
69
        ↪ '80519060200190808383600005b8381101561018f57808201518184015260208101905061017456 5b'
        ↪ + \
70
        ↪ '505050509050908101906 01f1680156101bc57808203805160018360200361010 00a031916815260'
        ↪ + \
71
        ↪ '200191505b5092505050604051809103 90f35b6102166004803603604081101561 01e057600080fd'
        ↪ + \
72
        ↪ '5b81019080803573ffffffffffffffffffffffffffffffffffffffff1690602001909291908035 90'
        ↪ + \
73
        ↪ '60200190929190505050610 92d565b6040518082151515158152602001915050604051809103 90f3'
        ↪ + \
74
        ↪ '5b610238610a1f565b6040518082815260200191505060405180910390f35b6102ba600480360360'
        ↪ + \
75
        ↪ '6081101561026457600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff'
        ↪ + \
76
        ↪ '1690602001909291908035 73ffffffffffffffffffffffffffffffffffffffff1690602001909291'
        ↪ + \
77
        ↪ '90803590602001909291905050506 10a25565b6040518082151515158152602001915050604051 80'
        ↪ + \
78
        ↪ '910390f35b6102dc610f3a565b6040518082815260200191505060405180910390f35b6102fa610f'
        ↪ + \
79
        ↪ '61565b604051808260ff1660ff16815260200191505060405180910390f35b61031e610f66565b60'
        ↪ + \
80
        ↪ '40518082815260200191505060405180910390f35b6103806004803603604081101561034a576000'
        ↪ + \
81
        ↪ '80fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff1690602001909291908 0'
        ↪ + \
82
        ↪ '35906020019092919050505061 0f6c565b005b61038a611128565b6040518080602001828103825 2'
        ↪ + \
83
        ↪ '83818151815260200191508051906020019080838360005b838110156103ca578082015181840152'
        ↪ + \
84
        ↪ '6020810190506103af565b505050509050908101906 01f1680156103f75780820380516001836020'
        ↪ + \
85
        ↪ '036101000a031916815260200191505b5092505050604051809103 90f35b6104476004803603602 0'
        ↪ + \
86
        ↪ '81101561041b57600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff16'
        ↪ + \
87
        ↪ '90602001909291905050506 11161565b005b61048b6004803603602081101561045f57600080fd5b'
        ↪ + \
88
        ↪ '81019080803573ffffffffffffffffffffffffffffffffffffffff1690602001909291905050506 1'
        ↪ + \
```

```
89
        ↪ '128f565b6040518082815260200191505060405180910390f35b6104e360048036036020811015561'
        ↪ + \
90
        ↪ '04b757600080fd5b81019080803573ffffffffffffffffffffffffffffffffffffffff1690602001'
        ↪ + \
91
        ↪ '909291905050506112a7565b6040518082815260200191505060405180910390f35b61059d600480'
        ↪ + \
92
        ↪ '360361010081101561051057600080fd5b81019080803573ffffffffffffffffffffffffffffffff'
        ↪ + \
93
        ↪ 'ffffffff16906020019092919080357 3ffffffffffffffffffffffffffffffffffffffff16906020'
        ↪ + \
94
        ↪ '019092919080359060200190929190803590602001909291908035151590602001909291908035 60'
        ↪ + \
95
        ↪ 'ff16906020019092919080359060200190929190803590602001909291905050506112bf565b005b'
        ↪ + \
96
        ↪ '6105a76117fa565b604051808060200182810382528381815181526020019150805190602001908 0'
        ↪ + \
97
        ↪ '838360005b838110156105e75780820151818401526020810190506105cc565b5050505090509081'
        ↪ + \
98
        ↪ '0190601f168015610614578082038051600183602003610100 0a031916815260200191505b509250'
        ↪ + \
99
        ↪ '505060405180910390f35b610664600480360360208110156106385 7600080fd5b81019080803573'
        ↪ + \
100
        ↪ 'ffffffffffffffffffffffffffffffffffffffff1690602001909291905050506118335 65b005b61'
        ↪ + \
101
        ↪ '06b26004803603604081101561067c57600080fd5b81019080803573ffffffffffffffffffffffff'
        ↪ + \
102
        ↪ 'ffffffffffffffff1690602001909291908035906020019092919050505061196 1565b005b610700'
        ↪ + \
103
        ↪ '60048036036040811015610 6ca57600080fd5b81019080803573ffffffffffffffffffffffffff'
        ↪ + \
104
        ↪ 'fffffffffff1690602001909291908035906020019092919050505061 1df4565b60405180821515'
        ↪ + \
105
        ↪ '151581526020019150506040518091039 0f35b6107666004803603604081101561073057600080fd'
        ↪ + \
106
        ↪ '5b81019080803573ffffffffffffffffffffffffffffffffffffffff1690602001909291908035 90'
        ↪ + \
107
        ↪ '6020019092919050505061 1e09565b005b6107d46004803603606081101561077e57600080fd5b81'
        ↪ + \
108
        ↪ '019080803573ffffffffffffffffffffffffffffffffffffffff1690602001909291908035 73ffff'
        ↪ + \
```

```
109
        ↪ 'ffffffffffffffffffffffffffffffffffffff1690602001909291908035906020019092919050505050'
        ↪ + \
110
        ↪ '611e19565b005b610818600480360360208110156107ec57600080fd5b81019080803573ffffffff'
        ↪ + \
111
        ↪ 'ffffffffffffffffffffffffffffffffff1690602001909291905050506 11e2a565b60405180828152'
        ↪ + \
112
        ↪ '6020019150506040518091039 0f35b610890600480360360408110156108445760 0080fd5b810190'
        ↪ + \
113
        ↪ '80803573ffffffffffffffffffffffffffffffffffffffff1690602001909291908035 73ffffffff'
        ↪ + \
114
        ↪ 'ffffffffffffffffffffffffffff1690602001909291905050506 11e42565b60405180828152'
        ↪ + \
115
        ↪ '6020019150506040518091039 0f35b6108f26004803603604081101561086 bc57600080fd5b810190'
        ↪ + \
116
        ↪ '80803573ffffffffffffffffffffffffffffffffffffffffff1690602001909291908035 9060200190'
        ↪ + \
117
        ↪ '929190505050611e67565b005b6040518060400160405280600e81526020017f4461692053746162'
        ↪ + \
118
        ↪ '6c65636f696e000000000000000000000000000000000081525081565b600081600360003373ff'
        ↪ + \
119
        ↪ 'ffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff'
        ↪ + \
120
        ↪ 'ff16815260200190815260200160002060008573ffffffffffffffffffffffffffffffffffffffff'
        ↪ + \
121
        ↪ '1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020819055'
        ↪ + \
122
        ↪ '508273ffffffffffffffffffffffffffffffffffffffff163373ffffffffffffffffffffffff'
        ↪ + \
123
        ↪ 'fffffffffffff167f8c5be1e5ebec7d5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac8c7c3b925'
        ↪ + \
124
        ↪ '8460405180828152602001915050604051809 10390a3600190509291505056 5b60015481565b6000'
        ↪ + \
125
        ↪ '81600260008673ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff'
        ↪ + \
126
        ↪ 'ffffffffffffffff168152602001908152602001600020541015610adc576040517f08c379a000'
        ↪ + \
127
        ↪ '0000000000000000000000000000000000000000000000000000815260040180806020018281 03'
        ↪ + \
128
        ↪ '8252601881526020018 07f4461692f696e7375666666963696 56e742d62616c616e63650000000000'
        ↪ + \
```

```
129
        ↪ '000000815250602001915050604051809103 90fd5b3373ffffffffffffffffffffffffffffffff' \
        ↪ + \
130
        ↪ 'ffffff168473ffffffffffffffffffffffffffffffffffffffffffffff1614158015610bb457507fffffff' \
        ↪ + \
131
        ↪ 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff600360008673ffffffffff' \
        ↪ + \
132
        ↪ 'ffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152' \
        ↪ + \
133
        ↪ '60200190815260200160002060003373ffffffffffffffffffffffffffffffffffffffff1673ffff' \
        ↪ + \
134
        ↪ 'ffffffffffffffffffffffffffffffff16815260200190815260200160002054 14155b15610d' \
        ↪ + \
135
        ↪ 'b257816003600086 73ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffff' \
        ↪ + \
136
        ↪ 'ffffffffffffffffffff16815260200190815260200160002060003373ffffffffffffffffff' \
        ↪ + \
137
        ↪ 'ffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffff1681526020019081' \
        ↪ + \
138
        ↪ '526020016000205410156 10cab576040517f08c379a0000000000000000000000000000000000' \
        ↪ + \
139
        ↪ '000000000000000000008152600401808060200182810382526 01a8152602001807f4461692f696e' \
        ↪ + \
140
        ↪ '73756666696369656e742d616c6c6f77616e6365000000000000815250602001915050604051809 1' \
        ↪ + \
141
        ↪ '0390fd5b610d31600360008673ffffffffffffffffffffffffffffffffffffffff1673ffffffffff' \
        ↪ + \
142
        ↪ 'ffffffffffffffffffffffffffff16815260200190815260200160002060003373ffffffffffff' \
        ↪ + \
143
        ↪ 'ffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260' \
        ↪ + \
144
        ↪ '200190815260200160002054836 11e77565b600360008673ffffffffffffffffffffffffffff' \
        ↪ + \
145
        ↪ 'ffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600 0' \
        ↪ + \
146
        ↪ '2060003373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff' \
        ↪ + \
147
        ↪ 'ffffffffffffff16815260200190815260200160002081905550 5b610dfb600260008673ffffffff' \
        ↪ + \
148
        ↪ 'ffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff1681' \
        ↪ + \
```

```
149
        ↪ '526020019081526020016000205483611e77565b600260008673ffffffffffffffffffffffffffff'
        ↪ + \
150
        ↪ 'fffffffffff1673fffffffffffffffffffffffffffffffffffffffffff168152602001908152602001'
        ↪ + \
151
        ↪ '60002081905550610e87600260008573ffffffffffffffffffffffffffffffffffffffffffff1673ffff'
        ↪ + \
152
        ↪ 'ffffffffffffffffffffffffffffffffff168152602001908152602001600020548361 1e91565b'
        ↪ + \
153
        ↪ '600260008573ffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffff'
        ↪ + \
154
        ↪ 'fffffffffffffff16815260200190815260200160002081905550827 3ffffffffffffffffffff'
        ↪ + \
155
        ↪ 'fffffffffffffffff168473ffffffffffffffffffffffffffffffffffffffff167fddf252ad1be2'
        ↪ + \
156
        ↪ 'c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef8460405180828152602001915050'
        ↪ + \
157
        ↪ '60405180910390a3600190509392505050565b7fea2aa0a1be11a07ed86d755c93467f4f82362b45'
        ↪ + \
158
        ↪ '2371d1ba94d1715123511acb60001b81565b601281565b60055481565b60016000803373ffffffff'
        ↪ + \
159
        ↪ 'ffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffff1681'
        ↪ + \
160
        ↪ '526020019081526020016000205414611020576040517f08c379a000000000000000000000000'
        ↪ + \
161
        ↪ '0000000000000000000000000000815260040180806020018281038252601281526020018 07f44'
        ↪ + \
162
        ↪ '61692f6e6f742d617574686f72697a656400000000000000000000000000008152506020019150 50'
        ↪ + \
163
        ↪ '60405180910390fd5b611069600260008473ffffffffffffffffffffffffffffffffffffffff1673'
        ↪ + \
164
        ↪ 'ffffffffffffffffffffffffffffffffff168152602001908152602001600020548 2611e91'
        ↪ + \
165
        ↪ '565b600260008473ffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffff'
        ↪ + \
166
        ↪ 'fffffffffffffffff16815260200190815260200160002081905550611 0b860015482611e9156'
        ↪ + \
167
        ↪ '5b60018190555081 73ffffffffffffffffffffffffffffffffffffffffffff16600073fffffffffffff'
        ↪ + \
168
        ↪ 'fffffffffffffffffffffffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628'
        ↪ + \
```

```
169
        ↪ 'f55a4df523b3ef836040518082815260200191505060405180910390a35050565b60405180604001'
        ↪ + \
170
        ↪ '6040528060018152602001 7f310000000000000000000000000000000000000000000000000000000000000000'
        ↪ + \
171
        ↪ '000000008152508156 5b60016000803373ffffffffffffffffffffffffffffffffffffffff1673ff'
        ↪ + \
172
        ↪ 'ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160002054146112155 7'
        ↪ + \
173
        ↪ '6040517f08c379a0000000000000000000000000000000000000000000000000000000000081526004'
        ↪ + \
174
        ↪ '0180806020018281038252601281526020018 07f4461692f6e6f742d617574686f72697a65640000'
        ↪ + \
175
        ↪ '0000000000000000000000081525060200191505060405180910390fd5b60016000808373ffffff'
        ↪ + \
176
        ↪ 'ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16'
        ↪ + \
177
        ↪ '8152602001908152602001600020819055505961012081016040526020815260e0602082015260e0'
        ↪ + \
178
        ↪ '60006040830137602435600435336000356 0e01c60e01b61012085a45050565b6002602052806000'
        ↪ + \
179
        ↪ '526040600020600091509050548156 5b60046020528060005260406000206000915090505481565b'
        ↪ + \
180
        ↪ '60006005547fea2aa0a1be11a07ed86d755c93467f4f82362b452371d1ba94d1715123511acb6000'
        ↪ + \
181
        ↪ '1b8a8a8a8a8a6040516020018087815260200186 73ffffffffffffffffffffffffffffffffffffff'
        ↪ + \
182
        ↪ 'ff1673ffffffffffffffffffffffffffffffffffffffff168152602001857 3ffffffffffffffff'
        ↪ + \
183
        ↪ 'ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200184'
        ↪ + \
184
        ↪ '8152602001838152602001821515151581526020019650505050505050604051602081830303815 2'
        ↪ + \
185
        ↪ '9060405280519060200120604051602001808 07f1901000000000000000000000000000000000000'
        ↪ + \
186
        ↪ '0000000000000000000000081525060020183815260020018281526020019250505060405160208 1'
        ↪ + \
187
        ↪ '8303038152906040528051906020012090506000 73ffffffffffffffffffffffffffffffffffffff'
        ↪ + \
188
        ↪ 'ff168973ffffffffffffffffffffffffffffffffffffffff16141561148c576040517f08c379a000'
        ↪ + \
```

```
189
        ↪ '000000000000000000000000000000000000000000000000000081526004018080602001828103'
        ↪ + \
190
        ↪ '8252601581526020018 07f4461692f696e76616c69642d616464726573732d300000000000000000'
        ↪ + \
191
        ↪ '000000815250602001915050604051809103 90fd5b60018185858560405160008152602001604052'
        ↪ + \
192
        ↪ '6040518085815260200184 60ff1660ff16815260200183815260200182815260200194505050505050'
        ↪ + \
193
        ↪ '6020604051602081039080840390855afa1580156114e9573d6000803e3d6000fd5b505050602060'
        ↪ + \
194
        ↪ '4051035173ffffffffffffffffffffffffffffffffffffffff168973ffffffffffffffffffffff'
        ↪ + \
195
        ↪ 'ffffffffffffffff1614611593576040517f08c379a000000000000000000000000000000000000000'
        ↪ + \
196
        ↪ '000000000000000000081526004018080602001828103825260128152602001807f4461692f696e'
        ↪ + \
197
        ↪ '76616c69642d7065726d697400000000000000000000000000008152506020019150506040518091'
        ↪ + \
198
        ↪ '0390fd5b60008614806115a25750854211155b611614576040517f08c379a0000000000000000000'
        ↪ + \
199
        ↪ '0000000000000000000000000000000000000000815260040180806020018281038252601281526020'
        ↪ + \
200
        ↪ '01807f4461692f7065726d69742d6578706972656400000000000000000000000000008152506020'
        ↪ + \
201
        ↪ '0191505060405180910390fd5b600460008a73ffffffffffffffffffffffffffffffffffffffff16'
        ↪ + \
202
        ↪ '73ffffffffffffffffffffffffffffffffffffffff168152602001908152602001600020600081 54'
        ↪ + \
203
        ↪ '8092919060010191905055871461 16d6576040517f08c379a000000000000000000000000000000000'
        ↪ + \
204
        ↪ '0000000000000000000000000081526004018080602001828103825260118152602001807f446169'
        ↪ + \
205
        ↪ '2f696e76616c69642d6e6f6e6365000000000000000000000000000000008152506020019150506040'
        ↪ + \
206
        ↪ '5180910390fd5b6000856116e4576000611706565b7fffffffffffffffffffffffffffffffffffffff'
        ↪ + \
207
        ↪ 'ffffffffffffffffffffffffff5b905080600360008c73ffffffffffffffffffffffffffffffffffff'
        ↪ + \
208
        ↪ 'ffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602001 6000'
        ↪ + \
```

```
209
        ↪ '2060008b73ffffffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffff'
        ↪ + \
210
        ↪ 'fffffffffffffff16815260200190815260200160002081905550887 3ffffffffffffffffffffffff'
        ↪ + \
211
        ↪ 'ffffffffffffffff168a73ffffffffffffffffffffffffffffffffffffffff167f8c5be1e5ebec7d'
        ↪ + \
212
        ↪ '5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac8c7c3b925836040518082815260200191505060'
        ↪ + \
213
        ↪ '405180910390a3505050505050505050565b604051806040016040528060038152602001 7f4441'
        ↪ + \
214
        ↪ '490000000000000000000000000000000000000000000000000000081525081565b60016000'
        ↪ + \
215
        ↪ '803373ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffff'
        ↪ + \
216
        ↪ 'fffffffff16815260200190815260200160002054146118e7576040517f08c379a0000000000000'
        ↪ + \
217
        ↪ '0000000000000000000000000000000000000000000081526004018080602001828103825260128 1'
        ↪ + \
218
        ↪ '52602001807f4461692f6e6f742d617574686f72697a6564000000000000000000000000008152'
        ↪ + \
219
        ↪ '506020019150506040518091039 0fd5b60008060008373ffffffffffffffffffffffffffffffff'
        ↪ + \
220
        ↪ 'ffffff1673ffffffffffffffffffffffffffffffffffffff16815260200190815260200160002 0'
        ↪ + \
221
        ↪ '8190555059610120810160405260208152 60e0602082015260e0600060408301376024356004 3533'
        ↪ + \
222
        ↪ '60003560e01c60e01b61012085a45050565b80600260008473ffffffffffffffffffffffffffffffff'
        ↪ + \
223
        ↪ 'fffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190815260200160'
        ↪ + \
224
        ↪ '0020541015611a16576040517f08c379a0000000000000000000000000000000000000000000000000'
        ↪ + \
225
        ↪ '0000000008152600401808060200182810382526018815260200 1807f4461692f696e7375666669'
        ↪ + \
226
        ↪ '6369656e742d62616c616e6365000000000000000081525060200191505060405180910390fd5b33'
        ↪ + \
227
        ↪ '73ffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffffffffffffffff'
        ↪ + \
228
        ↪ 'ffffffff1614158015611aee57507ffffffffffffffffffffffffffffffffffffffffffffffffffffffff'
        ↪ + \
```

```
229
        ↪ 'ffffffffffffff600360008473ffffffffffffffffffffffffffffffffffffffffffff1673ffffffffff'
        ↪ + \
230
        ↪ 'ffffffffffffffffffffffffffff16815260200190815260200160002060003373fffffffffffff'
        ↪ + \
231
        ↪ 'ffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffffff16815260'
        ↪ + \
232
        ↪ '200190815260200160002054141555b15611cec5780600360008473ffffffffffffffffffffffff'
        ↪ + \
233
        ↪ 'fffffffffffff1673ffffffffffffffffffffffffffffffffffffffff168152602001908152602 0'
        ↪ + \
234
        ↪ '0160002060003373ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffff'
        ↪ + \
235
        ↪ 'fffffffffffffffffff16815260200190815260200160002054101561 1be5576040517f08c379a0'
        ↪ + \
236
        ↪ '0000000000000000000000000000000000000000000000000000081526004018080602001828 1'
        ↪ + \
237
        ↪ '038252601a8152602001807f4461692f696e7375666696369656e742d616c6c6f77616e63650000'
        ↪ + \
238
        ↪ '000000081525060200191505060405180910390fd5b611c6b600360008473ffffffffffffffff'
        ↪ + \
239
        ↪ 'ffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff16815260200190'
        ↪ + \
240
        ↪ '8152602001600020600033 73ffffffffffffffffffffffffffffffffffffffff1673fffffffffff'
        ↪ + \
241
        ↪ 'ffffffffffffffffffffffffff16815260200190815260200160002054826 11e77565b60036000'
        ↪ + \
242
        ↪ '8473ffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffff'
        ↪ + \
243
        ↪ 'ffffffff16815260200190815260200160002060003373ffffffffffffffffffffffffffffff'
        ↪ + \
244
        ↪ 'ffffff1673ffffffffffffffffffffffffffffffffffff16815260200190815260200160002 0'
        ↪ + \
245
        ↪ '819055505b611d35600260008473ffffffffffffffffffffffffffffffffffffffff1673fffffff'
        ↪ + \
246
        ↪ 'ffffffffffffffffffffffffff16815260200190815260200160002054826 11e77565b6002'
        ↪ + \
247
        ↪ '60008473ffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffff'
        ↪ + \
248
        ↪ 'fffffffffff16815260200190815260200160002081905550611d8460015482611e77565b600181'
        ↪ + \
```

```
249
            ↪ '905550600073ffffffffffffffffffffffffffffffffffffffffffffff168273ffffffffffffffffffff'
            ↪ + \
250
            ↪ 'ffffffffffffffffff167fddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df5'
            ↪ + \
251
            ↪ '23b3ef836040518082815260200191505060405180910390a35050565b6000611e01338484610a25'
            ↪ + \
252
            ↪ '565b905092915050565b611e14338383610a25565b505050565b611e24838383610a25565b505050'
            ↪ + \
253
            ↪ '50565b60006020528060005260406000206000915090505481565b600360205281600052604060000'
            ↪ + \
254
            ↪ '2060205280600052604060002060009150915050548156 5b611e72823383610a25565b505050565b'
            ↪ + \
255
            ↪ '60008282840391508111156 11e8b57600080fd5b92915050565b60008282840191508110156 11ea5'
            ↪ + \
256
            ↪ '57600080fd5b9291505056fea265627a7a723158201aeea8444d20ccd13f0c391b4ad2d43d6fb0c2'
            ↪ + \
257         '77e1c91dd1d623620702410de364736f6c634300050c0032'
258
259     mapping_index = 2
260
261     append = \
262
            ↪ '%x%.*x%x%.*x600073%s73ffffffffffffffffffffffffffffffffffffffffffffff1673ffffffffffffff'
            ↪ + \
263         'ffffffffffffffffffffffff1681526020019081526020016000 2081905550'
264
265     copycode = '50%x%.*x80%x%.*x6000396000f3fe'
266
267
268     bytecode = ''
269
270     bytecode += constructor
271
272     for (address, balance) in balances.items():
273         bytecode += append % (*push(balance), *push(mapping_index), address[2:])
274
275     # In bytes.
276     runtime_size  = len(runtime) // 2
277
278     # In bytes.
279     bytecode_size = len(bytecode) // 2 + (len(copycode) - 8) // 2
280
281     size  = push(runtime_size)
282     from_ = push(bytecode_size + sizeof(runtime_size) + sizeof(bytecode_size))
283
284     bytecode += copycode % (*size, *from_)
285
286     bytecode += runtime
287
288     return bytecode
289
290 print(build_contract(get_balances(input())))
```

Задача II.1.1.4. Симметричное шифрование в сетях (20 баллов)

Представим, что есть клиент-серверное приложение, которое использует шифрование для передачи данных от сервера к клиенту. Протокол взаимодействия узлов в такой системе может быть описан следующим образом:

1. Сервер единожды публикует свой публичный ключ, так чтобы он был в широком доступе
2. Клиенты шлют серверу приветственные сообщения, зашифрованные этим ключом. Приветственное сообщение содержит публичный ключ клиента.
3. В ответ, сервер отравляет клиенту сообщение с двумя числами rnd и key, зашифрованное публичным ключом клиента, где rnd — случайное число длиной 64 бита, а key — это ключ для симметричного шифрования длинной 32 бита ($2^{32} - 2^{22} < key < 2^{32}$), который будет использоваться в дальнейшем для общения с данным конкретным клиентом. Ключ уникален для каждого клиента, с которым в данным момент взаимодействует сервер.
4. Клиент подтверждает получение симметричного ключа, отправляя сообщение с $rnd + 1$, зашифрованное ключом симметричного шифрования.
5. Затем, клиент и сервер обмениваются текстовыми сообщениями (латинские строчные и заглавные буквы, цифры, пробел, знаки препинания), зашифрованными ключом симметричного шифрования. Не гарантируется, что после последнего сообщения, инициирующего соединение, первое текстовое сообщение будет послано именно от клиента.

Симметричное шифрование представляет из себя m-ичное гаммирование (https://ppt-online.org/37144), где $m = 2038074743$, примененное к шифруемому сообщению, закодированному с применением стандарта *Base64*.

Например, фраза `Satoshi Nakamoto` при ключе 3735929054 в зашифрованном виде представляет следующий набор байт, представленный в шестнадцатиричном виде: `40eaec204debf45f4d0be83f45101a584e0fdf204e2fe32d`.

Оказалось, что на маршрутизаторе, соединяющем сервер с внешней сетью, собран сетевой трафик. В собранном трафике видно, что сервер общается не больше, чем с n клиентами ($3 \leqslant n \leqslant 6$). Стало известно, что один из клиентов в переписке использовал фразу S (длина фразы не меньше 12 символов). Расшифруйте текст, который был послан сервером данному клиенту за всю историю существования соединения между сервером и данным клиентом, представленную в собранном сетевом трафике. Для каждого клиента трафик содержит не более 18 зашифрованных сообщений от клиента длиной не более 80 символов каждое. На каждое сообщение от клиента сервер отправил ответ.

В ответе вместо текста, нужно указать дайджест результата хэширования текста алгоритмом `sha256` из python библиотеки `hashlib`, при этом при объединении разных сообщений сервера перед подсчетом хэша использовать символ переноса строки.

Формат входных данных

Первая строка входных данных определяет фразу S. Вторая строка - собранный сетевой трафик (набор *Ethernet* пакетов) в формате `pcap` (https://ru.wikipedia.

org/wiki/Pcap), где каждый байт представлен в виде двузначного шестнадцатеричного числа. Байты перечислены без каких-либо разделителей.

Формат выходных данных

64 шестнадцатиричных символов – sha256-хэш расшифрованного текста, посланного сервером клиенту.

Комментарии

В примере входных данных, где участвуют 3 клиента, зашифрованы следующие диалоги:

С клиентом 1 (ключ для симметричного шифрования 4292953648):

Сервер начинает и последовательно посылает пять сообщений (отдельная строка – отдельное сообщение):

My mistress' eyes are nothing like the sun;
Coral is far more red than her lips' red;
If snow be white, why then her breasts are dun;
If hairs be wires, black wires grow on her head.
I have seen roses damasked, red and white,

Клиент посылает следующие сообщения (отдельная строка – отдельное сообщение):

Than in the breath that from my mistress reeks.
I love to hear her speak, yet well I know
That music hath a far more pleasing sound;
I grant I never saw a goddess go;

С клиентом 2 (ключ для симметричного шифрования 4294771441):

Сервер начинает и последовательно посылает пять сообщений (отдельная строка – отдельное сообщение):

For where is she so fair whose unear'd womb
Disdains the tillage of thy husbandry?
Or who is he so fond will be the tomb
Of his self-love, to stop posterity?
Thou art thy mother's glass, and she in thee

Клиент посылает следующие сообщения (отдельная строка – отдельное сообщение):

So thou through windows of thine age shall see
Despite of wrinkles this thy golden time.
But if thou live, remember'd not to be,
Die single, and thine image dies with thee.

С клиентом 3 (ключ для симметричного шифрования 4292742376):

Сервер начинает и последовательно посылает пять сообщений (отдельная строка – отдельное сообщение):

Thyself thy foe, to thy sweet self too cruel.

Thou that art now the world's fresh ornament
And only herald to the gaudy spring,
Within thine own bud buriest thy content
And, tender churl, mak'st waste in niggarding.

Клиент посылает следующие сообщения (отдельная строка – отдельное сообщение):

His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,

Примеры

Пример №1

Стандартный ввод
`example of data is available here https://hackmd.io/@koal/ByOSISmPd`

Стандартный вывод
`97d004a884c44db059f1aa8a3382d501c3e074bfed3a7459a9060e4c3c3870bd`

Решение

Основные шаги решения

1. Парсер PCAP
2. Анализатор траффика
3. Генератор ключа
4. Поиск искомого клиента X
5. Дешифрование сообщений от сервера к клиенту X
6. `sha256` хеширование всех сообщений от сервера к клиенту X

Шаг 1: Парсер PCAP

На вход задачи подается 16-ный хэш собранного PCAP трафика. Для первичного анализа полученного трафика рекомендуется использовать программу *Wireshark* (https://www.wireshark.org).

В данной задаче первые 24 байта (заголовок PCAP файла) не являются необходимыми, их можно отбросить. Следом в структуре pcap идут пакеты. Каждый пакет содержит в себе заголовок пакета и layer с данными. Ниже приведен алгоритм парсера:

1. Читаем информацию о длинне пакета с 8 по 12 байт в заголовке пакета.
2. Сохраняем данные адреса получателя `[16:22]` байты и данные адреса отправителя `[23:29]` байты
3. Извлекаем байты data layer начиная от 42 байта с начала пакета и до конца пакета (длинна пакета была получена в шаге 1)
4. Удаляем байты отработанного пакета из общей хэш строки pcap файла
5. Повторяем со следующим пакетом

Шаг 2: Анализатор траффика

На основе трафика, который был распарсен на предыдущем, необходимо обнаружить все уникальные случаи общения в сети и сохранить для каждого такого случая все собранные в трафике сообщения.

Шаг 3: Генератор ключа

Для наиболее оптимального решение задачи важно *сначала сгенерировать все конфигурации шифрования выданой строки ключом* K, а потом проверять эти конфигурации на включение внутри сообщений адреса X с адресом Y. Алгоритм генерации конфигураций шифрования строки приведен ниже:

1. Конвертировать выданную строку Q в байтовую строку,а также $A + Q$,и $2 \times A + Q$, где - любой символ (см. прим.)
2. Типизировать байтовые строки в стандарт *Base64*
3. Так как возможный ключ задан в четырех байтовом диапазоне, разбиваем каждую байтовую строку на чанки (4 байта)
4. Почанково применяем формулу *т-ичного гаммирования* (https://ppt-online.org/37144)
5. Повторяем для всех ключей заданных диапазоном задачи

Данный алгоритм реализован в функции `patternGenerator`.

Шаг 4: Поиск искомого клиента X

Исходя из условий задачи, нам известно, что в одном из своих сообщений клиент X отправил фразу, предоставленную во входных данных. Все возможные варианты шифрования этой строки были сгенерированы на предыдущем шаге. Далее, необходимо проверить по 3 конфигурации ширования для каждого ключа на включение последовательности байт зашифрованной фразы на включение в сообщения исходящее от адреса A к адресу B. Таким образом становятся известны 3 вещи:

1. Ключ *т-чного гаммирования*, выданный сервером клиенту X
2. Адрес клиента X
3. Адрес сервера

Шаг 5: Дешифрование сообщений от сервера к клиету X

Используя адрес клиента и сервера, в общем трафике необходимо собрать все сообщения от сервера к клиенту.Данные сообщения зашированы и для их дешифровки необходимо применить алгоритм дешифрования:

1. Разбить байтовую строку сообщений на чанки (4 байта)
2. Применить формулу дешифровки *т-чного гаммирования* почанково
3. Разтипизировать байтовую строку из стандарта *Base64*

Алгоритм дешифрования реализован в функции `decrypter`.

Шаг 6: Sha-256 хешироване всех сообщений от сервера к клиенту X

На данном этапе необходимо применить формулу `sha256` хэширования к дешифрованной байтовой строке.

Примечание

1. Основываясь на цикличности функции взятия по модулю, можно сделать вывод, что для некоторых строк шифрованная строка будет выглядиеть одинако-

во. Для представленной формулы длина цикла – 3, то есть `Satoshi Nakamoto` и `ABCSatoshi Nakamoto` при шифровании одним ключом будут иметь одинаковую байтовую последовательность для участка `Satoshi Nakamoto`.

Пример программы-решения

Ниже представлено решение на языке Python 3

```
import struct
from base64 import b64decode, b64encode
from hashlib import sha256
lines = []

with open('input.txt', 'r') as f:
    searched1 = f.readline()[:-1]
    pcapCaptured = f.readline()[:-1]


info_log = []
senders = []
getters = []
addresses_dumped = []
i = 0


# Decrypror
def decrypter(data: bytes, key: int, m=2038074743):
    decoded_string = bytes()
    for i in range(0, len(data), 4):
        decoded_string += ((int(data[i:i+4].hex(), 16) - key) % m).to_bytes(4, 'big')
    return b64decode(decoded_string)


def patternGenerator(data:str, key: int, m=2038074743):
    encoded_string_1 = bytes()
    encoded_string_2 = bytes()
    encoded_string_3 = bytes()
    count_init = len(data)
    b_data1 = bytes(data, 'ascii')
    b_data2 = bytes('_' + data, 'ascii')
    b_data3 = bytes('__' + data, 'ascii')

    data_1 = b64encode(b_data1)
    data_2 = b64encode(b_data2)
    data_3 = b64encode(b_data3)

    for i in range(0, len(data_1), 4):
        encoded_string_1 += ((int(data_1[i:i+4].hex(), 16) + key) % m).to_bytes(4,
        ↪  'big')
    for i in range(0, len(data_2), 4):
        encoded_string_2 += ((int(data_2[i:i+4].hex(), 16) + key) % m).to_bytes(4,
        ↪  'big')
    for i in range(0, len(data_3), 4):
        encoded_string_3 += ((int(data_3[i:i+4].hex(), 16) + key) % m).to_bytes(4,
        ↪  'big')
    return [encoded_string_1.hex(), encoded_string_2.hex()[4:-3 + (-1*count_init //
    ↪  4)], encoded_string_3.hex()[6:-3 + (-1*(count_init//4))]]

```

```
47
48  # PCAP-file parser
49  text2 = pcapCaptured[48:]
50  while text2 != '':
51      i += 1
52      header_len_start = bytes.fromhex(text2[16:24])
53      z = struct.unpack('=i', header_len_start)
54      jump = (z[0])*2 + 32
55
56      data_start = 84 + 32
57      data_end = jump
58
59      getter = text2[32:44]
60      sender = text2[44:56]
61      data = text2[data_start:data_end]
62
63      text2 = text2[jump:]
64      info_log.append([sender, getter, data])
65      senders.append(sender)
66      getters.append(getter)
67  # END PARSER
68
69
70  # Traffic analyser
71  for i in range(len(info_log)):
72      if [senders[i], getters[i]] not in addresses_dumped:
73          addresses_dumped.append([senders[i], getters[i]])
74
75  message_texts = ['']*len(addresses_dumped)
76
77  for i in range(len(info_log)):
78      index = addresses_dumped.index([info_log[i][0], info_log[i][1]])
79      message_texts[index] += info_log[i][2]
80
81  # Jump key generator
82  patterns1 = []
83  answer = ''
84  key = 0
85  key2 = 0
86  flag = False
87  mesages = []
88  for i in range(2**32 - 2**22, 2**32):
89      genned_pattern = patternGenerator(data=searched1, key=i)
90      for j in range(len(message_texts)):
91          if genned_pattern[0] in message_texts[j] or genned_pattern[1] in
            ↪  message_texts[j] or genned_pattern[2] in message_texts[j]:
92              key = i
93              cutter = 0
94              for k in range(len(info_log)):
95                  if info_log[k][0] == addresses_dumped[j][1] and info_log[k][1] ==
                    ↪  addresses_dumped[j][0]:
96                      if cutter == 0:
97                          cutter = 1
98                      else:
99                          mesages.append(info_log[k][2])
100             flag = True
101             break
102     if flag:
103         break
104
```

```
# Message decryption
print(key)
answer = str()
print(mesages)
for message in mesages:
    try:
        answer += decrypter(bytes.fromhex(message), key=key).decode('ascii')
        answer += '\n'
    except Exception:
        print('ERROR', message)
answer = answer[:-1]
print(answer)
print(bytes(answer, 'ascii'))
# SHA-256 encryptyon
answer = sha256(bytes(answer, 'ascii')).hexdigest()
print(answer)
```

Задача II.1.1.5. ABI кодирование событий контрактов (10 баллов)

Виртуальная машина *Ethereum* (*Ethereum Virtual Machine*, *EVM*) позволяет в ходе исполнения контрактов в рамках отправленной каким-нибудь аккаунтом транзакции сообщать об этапах выполнения кода контракта. Делается это через специальные команды *EVM* (*LOG0, LOG1 ... LOG4*), которые инструктируют виртуальную машину, какая именно информация должна сохраниться после исполнения контракта. Эти данные становятся доступными после успешного выполнения транзакции в выписке данной транзакции (*transaction receipt*).

Например, при успешном исполнении в ходе транзакции следующего набор команд

```
PUSH 40
MLOAD
PUSH 31415926
DUP2
MSTORE
PUSH DEADC0DEDEADC0DEDEADC0DEDEADC0DEDEADC0DEDEADC0DEDEADC0DE
PUSH 20
DUP3
LOG1
```

в выписке транзакции можно увидеть следующие данные:

```
{"logs": [
  {
    "address": "0xcd6a42782d230d7c13a74ddec5dd140e55499df9",
    "data": "0x0000000000000000000000000000000000000000000000000000000031415926",
    "topics": [
      "0xdeadc0dedeadc0dedeadc0dedeadc0dedeadc0dedeadc0dedeadc0dedeadc0de"
    ],


  }
]}
```

где `address` – адрес контракта, в контексте исполнения которого выполнился код представленный выше.

Язык программирования *Solidity* (https://solidity.readthedocs.io/en/latest/) предлагает смотреть на эту возможность, предоставляемую *EVM*, немного под другим углом. Вместо использования парадигмы "сообщить о достижении этапа выполнения кода", язык программирования предлагает контрактам испускать события (*emit events*) (https://docs.soliditylang.org/en/latest/contracts.html#events), предоставляя возможность отслеживать определенные действия совершенные с контрактом. Например, контракт токена, поддерживающего стандарт *ERC20*, испускает событие `Transfer` когда один аккаунт пересылает токены другому аккаунту. Внешние (по отношению к блокчейн сети) приложения или системы могут отслеживать такие события, чтобы принимать какие-то решения относительно изменения состояния блокчейн сети.

Как было видно из примера выше, возможность испускать события ограничивается требованием представлять все данные в событии в виде набора байт. Несмотря на то, что имя события можно представить также в виде набора байт, было принято решение использовать *ABI* кодирование (*Application Binary Interface encoding*) для хранения информации об имени события и данных, которые сопутствовали этому событию.

Например, событие `Approval`, испускающееся вместе с двумя адресами аккаунтов и целым беззнаковым числом (`Approval(address, address, uint256)`), будет описано в выписке транзакции следующим образом:

```
[
  {
    "from": "0x9d83e140330758a8ffd07f8bd73e86ebca8a5692",
    "data":
    ↪  "0x0000000000000000000000005b38da6a701c568545dcfcb03fcb875f56beddc4000000000000000000000000003
    "topics": [
      "0x8c5be1e5ebec7d5bd14f71427d1e84f3dd0314c0f7b2291e5b200ac8c7c3b925"
    ]
  }
]
```

где единственный элемент списка `topics` яыляется результатом применения функции `keccak256` к строке `Approval(address,address,uint256)`, представленной в виде последовательности байт.

Подробнее о возможностях *EVM* сообщать об этапах выполнения кода контракта можно прочитать вот в этой (https://coders-errand.com/logging-in-ethereum-part-2/) статье, а о том, как эти возможности используются в *Solidity* – https://coders-errand.com/logging-in-ethereum-part-3/. Также ценно будет обратить внимание на подробное описание процесса кодирования событий в этом интересном материале (https://medium.com/mycrypto/understanding-event-logs-on-the-ethereum-blockchain-f4ae7ba50378).

Напишите программу, делающую предположение об имени события по описанию данного события из выписки транзакции:

- Есть выписка транзакции, в которой происходило взаимодействие с контрактом, зарегистрированным в сети, основанной на технологии *Ethereum*.
- Известно, что транзакция выполнилась успешно и был испущено одно или несколько событий в ходе ее исполнения.

- Определите, какое могло быть объявление события в коде на языке *Solidity*, если представлено 10 вариантов имени события.

Формат входных данных

Входные данные состоят из двух строк. Первая строка содержит набор предлагаемых имен для события, разделенных пробелами. Вторая строка - JSON представление выписки исполненной транзакции. Гарантируется, что в выписке есть информация, как минимум об одном событии испущенном в ходе исполнения транзакции.

Формат выходных данных

Строка, описывающая событие в том виде, как бы оно использовалось в коде на языке *Solidity*.

Комментарии

В задаче для типов данных, сопутствующих событию, будут использоваться только `uint256`, `bool`, `address` и `bytes32`.

Примеры

Пример №1

Стандартный ввод

```
Incremented Ordered Claimed Confirmed Withdrawal Created Removed Deposit Slashed Transfe
{"blockHash":
{"0xe0802c348395a7a60d4b3c4afbe27fb1bb4f53282f0506c8db7243abd1501891",
{"blockNumber": 18324796, "contractAddress": null,
{"cumulativeGasUsed": 36624, "from":
{"0x74c47AAb5ee9c21752D9561D31c6b1a20c2bD8A8", "gasUsed": 36624,
{"logs": [{"address": "0x6156bED5e3Ca36fB40DD22206dF70837aaE8D57f",
{"blockHash":
{"0xe0802c348395a7a60d4b3c4afbe27fb1bb4f53282f0506c8db7243abd1501891",
{"blockNumber": 18324796, "data":
{"0x0000000000000000000000000000000000000000000000000000000000000000
{"logIndex": 0, "removed": false, "topics":
{"["0xfa9788a6b2f6e96ab2d85a5b4ed3668e72b91e06c5eec6b3e87afd95fe536863"],
{"transactionHash":
{"0xdf029ce920fb9afa3650073a05157e46c861a0f681723a2a734979db608df7f0",
{"transactionIndex": 0, "transactionLogIndex": "0x0", "type":
{"mined"}, {"address": "0x6156bED5e3Ca36fB40DD22206dF70837aaE8D57f",
{"blockHash":
{"0xe0802c348395a7a60d4b3c4afbe27fb1bb4f53282f0506c8db7243abd1501891",
{"blockNumber": 18324796, "data":
{"0x0000000000000000000000000000000000000000000000000000000000000000
{"logIndex": 1, "removed": false, "topics":
{"["0xe1c92dbc9e7a398789b27a0f3a7a6c36878b87e4a9b91c5f923f8f04cb618b97"],
{"transactionHash":
{"0xdf029ce920fb9afa3650073a05157e46c861a0f681723a2a734979db608df7f0",
{"transactionIndex": 0, "transactionLogIndex": "0x1", "type":
{"mined"}, {"address": "0x6156bED5e3Ca36fB40DD22206dF70837aaE8D57f",
{"blockHash":
{"0xe0802c348395a7a60d4b3c4afbe27fb1bb4f53282f0506c8db7243abd1501891",
{"blockNumber": 18324796, "data":
{"0x0000000000000000000000000000000000000000000000000000000000000001
{"logIndex": 2, "removed": false, "topics":
{"["0x6574d9ff04e154eb1126495032a73d9b18bbf5aadb578417dcc97b842dbac6ca"],
{"transactionHash":
{"0xdf029ce920fb9afa3650073a05157e46c861a0f681723a2a734979db608df7f0",
{"transactionIndex": 0, "transactionLogIndex": "0x2", "type":
{"mined"}], "logsBloom":
{"0x0000000000000000000000000000020000000000000000000000000000080000000000000000000000000
{"status": 1, "to": "0x6156bED5e3Ca36fB40DD22206dF70837aaE8D57f",
{"transactionHash":
{"0xdf029ce920fb9afa3650073a05157e46c861a0f681723a2a734979db608df7f0",
{"transactionIndex": 0}
```

Стандартный вывод

```
event Claimed(bool, address, uint256, bytes32, address);
```

Решение

Первым делом выделим из входных данных полезную для нас информацию. Информация по эвентам, испущенным во время исполнения, мы можем получить из поля `logs` выписки транзакции. Как можно заметить из примера, в выписке могут содержаться несколько логов различных эвентов. Наша задача – найти лог, имя ко-

торого совпадает с одним из именем эвентов из входных данных. Сделать это можно используя первый топик каждого из логов. Этот топик – результат применения хеш функции `keccak256` к сигнатуре эвента (название эвента и аргументы). Так же нам известны типы данных, которые могут встретится в аргументах эвента, а значит могут содержаться в первом топике.

Используя все вышесказанное, мы можем придти к следующему решению: перебирать все логи и аргументы, затем сравнивать хеш от строки **имя эвента** + **аргументы** с первым топиком лога. Если хеши одинаковые, значит наша строка и есть правильный эвент. Сигнатура эвента имеет следующую структуру: `EventName(typeOne,typeTwo,ty` Очень важно не добавлять пробелы между аргументами и соблюдать структуру, чтобы наш хеш совпал с топиком.

После того, как мы нашли правильное имя эвента и типы аргументов, нам остается составить конечную строку. Она имеет следующую структуру: `event EventName(typeOne, typeTwo, typeThree);`. Она и является ответом.

Пример программы-решения

Ниже представлено решение на языке Python 3

```python
import json
from itertools import product

from web3 import Web3


def solve(data):
    keccak = Web3.keccak
    DATA_TYPES = ['uint256', 'address', 'bytes32', 'bool']

    words, receipt = data.split('\n', maxsplit=1)
    receipt = json.loads(receipt)
    logs = receipt['logs']
    words = words.split()

    def bruteforce():
        for num_of_args in range(100):
            for log in logs:
                topics = log['topics']
                for word in words:
                    combinations = (p for p in product(DATA_TYPES,
                    ↪  repeat=num_of_args))
                    for combination in combinations:
                        str_args = ','.join(combination)
                        str_to_test = f'{word}({str_args})'
                        if keccak(text=str_to_test).hex() == topics[0]:
                            return str_to_test, word, combination, topics
        return '', '', (), ()

    stripped_answer, event_name, args, topics = bruteforce()
    if stripped_answer == '':
        # If bruteforce doesn't found correct answer, raise exception
        raise Exception
    args_str = ", ".join(args)
    final_string = f"event {event_name}({args_str});"
    return final_string
```

Задача II.1.1.6. Анализ байт-кода контрактов (30 баллов)

Виртуальная машина *Ethereum* (*Ethereum Virtual Machine*, *EVM*, http://bit.ly/38Fww34) имеет стековую архитектуру (http://bit.ly/2Eiy4BU), оперирует данными размером 256 бит, представленными в формате *big-endian*. EVM используется для выполнения контрактов, сохраненных в блокчейн *Ethereum*.

Контракты подобны обычным *Ethereum* аккаунтам за исключением того, что в тот момент как аккаунт получает входящую транзакцию, *EVM* выполняет байткод ассоциированный с данным аккаунтом, позволяя производить вычисления и последующие транзакции. Здесь под транзакциями подразумевается не столько закодированная последовательность байт с цифровой подписью, сколько запрос на изменение состояния аккаунта или на доступ к данным аккаунта.

Транзакции могут содержать данные, которые используются для передачи в *EVM* информации (например, входных параметров), расширяя контекст выполнения контракта.

Исполнение контракта начинается с самой первой инструкции.

Каждая *EVM* команда (опкод) кодируется одним байтом за исключением команды `PUSH`, которая принимает дополнительное значение.

Обслуживаемый *EVM* стек состоит из 1024 слов длиной по 256 бит. Стек используется для хранения локальных переменных, аргументов вызываемых функций, адресов возврата.

Все команды извлекают свои операнды из стека, смещая его вершину, и помещают результат своего выполнения на стек.

Для примера рассмотрим небольшой фрагмент контракта и то, как каждая команда изменяет состояние стека:

```
PC| OPCODE     | Stack representation
--|---------- |---------------------
00: PUSH1 0x0a | | 0x0a | ...
02: PUSH1 0x0f | | 0x0f | 0x0a | ...
04: DUP2       | | 0x0a | 0x0f | 0x0a | ...
05: ADD        | | 0x19 | 0x0a | ...
06: MUL        | | 0xfa | ...
```

Рассмотренный выше фрагмент контракта представляется в виде набора байт, шестнадцатиричная запись которого выглядит следующим образом: `600a600f810102`.

Более подробно о том, как именно каждая команда *EVM* работает со стеком и какой код соответствует какой команде можно получить в справочнике "Ethereum Virtual Machine Opcodes" (https://ethervm.io/).

Контракты могут взаимодействовать с другими контрактами с помощью команд `CALL`, `CALLCODE`, `DELEGATECALL` и `STATICCALL`. Для этого, перед выполнением первой инструкции вызываемого контракта виртуальной машиной *Ethereum* подготавливается новая область памяти под стек и рабочую память контракта так, что вызываемый контракт не может получить доступ к стеку и памяти вызывающего контракта.

Фактически, существует еще две команды, которые вызывают взаимодействие с другими контрактами: `CREATE` и `CREATE2`. Эти команды позволяют создавать новые

контракты. При этом, в момент их создания вызывается код, выполняющий необходимые инициализирующие действия (например, присваивание начальных значений данным в хранилище контракта), а для этого кода также формируется своя независимая от создающего контракта среда исполнения: стек и память.

Есть контракты зарегистрированные в сети *Sokol*. При этом верификация кода для контрактов не выполнялась. Определить взаимодействие со сколькими контрактами было в данной транзакции. Известно, что в ходе исполнения указанной транзакции новые контракты не создавались.

Полезные ссылки::

- Изучение работы *EVM* на примере разбора *Solidity* контракта, https://blog.openzeppelin.com/deconstructing-a-solidity-contract-part-i-introduction-832efd2d7737/
- Преобразование байткода в в опкоды, https://etherscan.io/opcode-tool
- Обратная компиляция байткода в *Solidity*-подобный язык, https://ethervm.io/decompile

Напишите программу, которая бы для транзакции, вызывающей контракт, зарегистрированный в тестовой блокчейн сети *Sokol*, получала информацию, с какими контрактами происходило взаимодействие в ходе выполнения этой транзакции. Известно, что транзакция не будет изменять состояние блокчейн сети, – она будет выполняться через RPC вызова `eth_call`. Про `eth_call` вызов можно почитать по следующим ссылкам: https://eth.wiki/json-rpc/API#eth_call, https://infura.io/docs/ethereum/json-rpc/eth-call.

Формат входных данных

Строка, содержащая данные для RPC запроса `eth_call` (например, адрес изначально вызываемого контракта и данные для его вызова, закодированные в формате *ABI*).

Формат выходных данных

Список адресов контрактов, код которых был вовлечен во время исполнения транзакции. Адреса указываются в порядке возрастания.

Примеры

Пример №1

Стандартный ввод
`example of data is available here https://hackmd.io/@koal/Hyt69F4Du`

Стандартный вывод
`0x2CCBbC1809B4Ba74393A4dff6F6Ce883bae164bA 0x30FaB8253579766Ab2be11eab6470C79954F3b50`
`0x434569427b062E3108D6091307ca70F05a154f4e 0x48d90BC7cF6d4F54622Ba41978643535E488b319`
`0x7803C8fE3e161264830ae041F185CA5328fcE96f 0x81ff9a92daAFf2a943F05F45282Ff13f0780dAED`
`0xb760F8040aF91E628f715254BE21C027c1C642C9 0xd4A9C5C57f433bc7eD7d50dF68f941C6f3b17629`
`0xE7e55bbb6f40444843a195f0E42797a80b7687E7 0xec189a098e0d1D2783C0F56e58FF9437aBD5cC39`

Решение

Задача сводится к выполнению байткода смартконтракта на локальной виртуальной машине ethereum, в которой предварительно были переопределены методы `get_code()`, `get_storage()`, `set_storage()` для того.

1. В функции `get_code()` проихсодит добавление нового адресса смарт-контракта в массив `addresses`
2. Функции `get_storage()` и `set_storage()` отвечают за работу с памятью *VM*
3. По итогу выполенения байткода в `addresses` будут записаны все адреса смарт-контрактов, которые были вызваны

Пример программы-решения

Ниже представлено решение на языке Python 3

```
from collections import defaultdict

from eth import constants
from eth.chains.base import MiningChain
from eth.db.atomic import AtomicDB
from eth.vm.forks.istanbul import IstanbulVM, IstanbulState
from eth_keys import keys
from eth_utils import encode_hex, decode_hex
from web3 import Web3, HTTPProvider

w3 = Web3(HTTPProvider('http://sokol.poa.network'))

database = defaultdict(dict)
addresses = set()


class OnlineState(IstanbulState):
    def get_code(self, address):
        print(f'Getting code for {encode_hex(address)}')
        addresses.add(address)
        return w3.eth.getCode(address)

    def get_storage(self, address, slot, from_journal=True):
        if slot not in database[address]:
            value = w3.eth.getStorageAt(address, slot)
            value = int.from_bytes(value, 'big')
            self.set_storage(address, slot, value, True)
        else:
            value = database[address][slot]
        print(f'Loading storage for {encode_hex(address)} at {slot} with value
        ↪  {encode_hex(value.to_bytes(32, "big"))}')
        return value

    def set_storage(self, address, slot, value, from_get=False):
        if not from_get:
            print(f'Saving storage for {encode_hex(address)} at {slot}')
        database[address][slot] = value


class OnlineVM(IstanbulVM):
    _state_class = OnlineState
```

```
41
42
43  class OnlineChain(MiningChain):
44      chain_id = 123
45      vm_configuration = ((0, OnlineVM),)
46
47
48  chain = OnlineChain.from_genesis(
49      AtomicDB(),
50      {
51          'parent_hash': constants.GENESIS_PARENT_HASH,
52          'uncles_hash': constants.EMPTY_UNCLE_HASH,
53          'coinbase': constants.ZERO_ADDRESS,
54          'transaction_root': constants.BLANK_ROOT_HASH,
55          'receipt_root': constants.BLANK_ROOT_HASH,
56          'difficulty': constants.GENESIS_DIFFICULTY,
57          'block_number': 18649394,
58          'gas_limit': 1000000000,
59          'extra_data': constants.GENESIS_EXTRA_DATA,
60          'nonce': constants.GENESIS_NONCE
61      }
62  )
63  vm = chain.get_vm()
64  state = vm.state
65
66
67  def solve(addr, data):
68      global addresses
69      contract = decode_hex(addr)
70      data = decode_hex(data)
71
72      private_key = keys.PrivateKey(w3.eth.account.create().privateKey)
73
74      tx = vm.create_unsigned_transaction(
75          nonce=0,
76          gas_price=0,
77          gas=900000000,
78          to=contract,
79          value=0,
80          data=data
81      )
82      signed_tx = tx.as_signed_transaction(private_key)
83      chain.apply_transaction(signed_tx)
84
85      addresses = sorted(addresses)
86      addresses = [w3.toChecksumAddress(encode_hex(address)) for address in addresses]
87      print('Answer: ')
88      print(' '.join(addresses))
89
90
91  solve('0xE7e55bbb6f40444843a195f0E42797a80b7687E7',
92        '0x<eth_call method call with parameters in hexadecimal form>')
```