Homework 1: Structure of the blockchain and how cryptography plays a role

Patrick McCorry

Kings College London, UK
patrick.mccorry@kcl.ac.uk

Abstract. We'll focus on the blockchain as a data structure to understand how crytography provides a role to self-enforce its integrity. Next we'll look at how the database is structured in Bitcoin (UTXO model) and Ethereum (Account-based model), and how transactions are processed to update the database. It is recommended to make notes on this homework sheet for future use.

1 Cypherpunks write code

Before deep-diving into the blockchain's structure, its worth taking a step back to understand the motivation for why something like Bitcoin even exists. On several occasions, Satoshi Nakamoto made it clear that Bitcoin emerged due to the mis-trust of banks and the lack of transparency in the global financial system:

The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies is full of breaches of that trust. Banks must be trusted to hold our money and transfer it electronically, but they lend it out in waves of credit bubbles with barely a fraction in reserve. We have to trust them with our privacy, trust them not to let identity thieves drain our accounts. - Satoshi Nakamoto

Remember, the bitcoin whitepaper was first posted on the 31st October 2008 and the running code was later published on the 3rd January 2009. Satoshi Nakamoto was clearly working on Bitcoin during the Great Financial Cirsis that was sparked due to the sub-prime mortgage crisis. The banks were at the heart of this crisis as they sold mortgages to people with bad credit ratings with the expectation that if the price of houses kept going up, then both parties could profit. When the housing market's bubble burst in 2006, it triggered defaults as the initial loans were worth more than the purchased house. The toxic loans and risk spread throughout the markets (i.e. securities, pension funds, etc) and eventually nearly took them down. As Satoshi Nakamoto highlighted, this was all possible because banks could lend more money than they held in deposits, and they had full custody over all deposits. Even worse and to Satoshi Nakamoto's frustration, the banks *got away with it* as national governments bailed them out:

The Times 03/Jan/2009 Chancellor on brink of second bailout for banks. - Genesis Block

Satoshi Nakamoto was clearly versed in the Cypherpunk's Manifesto[1] which at its heart focuses on how cryptography can be used to remove power-imbalances in society. We can speculate that during the financial crisis, Satoshi Nakamoto was trying to work out how cryptography can be used to build a global currency that empowers the individual, and not any single nation state. On hindsight (and in spite of 30+ years research), Satoshi Nakamoto was the first to investigate how to build a peer-to-peer electronic cash system without the support of a central authority/currency issuer. This eventually led to the *blockchain* which is a cryptographic audit log that lets anyone re-compute the contents of a database and *Nakamoto Consensus* that lets financially motivated peers compete to update the database (and thus remove the need for any appointed authority).

For this homework, we'll solely focus on how cryptography provides integrity to the blockchain, how the database is structured and how transactions are processed.

2 Two Cryptographic Primitives

While Bitcoin, Ethereum and their derivatives are called *cryptocurrencies*, at heart they are built upon two basic cryptographic primitives.

2.1 Cryptographic Hash Functions

First let's cover the basic idea of a hash function before we explore how it is used in a cryptocurrency.

- What is a hash function? [2 marks]
- A hash function is simply a one-way function. It takes an input of arbitrary sized, and computes a fixed, pseudo-random output.
- Identify and explain the three properties that make a hash function cryptographic? [6 marks]
- **Preimage Resistance:** Given a hash h, it should be difficulty to compute the pre-image x. In other words, the hash gives no clue to the preimage.
  Second Preimage Sesistance: Whoever computes $h = H(x)$, should not be able to compute another preimage x' such that $h = H(x')$. In other words, it can act as a real commitment to a secret value.
  Collision resistance: Given any hash h, it should be impossible to compute any pair of inputs x, x' that corresponds to h. In other words, there can never be a collision!

Good job! Let's now cover how a cryptographic hash function is used in Bitcoin to support *simplified payment verification* for light clients.

[1] https://www.activism.net/cypherpunk/manifesto.html

- What is a merkle tree and a merkle tree root? [2 marks]
- A merkle tree has a list of leaf nodes representing real data items (i.e. transactions), and every non-leaf node is a hash of its child nodes. A merkle tree root is simply a commitment to the entire tree and every data item.
- How are the blocks chained together to form the blockchain? [2 marks]
- Every hash in the chain is the hash of the previous block header.
- What information is required to prove a transaction is in a block? And how does a light client verify it? [6 marks]
- The light client must be provided with the block id, the transaction, and several non-leaf hashes that represent a branch in the merkle tree. The light client hashes the transaction, recomputes the leaf branch to compute a new root'. Finally the light client compares the computed root' with the block header's root.
- Why does simplified payment verification have weaker security guarantees compared to processing the entire blockchain? [4 marks]
- In this mode, a client will not receive or execute every transaction on the blockchain. They can only verify that a transaction is in the blockchain, but not that it is valid. Furthermore, they follow the longest and heaviest chain, and cannot confirm if that blockchain is valid.

2.2 Digital Signatures

Next we need to consider how users can register to use a cryptocurrencya nd how they can prove they are entitled to spend their new coins.

- What is a digital signature? And what digital signature algorithm is typically used in cryptocurrencies? [4 marks]
- Given a private key and the corresponding public key, a user can prove mathemtically they have signed a digital document. Cryptocurrencies typically use ECDSA (and hopefully someday Schnorr Signatures).
- Why is it important to always use fresh randomness when computing a digital signature? [2 marks]
- The randomness used in a digital signature (r,s) is public knowledge. Anyone can detect if two signatures rely on the same randomness. If they do, then it is straight-forward to break it by computing the private key.

3 Database Structure

The blockchain's only purpose is to let anyone re-execute all transactions in sequential order to re-compute a database. In class, we tried to focus on the database's structure for both Bitcoin and Ethereum:

- Bitcoin's database follows a UTXO model as the ledger simply consists of a list of unspent transaction outputs.
- Ethereum's database follows an Account-based model where the ledger records the current balance, storage and nonce for every account.

Let's take this opportunity to assess our understanding of the blockchain's structure. To begin, we'll focus on Bitcoin:

- What is the role of an input and output in a Bitcoin transaction? And why is there no real concept of a coin? [4 marks]
- An output specifies the spending conditions before a coin can be spent, and an input specifies the cryptographic evidence of why the spender is authorised to spend the coin. A transaction can combine the coins from all inputs into a single output or split the coins into several outputs. As such, there is no fixed coin that we can follow.
- Name three typical spending conditions that is supported in Bitcoin. [3 marks]
- A digital signature from a specified Bitcoin address. If the secret x of a hash h is revealed. If the transaction is spent before time t.
- What is the UTXO set? [2 marks]
- A list of unspent transaction outputs. Each script is associated with a set of coins.
- When the wallet software processes a transaction inside a block, how is the UTXO set updated? [4 marks]
- The software will first check all transaction inputs. It will remove spent transaction outputs from the UTXO set. Afterwards it will add the new unspent outputs to the UTXO set.
- Let's pretend 10 transactions are accepted into a block, and each transaction has an output that sends 1 coin to the bitcoin address A. How many entries in the UTXO set will be recorded and why? [2 marks]
- 10. We don't store a users balance, only a list of unspent transaction outputs.
- Why is it difficult to assume that if a transaction is spending two or more sets of coins, then they all belong to the same user? [4 marks]
- Multiple users can collaborate to combine their inputs into a single transaction. This can be done in such a way where users don't need to trust each other, and can be used to make it difficult to link the inputs/outputs. Coinjoin is an example tumbling technique for doing it.

The UTXO model is popular for its simplicity, its statelessness, and its determinism. Except for some quirks, it works great for processing financial transactions and supporting obfuscation of ownership. However it has so far proven difficult for designing fair-exchange protocols that run on top of Bitcoin which eventually led to the development of Ethereum's account-based model.[2] Let's explore some questions around it:

- What information is stored for each account? [5 marks] In class, we mentioned it was the account hash, storage, contract code, coins and nonce. If a student answers with "storage trie root" this is acceptable, but they must mention it contains the address and contract storage.

[2] A long running joke is that designing a new protocol in Bitcoin can get you a top-tier Oakland publication, whereas the same protocol in Ethereum gets you a blog post.

- What is a replay attack? And why does the transaction nonce stop it? [4 marks]
  A replay attack lets an attacker re-submit the same transaction to the network several times and as a result it is executed several times. The transaction nonce ensures that a transaction is only accepted once into the blockchain.
- When processing an Ethereum transaction that interacts with a contract, what basic validation checks are performed on the transaction and how does the software update the account database? [6 marks]
  Transaction Validation Does the user have a sufficient balance to cover the maximum fee? i.e. userBalance > gasPriced * gasLimit. Does this transaction have a larger nonce than what is already stored for the user? i.e. to prevent replay attacks.
  Update contract storage The contract's function is executed to compute the contract's new state. If the transaction does not run out of gas, then the database is updated to store the contract's new state. i.e. if I play the winning move in a game, then the outcome (I've won) should be recorded in the contract! If the transaction runs out of gas while executing the function, it simply fails and there is no update for the contract.
  Update user's account The transaction signer's balance should be updated to deduct the transaction fee (and any coins sent to the smart contract). This transaction's nonce is also stored (i.e. to prevent replay attacks).

For the final question let's discuss:

- What are the subtle differences between Bitcoin's UTXO model and Ethereum's account-based model? [8 marks]
  Number of signers In Bitcoin's UTXO model, there can be one or more signers per transaction. So far, Ethereum's account-based model only supports one signer per transaction.
  Evidence supplied when spending a coin In Bitcoin, the signer needs to specify which coins they are spending (i.e. what unspent transaction output do we care about?). In Ethereum, no evidence is required - we just deduct from the user's balance.
  Entries in the database In Bitcoin, the balance of a user is not aggregated and every time a user receives a coin it will create a new entry in the database. In Ethereum, the balance of the account is simply updated.
  Protection agaisnt replay attacks In Bitcoin - there is no risk of a replay attack. If a transaction is signed, then it will always spend the desired inputs and always have the same outcome. In Ethereum - a nonce is required to prevent the same transaction being accepted more than once.
  Stateless vs Stateful In Bitcoin - the state and transaction are interwined. When we spend an UTXO - the state is destroyed. While there are techniques like Covenants that may someday allow an unspent output restrict how a future output is created, right now it doesn't exist in Bitcoin. In Ethereum - the state and transaction are not interwined. All state is stored in the

database, and the contract is responsible for self-enforcing how the state can be transitioned.

While the answers will be released in a week's time, if you found any of the questions difficult then please visit Patrick McCorry during his office hours.