IIC3272 — Criptomonedas y contratos inteligentes

Homework 1: explanation and projects

1. Administrative stuff

You should form groups of three people each. Your group will select one of the projects detailed below, investigate the topic, write a short report on it, *and* give a 30 minute presentation on the topic. The presentations can be recorded and shared via dropbox/google drive, or we can agree on date/time and do a Zoom session if you prefer it that way.

Deadline to hand in your work: Monday, June 1st, 2020.

2. FAQ

Do we need to hand in the work we do?

Yes, you need to write a short report detailing your findings. If you did some implementation, a link to the repository containing it should be provided.

What should be the structure of my presentation?

You have 30 minutes, so there is sufficient time to quickly explain the setting and background, and then detail the specific problem that is tackled, and how is this resolved.

We have a size 9 font in our presentation so that all the information fits on the slides. Is this OK?

No! An important part of the evaluation is your ability to summarize the work you did in a short presentation. This is no easy task. In particular, a good short presentation requires better understanding of the topic than a bad long presentation. In this course you will need to be able to isolate the important details of your work and present it in an accessible way.

I don't like any of the proposed topics.

Propose a new topic. For this, send me an email (dvrgoc@ing.puc.cl).

We're stuck, we don't know how to proceed, or can not decide what to do next.

That's what I'm here for. Send an email, or arrange a consultation time! Also, some projects have a lot of work in them, so if you do not finish all of it you still might get the highest score if sufficient effort is put into it.

3. Projects

A representative of the group should send me an email ccing the other people in the group with two preferences for a project they would like to do. It is recommendable that you do this as soon as possible, in order to get the topic you wish to work on. Note that one topic can be covered by two groups, however, in this case you should put some spin on the problem you study (e.g. group one studies the MD5 hash function, and group two sha512). Also, if too many groups select the same topic, the ones that announced their preference last will be nudged in another direction.

Bellow follows a list of proposed projects.

3.1. Historians of Bitcoin

Explore the history of Bitcoin, the ideas that led to Bitcoin, and previous payment systems in detail. A good starting point for this is to follow up on the references in the Bitcoin book (preface). Another excellent reference is the paper `https://cacm.acm.org/magazines/2017/12/223058-bitcoins-academic-pedigree/fulltext`. Also, review early Bitcoin drafts, see what features have been excluded in the actual version, a provide a list of failed implementation ideas in Bitcoin.

Since several groups can take the same project, they will be required to cover different topics. For instance, one group can explore the history of Bitcoin itself, another one can study some previous attempts to define cryptocurrencies, and explain why they failed.

3.2. Hashing

Study how one particular hash function is implemented, and what makes it secure. For this, you can use something relatively complicated as SHA256, or an easier one such as MD5. In both cases you should be able to explain what makes the function secure, and discuss which theoretical weaknesses they have.

Reading material for SHA256:

- SHA-256 formal specification(NIST): `https://csrc.nist.gov/publications/detail/fips/180/4/final`
- SHA2 Wiki: `https://en.wikipedia.org/wiki/SHA-2`
- A nice resource: `https://web.archive.org/web/20160330153520/http://www.staff.science.uu.nl/~werkh108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf`

Note that more than one group can take up this project, in which case I will ask you to select different hash functions to study. I.e. one group could take SHA256, and the other one MD5, ripemd160, or anything similar.

3.3. Math behind elliptic curve cryptography

In class we presented the bare bones of elliptic curve digital signature algorithms, however, we did not dwell deep into the math of elliptic curves. In this project you are asked to do precisely that.

There is a number of subjects to cover here: from why the point at infinity has a natural interpretation in projective geometry, to bounding the number of points on an elliptic curve over a finite field (also known as Hasse's theorem). Similarly, you might dwell into how to use elliptic curves to test whether a number is prime, or how to factorize numbers using Pollard's rho method, and Lenstra's algorithms.

All of these topics are covered in chapter 6 of the book: *A Course in Number Theory and Cryptography* by Neal Koblitz. In case you decide to take the topic we will make the book available to you (also, you can try entering the book's name followed by 'pdf' into a search engine and see what pops up).

3.4. Breaking the Discrete Log problem

When describing elliptic curve cryptography, we said that the security is based on the assumption that solving the discrete log problem is an exponential time algorithm, which basically checks all the possible solutions. However, there are some options that can significantly reduce the search space. A good list of such algorithms can be found (in the section called Algorithms) at: `https://en.wikipedia.org/wiki/Discrete_logarithm`.

In this project you should select at least three of these algorithms, explain in detail how, why, and when they work, and, if possible, do some experimentation on breaking the problem in some simple group.

3.5. Bitcoin wallets

Study how Bitcoin wallets operate. Explain what are hierarchical deterministic wallets. Explain seeds and mnemonic words. Also explain how these are implemented (or implement parts of one yourself).

Reading material:

- Keys, addresses, ECC basics: `https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch04.asciidoc`
- Wallets, HD wallets, mnemonics: `https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch05.asciidoc`
- HD wallets: `https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki`
- Mnemonic seeds: `https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki`

3.6. Scroogecoin galore

Implement complete Scroogecoin, including a web page, or a client (i.e. in Python, or just in terminal) simulating Scrooge. Note that you will have to tidy up our basic implementation, and link all the parts together.

Scrooge will be publishing the blockchain with all the transactions that occurred. The main novelty here is that Scrooge will not publish a single transaction per block, but will group up to five transactions in a single block. For this, Scrooge will need to maintain a pool of new transactions that arrive, and make sure that there are no inconsistencies. Scrooge should also be able to add two transactions A and B in the same block, such that A is executed first, and then B spends A's outputs.

Your implementation will need to have an active UTXO pool, and an active new_transaction pool. Assume that Scrooge will maintain the blockchain, and publish a new block: (i) for each 5 new transactions; or (ii) every 2 minutes. Once you implement this, have various actors (at least 3) send transactions (including Scrooge who creates the coins). Test how your transaction pool and UTXO pool handle higher loads in terms of verification (i.e. 50 transactions with multiple signatures).

3.7. Bitcoin Core source code

If you really want to get into dirty details of Bitcoin's implementation, the only real place to do this is in the source code of the most widely used Bitcoin client. We are talking here about *Bitcoin Core*, and it's github is available at: `https://github.com/bitcoin/bitcoin`.

For this project, you should familiarize yourselves with the code, and study how some aspects that we described and/or implemented in the class are actually implemented here. You might have to read through Bitcoin BIPs to make sense of some things that are being discussed, although this is not always necessary.

Some examples of the part of the code that you could study are: Merkle trees, and flag bit generation, the proof of work computation, or parsing and serializing transactions. You are, of course, free to select whichever part of the code you'd like to cover.

Note that knowledge of C++ is helpful, but if you're not a C++ crack do not worry, you just need to be able to read the code, not much more.

3.8. A cryptocurrency network

Implement a simple p2p client for managing a blockchain. The idea is that each node in the network guards a blockchain in which each block contains one or more transactions. You can use any sort of structure for guarding transaction (i.e. they do not need to be Bitcoin/Scroogecoin transactions). I suggest that you use dictionaries of the form:

```
{
   "type": "transaction",
   "value": "Hello",
   "uniqueID": ID
}.
```

Block in the blockchain can simply be a dictionary of the form

```
{
   "type": "block",
   "transactions": [tx1,...,txn]
}.
```

Your client should allow: (i) connecting to and existing node on the network; (ii) sync the blockchain with an existing node. The basic messages that the nodes are sending are: (i) version – containing the blockchain height of the node (number of blocks); (ii) inventory (announcing blocks/new transactions that the node knows); (iii) getdata(hash) – ask for data identified by hash.

You should use the ideas presented in the class, and use a minimal amount of messages you need to establish the desired behaviour of the network. Note that you do not have to worry about proof of work, or verifying transactions. You should only implement the logic of exchanging inventories. Note that you will have to keep track of new transactions that each node knows, but which are not in the blockchain.

Once completed, your implementation should be tested. For this, create several classes of experiments:

1) A network with two nodes which start with different size blockchain, and have to sync. Assume that one node has no blocks apart from genesis, and the other node has a 5-block blockchain. Test that your implementation does this correctly.

2) After the nodes are synced, start simulating their communication when they receive different messages. For starters, let one node receive 2 transactions, and announce them to the other node. Update the corresponding outstanding transaction pools.

3) Once new transactions have been propagated, let one of the nodes announce the next block with these two transactions. Make sure that they sync.

4) Once done, repeat the exercise with 3 nodes.

A good option for implementation is to use socket library in Python, but you are allowed to use any other library for establishing connections between two nodes.

For more information on networks check out the links I left in the class for module 6.

As this is a big project, you can start developing it now, and finish it for your final project. If you do not manage to finish everything before the presentation date, that is OK, you can still get a full grade if you do a good job.

3.9. Consensus from trust

In this project we will be implementing a consensus from trust, which does not use mining via proof of work to propagate messages. In a consensus from trust, we have a network of nodes, and any node can subscribe to another node that it trusts in order to receive messages. This is naturally modelled as a directed graph in which an edge (A,B) signals that B trusts A (i.e. receives its messages and sends them to its followers). In such a consensus, if A heard a message tx, it will send this message to B, and now both A and B will agree to include this message in their list of received messages.

Simulate what happens in a network of communicating nodes and how they synchronize messages they receeve using consensus from trust. You will assume that your nodes maintain a list of transactions. Transactions will be the same as in problem 1, and the simulation will work in rounds.

You will start by generating a random graph that will represent the nodes of the network. That is, create a procedure $GenNetwork(n, p)$ that takes as its input a number n, and a probability p. The procedure will generate a graph with n nodes, and for each pair (A, B) of nodes, assign a $p\,\%$ chance that there is a directed edge between A and B. If there is an edge (A, B) in the network, this means that B will listen to A's messages (B is a follower of A), and B will send these messages to its followers. There is also a set probability of $ppp\,\%$ that a node is malicious. $GenNetwork$ keeps tracks of malicious nodes as well. If you know how, make sure that the graph you generate is connected.

You will use the graph generated by the procedure $GenNetwork(n, p)$ in order to see how nodes in such a network reach a consensus on the set of transactions they receive. For this, generate a list of $10 * k$ different transactions, where k is a natural number. Your simulation will have k rounds, and in each round 10 different messages will be broadcast to the network. Each node has a $pp\,\%$ probability to receive each message. If a node is not malicious, it will receive a message, add it to its list of transactions, and sends the message to its followers. On the other hand, if a node is malicious, it can deploy one of the following three strategies:

(i) it can act as dead, and never broadcast any messages to its followers;

(ii) it can only broadcast the messages sent at the beginning of the round, and it will never broadcast the messages it heard fro other nodes;

(iii) it can alternate between these two behaviours in different rounds.

This flooding algorithm is then applied recursively until all the messages have been sent to followers (including the messages a node receives from the nodes it follows). At the beginning of the next round the next block of 10 messages is being broadcast, and the simulation continues.

Your simulation should determine how the interplay between the parameters affects the list of transactions that the nodes agree on (have reached a consensus). Two nodes are in consensus about the messages they received, if their list of received transactions is precisely the same. Track the maximum number of nodes in consensus at the end of each round. Note that two

sets of nodes can be in consensus about different sets of transactions (e.g. when they are in two disconnected components of the graph).

You can start with $n = 20$ (network size), and $p = 40\,\%$ (connectivity rate). You can pick $ppp = 10\,\%$ (percentage of malicious nodes), and $k = 10$ (number of rounds). Starting from there increase the network size, number of rounds, etc. Track how the number of nodes in consensus changes based on the network size, number of malicious nodes, connectivity, and number of rounds. As a sanity check, you should verify that in a connected graph with 0 malicious nodes, all the nodes will be in consensus after each round.

3.10. Cryptography behind Monero

Well, Bitcoin sucks (kind of) in terms of privacy. One proposed solution to this problem is the Monero cryptocurrency, which allows anonymous transactions, which anyone can broadcast, bot no one can tell who they come from.

In this project you should study what makes Monero work. That is you should study the main cryptographic premises behind Monero such as ring signatures, and show how it achieves anonymity.

A good place to start is the *Zero to Monero* book by Alonso and Koe: `https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf`.

Note that this project might have a lot of work behind it, so it can be extended to your final project as well, in which case you need to cover one part of Monero for project one, and the rest for project 2. It is up to you how to split the work here.

3.11. Proof of stake

In Bitcoin one uses a proof-of-work based protocol to reach a distributed consensus. The other often proclaimed option is proof of stake, in which participants guarantee the correctness at the danger of losing money they have (note that this is a gross simplification).

For this project you should understand what proof of stake is, what are its advantages, and what are its main weaknesses. You should also check out some concrete proposals of how a proof of stake protocol could be implemented. You can start reference hunting at the following links:

- `https://en.wikipedia.org/wiki/Proof_of_stake`
- `https://en.bitcoinwiki.org/wiki/Proof-of-stake`

3.12. Propose your own topic

Quite simple: send me an email (dvrgoc@ing.puc.cl) to confirm the topic you would like to explore.