Coding Challenge

Introduction

Summitto is pioneering a triple entry accounting system, which will catalyze the creation of new markets, and increase efficiency of numerous governmental and business activities throughout the world over the next decade. In order to do this, we are building a blockchain protocol with a very strong focus on confidentiality. Though we are building a product which will be used by governmental services, we are a fully technology focused company.

This is why we'd like to give you this coding challenge. We want you to add new functionality to the Monero source code. The purpose of this coding challenge is to assess:

- Your ability to solve problems through simple yet effective code
- Your understanding of modern C++ and its intricacies

Background

Blockchains are distributed databases in which each node has a local copy of the entire state. A blockchain is a linked list of blocks, each of which contains a list of transactions. Creation of a new block is called "mining". Transactions in these blocks specify at the very least three elements: 1) old transaction outputs to spend 2) new transaction outputs to create 3) cryptographic signatures. In the cryptocurrency Monero, a transaction can *optionally* contain a payment ID of either 32 unencrypted bytes or 8 encrypted bytes.[1] We want you to adjust the code to allow data of 32 bytes to be encrypted. Important to understand is that all of the functionality you need to use is already available in the official code. That means you just need to identify how it works today with the encrypted 8 bytes and extend it to 32 bytes.

In Monero, an important source of anonymity is the so called 32-byte "shared secret". This shared secret is established using the Diffie-Hellman algorithm, and allows you to encrypt and decrypt 32 bytes of data by using it as a one-time pad. The shared secret enables the sender to for example reveal amounts in a transaction only to the receiver, without requiring interaction. This same shared secret is used to encrypt the payment ID. In order to use transactions for VAT registrations, each transaction should contain a unique reference to a corresponding invoice. The payment ID data field plays a perfect role for this functionality.

The Monero codebase can be difficult to work through. However, at the same time, this software contains some of the most high quality encryption primitives in the world. Whilst summitto's own libraries contain only well-documented C++17 code, we're keen to evaluate your ability to quickly ground yourself in this interesting but rough codebase.

[1] https://getmonero.org/resources/moneropedia/paymentid.html

Part 1

The end-result of this challenge is to allow a user to create a transaction which contains a unique reference to a corresponding invoice. You are free to use whichever C++-version you are most comfortable with, but we prefer to see modern features. You will:

- Take a simple invoice document in pdf format.[2]
- Create a new executable in the Monero codebase, re-using cryptographic functions directly from Monero. The source code of your new executable can for example be situated in *monero/src/summitto* . When you run the program, it should:
    - take the path of a pdf (or any other file) as input
    - create a 32-byte hash from the pdf
    - truncate the hash to 26-bytes
    - ask a user for a decimal number of at most 6 bytes through the CLI.
    - Convert this number to an appropriate type and append it to the hash to create a 32-byte value
- Create unit tests for your new executable similar in scope to those in the monero/tests/unit_tests folder.
- Start a monerod process and two monero-wallet-cli processes (see page 4 for info)
- Mine 100 blocks using the curl command at the bottom of this document
- Send a transaction from one wallet to the other using the "transfer_split" command, including the 32-byte value in the payment id field. Ensure that 32-byte payment id's are always encrypted (only the short version is encrypted in the official client). Do not create an integrated address. Using a payment id does not require an integrated address, normal addresses work just fine.
- Mine 1 block
- Adjust the Monero wallet to decrypt the received 32-byte payment id
- Write a max. 1-page document explaining your design choices. Please add written explanation of how to evaluate whether the appended hash is successfully encrypted by the sender and successfully decrypted by the receiver.[3]

Part 2

Explain whether there exists an alternative secure way in which you can use the 32 byte shared secret to encrypt both the 32-byte invoice hash and 6-byte VAT number, and if so, why you should or shouldn't use this new method.

[2] E.g. https://www.getharvest.com/downloads/Invoice_Template.pdf

[3] Logs can be activated in the daemon and wallet rpc using the set_log command.

Prerequisites

The challenge has been tested on macOS Sierra and ubuntu 18.04, using clang 7.0.0. It may work on other systems too, but if you encounter issues we recommend these compiler versions.

If you have persistent compiler issues or you have trouble setting up a virtual environment[4] we can send you a virtualbox image instead (nick/password: summitto).

Compiling:
You will get an account on our git server, the password is: summitto.
Clone the challenge repository: https://challenge.summitto.com/summitto/challenge
Follow the steps to install dependencies and to compile from source in the README.

A note on libgtest-dev on ubuntu
If you encounter compiler problems due to gTest please remove the pre-installed version from your system or virtual machine and recompile. Here is an example command for ubuntu:

```
sudo apt remove libgtest-dev && make clean
```

Results
You can push your results to the challenge_result repository in our git server.

[4] We recommend 5 GB memory and 12 GB storage

Useful commands

The following commands may help you run your local testnet wallets and nodes. Most commands are one-liners, you may need to copy them into a text editor to ensure they are placed in one line.

You can start an ***offline*** local node as follows.

```
mkdir ~/wallets

./monerod --regtest --fixed-difficulty=1 --p2p-bind-port 20170
--rpc-bind-port 20171 --no-igd --hide-my-port --log-level 0
--p2p-bind-ip 127.0.0.1 --log-file  ~/wallets/node_01.log --data-dir
~/wallets/node_01 --zmq-rpc-bind-ip 127.0.0.1  --zmq-rpc-bind-port 20172
--offline
```

Create two wallets. Make sure to exit the processes after the wallets are generated:

```
./monero-wallet-cli --daemon-port 20171 --generate-new-wallet
~/wallets/wallet_01.bin  --restore-deterministic-wallet
--electrum-seed="sequence atlas unveil summon pebbles tuesday beer
rudely snake rockets different fuselage woven tagged bested dented vegan
hover rapid fawns obvious muppet randomly seasons randomly" --password
"" --log-file ~/wallets/wallet_01.log;

./monero-wallet-cli --daemon-port 20171 --generate-new-wallet
~/wallets/wallet_02.bin  --restore-deterministic-wallet
--electrum-seed="deftly large tirade gumball android leech sidekick
opened iguana voice gels focus poaching itches network espionage much
jailed vaults winter oatmeal eleven science siren winter" --password ""
--log-file ~/wallets/wallet_02.log;
```

You can start wallet RPC servers as follows (in two separate terminals):

```
./monero-wallet-rpc --daemon-port 20171 --rpc-bind-port 20173
--disable-rpc-login --wallet-file ~/wallets/wallet_01.bin --password ""
--log-file ~/wallets/wallet_01.log --no-dns

./monero-wallet-rpc --daemon-port 20171 --rpc-bind-port 20183
--disable-rpc-login --wallet-file ~/wallets/wallet_02.bin --password ""
--log-file  ~/wallets/wallet_02.log --no-dns
```

Start mining blocks

This command will generate 100 blocks for you and send the miners fee to the address provided. This allows you to get coins to send to other wallets. If you want to increase the balance of your wallet you can run this repeatedly. If you “mine” blocks, which is how you get coins in the first place, they are “locked” for a certain number of blocks. This is why we require you to mine a decent amount before you can transfer funds.

```
curl -X POST http://127.0.0.1:20171/json_rpc -d
'{"jsonrpc":"2.0","id":"0","method":"generateblocks","params":{"amount_o
f_blocks": 100, "wallet_address":
"46PAiPrNjr2XS82k2ovp5EUYLzBt9pYNW2LXUFsZiv8S3Mt21FZ5qQaAroko1enzw3eGr9q
C7X1D7Geoo2RrAotYPvzt9vB"}}' -H 'Content-Type: application/json'
```

Send a transaction

You can send a transaction as follows:

```
curl -X POST http://127.0.0.1:20173/json_rpc -d
'{"jsonrpc":"2.0","id":"0","method":"transfer_split","params":{"destinat
ions":[{"amount":100,"address":"46HZen5tbbPZiLn66S3Qzv8QfmtcwkdXgM5cWGsX
APxoQeMQ79md51PLPCijvzk1iHbuHi91pws5B7iajTX9KTtJ4aFuHcv"}],"account_inde
x":0,"subaddr_indices":[0],"priority":0,"ring_size":11,"get_tx_key":
true, "payment_id":
"1234567812345678123456781234567812345678123456781234567812345678"}}' -H
'Content-Type: application/json'
```

Get transfers

You can get transfers as follows:

```
curl -X POST http://127.0.0.1:20173/json_rpc -d
'{"jsonrpc":"2.0","id":"0","method":"get_transfers","params":{"in":true,
"out":true}}' -H 'Content-Type: application/json'
```