Blockchain

Wednesday, September 12, 2018 10:41 AM

Cryptography - science or (art) of encryption
- Extended to other stuff
  - Hash function, authentication, Random Number, Digital Signatures, Trusted network
  - Zero Knowledge Proof
- Higher Level Construction
  - Secure channel, key server, PKI

Threat model -> understand what & against whom you are going to protect
- Physi8cal? Network adversary? Presumptions that need to be made> Cryptography is very difficult
  - Hard to get it right, very difficult to get data
    - Cryptography is easiest because you know how to build and analyze. So there is a 'not-solution'L can generate but not their management

2 primitives -hash functions & signatures
- H: {0,1}* -> {0, 1}^n strings
  - Set of all binary strings 0~1 and infinity So there is an infinite set of all strings
  - Set of all string log n bits. All strings which are log of n bits
- So from infinite set to limited set
  - For any string, we produce same fixed length set. That output string is called a digest, hash or fingerprint. What publishers do is to hash the images, and then download and re-intepret
- There are many applications that use both functions

Requirements
- Collision resistance
  - M1 and m2 are different and difficult to find messages that give the same hash.
  - Difficult? Yes, because cannot map infinite to finite
- Pre-image resistance (One way property)
  - Given a hash value x + should be difficult to find any message m such that x = H(m)
  - Given some fixed value, difficult to find some input to get that output
  - Adversary has low probability of finding
- 2nd Pre-Image resistance
  - Input m, should be hard to find different message m2 which gives us H(m1) = H(m2)
- Collision resistance is easier compared to pre-image resistance
  - Collision resistance associated with node-bond

Birthday Attack:
- Same birthday exceeds 50%. How many ppl in the room? 23.
- Those birthdays are not distributed randomly.
  - N-values, choose k. So NCK
  - Such that thare are k(k-1)/2 : Summation of 1 + 2 + …k
- Pair of elements each of which has 1/N chance of being a pair of equal values.
  - Chance of finding collision close to k(k-1)/2N So k~=sqrt(N) and then close to 50%
- Need to have 2 ^n/2 steps as sqrt(2^n) = 2^n/2
  - Adversarial can find as hash functions are bound

256 bits to find collision, adversary needs to use 2^8 steps, 128 bits for brute force method.
1^128 steps: secure? Short term yes, long term… no
They found 1 binary block, 2 binary block… That is the same? Yes.

Security: ideal hash function should behave like a random mapping: truly random function with no bias.

Attack is a non-generic method of distinguishing the hash function from an identical hash function

Collision attack 2^n/2 steps
Pre-image attack 2^n asteps

Real hash functions: deterministic, fast, secure, know that it works (and how to analyze it !)

SHA3 is still secure, but SHA1 was proven insecure last year

Proof of Work?

Alice and Bob wants to communicate
Alice tells Bob, initiates conversation. Bob runs random function in his head to get a number and hashes it.
Bob tells her to prove the importance by taking this 32 bit hash and try to find my random number.
Alice runs the hash function so she cannot break it. She bruteforces it and finds H(n)
Alice finds and sends the number to Bob
Bob verifies if this N is correct. Bob is now convinced that Alice did some work. She has bruteforced her random number. Therefore, he agrees to talk.

How to apply? If Bob is under heavy load, he needs Alice to prove she's not a spammer.
Sometimes you see this strange page, need to check browser -< prove that you're not a bot.
Prevent others or cause adversary to slow down as his browser proves to be not a bot. This is an interactive protocol

An example of a non-interactive POW

15 years ago, Spam message prevention

Want to send email, have email agent send to another mail server

Got 1 interface now, but previously multiple idea is for client is that it's acceptable for client/user to wait for 2 seconds, then it would be inefficient for spammers
To send an email, the client introduces a new header field

Hash cash takes email and manipulates a field going from 0 to infinite and find some value that when hashed, would get us 0s at the beginning
Then the mail server computes a hash, checks if start with zero and send it along. Difficult to send email without doing some work.
But this never got deployed? Because not all mass email is spammer

Digital Signature: to sign a document, sign with private key and message -> returns signature

Verification

Verifies signatures of message using public key. Returns boolean (true/false)

Message -> sign <- Alice
|
Bob receives <-Verify <- Alice's public Key

Provided properties

Authentication

Bob sure signed by Alice. Bob knows public key bound to Alice
Signed by owner of private key

Non-Repudiation -- cannot deny having signed it. It's like opening new contracts, property you cannot deny

Integrity -> Eavesdropping, malicious
The message is modified, can detect

Security Property
Unforgeability: Given one or more message : signed pair [Xi, signed(Xi)]m it is computationally infeasible to compare any message signature pair
Oracle-- interface between signature And adversary
Oracle has the signature of secret key: Adversary can create message and send it to Oracle and signature returned
Adversary can get signature but not secret key from oracle.
Can get any message she wants; set of message to signature and at the end. Can adversary produce new message and its signature
Collect as many signature. You want and produce any signature

Hash Functions, signatures and computations, Commitments
Commitment scheme: A commitment scheme consists of two algorithms:
Com := commit(msg, nonce) The commit function takes a message and secret random value, called a nonce, as input and returns a commitment
Verify(com, msg, nonce) The verify function takes a commitment, nonce, and message as input. It returns true if com -- commit(msg, nonce) and false otherwise

We require that the following two security properties hold:
Hiding: Giving com, it is infeasible to find msg
Binding: It is infeasible to find two pairs (msg, nonce) and (msg', nonce') such that msg != msg' and commit(msg, nonce) == commit(msg',nonce')
Prove that you that something will happen
Putting message, hash it and reveal the message
After giving you that, not able to change it.
Wikileaks hashed something about John Kerry and commits it in a hash on tweet.

How to realize whoever was right? Coin flips?

Hash chain - get a secret value and hash it H(x), H(H(x)
Application: OTP, 2FA
We get Alice and Bob Server.
Alice computes H1000(S) and bootstraps server with H1000(s) preloaded with Bob
Alice signs in Bob to H1000-1 = H999(S). Only Alice can generates H999(s) as Bob only knows the hash.
And in the second session, H998(s) and so on.

What are the Disadvantages?
You can only use 999 times before changing it
Why preload it? And not everything else? Previous value she can prove any part of chain.

Hash Chains with Data
They are one-way, infeasible to modify as it will not terminate.
Can use timestamps and integrity
Idea was that let's get this database with this document and put it in the hash chain
Cannot modify any part of message
If you buy a newspaper, you can basically hash it and prove that our newspaper isn't modified such that it merges with more inputs of this document

Improvement of Hash Chain

Merkle Hash Tree/ Hash Tree
Documents/Data we have, we compute hash of these data, and these are leaves.
Non-leaf is the hash of concatenated leaves
We aggregate this data into 1 value and the integrity of this set by using short value.

What's so good about the hash tree?
Once you give 1 short value, everyone with access to this data can prove message to be in this data.
Because this data was produced
You don't get root, but can prove that this small part was added by you.
Since you have d, H(d1) can be found. Person who wants to prove you're in can give you H(d2) , following person gives you H(H(d1)||H(d2))

Root is unique, but identifies yourself. Size of proof is log2(n)

How to use Merkle Tree?
Data aggregated in hash tree. So everyone has to prove the following chain before you can insert your own malicious data.

Bloom Filter
Membership test (check if an element is in the set, without storing the element.
Initialize a bit array of n bits (initially, all bits set to 0).
K = (m/n) ln2 = 4
For each new element, apply each hash function. The value generated is a position in the array. For each position generated, Switch that position to 1.

To see if an element is in the set:
Apply the hash functions to thee element and find the positions
If all positions are 0, the element is definitely not in the set
If all the positions are 1, the element may well be in the set.
K-different hash functions, each maps set's element to one of a array position

Adding element -> If w is not in set, the hash function should collide
Hash element with k hash functions and set 1 on the obtained position
Querying element
If bits on all positions equal, return True. Otherwise, False
So possible if element is not in but check says if in.. But no false negatives.

Application
Database lookup
Object no need to look to source storage.

From the Textbook

Wednesday, September 12, 2018 3:35 PM

Block Chain: This data structure with a hash pointer. Each block not only tells us where the value of the previous block was, but also contains a digest of that value that allows us to verify that the value hasn't changed. We store the head of the list, which is just a regular hash-pointer that points to the most recent data block

Hash Pointer: a hash pointer is a pointer to where data is stored together with a cryptographic hash of the value of that data at some fixed point in time.

Why are block chains tamper-evident? Adversary's goal is to do it in such a way that someone who remembers only the hash pointer at the head of the block chain won't be able to detect the tampering. To achieve this goal, the adversary changes the data of some block k. Singe the data has been changed, the hash in block k+ 1, which is a hash of the entire block k, is not going to match up. Remember that we are statistically guaranteed that the new hash will not match the altered content since the hash function is collision resistant.

We will be able to detect the inconsistency between the new data in block k and the hash pointer in block k + 1. Of course, the adversary can continue to try to cover up by changing the next block's hash as well. As long as we store the hash pointer at the head of the list in a place where the adversary cannot change it, the adversary will be unable to change any block without being detected.

Merkle Trees. These blocks containing data comprise the leaves of our tree. We group these data blocks into pairs of two, and then for each pair, we build a data structure that has two hash pointers, one to each of these blocks. These data structures make the next level up of the tree. We in turn group these into groups of two and so on. This continues all the way up the tree until we reach the root node.

Adversary: If tampered, the hash pointer that's one level up not to match. Change will eventually propagate to the top of the tree where he won't be able to tamper with the hash pointer that we've stored .

Proof of membership. Someone wants to prove that a certain data block is a member of the Merkle Tree. As usual, we remember just the root. Then need to show up this data block. And blocks on the path from the datablock to the root.

N nodes: So only about log(n) items need to be shown. Since each step just requires computing the hash of the childblock, it takes about log(n) time for us to verify it. And so even if the Merkle Tree contains a very large number of blocks, we can still prove membership in a relatively short time.

Digital Signatures

: Desire 2 properties

> Only you can make your signature, but anyone who sees it can verify that it's valid. Secondly, we want the signature to be tied to a particular document so that the signature cannot be used to indicate your agreement or endorsement of a different document.

Digital signature scheme. A digital signature scheme consists of the following three algorithms:

- **(sk, pk) := generateKeys(*keysize*)** The generateKeys method takes a key size and generates a key pair. The secret key *sk* is kept privately and used to sign messages. *pk* is the public verification key that you give to everybody. Anyone with this key can verify your signature.
- **sig := sign(*sk, message*)** The sign method takes a message and a secret key, *sk*, as input and outputs a signature for *message* under *sk*
- **isValid := verify(*pk, message, sig*)** The verify method takes a message, a signature, and a public key as input. It returns a boolean value, *isValid*, that will be ***true*** if *sig* is a valid signature for *message* under public key *pk*, and ***false*** otherwise.

We require that the following two properties hold:

- *Valid signatures must verify*

  verify(*pk*, *message*, **sign**(*sk*, *message*)) == **true**
- Signatures are ***existentially unforgeable***

2 Properties that we require of a digital signature scheme

1. Valid signatures must verify. If I sign a message with sk, my secret key, and someone later tries to validate that signature over that same message using my public key, pk, the signature must validate correctly.
2. Unforgeability: computationally infeasible to forge signatures.
   a. Adversary claims that can forge signature
   b. Generate Key to generate a secret signing key and a corresponding public verification key. We give the secret key to the challenger and we give the public key to both the challenger and to the adversary. So the adversary only knows information that's public.

Lesson 3

Wednesday, September 19, 2018 10:34 AM

Everything starts in 2008, some paper called Bitcoin: A peer to peer electronic cash system. Links were give to some internet forum. After reading this paper, what's really interesting is the proposal that's it's open and distributed consensus with incentives and no trusted parties.

It is peer to peer; participants of the system are equal. There's no authority that decides something. In the design, bitcoin was intended to be cheap, anonymous, private, instant, and censorship-free payments.

We have 17MB and would rich 21 MB. 1 bitcoin is 10**8 satoshi.

This connected is basically connected with Banks. Whenever you want to pay, this is your payment. Moves money to whatever merchant you are buying for. Problem? Wikileaks got blocked by Visa and Mastercard. They received payment in bitcoins.

There was privacy: to see what kind of payments you're doing. Ecash: Bank gives signed statement that I have some coin, and I can do some business with this cheque. You couldn't link the transactions between deposits and withdrawals. Blind signature was used.

Cypherpunks: community based on anonymity, secret communications systems. Bitcoin was born in this community; developed there; strong political connotations.

Issues with the private systems were Centralization and Trust: we had to trust a network or some banks. You need to trust this device works; and then a few other entities. Ideally you won't trust. If we trust each other, then the community is much more efficient. How easy it is to do business. If you just talk to a guy, then you pay at some point.

In computer science, you don't want to give any trusted third parties. How to keep the system running? Peer 2 Peer for sharing data. People used it for sharing data, most users didn't want to share, just download. Some of these networks died because no one wanted to share it.

Security: No nakamoto consensus, no open and secure system.

After bitcoin, you have hundreds of forks, clones of bitcoin. We'll talk mainly about Bitcoin (Core). That's the most popular. 95% of system are bitcoins with some modifications.

We have hash chain of blocks, block has a set of transactions, these transactions cause hash tree. A special number, and a hash pointer and a link to the previous block.

In the bitcoin, hash function is defined as sha256. It's sha256 and sha 256 again.

How it works with transaction? We have miners, who run the Nakamoto consensus. Each of them keep a blockchain. Alice sends a transaction to Bob, and signs the transactions. This transaction is propagated amongst the Bitcoin network. You have peer to peer network which provides information to miners.

Then they start collecting more transactions and this proof of work basically starts. At some point, one miner is live. This miner on the right is lucky, found a hash that satisfies the target. He is propagating this bitcoin network. This block is the next block of the main chain. If it's appended to the main chain. Alice is -1.5$ from her balance. Bob can use 1.5$ to spend. Miner is getting 12.5$

Block has two components. We have header and transactions which are not so similar to Transaction

Class. In the header version. We have a hash of the previous block, which points to the previous one. And then we have a Merkle Tree root. Then we have timestamp. Then we have bits and nodes.

Bit is what is difficult, how many zeroes. Nonce is the number we have to find.

Proof of Work: Nakamoto Consensus maintained by anonymous peers. Hash of block header < target (proves that a lot of POW was invested). There is security and fast verification. If you would like to chain something, you would need to invest a lot of power. If it's appended to the block chain. To change it, it's very expensive. You don't know how to provide that. Transactions' there, you need to spend a lot of money to change it later. Very fast to verify: miner get a new block and check if header is okay.

We have two concepts. Target: value, hash to be less than that. It is some constant. Usually it's defined by 32 zeros, and rest of 1s. This is divided by current target. If target is decreasing, difficulty grows larger as difficulty = constant/current target. You need to find a hash less than that. You need to have 9 bytes of zeros at the beginning.

It's 72 bits! It's a lot.

What is very interesting is that bitcoin can change difficulty over time. It's adjusted every 2016 blocks. The value you want to target is to create 2016 blocks in 2 weeks. 1 block per 10 minutes. So they calculate this delta; this timestamp of the current block. Minus the previous one. So we basically measure how long it took to 2016 blocks. If it takes less than 2 weeks than we increase difficulty proportionally. If it's more, then we decrease difficulty. Maximum number for increasing is 300% and decreasing is 75%

Just to show you how it works in practice, at the top you could see hash array versus difficulty.

When miners propagate blocks, a new block is accepted if size <= Max_block_size. It's 1MB, but it's controversial. We check if the hash it's header < current target. We check that timestamp is not deviating from network time. Previous block's hash is correct.

What are full nodes? Replicated state machine. It's logic that implements sharing state which is managed by other replicas. Participants of the system basically do this set. This share state: set of unspent coins. To this set, we basically apply operations to modify it over time. Transactions operate on set of unspent coins. We have to receive, validate and store all blocks. We apply the same logic to save transactions. Miners are full nodes running the consensus protocol. You can create new blocks.

Let's talk more about transactions: If you think about transaction model. You have sender and receiver amount. This is something natural in the banks. Sender is implied, but transaction has sender, receiver and amount. And we just send it. How bank can actually keep it? The banks keep batches of this transaction, and what banks keep at the end is our balance. At the end our balance is this.

It has this like key value store; our back account, and our value is basically our balance. It is the easier model about receiving transactions, applying transactions to key value storage and applying this value's case in this storage. In bitcoin, there is no key value storage, this account has this amount. It just used completely difficult transactions.

Basically, we have a sender and a receiver, and we have a fee. Sender is coins going to be spend. Receiver is a condition how to spend them later. Fee is paying miners to mine. One or more transactions per block, and transactions are identified by their hashes. TXID = H(transaction). First we compute hash of transaction, and then we calculate onwards.

So the model is called Unspent Transaction Output (UTXO) Model. Each transaction spends coins previously received in one or more earlier transactions. The input of one transaction is the output of a previous transaction. Each transaction is creating multiple outputs. Each output can only be used

as an input once. We've seen the transaction, you can have multiple outputs. However, each output can only be used once as input. Otherwise, double spending could be possible.

Transaction Is valid only when it uses UTXO as inputs. Value of its outputs is less than or equal to inputs. WE are referring to previous unspent coins, and thinking of them as something to spend. There are unspent, so they've never been spent before, so we can use them as coins to spend. If we spend them, we cannot access them anymore. We can keep sending money to ourselves all the time. Value of our outputs. The difference (inputs-outputs) can be claimed as a fee by the miner who creates the block containing this transaction. Basically, you're going to spend, create a transaction, and put 1/2 bitcoin into transaction and miner could get another 1/2 bitcoin as fee.

These are transactions. Transactions have list of inputs and outputs. Our invariant is that when we are using this transaction, we cannot use the output for one transaction for two transactions for inputs. 1 input, many outputs. It doesn't matter.

Any advantage of this model? You can track people, all of his payments? From a miner's perspective: They care only about unspent outputs. Miners keep unspent outputs, unlike banks which keep track of all transactions.

Input refers to previous transaction, and index of output. For instance, this input would refer to the hash of this transaction, and to index 1, which takes half of previous transaction.

So the version determines set of rules to apply and an input uses a TXID and an output index number to identify a particular output to be spent. It also has a signature script which allows it to provide data parameters that satisfy the conditions (pubkey) script to spend the output.

Then we have locktime: we'll see applications later, but transactions can be cancelled. The earliest time a transaction can be added to the blockchain. It's in the blockchain, but it's inactive. After this time expires, this transaction is officially appended. If you have locktime transaction, you can send conflicting transactions and lock some coins, but you can send conflict to transaction. So there is a new non-locked transaction can invalidate it (use the same outputs). Input needs to provide some signature script to specify conditions to spend the coins.

Pay to Public Key Hash (P2PKH). We have this conditions and to satisfy these conditions. Usually, most of these transactions are P2PKH. We have Alice and Bob and they generate public and private keys. Alice is sending transaction to Bob's Address, and it is encoded as public key's hash. (base 58). It's similar to a bank transaction.

P2PKH transaction output that allows to spend it anyone who can prove that she posesses private key ownership that corresponds to Bob's hashed public key. This set of transactions are called public key script. It is a set of conditions how to spend. We are specifying conditions to be satisfied.

Transaction is propagated, and appended to the blockchain. Bob, now that he can satisfy this transaction, treats it as a spendable balance. He needs to create a new transaction, and in the input of thist ransaction, refers to Alice's TXID and its output. And Bob is creating a signature script that satisfies the conditions of this output's pubkey script. He proves that he's the owner who hashes it in this pubkey script.

He needs to put this public key (so that the script can check its hash) and his signature to prove that he has the corresponding private key and makes the transaction secure.

https://blog.trezor.io/a95e898e152c
In the previous one, we have one private key that we can spend our bitcoins. If you have a lot of money, you have a scheme where we have like 5 owners and only 3 of them can spend. Multisignature. If we have m of n owners who can basically spend it. In Pay 2 Script Hash. Only some of these rules are supported in practice.

It takes like one argument, data, and returns data, which cannot be spent. If we have output of return code. The coins here is wasted, but this is like used a lot. It is used to encode stuff. You have price per bytes, you would prioritize transactions, and you would change outputs. If there is a mistaken transaction, then there would be no change output specified.

Coinbase transactions: (first transaction of every block). Collecting and spending block reward plus transaction fees paid by transactions of this block. The output defines the miner how to spend. There are different rules: They cannot be spent fast: wait 100 blocks to spend the bitcoins. This is to prevent attacks. Just to prevent spending. If you have new coins, and you're trying to spend them, the receiver might figure out that these coins don't exist because of a fork in a network.

There is a reward scheme, how many bitcoins a miner can claim as a reward of mining. How every few years. They are getting less and less towards money. There is 12.5 $ per new block.

Everyone is trying to get a new block. At some point, they found that these miners agreed on this, and these miners agreed on this fork. They want their fork to win by competing. Because they have more hashing power. Everyone jumps to the longest chain, the fork grows stale.

Forks: In the case of a one-block fork, nodes individually decide which to accept (usually the first seen). Nodes follow the difficult chain and discard stale blocks of weaker chains. Long forks are very unlikely. Sometimes these stale blocks are incorrectly called orphans.

Why are forks bad? Wasted time and resources. The users can also be attacked. It's the worst thing that can happen to cryptocurrency. Bob creates transaction, sends 80 $ to Alice. The transaction is appended. Alice is +80$ and Bob is -80$. So Bob has the car sent to him. Bob can tell the miners, create fork w/o my transaction. So Bob can spend this 80$ again as the transaction gets cancelled out.

How to prevent this? If you want to follow the protocol, and play according to the protocol by confirmations. With 6 confirmation block, so we need to wait 1 hr (probabilistic) for transaction to take place. Majority of hash power is malicious, you cannot prevent it. You don't know what is hash power, but believe in it. There are software protection, they implemented checkpoints:

If you have blockchain in the software, then you have 10,000 and 1240, and have 1500. Then it cannot be changed. You see the same thing. It's protocol violation, completely different from the chain we know. WE should just treat it as true. So software vendors are pragmatic and this is a protocol violation.

Consensus Rule Changes: There are some bugs/features you would like to change. People are mining, doing transactions. So how to update? You need to fix bugs. Nodes use the same consensus rules (replaicated state machine) so what about updates? Updated and non-updated nodes will still follow different rules.
Hard fork: block following the new consensus rules is accepted by upgraded but rejected by non-upgraded nodes. Chains will be divergent,nodes will never agree.
Soft fork: block violating the new consensus rules is rejected by upgraded but accepted by non-upgraded nodes. Chains will converge, nodes will agree (when majority of the hash rate applies new rule
For example changing block size, rejecting malformed transactions. So there would be a waiting for hash power, flag days, user/miner activated soft forks.

Lesson 4

Tuesday, October 2, 2018 8:28 AM

What's a drawback of bitcoin? An issue that you can have when you see this protocol. This block, this transaction. One problem is to interact with that, you need to know the how thing. It's like a lot. You have like locks, which is header and transaction, and this is like capped at 1MB per ten minutes. Over 1 year, you will have a lot of data stored.

You cannot run it on your mobile device. That's a bit thing. You need to validate them. You need to build your set of outputs. One observation is that if you are running on a client, you'll be interested in node transactions.

You are interested in transactions with your outputs and your inputs. Most clients are not interested in all transactions. We'll learn why you have this merkle tree, why it's needed and what it will give us.

We have a header, and each header is associated with a set of transactions. This is like 8 bytes. Each one is 8 bytes, each one is like 1 header byte. We have this Simple Payment Validation which receive headers from full nodes. With headers, such clients can be provided with transactions and proof that this transaction is in this block. So if we have Alice and Bob.

Bob is SPV (Light) client. He has 4 blocks, with 4 MB. 80 Bytes per minute, 4MB per one year. If they talk to each other and have transactions, and Alice can send transactions to the Bitcoin network. That she's paying to Bob.

What will happen is that some miner will add this transaction as a new block. Bob needs to learn that this transaction is in. He basically knows that he's doing business with Alice. So he actually can ask the bitcoin network to give him this transaction and his proof. And the network would return transaction and the proof. He can validate this transaction because this transaction belongs this block and he will check the proof in a Merkle Tree. This is logarithmic thinking.

Bob is getting this transaction and the proof that it was added to the block chain and he verifies this proof against the headers he get. These headers are some kind of authentication, the network has a lot of proof of work. He can verify that this transaction is in this block. They rely on miners to validate transactions to make sure that this transaction is using all of these unspent outputs. SPV client could ask for the previous transaction and which output, but usually not implemented like that. This tree could be completely wrong because SPV clients could get headers.

With Merkle Trees that we presented, in a random order transaction. Easy to prove that the transaction is in the block, but not if transaction is not in the block. It's inefficient for sending a client. Bob has to go to the network and ask for transaction. The problem is privacy. The miner who receives that will know that the receiver in the transaction is Bob. Bob reveals his blockchain identity.

One way to do this is for Alice to provide Bob this proof; but then obviously Alice reveals that she's the sender of this message. What they thought is to have bloom filters for this transaction.

Bloom filters have false positives. Bob will ask for his public key hash. Bob asks for the number of times, they can just check the transactions. Miner will check for old transactions and if they are in the bloom filter. They will ask if it is in the block, it will be returned. The miner will return a lot of transactions. That the transaction set will return the transaction, but miner not sure if they are in.

If miners collaborate/collude, or if you are asking the miner multiple times. Miner..

In Bitcoin Wallet, a wallet is a collection of private keys (and client software) used to make transactions on the Bitcoin network. Your wallet is just a set of secretes, which allow you to manage this coin. You have some transactions, interacting with the network and allowing you to move bitcoins around. Wallet which generates key, stores the key and manage transaction.

This operation is pretty criticial, and this offline should have private key. You can create multiple subwallets, don't store in one public key hash, so let's split it into multiple wallets. Wallets that would delegate subwallets and the keys to operate that. Someone has to sign transactions, some has to get data from the network.

The big issue is that we have the same key for all of our transactions. Anonymization problem. So people started thinking how to solve it. What can we do with Bitcoin? You can use ECDSA. You will introduce some background about ECDSA but it will be really vague.

1 big problem in cryptography is this logarithmic problem. You have a huge prime number, and a number G with any prime number. If you are doing this operation g^x. If you are doing a modulo, (divide and take rest) to p. If you take this number and take some random value and you are getting y. And you are giving y to someone. Then he cannot find x. So we know what is g and what is y. But we cannot find x.

First we connect to some peers. Then we get connected to full nodes (block providers). These are guys that are getting new blocks and they are validating blocks and disseminating but not active in mining. Waiting for new miners to come, disseminating and so on. We cannot connect to the miners, but go to the Block finders.

Then we have header forwarders. From the Blockforwarders they can get headers and validate headers. They can be SPV clients, but they are another part of the infrastructure. Then we have lite users, wallets. Wallets mainly talk to the header and block forwarders asking for transactions. You cannot find the exact size of the bitcoin network, they will not tell you how they peer and return some other's peers.

Miners don't know the topology and the bit miners know each other. So it's not like they know it and so on. So over time, you have miners here, let's do connection, and we are like seeing a lot of blocks from you. One miner is not here. If you take minutes to propagate blocks, so the delay is pretty important. So there are many message types, like pushing new blocks, produce new blocks and something else.

You have data integrity by design, each block is like known by the previous one. Another thing is like memory pool, they keep buffer for unconfirmed transactions because this is something they want to add to the block. When there is a lot of transactions going in the network, the memory is constantly growing. However, it's not critical that this is synchronized. If they don't have incentives to basically exchange that. They don't want to exchange to other miners and they don't want to carry it soon.

They can be banned. What is like critical in the whole network, critical to connect to at least one on this node. It could be completely disconnected, and to some malicious nodes. And this malicious node could show some completely faked blockchain. It's critical for her to be connected to at least one.

It is critical that a node is connected to at least one 'honest' node.

Bitcoins are full of incentives. How to basically analyze the systems. We should be able to sum up the model. We have players and moves that we can make. Then we have a payoff function: gain for following a strategy. Nash equilibrium: when a player deviates from them, no gain by player. Rational players should follow this equilibrium.

Which transactions to include? Which block to mine? We didn't define payoff function, but it's just what transactions to include. You have forks, and make sense to mine on something different.

Obviously, protocol says to mine immediately, but if you are selfish. Many transactions make sense with fees. The equal reward for each miner is bitcoin, and they will start fighting for the transaction because it would compensate for this transaction.

If a miner has a large portion of the system, miner doesn't announce found blocks immediately. All of the miners honest work is lost. By witholding block and revealing at a strategic point, the selfish miner can benefit by wasting this honest power. If this selfish miner is big enough, he will be winning this game. This protocol says if they are starting to win, I should just publish immediately. If everyone seeks this fork, they will just pick randomly.

Mining Pool: How to share it? Remember that the block is less than target. This block will be accepted, reward will be paid. There will be high difficulty hashes, and show that we really work on the same block. Hash with higher difficulty than you, so I should be paid more. The pool manager collects and distributes rewards according to miners' individual contributions. If they can find a new block the pool manager gets the reward and distribute among miners. Hashing the difficulty I found.

By then, they proved that they really worked on this problem. If you are able to show a high difficulty hash, they will just pay you. That's really good for miners; they are paid for every proof of hash and all the risk is on the manager as if they cannot find anything, the pool basically dies. This flat rate is not high; they are paid even if they didn't find.
They are figuring how to find how much was contributed. And they distribute this money according to hashes.

Proportional pools: when they have new blocks, if you have a pool that is proportional and another with a flat rate is that you start mining in proportional, and then you switch to another if you can't find it. This proportional pool promote early findings. If you can find block very early, there won't be too difficult to find it. I can switch to a flat rate pool and get paid for every hash.

If I mine honestly, I'll get 70% of rewards, 70% of system. What happens if I move 10% of my power to another pool? So split 60%, and 1/4 of the other pool. 25% of this pool's rewards will come to me. If we mine honestly, pool 1 will get 60% from this one, and 25% of this pool will also come to me. Whatever I find here, I reject. How they pay is that they pay me for my contributions; use this hash which are long enough that I compute something, and this hash is not so low that I get reward from the system. So this 10% got this block, it will just drop it.

However, this pool holding is not as bad as withholding a block against pools. You have two pools in the system: one and 2. 70% of mining power, P2 has 30% of mining power. Information gotten by one pool would be 70 and other would be 30. They delegate some hashing power to the smaller pool. Let's say 10 percent of this pool goes to another one. P2 plus p1 10 percent. So basically, we have two pools which Is like 70 30 and 60, 10 and 30. So in total 40.

So a bigger pool can lend its computing power, and get the 10 percent of the block that goes to the new balaance, and the bigger pool gets 1/4 of 30% and there form 67.5 percent of the profit. The 10 percent that is rejected is found by the bigger pool 66 percent of the time. So now the distribution before the 1/4 is now 66 33. So you get 66% plus 8.25% from the 1/4 and gets 74.25. So he has the majority of this.

In other words, if this 30% honest other pool finds the new coin, 7.5% of bitcoin total will go to the 60% majority, and the 10% that is never published is also going to the 70% of this pool.

At some point, there will be no block reward. So you will be waiting for new transaction fees. Miners can get advantage of not validating transactions. So they take risks that the blocks they find will be invalidated by the network, but can take this risk sometimes. So mining without validating is called SPV mining.

Freshness: related to time, tells us about order of data. One is weak freshness; it allows us to order events. One event happened after another event. As you can imagine, this is provided by default You

have this pointer to the previous header and we can prove cryptographically that given block is mined after given block. We can exactly tell when something happened. It's not on the order, but also timestamp. So miners put timestamps. However, there is no time source specified. They don't do NPT protocol, so miners put whatever time they think it is correct. In the network, we have the orders of this block and in many cases, these timestamps could be conflicting.

Node considers a new block's timestampas valid as the median timestamp of previous eleven blocks and smaller than network time (median of timestamps returned by all nodes connected to the node).

Properties:
Consensus:
Append-only
Transparency
Availability
Openness and Decentralization
Censorship Resistance.

Cap Theorem: no system provides consistency and availability.
Partition tolerant. You have two views of transaction. If the bitcoin network has a split, in one part you have one view and another. This is strange for transaction systems. You cannot detect partitions

Scalability: Better throughput elsewhere. More people use it the more expensive it is. More people using this, it's more expensive.

Lesson 5

Wednesday, October 3, 2018 10:39 AM

The longest chain: We have chain and which one is longer and why not? Proof of work invested here is more than the proof of work invested here. Initially, the longest chain was the stronger chain. Obviously, it's very easy to give a very long chain with low difficulty. It is very difficult to give a blockchain with higher difficulty.

What is security property for signatures? We have message and corresponding signature. We could not identify the private key. But the security property? You cannot claim that you didn't sign something? If you think about digital signatures, what is impossible for adversary? After the number of messages, you can guess the signature. Basically unforgeability and unable to generate key message pair for new message.

It is impossible to find mi and the corresponding signature for that, such that the verification with the public key will return true. It is impossible to find matching signature for new message. This makes sense; adversary cannot create signature for any different message. Adversary can replay, but not modify and new message.

However, what is very interesting is malleability. Let's say adversary has one message and one signature. Can create message and signature 3 which would be modified and then accepted. So you cannot produce any signature for a new message, but you can use an old signature for a new message.

You can modify signatures; it's not a violation of unforgeability. And this property of some signature schemes, ecdsa has this signature. You have signatures like two values, and how you could modify it. They are encoded as points on the thing. Can create two message signature pairs. It is correct, just a different form of signature.

Mt.Gox had transactions. How was transactions identified in Bitcoin? TXID. It's just hashed: H(tx). This transaction was propagated to the different network, and the system remembers the hash of the transaction and waits until the transaction is appended.

The adversary was interested in modifying this transaction. Here transaction tx is sent to the network. He was modifying signature within the transaction, so everything looked okay and went through validation checks. However there was a race condition. If this was appended first, the adversary could claim that: Exchange, your transaction wasn't appended. There is no transaction ID in the blockchain. So the exchange sent money to the adversary.

Instead of storing hashed transactions, which are not very precise, we should store in some unique identifier of transactions. Instead of storing transaction ID, store hash of multiple transactions. Without changing format, just change somet stuff.

SPV client has only headers; has private/public key, can receive transactions. Validate whether it can transact correctly, whether it is in the blockchain. Needs to check if the Merkle Proof validates against it.

SPV client needs a block, a transaction, headers, and should be able to query for transactions and check if they are in the blockchain. Miner does not need it. But SPV client needs the transaction and proof. Final thing is to implement selfish mining; this is easy to show when the mining power is large.

It should be at least 30%. Following selfish mining. Get resource/processing in the system, give higher or lower. Give comments like nice (kernel scheduling, giving more resources to each miner). Miners would be malicious, show malicious spending. Miners basically publish their transaction and the SPV client got a bit confused?

Lesson 6

Tuesday, October 9, 2018 8:35 AM

Resources: it's a waste of resources. So people start basically thinking: can you prove it?
Proof of Work is needed to protect from Sybil Attacks: Adversary creating many identities, and this protection has to be resource-base. It has to be based on CPU consumption, memory bandwidth. In the past systems, which are related to bitcoin in this aspect.

You are basically running all of these farms, to prove that you are a waste of energy. Also, the hash functions are easy to optimize, they are designed to be fast and easy. If the faction is pretty easy, then you can implement it in hardware. This Asic format (super computer) is easy to optimize.

Can we have some mining with some better properties? Bitcoin is hard to change but other systems could benefit. If you decide to waste resources, to go on the attack, then maybe you could do something useful.

What would you like to replace with this, verifying from another form of function. Whoever finds larger number wins. You would like to replace proof of work with something. What is that something? Mining and verifying.

Everyone calls verify: It should be fast, as everyone runs it. So verification should be easy.

So afew requirements

1. Quick to verify: every full node validates solution. If verification takes 1 second, you can spam the network, and it would take them forever to verify. You verified very quickly. You have millions of hash functions per second. You can verify very quickly
2. Adjustable difficulty {in, de}creasing network capabilities. Bitcoin has this property; networking capabilities in terms of computing power can be increased or decreased. Otherwise; it won't be stable.
3. Fair: What is fair? Intuitively, fair means that finding solutions should be proportional to resources available. If we have hashing equipment, and 10 percent of hashing power, you should get 10 percent over time. Over time, you should get basically proportional reward
    a. Progress free: finding solutions should be independent (probability)What is the implication of that? For every input, you have the same probability that the output would be less than target. If you are computing one hour, you don't have higher probability for the next input
    b. This kind of process is memoryless: finding time forms exponential distribution. Events have the same probability and they are independent, if not it would be difficult to compete with high resources. Because he or she would be winning all the time. So that's something very bad.
4. SHA256(sha256(.) < Target satisfies them. It does hash of hash, it's quick to verify. Adjust the target as you want, and it's fair IF the hash function is pseudorandom, because for every input, you have the same probability.

The problem with hash function is that you could get this ASIC Computing. The idea of Bitcoin was that we mine with CPUS. You contribute to the network. To Corrupt 50% of it should be difficult.

With Asics, your CPUs are worthless in terms of mining. If you buy your own FPGAs or GPUs, it is useless if the company makes this hardware which makes this 2 times hash it makes it insanely fast. What consequences is that mining is getting professional and concentrated; it's a business when people just buy it. Large players are bad; difficult to enter the system. You cannot use your own free cycles to contribute to the system and buy this asics because only then could you compete
This ASICs wastes a lot of energy; it's efficient, they take a lot of energy because it is an integrated circuit. It's more efficient than CPU but for the same price, the equipment has a much more energy consumption

There are few people to argue against this: people who use ASICs say that it is proven/secure, it is always possible to specialize hardware. Nothing bad happens so far and we are mining with ASICs and it makes sense.

Always you can build something more specialized. Hardware or software, all of which you are able to compete. Wonderful method where you have this visualization: you can divide money to everyone, would they still have the same amount of money. Someone is better at doing business than others; probably if you have anonymous market, everyone would not have the same as the rest.

The goal is to build such mining algorithm that building ASICs is unprofitable. It is expensive, challenging, and the algorithm can change in a way that the ASICs is useless after some time. We would like to get something called how the 1 computer/1 vote is implemented.

If we build something, we can mine efficiently on the standard PCs. Probably miners who lent AWS machines and invest a lot of money in having 1 million servers and doing this proof of something. This is very interesting trend in this mining and it is leads to currency wars?

Memory-hard mining: Instead of computing, we want to have a function which requires a lot of memory. Access to memory is not as expensive as computing/CPU. It's greener.
Memory-bound mining: similar concept where the memory access time is important.

You can outsource it to harddrive (hard) and keep it in RAM (bound). Memory access is important. So usually, we aim for both. Memory hard and memory bound. It's really true that this kind of ASICs are hard to design. They are good in computing something, without accessing some memory, without doing something with this memory, and it is very hard to build ASICs that need to compute something + access memory

But we can't use SHA256 as it is not memory hard

This problem is similar to other problems in security; Passwords security: The user has username, hash of the password. Why would you not want to expose your password? People can steal it (your server) when they compromise your computer and the list of old passwords are visible.

Let's hash passwords, so adversary would see hash because they won't be able to login, he knows only the hash. One function is called like scrypt.

Scrypt is to mitigate this kind of problem:
It's some kind of memory card, so the definition of the function, which takes N, a pretty large number, and a seed is a message to hash.
So first, it initializes this large buffer of zeros.

Then what we'll do is compute the hash of the previous number. For all the N zeros. The trick here is to access it, so the elements are pseudorandom.

We compute X has hash of the previous one for the last one. However, we compute like hashes. X is basically hash of this. So X is now pseudorandom. And we go through from 1 to N, and we do modulo of N. For every iteration, we take random units.

So X is basically sha256 of this previous one, XOR of the previous index. After this N rounds, we return X. Complexity is O(N). Memory is O(N).

With only seed and N, can you compute it? Without keeping this list in the memory, I can just keep hashing it and I don't have to store intermediate values. So what happens here; for every call of line 9, I need to compute this value if I don't have to store it. So the complexity of computing is O(N). So to compute X, we need to compute this 1 for each N.

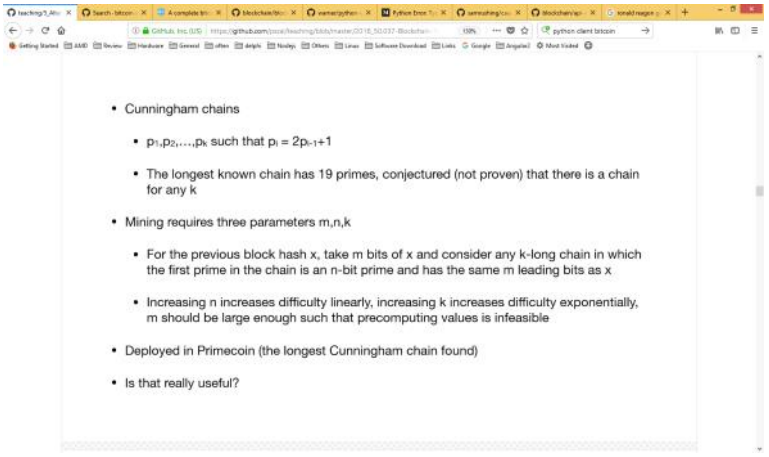
So our complexity without this buffer space is O(N^2). The difference between O(N) and O(N^2) is significant. There is a lot of processing, storage overhead, but at the end it's like N^2. So what is the issue here? Verification is hard, because you need to run it once. For mining, you can hear the message with another node, and your array is growing, but for verification is pretty hard. Less efficient then with bitcoin. But It's still very fast.

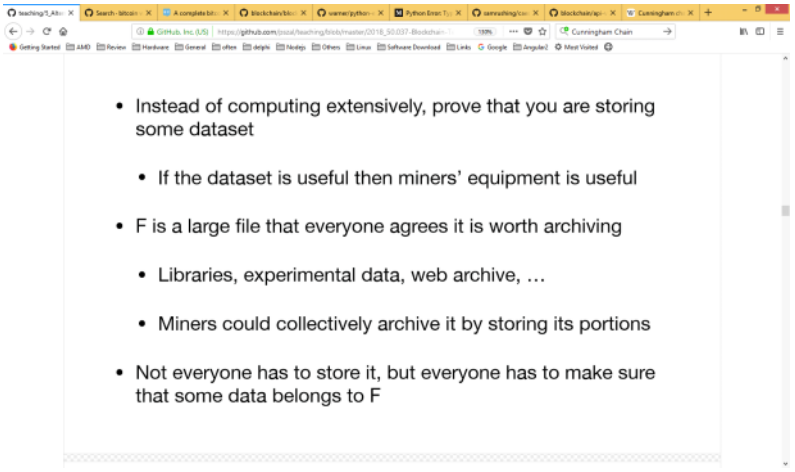
With SPV Client, N is too large, the verification requires significant memory. With some sane N, it's okay. If it is too low, then ASICs can be built. It used in many operating systems. So why you hash passwords. So other approaches are Cuckoo Cycle (build a graph, there's a cycle, and it's memory hard to keep the structure of the graph). X11 (11 hash functions. You have this property that it is difficult to build)

What is the bad thing about this memory hard find. It's still like users. It's a bit better, because you don't waste as much energy. However, it's again a waste of resources. You have many servers, just to compute this function. It would be great to have something useful, can go for many years of research. Need to verify, adjustable (difficulty down up) and fair.

Distributed computing has many cycles: use the spare cycles of users. Probably was SETI@home. It uses data from regular telescopes, listen to extra noise from space and have patterns. NASA never has enough budget, let's use this application which would download a sample of this and run the algorithm. By that, we'll find something.

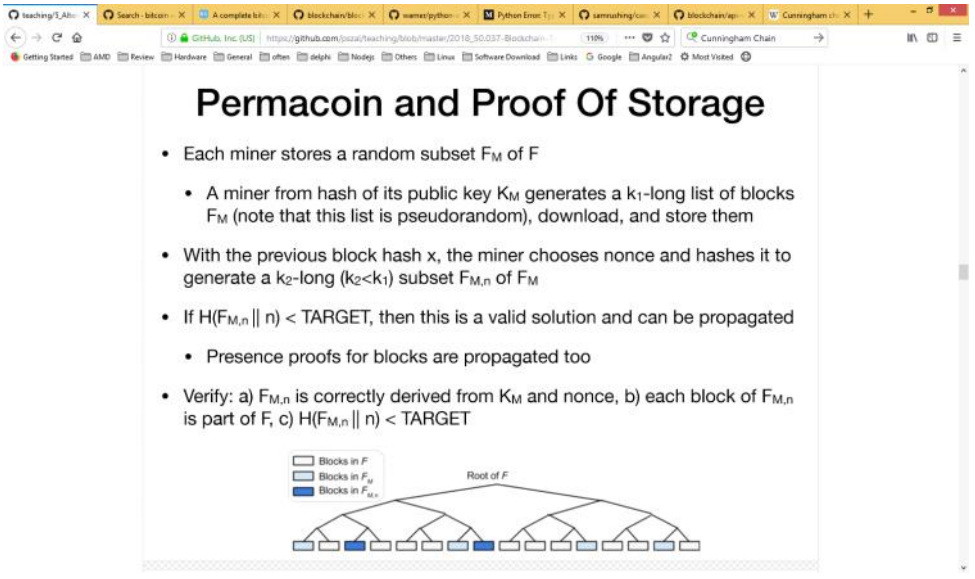
We are looking for proof of useful work. We just compete to find something. In terms of distributed computing, it's fine if you get the results. You want to maximize your chances of finding something.





Miners could not afford to have this file stored on their hardware. If you have many miners, everyone can store some part, and they need to agree that some data belongs to F.
So store it in a Merkle tree. Root is hardcoded in the system, in the genesis block, and it is worth to store this file. Basically, miner M, joining the system, generates public key K. Miner, from this public key, better mines which block that we need to store. Miner basically computes hash function, and mapping to some random block. Basically, this hash function results in hashing to K1 blocks. K = 6. So basically, we generate some pseudo random thing, and generate this pseudo random thing on k 1 blocks.

For the previous hash X, the miner would basically put the hash of X and generates a subset FMn of Fm. This would be a selection from this subset. Trying to tell that a hash of this set, concatenated with nonce, whether this hash is less than target. If it is less than target, it is propagated, and proved of inclusion.



Mining Misc:
Proof of luck /elapsed time depend on trusted hardware eg former unbroken intel SGX
But this causes one trusted party (INTEL) yet this is efficient as well?

Blockchain is an open system and you cannot stimulate it without resources.

Proof of stake depends on your number of coins (don't need resources) but how does it work?

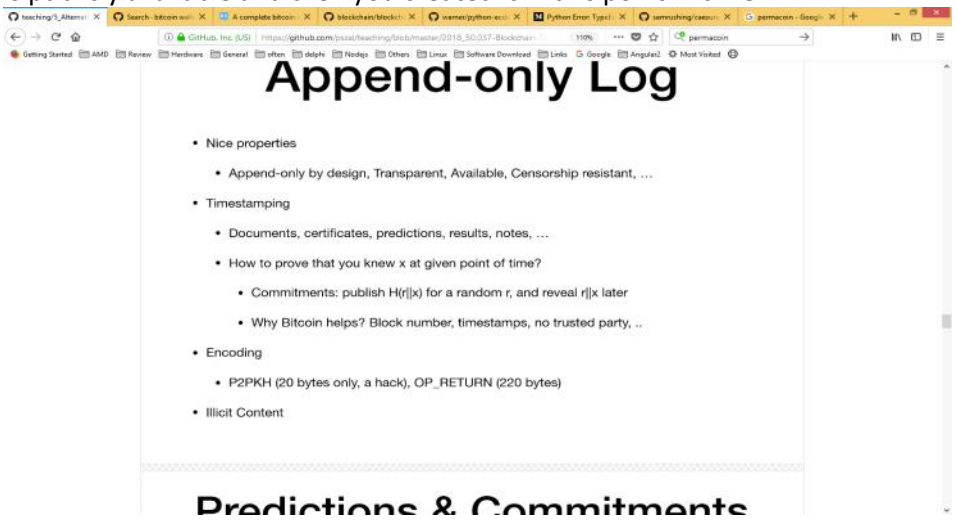
Someone needs to give you stake to participate but they can withhold it.

Noun-outsourceable mining: Make a system that you use your private key to hash and get signatures, then people will be hesitant to use say Amazon Web Services

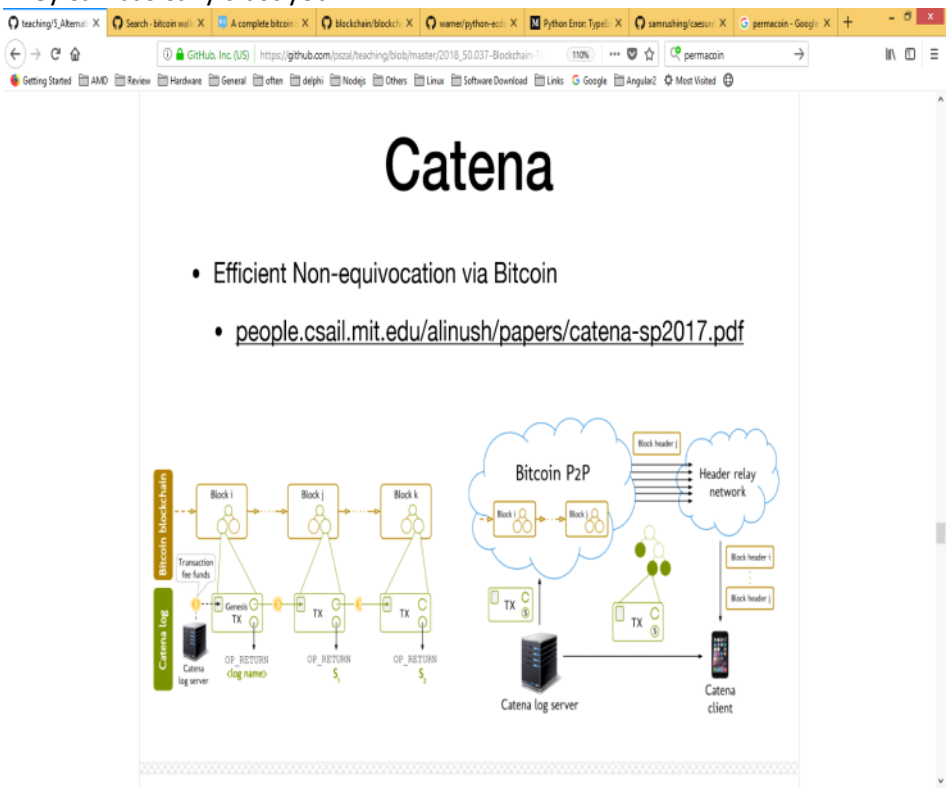
Bitcoin is append only by design: You have enough confirmations, it will stay there forever, you cannot change it. Everyone can look at it, it can arrive at any time. My transactions can go through.

So where do we need that append only block? In many cases, we need to timestamp. Your graduation cert is time stamped. You would like to show that something happened, you predict that something is gonna happen, you need to prove this prediction?

If you know X at some time, you can publish the hash of X concatenated with some number. People would not be able to enumerate that, R is a random number, you would be able to make sure that it is publicly available and then you created it in this point in time.

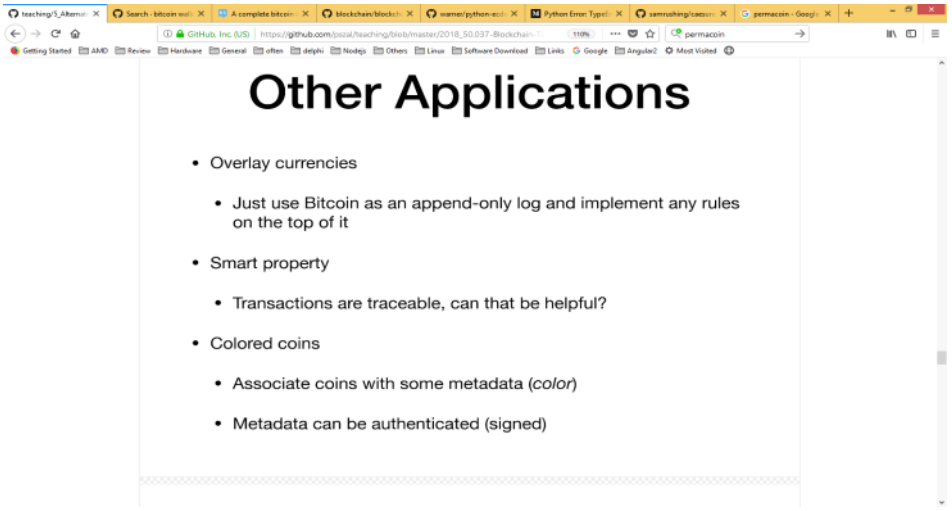


They can basically trust you.



Keeps servers accountable. If a given server gives different view of its contents, then it becomes security critical. They bootstrap it and give transactions to everyone. When they change the database, they publish the hash.

Merkle tree (database) and transaction = 0. And this output = 0, output = 1 And output 1 is basically op_return and through zero. They publish, op return, root of the hash tree.



Other Applications

- Overlay currencies
  - Just use Bitcoin as an append-only log and implement any rules on the top of it
- Smart property
  - Transactions are traceable, can that be helpful?
- Colored coins
  - Associate coins with some metadata (*color*)
  - Metadata can be authenticated (signed)

Lesson 7

Tuesday, October 16, 2018 8:42 AM

Smart Contract:

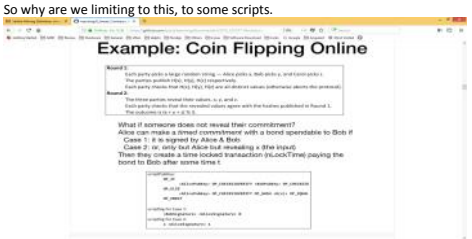
1. Agreement (represented by code) If we agree on this code, this code is law. This is an old idea, that the code is law. (Eliminate lawyers)
   a. Insurance
   b. Financial Contracts
   c. Gambling
   d. Voting

We interact with this logic, because we can verify it.

Nullification: We cannot modify it. Because it is in the blockchain

We have a platform where we can monitor transfers. This is something that people start thinking again. This distributed concepts to have something more than only transfers. And very interesting, that bitcoin has this scripted language. This scripted language was to encode logic related to transferring funds. We can prove some ease, some multi-signature scheme. This is like language.

So why are we limiting to this, to some scripts.



We would like to decide who won. We have Alice, Bob and Carol. All have the same choice of win. We would like to write our numbers down, sum up these numbers like A, B, C modulo 3, then 0,1,2 goes to A, B, C respectively.

If you know A and B, then Carol can give an appropriate C to win.

Give first commitment?

1. Publish commitments: They just compute hash over number, the number should be large and they should just show it. They cannot claim for another number, they need to reveal A, B, C.
2. Then reveal the actual number, show that the hashes are derived from original number and then they would have the same chances of winning

Is there a problem with this? What is the trusted problem?
C goes offline: protocol aborted! If the last guy sees he is not getting advantage, he would go offline. You would prove that someone is unavailable.

What we could do? They can do a smart contract over bitcoin.
If some does not reveal, the commitment, when they start playing, Alice can make a timed commitment. So special transaction: which can be spent to Bob in two cases.

If it is signed by Bob and Alice, if they both agree. Or if it is signed only by Alice, but if Alice reveals her commitment.
We have this output, both agree to sign it, and it would be spent. Only Alice signs, however, she reveals her commitment. So they started playing, they put this commitment phase. We have X, so Alice committed this hash, and she creates a deposit transaction which can be spent if it is signed by Alice and Bob, or when Alice reveals the input.

So Alice creates the output, so they create the first case, they create a transaction that signs money to Bob. However, this transaction is time locked: it will only be executed one day later. Or five hours. Or some time they agree. So Bob is secured. Because that transaction is there. Either he will get this deposit, or Alice will reveal X.

If Alice doesn't reveal, he gets money, if Alice reveals, he gets what is committed.

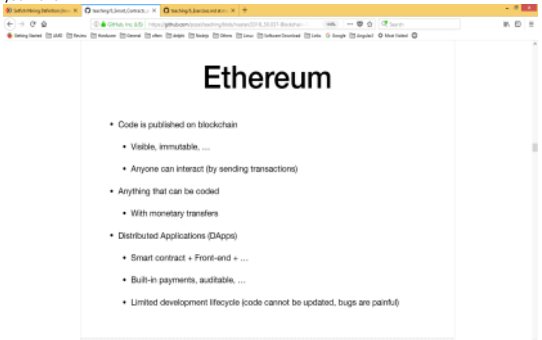
For each player, you have to deploy this to protect each other; so that in most multi-party computations, you actually need to have something like the bottom packet.

They introduced this opcode which was like OR, or NAMEREGISTER. OR_NAME_UPDATE. And OP_NAME_FIRST_UPDATE. And they also extend that. Unspent transaction outputs, and to have mappings.

Name, and which is mapped to some IP address or some public key.

You could register it using that, and the miners would keep it in the state and if you wanted to access to it you can use that. Name register is pretty annoying, it was like bitcoin script. So you have to write this kind of thing. It's application specific.

Ethereum is the second for a long time. There is very active development.
What distinguished this from bitcoin? This scripting language: we don't have this scripting language that bitcoin has. We have a turing complete, high level scripting language which can code whatever you want.



What else we can publish on blockchain is code. Write an application, and this application will be appended to the blockchain. Codes are published as transactions in bitcoin. Everyone will see your application. Another is that everyone can interact with this. If something is in blockchain, anyone can hold this application. Another one is that you cannot change this application; if you come back, you can't modify this.

This application can have logic connected to transfers. You can encode transactions, and payments within this. You can have monetary transfers with this application.

How it's usually done. We have some applications which is on the blockchain, you have a website which is a front end which interacts with this smart contract and this application is accessible by users.

You can have built in payments, you can look what the state of this contract is going on, what this code is and so on. It's pretty limited in terms of development cycle because you cannot avoid it.

Namecoin in Ethereum

- Namecoin (recap)
  - Bitcoin's fork
  - New OPCODES (name register, update, …)
- Ethereum (pseudocode)

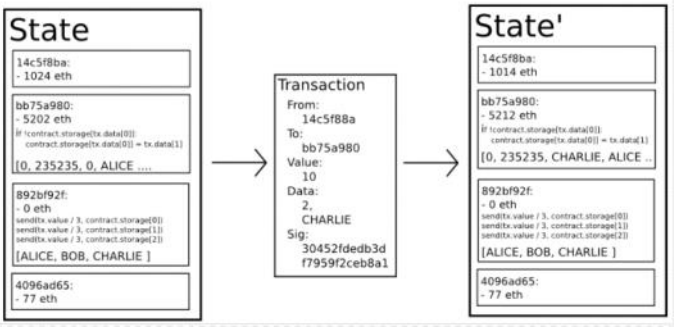
```
def register(name, value):
    if !self.storage[name]:
       self.storage[name] = value
       return 1
    return 0
```

This name is visible on the bitcoin blockchain. This will update, miners will do this to some purlieu. You have to add new opcodes. Start the blockchain and it is not really used.

You would write such applications. You would define a register, which takes name and value. If this name is taken, then you return zero, not taken, then just assign the name to value.

High-level Overview

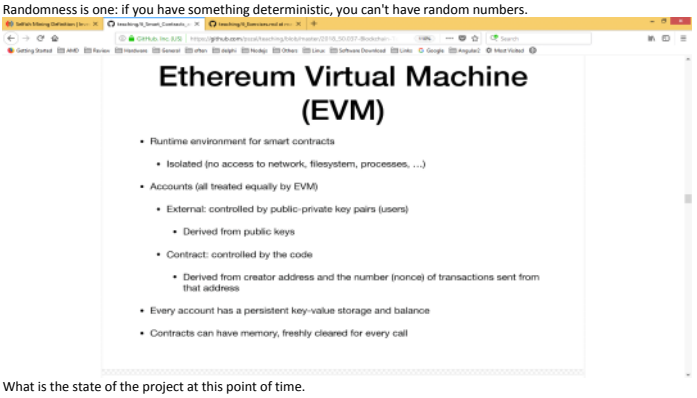
- Miners run a consensus protocol
  - Monetary transfers, code, code calls on the top
- Miners keep the current state
  - State transition



They keep a global state, keeps some info about the state of this application. Whether it has money, internal storage, and this should be consistent. You have application, transfer. With ethereum you can have some direct transfers. They basically store the state, has some money, has some code. You can see some random state. For every incoming transaction, they basically code and they run this code and transaction. And this would be caused of the state.

You can transaction which calls this code, and chains some. If we are getting the same output, then this becomes deterministic. All of these code has to be deterministic to get the same output from the same input.

Randomness is one: if you have something deterministic, you can't have random numbers.



What is the state of the project at this point of time.

This machine takes this code. This code is byte code, so a part of a code. This code runs for every code. Address of this code. Every miner running the same code, for the same inputs. So this virtual machine can maintain state.

What is critical? It has to be isolated. It cannot have connections to some website. For example, hacking. So if you post something, like a TCP connection. Miner validates it, calls some function of your code. There will be multiple connections via network, not making it isolated.

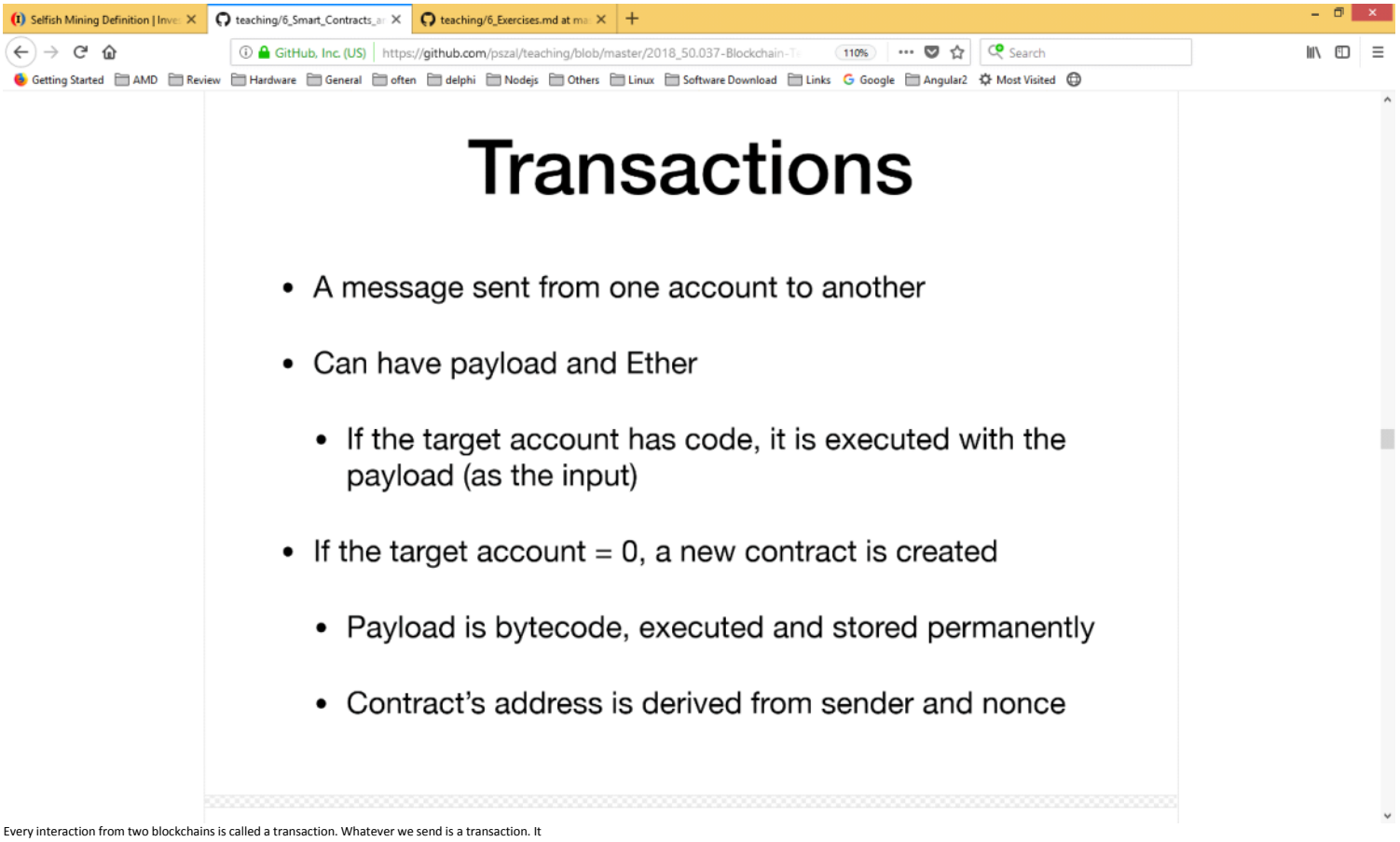
Smart Contracts cannot be read from your hard drives. So your smart contracts should not talk to your browser. So what is interesting is that accounts are unified in Ethereum.

We have one entity which is account. In Bitcoin, we don't have accounts, just conditions as to spend outputs. We can identify public keys. This is not something embedded in the system. In ethereum, we have identifiers which are unique and special.

There are two types of accounts: first we have external. If you are interacting with Ethereum, you have external accounts, and this is under control with your public key. Address of your account is your hash of a public key. You generate a public private key and this is your account.

Contract accounts: instantiation of your fork. This contract gets like address. This address is derived of hash from external account.. So if I'm creating this public key, concatenating with some nonce. It's not true, because accounts can create other contracts, but can put other accounts. This nonce is like a counter; it counts how many transactions sent from the address.

One can prove that this contract was taken from this account. This account has a persistent key-value storage and balance. You can associate data with your address. Contracts can have memory. The distinction between storage and memory, that storage is persistent, memory is volatile, so your memory is set to zero once you call the contract.



Every interaction from two blockchains is called a transaction. Whatever we send is a transaction. It is a message sent from one account to another. Transactions in Ethereum contains messages. It's message + metadata.

So a transaction is something that the user is sent to the blockchain, modified, and then send a message

It can have payload and Ether (a value). IF you are sending this transaction to the contract account, it will be executed with this data. It is a coin function. This transaction will contain data and address of the function, and what to call with what data.

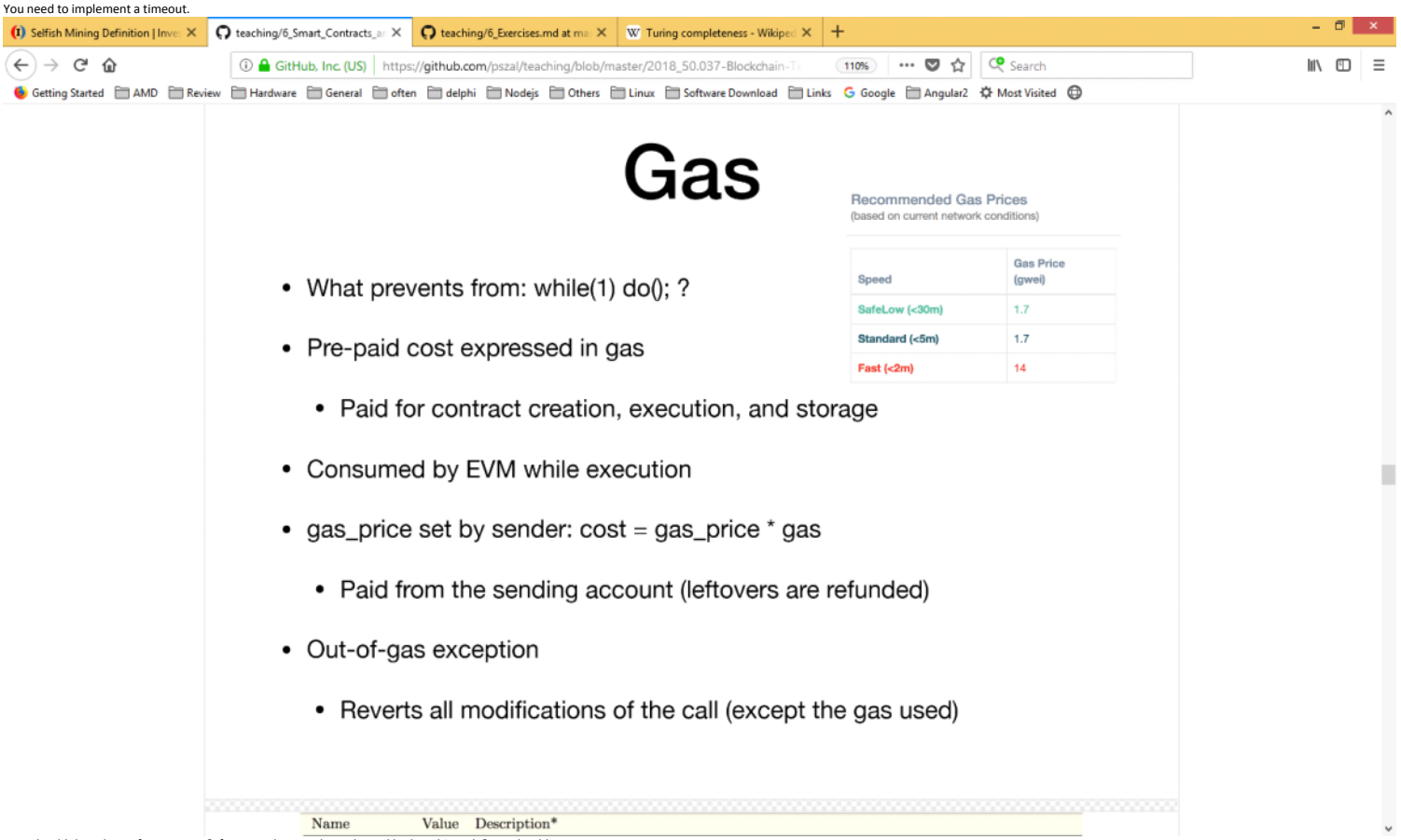
If you are sending this address and destination. This address is zero, you want to create new contract. IF I put transaction = 0, then I want to create a new contract. The payload is bytecode, executed, and stored permanently. The special address is zero, this code is executed and create an

instance of this contract. And assign its address which is derived from the sender and nonce.

It would be funny if you hash the public key = 0, but this is how it works.

It's turing complete language, so you can do some interesting stuff. If I send a contract that does the following: while (1) do something (infinite loop). This EVM has implemented jumps; so what would happen? I would go to this function, and you would go again and again.

You need to implement a timeout.



How should they charge for contracts? If you need to run the code, and look at this code? We should pay them for execution cycles.
Ethereum introduces the concept of gas: for every operation, in this EVM, you need to pay for it. You put your resources, so when you set the transaction here, you can do it like Gas. How much you can compute?

So you would get through this loop, and start looping and start getting money. You are running out of this gas, and you start charging. For every cycle you will pay. As long as you pay it, you can compute. Ethereum has this global computer, it is shared but you are paying for computation. So what you are computing for?

You are paying for contract creation, for execution, and for storing. Storage is the most expensive. You need to specify your price: How much are you willing to pay for executing this? I'm going to pay you this fraction of ether for one gas. This is one mechanism, for prioritizing things. If I want to incentivize them to put my transactions faster. Change the state, and pay you three times more than current other actions.

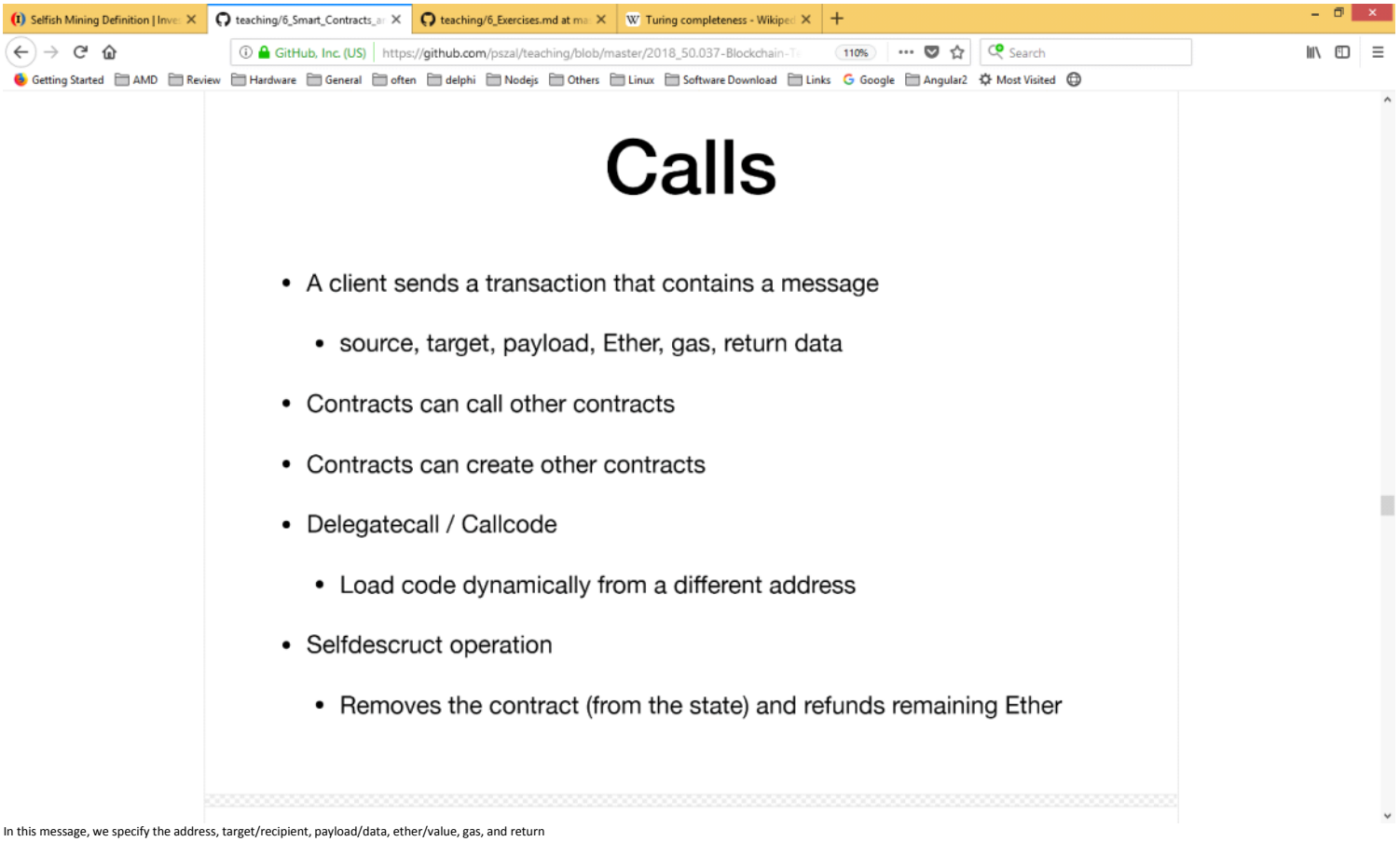
This is dynamic, this pricing market is dynamic.

1 billionth of gwei is ether. If network is congested, you need to pay more gas. You need to pay a higher price for your gas. So cost = gas_price*gas. Leftovers are refunded, so if you compute it, you overestimated how much you need, and the miner will return it. Unlike bitcoin, where miner takes everything.

(Miner all validating, so no cheating!) Out of Gas: if you think about all the modifications, then the developer or user knows its fails, and what's at stake. If you apply partial changes. That may be retrieved. What is annoying here is to estimate the gas consumption

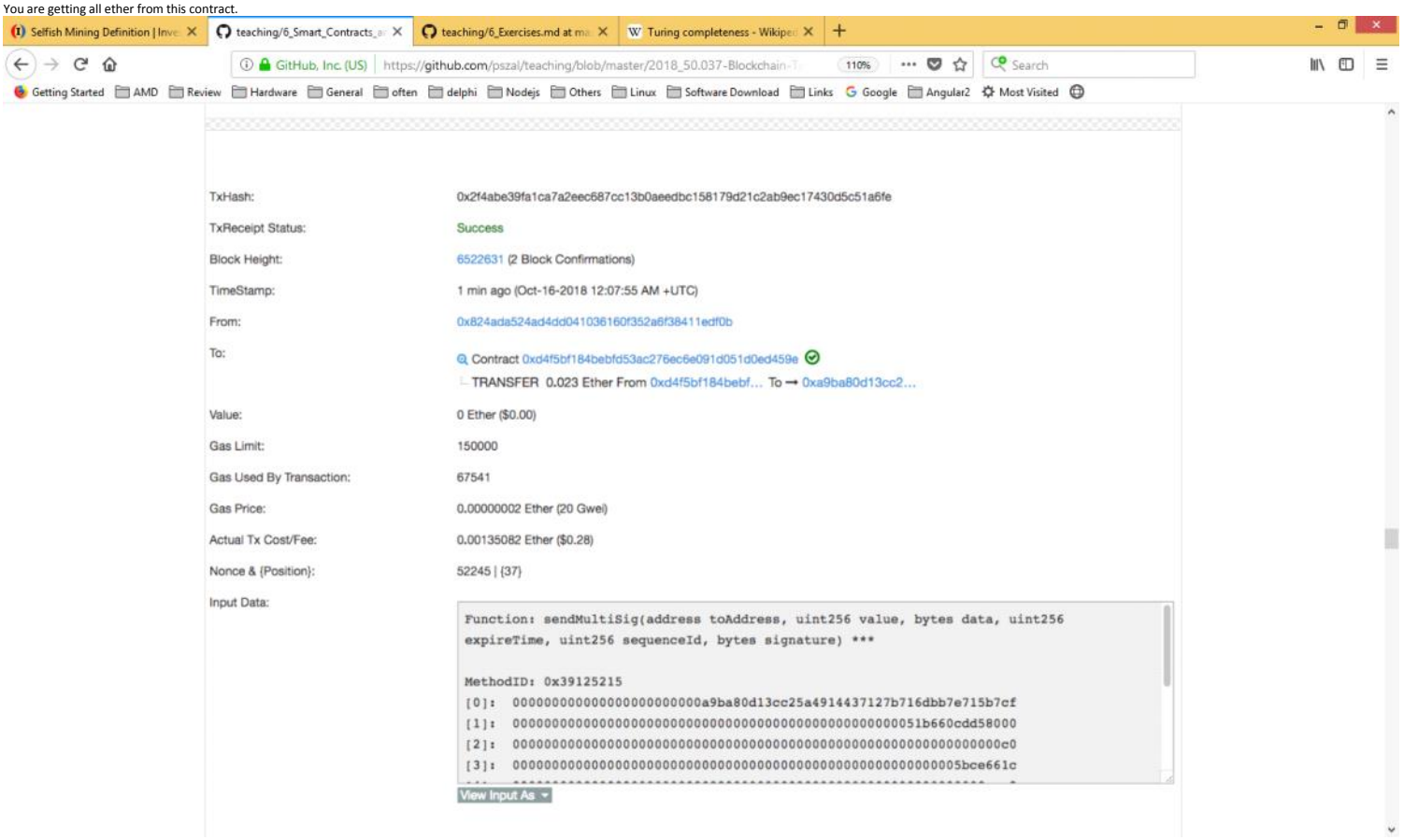
The optimization of the compiler makes it difficult to estimate how much. You can get pretty good estimation from running your local blockchain. Here we have names of instruction, the OPCODE in EVM, and the description and gas price. What is expensive is creating a new contract

If you remove storage, you are getting some gas back. So you can call some cryptography embedded in it.

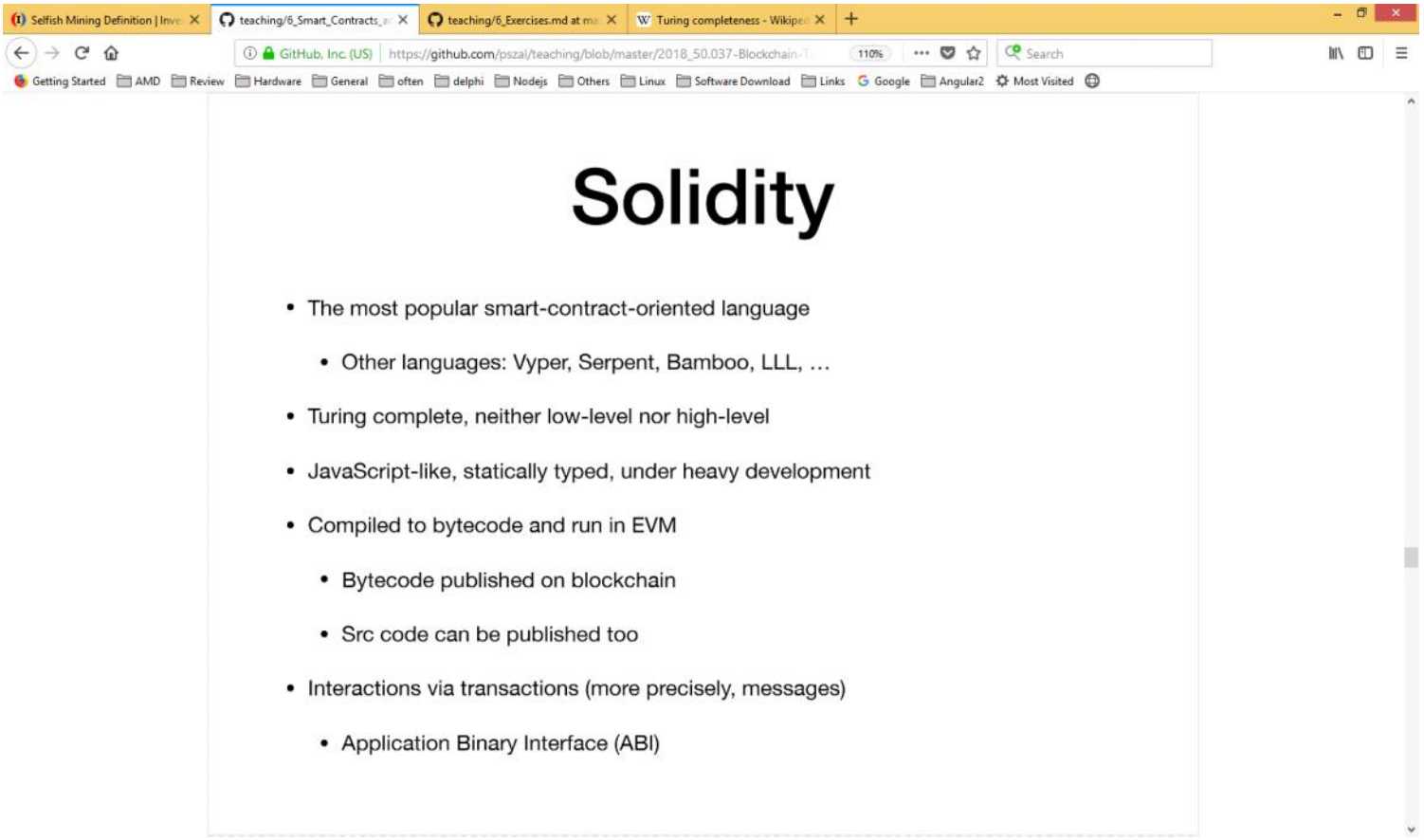


In this message, we specify the address, target/recipient, payload/data, ether/value, gas, and return data. The place where function should return result. Contracts can call other contracts; so you can interact not only between users, but with other contracts. You can have something like load code from a different location. This is really unsafe if you don't know the source of your code; you need to make sure what you are calling. You can remove contracts. By removing, I mean remove the state from EVM. You cannot remove from the blockchain; all transactions stay there, but you can remove it from the current state.

You are getting all ether from this contract.

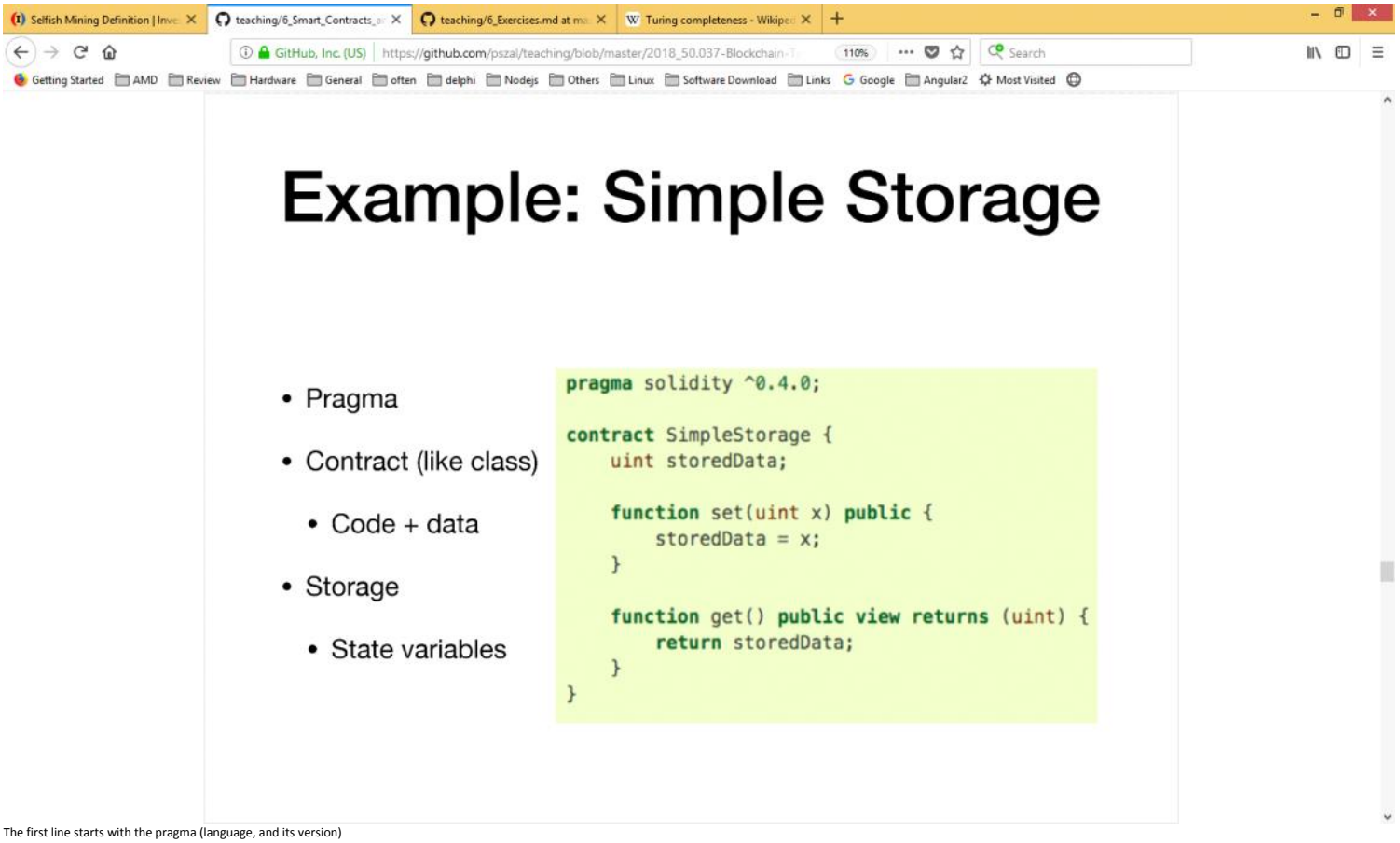


They have smart contracts that implement this arguments, and this is sending a function (sending this money from one place to another). Something you can implement as your code. Predefine and it is limited. Whatever conditions you want to send this money from one place to another.



Solidity is something we should use? Vyper is not very popular. Only the creator of Ethereum is using it. Solidity is Turing complete, no idea if low level or high level, it is Java Script like. Types are static, and no on the fly stuff, and it is under heavy development. See what is possible or impossible. There are many weird things such as multi-dimensional arrays and this kind of stuff. This is something you should be careful about.

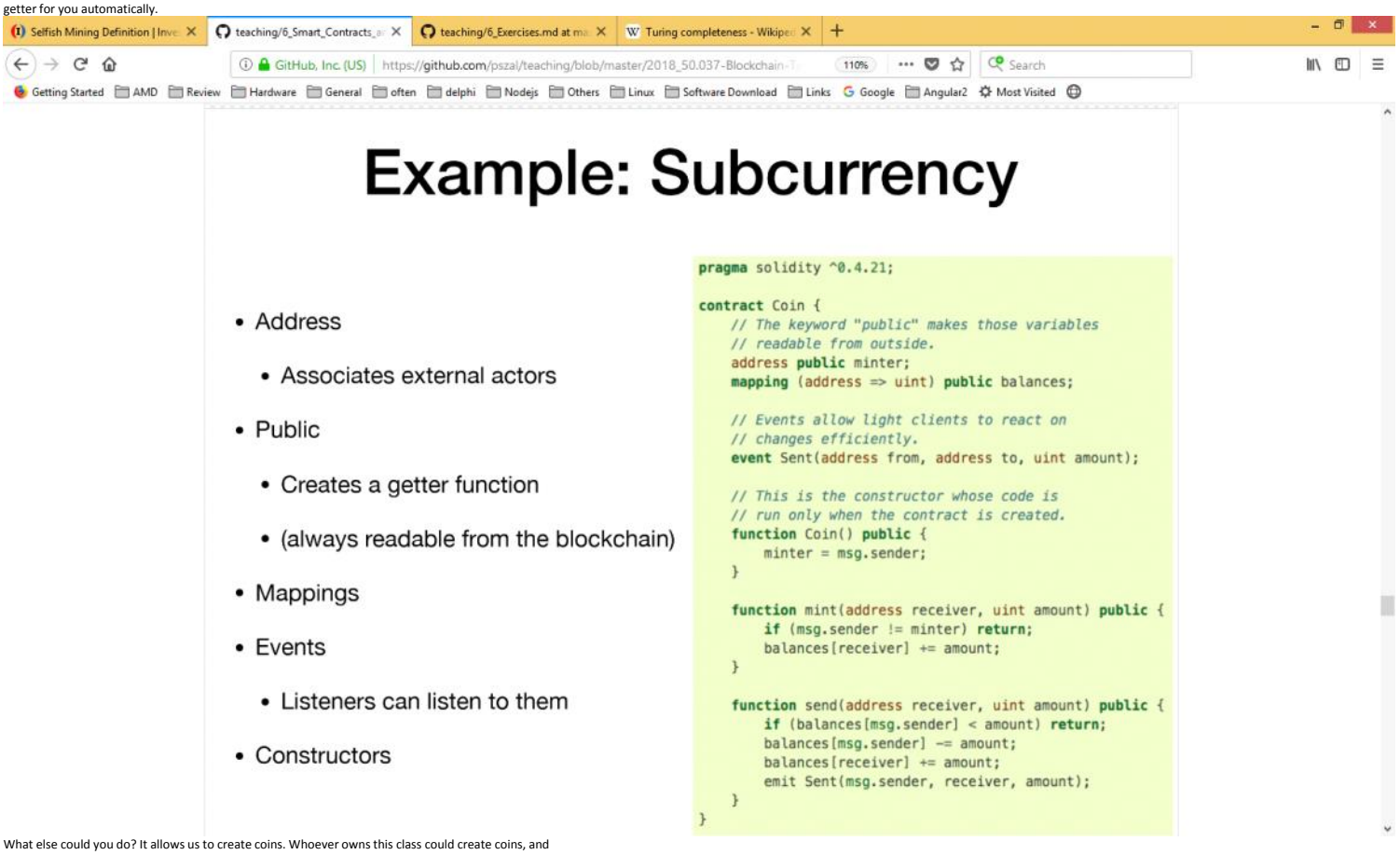
Everything compiles to the byte code, and there are other languages. You can also publish source code. When you are creating contract, you are sending by byte code. But you are publishing the source code. At least it would read like source code and byte code. For this interaction, with this messages. Application, Binary iNterface. Once my contract calls another one, good to know what the arguments, how to call it and this is specified in ABI



The first line starts with the pragma (language, and its version)
Why do we need this? There are many versions running. Some code playing some odd language, and this compiler would work with this version. With the new version, they modified something but it would be nice if they could still use it.

Then we have something like classes. A contract is like a class, we have simple storage, the name of this class. So contract has storage and code. So we assign an unsigned integer as stored data. So you pay for that so you are storing it. Then we have two functions: public (so everyone can call it) and takes some argument X and sets this storage to X.

If this function is called, they will just execute it, and store value x, and modify the state of this instantiation. So getter (public) will return the data. You need to declare what it is likely to return, and will return this function. To avoid anything wrong, you can put it as public so solidity will apply getter for you automatically.



What else could you do? It allows us to create coins. Whoever owns this class could create coins, and allows us to transfer this class to other people. You can have any coin within this contract and send to others. Something similar was running all this ICOs, for people who pay a lot of money for this document

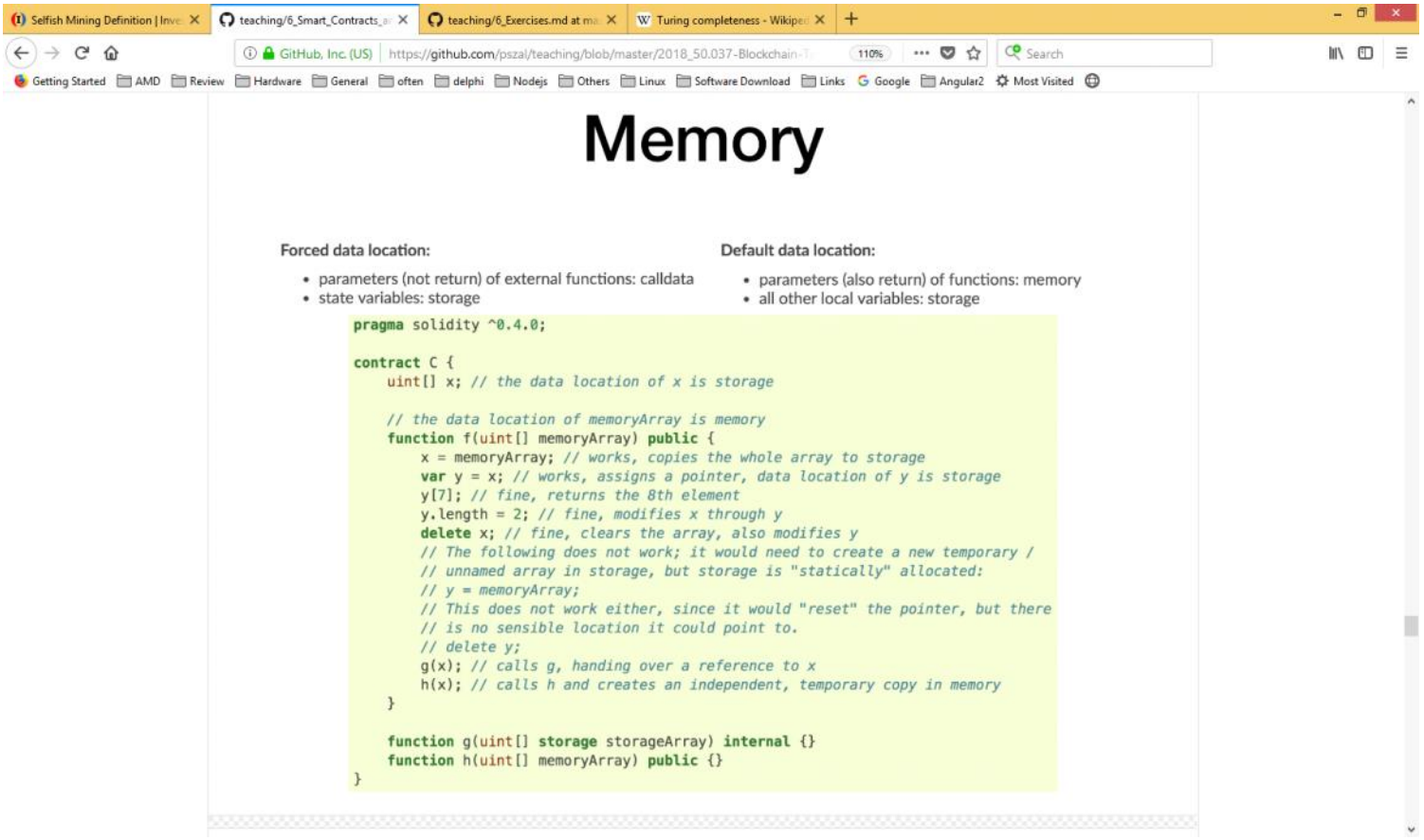
We can use addresses in smart contracts (external actor like smart contracts or users). Here we have the address is public, so the getter is created for that. Then we have mappings (some dictionary, but works the same for accessing and writing to that but we cannot traverse the values) This is in permanent storage, stored by the network and every miner stores that. So you can implement something permanent.

What is also important is that you can declare something like climate: so other processes cannot take it; this is part of abstraction, you can derive what is in this EVM, it just means that other players cannot directly access it. This is a public blockchain though, they can't do it.

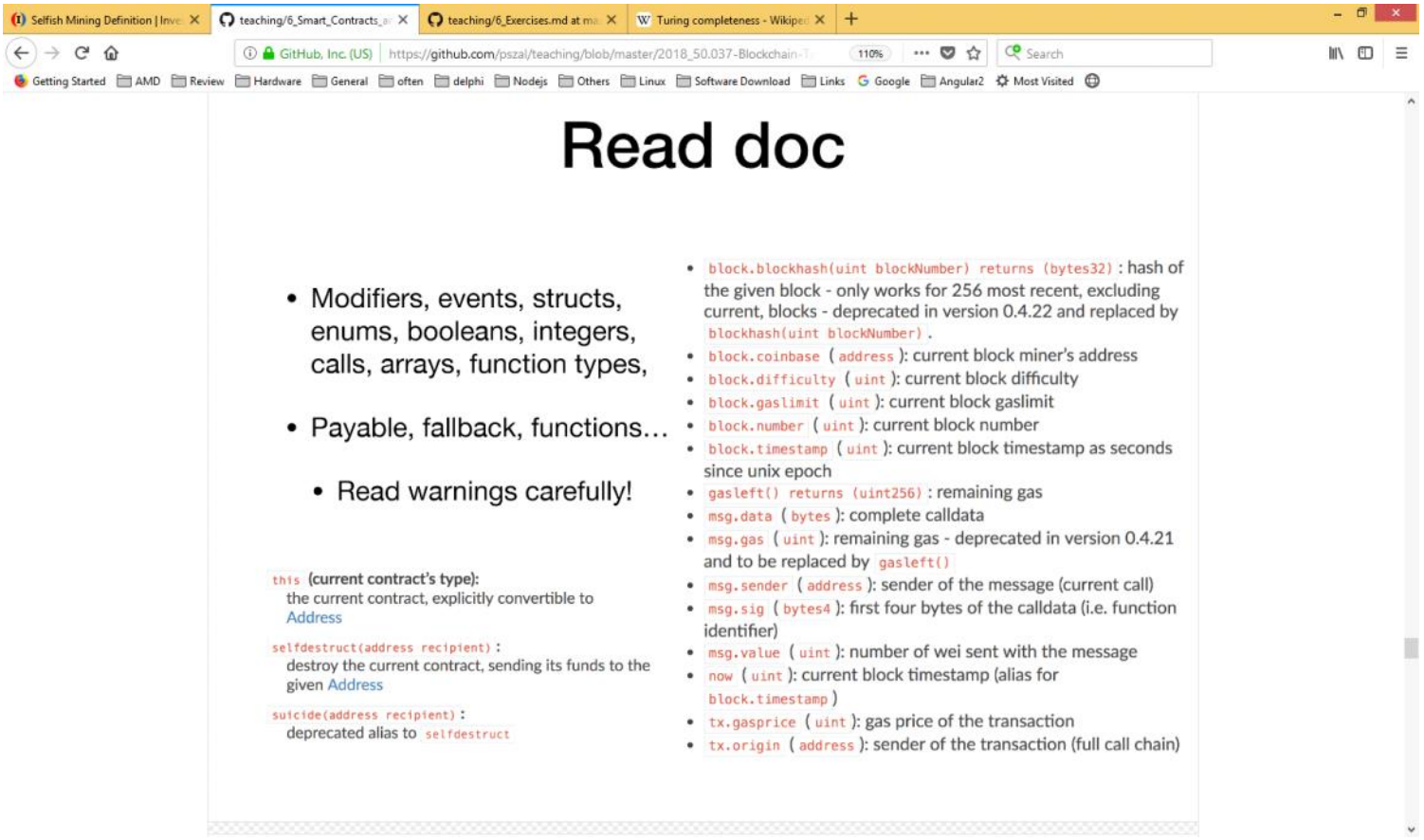
We can register Listeners, who are listening to some events of the smart contracts. If it is even, which is sent. This would receive some refunds, let me check what I receive.

Constructors: before it was the same name as the class name, so they do copy and paste, and this was bad for security. It just calls once, executes and then finishes. Whoever sent this message would be the owner of this contract. So if you want to create new coin, this called is sent by the minter, if it sent by the minter, it will add some amount to receiver. This object is stored in dictionary. So everything at the beginning is zero. And then we have sent function, send function checks whatever it should check. We send the balances to the receiver and sends the balance from us.

This currency transfer is not like Ether transfer; they can create their own ledger within the smart contract and this smart contract is very simple. For the security, if you change it, if you change only this name, this would be a standard function. Everyone would be able to change it to himself/herself. So initially, minter was only called once at the beginning, and if you have liked this much, so everyone would be able to call it and get any coins.

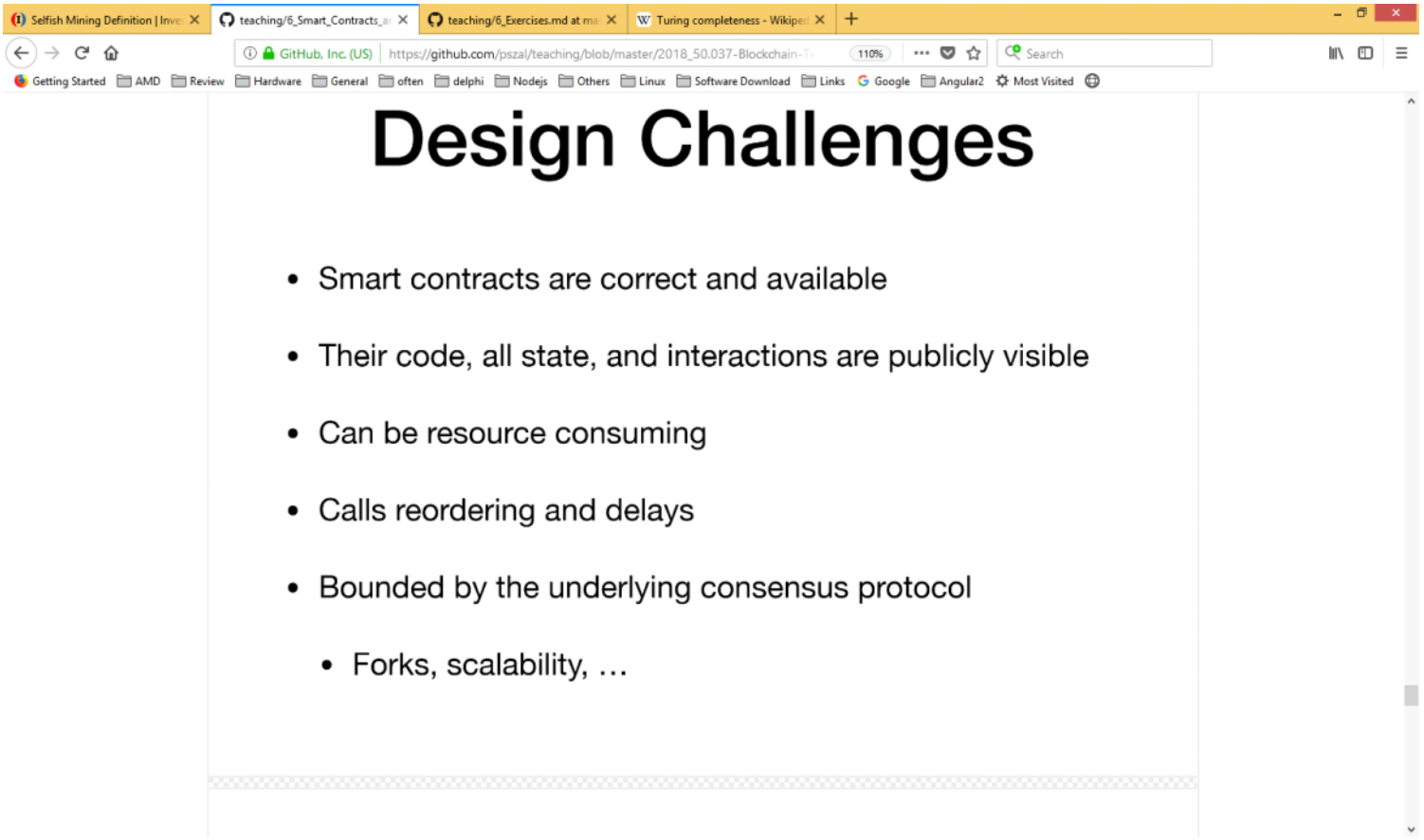


You have data allocation of X. Everything by default is storage in the contract. There are functions (memoryArray). We put the part as storage to. So we can assign (just copy) and we can use variables. We'll just use cast through this type of X. Accessing elements and modify length. Also you can remove arrays.



You cannot return structures or pass structures. You can declare functions, to which you can pay ether.

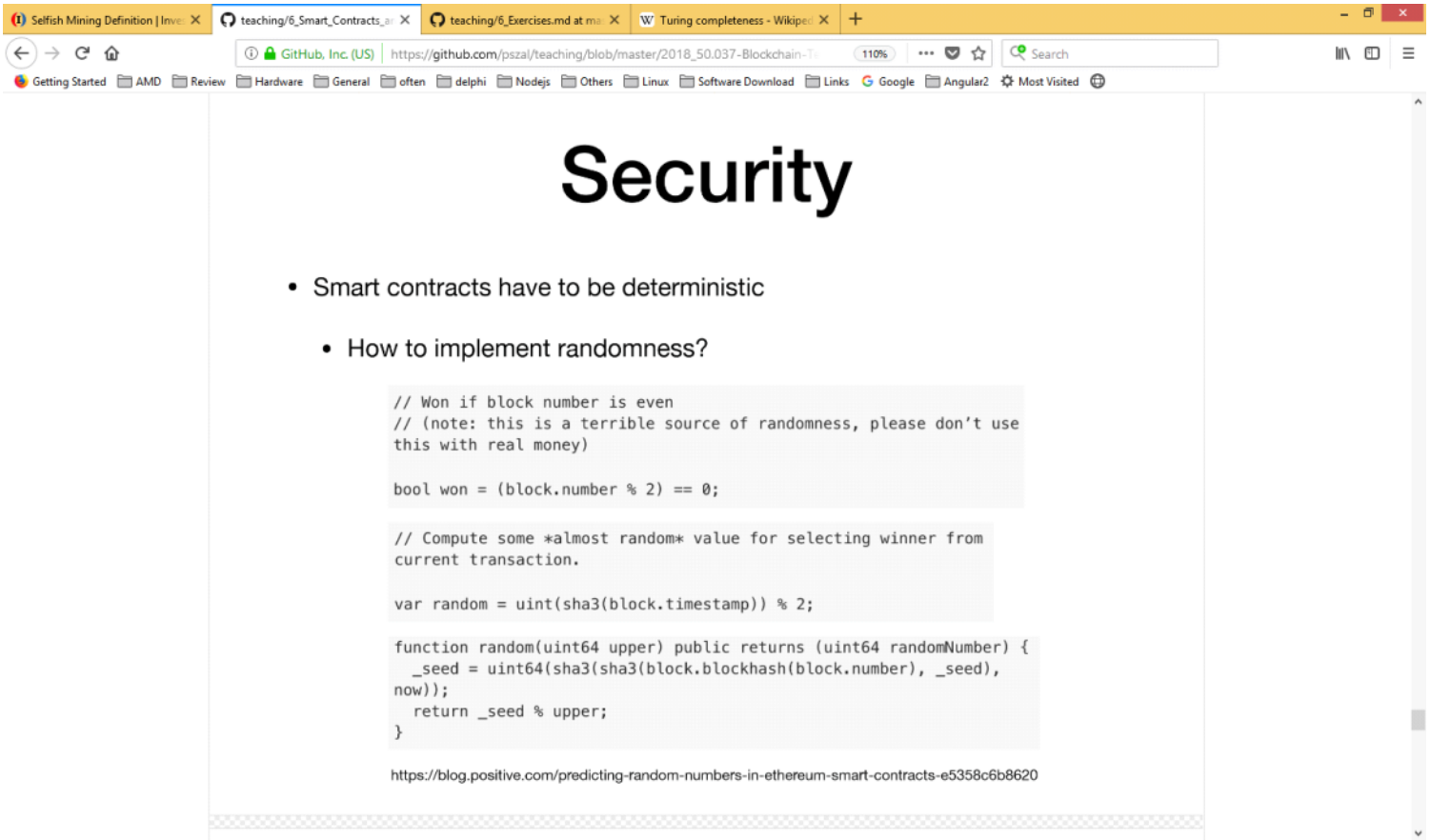
Fallback function is very interesting. If you have smart contracts A and B. If A sends ether to B, B will call fallback. It has no name, so you can implement something (I will do calculate plus for this ether). This contract will be called if this fallback is implemented.



We can assume that if they are on the blockchain, we know they are correct. Because this is by every miner. They are available. This is something given by blockchain.

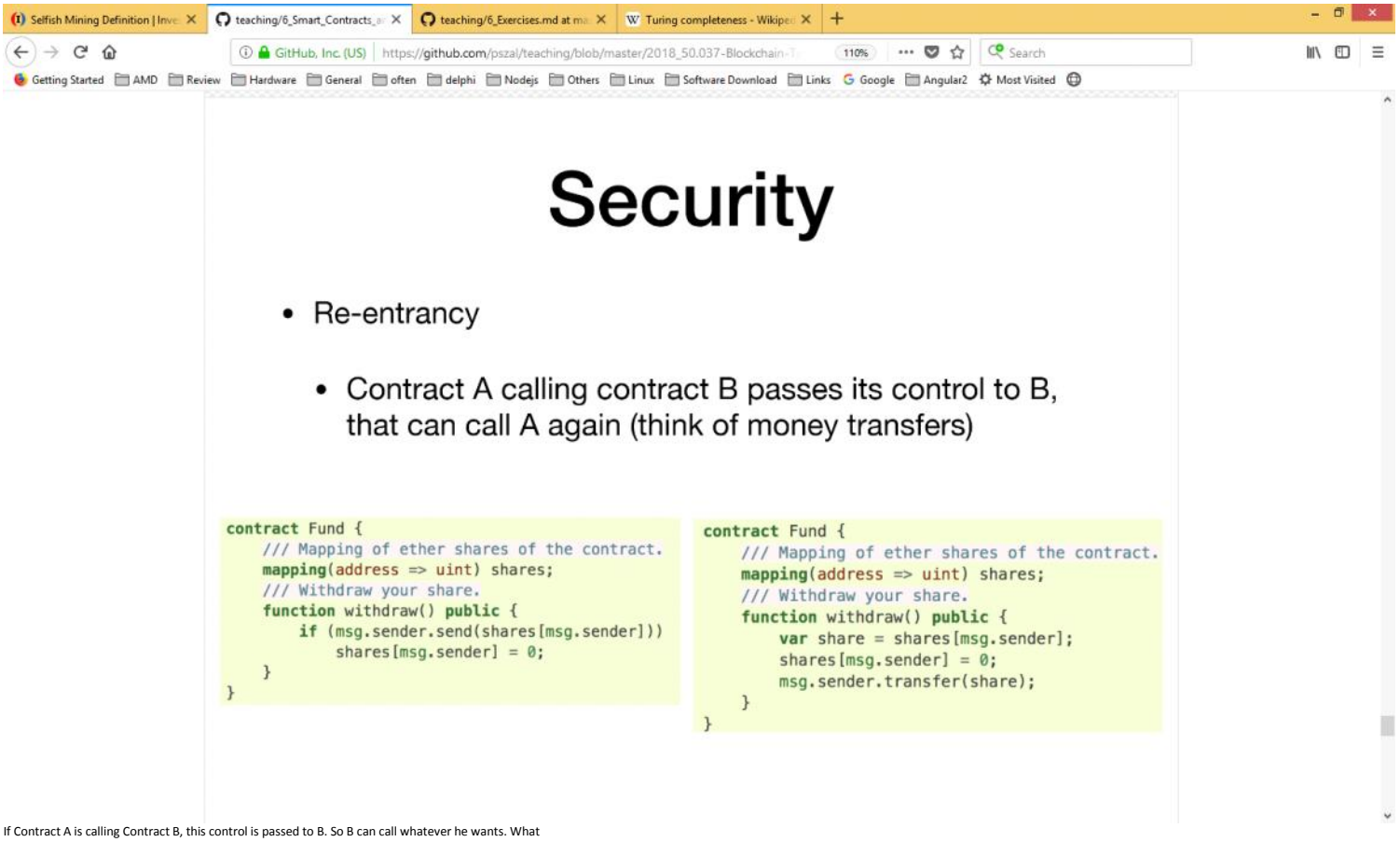
You can try to find bugs, see what's the state, and how much sent resources there are. And it's pretty challenging to write smart contracts nicely, because it may be resource challenging. You have to be careful with optimization because execution time is roughly constant. It takes a lot of gas and it takes no gas.

You should be sure that calls have no order. I don't have guarantee whether they can be used on the blockchain. So the things that are stable are in the blockchain. So the order is not like you call something in your computer. We can have forks on Ethereum. We have scalability which is similar on some bitcoin.



One way is to get like block number. If you are a miner, this is not really random. If you have random, but you use timestamp, sha over this timestamp. Miner could control it; this timestamp is not good, so I'll add it to another timestamp. It is very difficult, it takes like previous block hash. How

it is implemented is to get the future hash of the block. Because then miner can misbehave but he would get his reward. So I will try to mine again. So there are nice ways of playing with it.

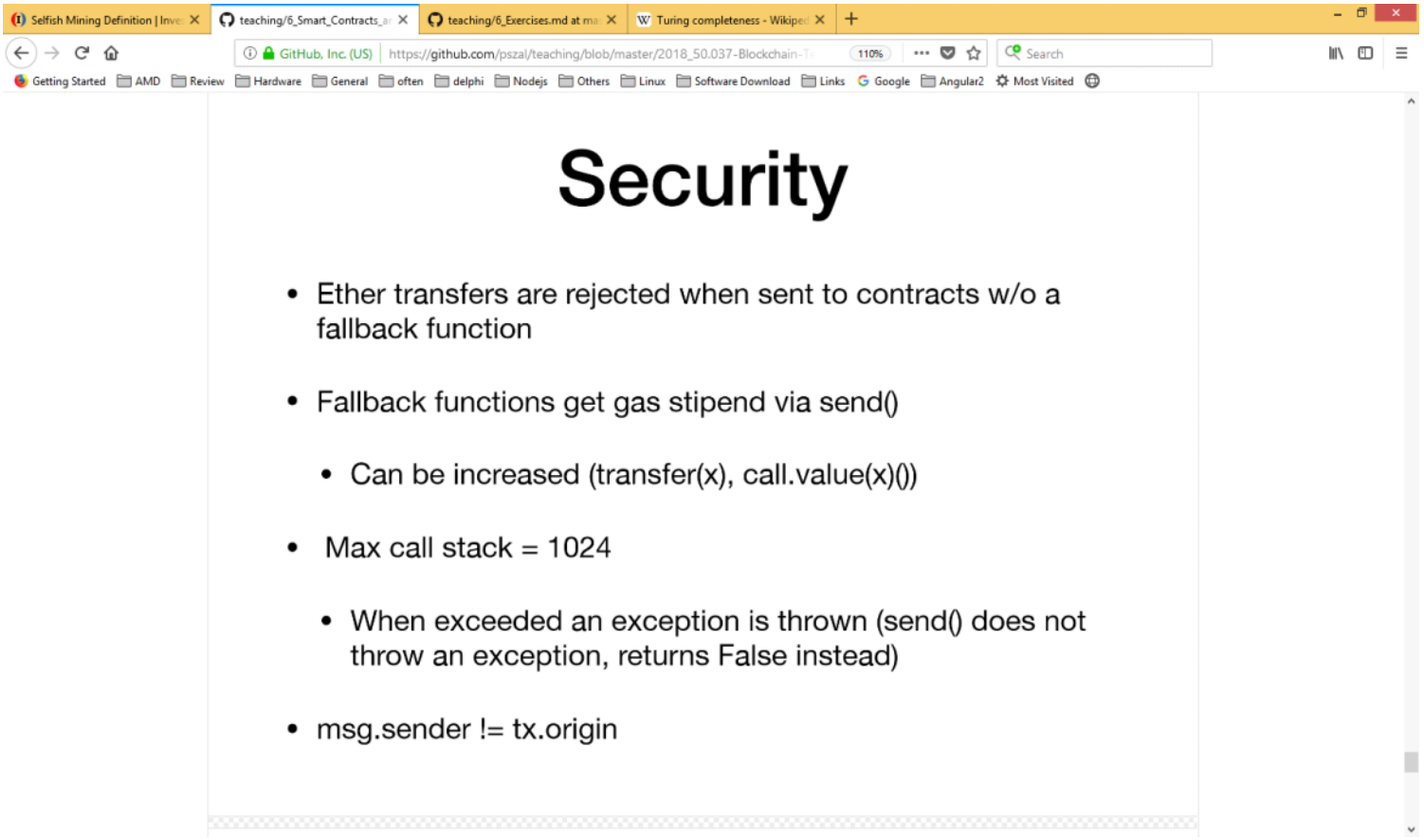


If Contract A is calling Contract B, this control is passed to B. So B can call whatever he wants. What if B calls A again? It can be done. So that's alright, it can be done. So we have code. We have some money in this and we would like to call withdraw. We are sending to sender whatever sender has in their shares.

So let's say that Contract B, in this table of shares has 100. contract A will send 100 to B. So if it is executed, we call shares[sender] = 0. So we have sent everything. Do you remember the fallback function? The contract will receive this 100. this contract can call A (withdraw). So what will happen?

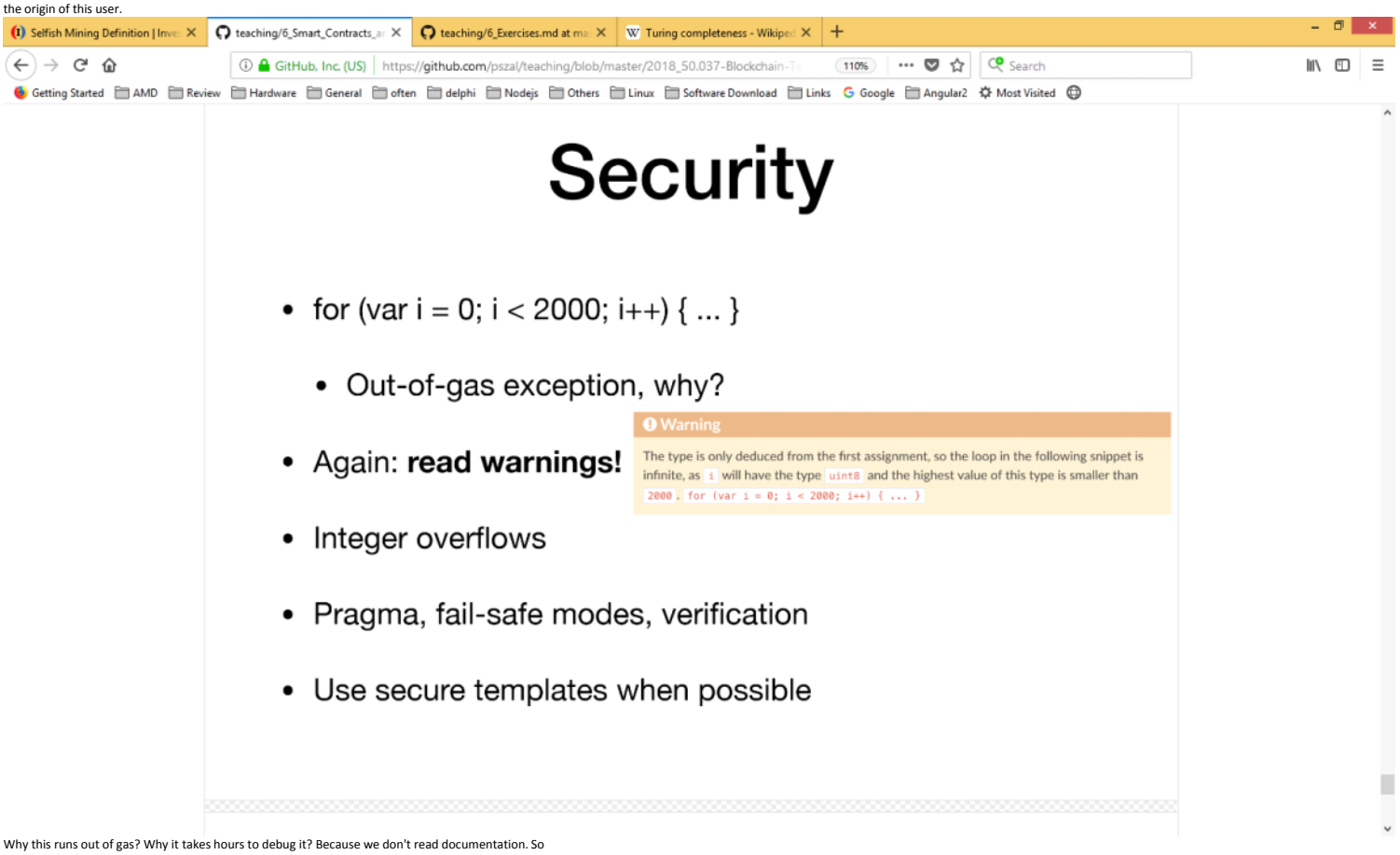
This function will call the withdraw, so if message sends 100, it will send 100 again. So what will happen? This fallback will be called again? This function will run out of gas, except transfer will go through. So how to prevent it?

You can zeroed it before calling transfer. So the fallback function is called, and A will be sending 0 instead of 100.



If we don't have this fallback function, this transaction will be rejected. With this send, it wouldn't work. If it is sent, this sent is giving some gas for this fallback function. So the way it is usually implemented is that it is using other function.

If you call send then exception is not thrown. So user could initiate transaction, but this call from B is the origin of this user.



Why this runs out of gas? Why it takes hours to debug it? Because we don't read documentation. So this function will run out of gas exception. Unsigned integer of one byte is the type of I, so it will take type of I of integer and assign from zero to 255. So this is done dynamically.

It will never go after 255, so it will be looping.

Lesson 8

Tuesday, October 30, 2018 8:35 AM

Ethereum: Project was initiated 2013, released 2015. It was the first project for implementing the smart contract.
In Bitcoin: capable of executing code, but inefficient and difficult to implement this kind of agreement. Ethereum wanted to do it better.

So like Cryptocurrency, we have this machine with miners inside. Who run some sort of consensus protocol. Out of this protocol we have a blockchain. And how users can interact with this blockchain, they can send money to themselves. So we can have standard transactions that are appended to the blockchain. So user B sends money to user A. User A can publish some code. And this code would be appended here.

So then another thing which is nice is the submission of data. B can now submit some data, the network will append it and this data will be associated with this contract. As it is a code, we can send calls.

So we can have user C to send to this network. He just calls function 1. This is also sent to the blockchain, does some work later. And this execution fold will be added here. Every miner will execute it. So this function 1, can do many other things like executing another contract which transfers money from A to B. So this code changes the state of the network. Which means that with every block, we need to encode some state. This state could be derived.

Ethereum associate some state with every block. So we append everything to the blockchain and determine the consensus. We can send money, we can publish code. We can associate data with this code. If we have all of those, the miners run all sequences of this event. Blockchain only orders that, but we run them just to keep this state. This is called like replicated state machine. Everyone does the same and should get the same result.

It is difficult to do some probabilistic thing, but there are assumptions. How to have random functions within solidity? You can have some kind of construction which are time locked. This is verifiable. So you calling something, and the block is mined. So it is some kind of proof of work, but randomness is difficult. It's from an external source, not our operational system. So it is a deterministic kind of thing.

Everyone does the same thing. It is designed to be consistent. If we can agree on the sequence of the events that we want to agree on this blockchain, then we run the same outputs, inputs for the same code because the language is deterministic.

If we assume that the network is ideal and there is no latency. So the speed of the slowest CPU is the speed of the network. You cannot parallize it; so everyone has to calculate this output and agree with others. If everyone has to verify that they did it correctly. If this function which is called here, which call another, and which transfer money. If the miner was misbehaving, this block will be rejected. Other miners will reject. If you spend output which is invalid, this block would be not accepted. To verify it, this takes time, so the whole thing cannot be faster than one CPU.

It's like a global computer. It's a distributed computing machine with money transfer. If you think about it more, the worst thing is that they are replicated, so throughput is limited by capabilities of one single miner. What is good about that?

Backups! It is completely robust, this system. Half of these nodes can crash and it will still work. If some are alive, it will be working. If you have replication you are limited by CPU. But on the other hand, it is difficult to deal with this tradeoff. They think, how many machines should replicate, should they pass it. So like Bitcoin, the state was much more complex, so there is unspent output, but more variables to be set. So they basically run, they see a different view, it means that the block is corrupted.

They knew about selfish mining, difficult to modify the system, with slow blocks. In Bitcoin, we have 10 minutes. ASICS: mining is very centralized in bitcoin and they want to improve on that.

Design Principles

Where is complexity in OSI? It's in the middle layer, the transport layer. So we have the internet, with points A and B and they want to talk. A wants to send packets to B. So there are like three or two to describe. The guys in between know how to send it. They are sending here. It's like sending. So post office, this should go abroad, this should go to Malaysia. And this is received. There are many issues here, but like the concept, it's very easy.

However, if you want reliable transport, this is where things can get complicated. This layer gives you best effort, so this is the unlikely thing because you may never know. If you see the API in the application layer, it's easier compared to understanding the TCP. They want to build upon UDP and change operation systems. So they insert a layer (presentation?)

So in our case, we have common space: blocks in the order. This should give us the order of events. We get the Ethereum Virtual Machine and State. We execute everything given by this layer. We are pushing up transactions, codes whatsoever. What do we have here? Solidity or other languages on top of that layer. This EVM + State is difficult, but is sandwiched by the common space and the Solidity code.

So this consensus protocol, this is something new, no one understands how it works. Keep it easy to implement. It is complex, this EVM stuff is not as complex as JVM or other virtual machines. We know how to build the graphing. And Solidity is high level (not as much, but developers should find it easy)

Freedom: do not restrict users (unless necessary) and combine this with the sandwich complexity model with generalization. It's great to have blockchain, but no application for that. Ethereum says that we don't know, we don't want to guess, so let's give like everyone the power to implement their own application, and the goal is that Ethereum is general as it is difficult to predict use-case. With no constraints or negligible.

NO Features: They don't want to change too much here. I'm running this application, why don't you optimize Ethereum for my application. But they didn't want to optimize it for special usage. They do it a bit though?

Non-risk adversion: The community is pragmatic, so if you do something that changes the layer, and does a big job, they will implement it. They want to improve the code, and they make risky decisions.

Accounts Model
The core thing here is an account balance model. In bitcoin, we have unspent transaction outputs. There's a discussion which is better: they claim efficiency, but...

E.g. make a lot of empty machines. A lot of transactions make random accounts. So small amount of Ether. So preventing them from doing that is to charge them for transaction fees. So one big thing: the table of addresses and accounts would be huge.

Simplicity: This address balance is much simpler to keep and implement.

Transactions and Message
Transactions are sent from this external guy, pushed to the network. With transactions we have messages. If one smart contract calls another, this sends messages to the address. Transaction contains addresses.

We have message calls which transfers money, which calls the function, and we have contract

creation. What do they do? Each execution is paid; everything you send is paid using the Gas Price. Gas Price is how much you want to pay for the execution of your transaction. It goes down to this EVM, and the state change, and you need to specify how much Ether you are willing to pay for this execution. So the cost of execution is spent in gas. Now we can give a price to this gas. So this transaction is important, but I'll pay you more for it. They will execute it, they will apply state change and they will get this money.

Then we have a Gas Limit: the maximum amount of gas that the sender uses to be consumed by this transaction. IE. The maximum amount of gas to be used in while executing a transaction (called also the Gas Start)
Usually, the person doesn't know how much gas, so they try to estimate that but it's very inaccurate.

To: The 20-character recipient of the message call. It can be a smart contract. It depends on the compiler version. EVM has some internal optimization. You can simulate that, estimate that. So we are sending To: it's the address.

Then we can send like money to the contract. (Value: # of Wei to be transferred to the recipient of a message call)
Nonce: # of transactions sent by the sender
Signature: ECDSA signature to authenticate ender
INIT: (opt) an EVM code that initializes new contract accounts
Data: (opt) parameters of message calls.

Consensus:
Minimize the Bitcoin's 10 minute limit? Why does it have that? Because of network propagation time? If you decrease it, it will take longer to mine the current branch. So you have to wait one hour to converge. So you don't know what's going on in this network because of latency. So they want to have 12 seconds (stale blocks are used for determining the strongest chain, so their finders are rewarded but w/o tx fees)

In the Bitcoin protocol, the longest is the strongest rule, they implemented this protocol called GHOST. It doesn't get the longest chain, but it analyzes the graph and it takes the heaviest subtree.

So it goes seven blocks back, and calculates the heaviest subtree. What the heaviest subtree? They'll take the longest chain. If you have a chain which is very long, this big ass subtree with multiple branches wins despite the long chain because it proves most of the miners are on the big ass subtree. Selfish miners do not win. We don't know whether what is implemented in Etherum whether it implements this property?

So the big ass subtree is called an anchor, so the finders of this block has rewards. So anyone in the big ass subtree/anchor will get reward, with who gets what being on the long chain. Those who build those heavier trees, the inflation is higher than predicted. That's because of the reward of the output.

How often the bitcoin updated? Every 2 weeks, difficulty is updated. But Ethereum? The difficulty is updated with every block. In ethereum, they have really good synchronize time. They require you to synchronize with NTP server.

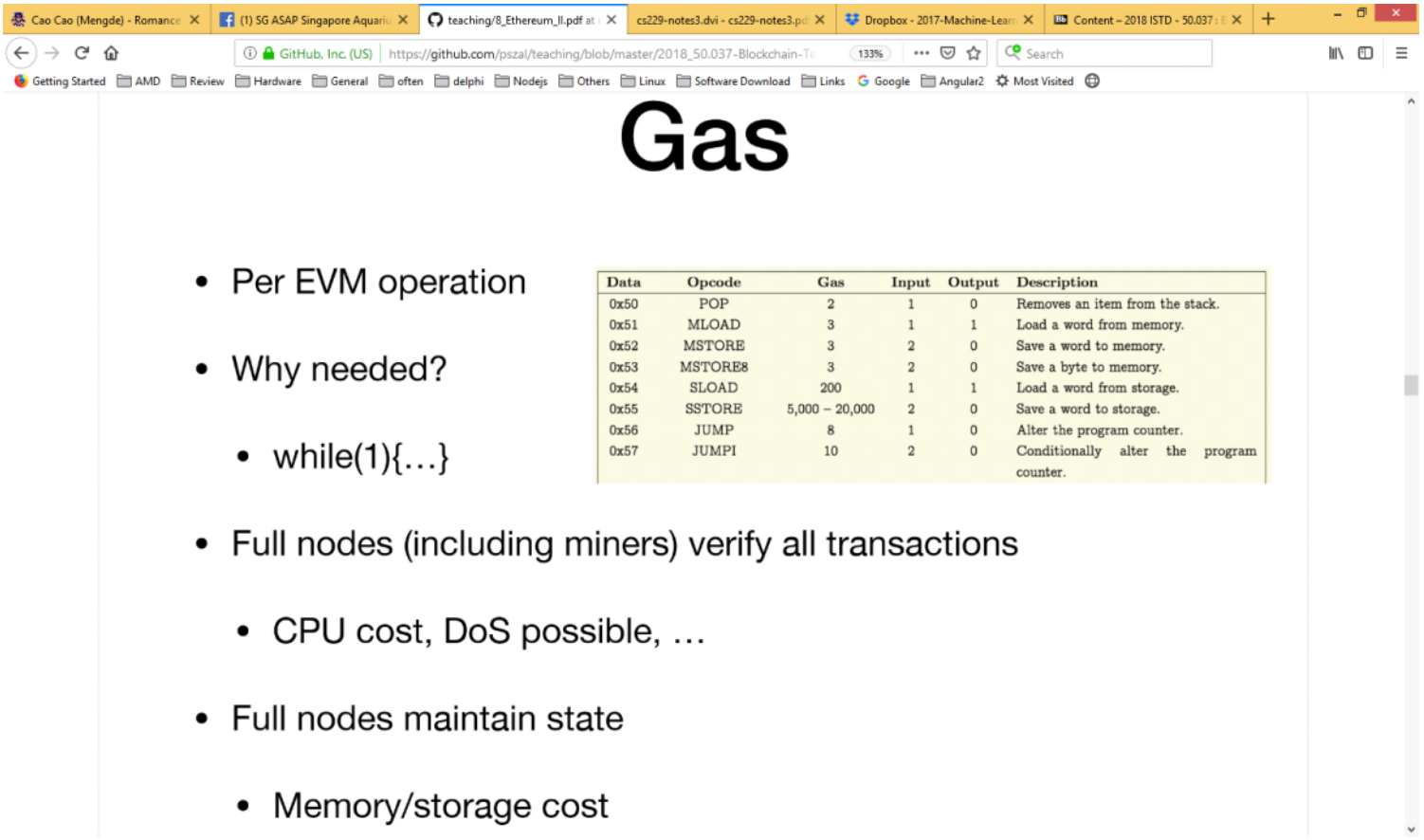
Mining Design (POW)
Mining works exactly as Bitcoin. We have some function and blockheader and we try to brute force this function using the header function. So the miner keeps computing some function, check whether the output is less than some target. They want to modify this function. In Bitcoin, it was SHa256. Here they want to change it.

So what was the design of miners. First point is

1. IO Saturation: They want to prevent ASICS. So no supercomputers! Memory is pretty close to the computing unit in computers, so if you want to retrieve from memory is fast. So if you optimize your function to have many lookups, you can have a dedicated unit to talk to. Consume nearly the entire available memory access bandwidth.
2. GPU friendliness. So ideally you optimize for CPU, because there are too many challenges. And what's the difference between CPU and GPU? Slowness? It's impossible to optimize CPU than it will be perfect
3. Light Client verifiability: A light client should be able to verify a round of mining/check the headers in a fraction of seconds and a few megabytes. In Bitcoin it's easier Here, they have some parallels.
4. Light Client slowdown: If you like to mine using light clients. So I will not do all of this validation, I will just be publishing all solutions without validations. So the user send whatever they want to have, I will just append this transaciton and append the blocks. Light clients are slower than full miners
5. Light Client Fast Startup: a light client should be able to become fully operational and able to verify blocks within seconds.

ETHash
You need to have a lot of memory, and you need to access the memory pretty frequently. Do you remember Bcrypt, Scrypt? They do something like this:

1. You have blocks. New block has arrived. So from this block, you generate some seeds. In a deterministic, simple way.
2. From this seed, this seed is short, so you will generate some cache. This is 16MB. You will store the cache
3. From this cache, you generate some data. And this data is heavy, 1GB. If you generated that, you want to create some new block header. You run the function such that you access the data from this function and you concatenate the data from this 1GB. If you have less than target, you have found the solution.
4. If you find it, announce the solution. Light clients can verify the solution only from the seed and the function. So the client can get the seed and the hash and verify if the block is correct.
5. Client knows which data and how to get it from here. From the cache, a 1GB dataset is computed (each database item depends on only a small number of items from the cache). Full nodes store the dataset. The dataset grows linearly with time
6. Mining involves grabbing random slices of the dataset and hashing them together.
7. Verification can be done with low memory by using the cache to regenerate the specific pieces of the dataset that you need.



Gas

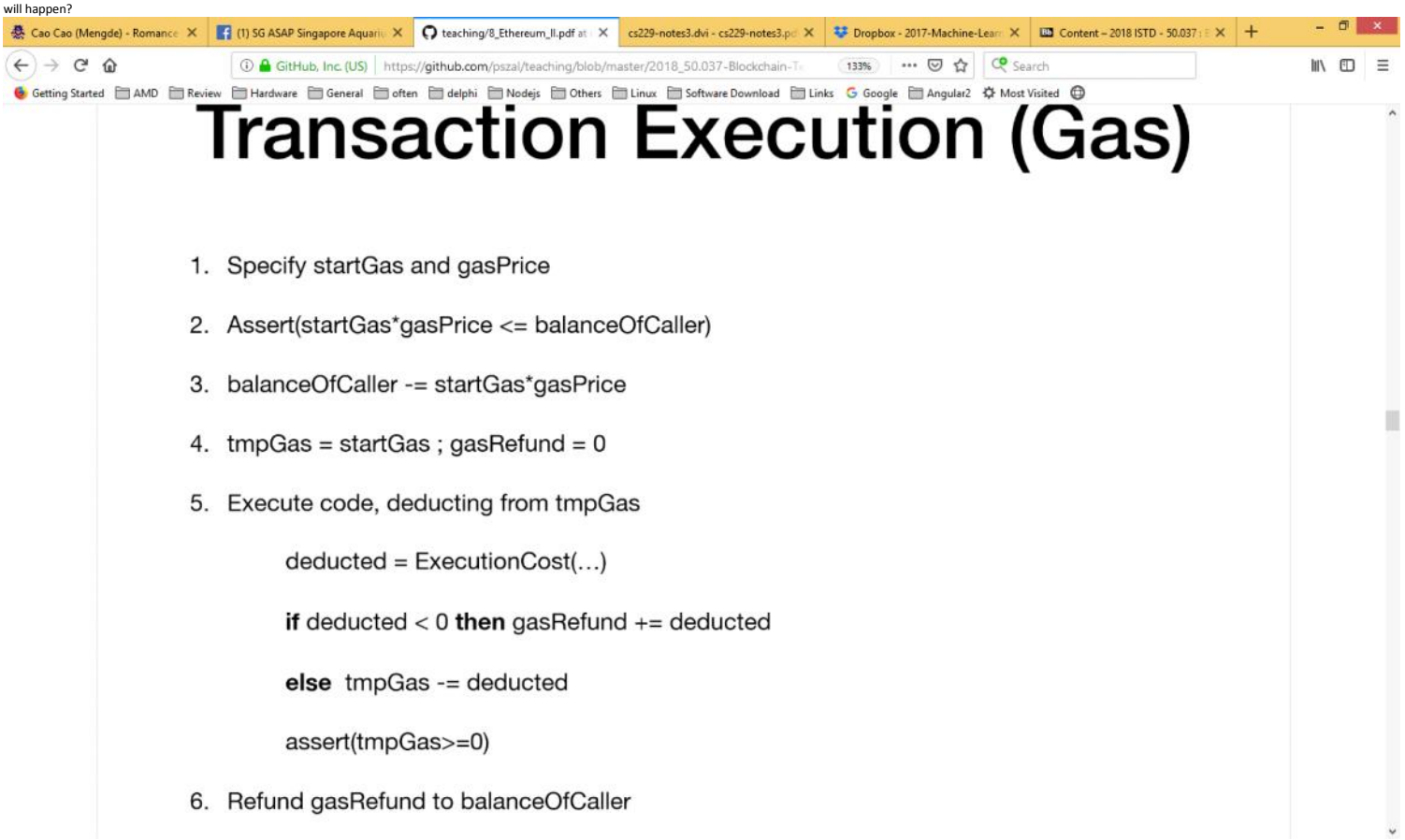
- Per EVM operation
- Why needed?
  - while(1){…}
- Full nodes (including miners) verify all transactions
  - CPU cost, DoS possible, …
- Full nodes maintain state
  - Memory/storage cost

Data	Opcode	Gas	Input	Output	Description
0x50	POP	2	1	0	Removes an item from the stack.
0x51	MLOAD	3	1	1	Load a word from memory.
0x52	MSTORE	3	2	0	Save a word to memory.
0x53	MSTORE8	3	2	0	Save a byte to memory.
0x54	SLOAD	200	1	1	Load a word from storage.
0x55	SSTORE	5,000 – 20,000	2	0	Save a word to storage.
0x56	JUMP	8	1	0	Alter the program counter.
0x57	JUMPI	10	2	0	Conditionally alter the program counter.

We have this publishing of code. The code is turing complete, can do what we want. So the pricing is pretty complex.
Why do we have this prices? We need to pay something. How much you are willing to pay. This is my transaction, I will give you 0 fees. It may not go through but I won't care.
Needs to prevent the while eternal loop and this will not terminate. If you are paying for each execution, they will be happy to pay for that.

So storing data in permanent storage is quite expensive, you pay a lot of gas products.
Full nodes verify all transactions.
In Bitcoin, this script is longer. And you have different costs such as CPU cost, DoS where you have a block with a lot of content, there's a lot of transaction and there might be a fake transaction? There are limits to prevent these.

There are memory/storage cost, where miners can maintain the state. If you send transaction, what will happen?



Transaction Execution (Gas)

1. Specify startGas and gasPrice
2. Assert(startGas*gasPrice <= balanceOfCaller)
3. balanceOfCaller -= startGas*gasPrice
4. tmpGas = startGas ; gasRefund = 0
5. Execute code, deducting from tmpGas

   deducted = ExecutionCost(…)

   if deducted < 0 **then** gasRefund += deducted

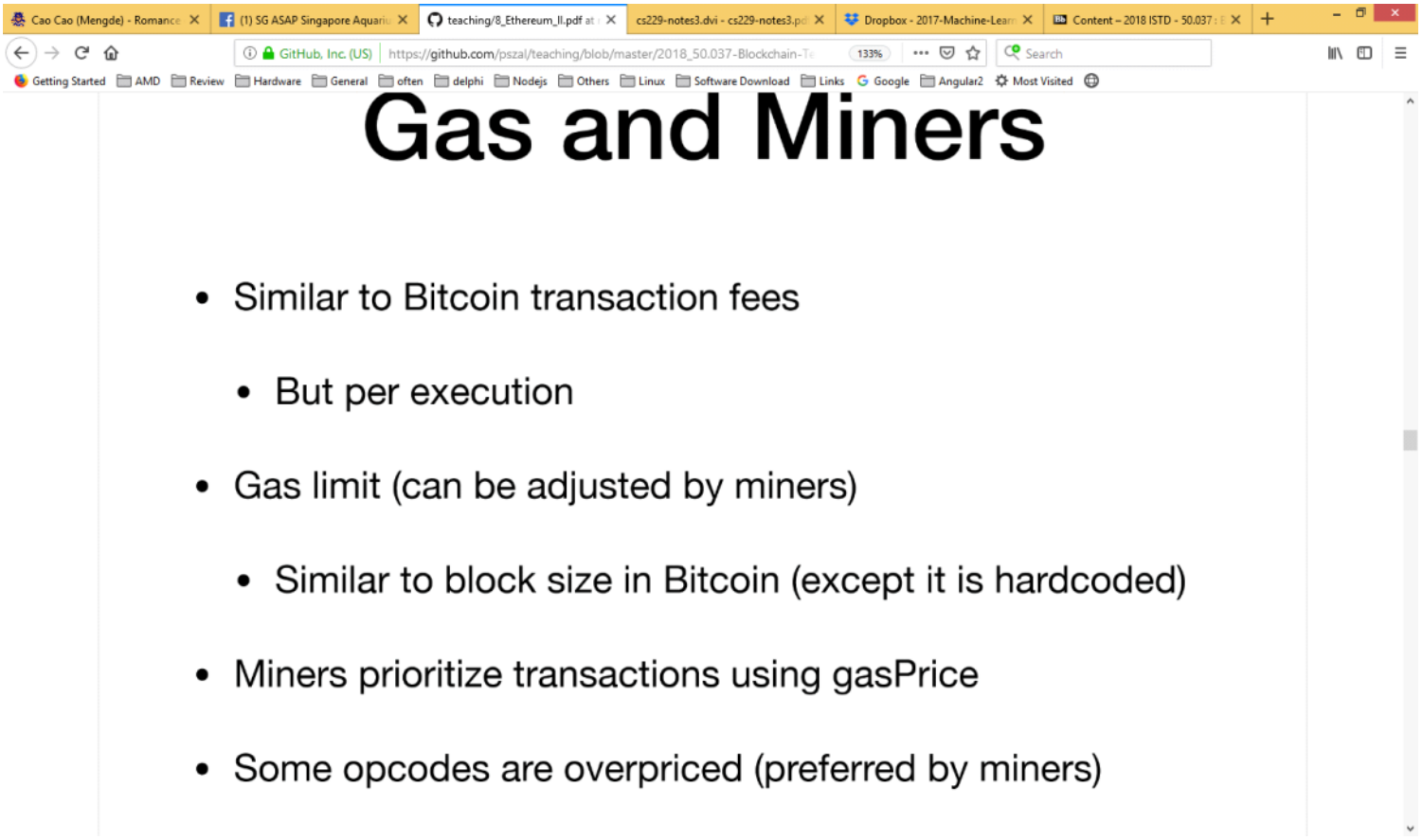
   else tmpGas -= deducted

   assert(tmpGas>=0)
6. Refund gasRefund to balanceOfCaller

In your transaction you have starting gas and gas price.

So the sender will have gas price and the sender asserts start gas price. I will want to pay 5 for each cycle. Can you afford that? In your balance, we know who sent this transaction. You have this money, I will execute. So then what happens? In this EVM machine, this transaction is executed.

If you are creating a new contract, some code is created to execute this contract. How can the cost be less than zero? A miner should be paid for executing something? Some operations are accessing storage, so if you are clearing your storage, they will refund it. Otherwise, we are deducting from tempGas.
At the end, we assert that we can deduct, but not add temp gas.
We assert that if Temp gas is like that. If it is, we execute something we shouldn't be executing, the gas refund is given back to the caller. It will revert all state of this operation.

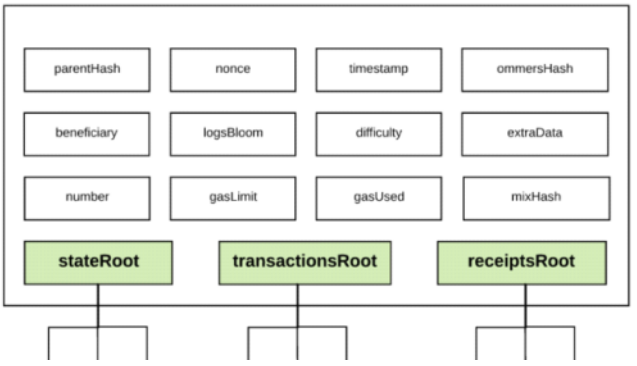


We have gas limit per block. We have how much gas is aggregated. So all transactions are computed and it has to be less than the gas. This is again for what? Why do we have this gas limit per block? So that the block is not too big?

What does it mean that miners are agnostic to what code you are running? Some opcodes are overpriced (preferred by miners). Price is set pretty high, so the miners will take this overpriced opcode. Storing the data is not very computing demanding, but they prefer for instance those.

Block Header

- Fields known from Bitcoin
  - parentHash, nonce, timestamp, and difficulty
- Uncles (ommers) hash
- Beneficiary: instead of coinbase TX
- Gas limit: gas limit per block
- Gas used: sum of total gas used by block transactions
- Extra data
- mixHash: helps PoW
- Roots and logsBloom



We have transactions and we have blocks. So we have some fields from bitcoins. We have nonce.
We have timestamp. We have some fields that encode difficulty.
We have the hash of Uncles/Huge ass Subtree. This has to be encoded in the header to verify it.
We don't have coinbase Transaction. We have only the beneficiaries, those who get money from the system.
We have gas limit, gas used /sum of total gas used by block transactions. Everything was run and so on.
We have some extra data. This is like for future use cases, and differentiate another header. This is to verify that a given block header is for some efficient proof of work.
We have some tree and a bloomfilter.

Cao Cao (Mengde) - Romance | (1) SG ASAP Singapore Aquari | teaching/8_Ethereum_II.pdf at | cs229-notes3.dvi - cs229-notes3.pd | Dropbox - 2017-Machine-Learn | Content – 2018 ISTD - 50.037
GitHub, Inc. (US) https://github.com/pszal/teaching/blob/master/2018_50.037-Blockchain-Te 113% Search
Getting Started AMD Review Hardware General often delphi Nodejs Others Linux Software Download Links Google Angular2 Most Visited

Block Header

- Every header contains roots of three trees:
  - state tree: representing the entire state after the block creation
  - transaction tree: representing all transactions in the block
  - receipt tree: representing the "receipts" corresponding to each transaction
- Receipt:
  - medstate: state trie root after processing the transaction
  - gas_used: the amount of gas used after processing the transaction
  - logs: is a list of log items generated during the execution of the transaction
  - logbloom: is a bloom filter made up of the logs in the transaction.
- logsBloom in the header, accumulates all transaction BFs

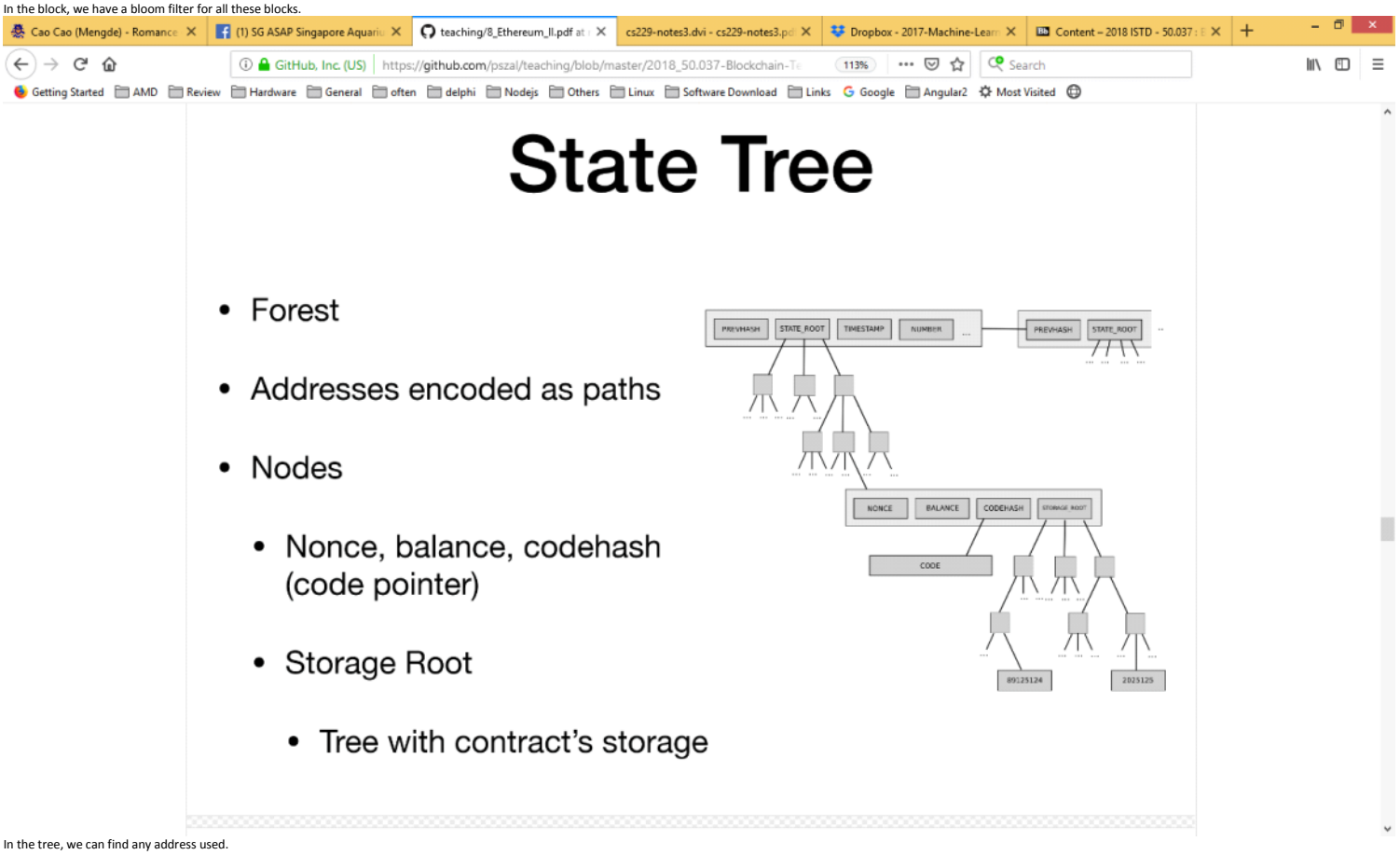
We have a transaction tree. Collecting all these transaction sent to the network and then appending them. Another one is receipt tree which calls the objects called to receipts. It's basically some kind of metadata which allows you to debug this stuff. If you play with solidity. For every transaction, you can check how much it is consumed and how much you are paying and so on.

Receipt: It contains the medstate: your transaction was applied and what was your state change for

your transaction. How the transaction stores the global state. Then we have gas used: how much gas your transaction consumed. There are special opcodes that allow to log some things. You can put some debug data here.

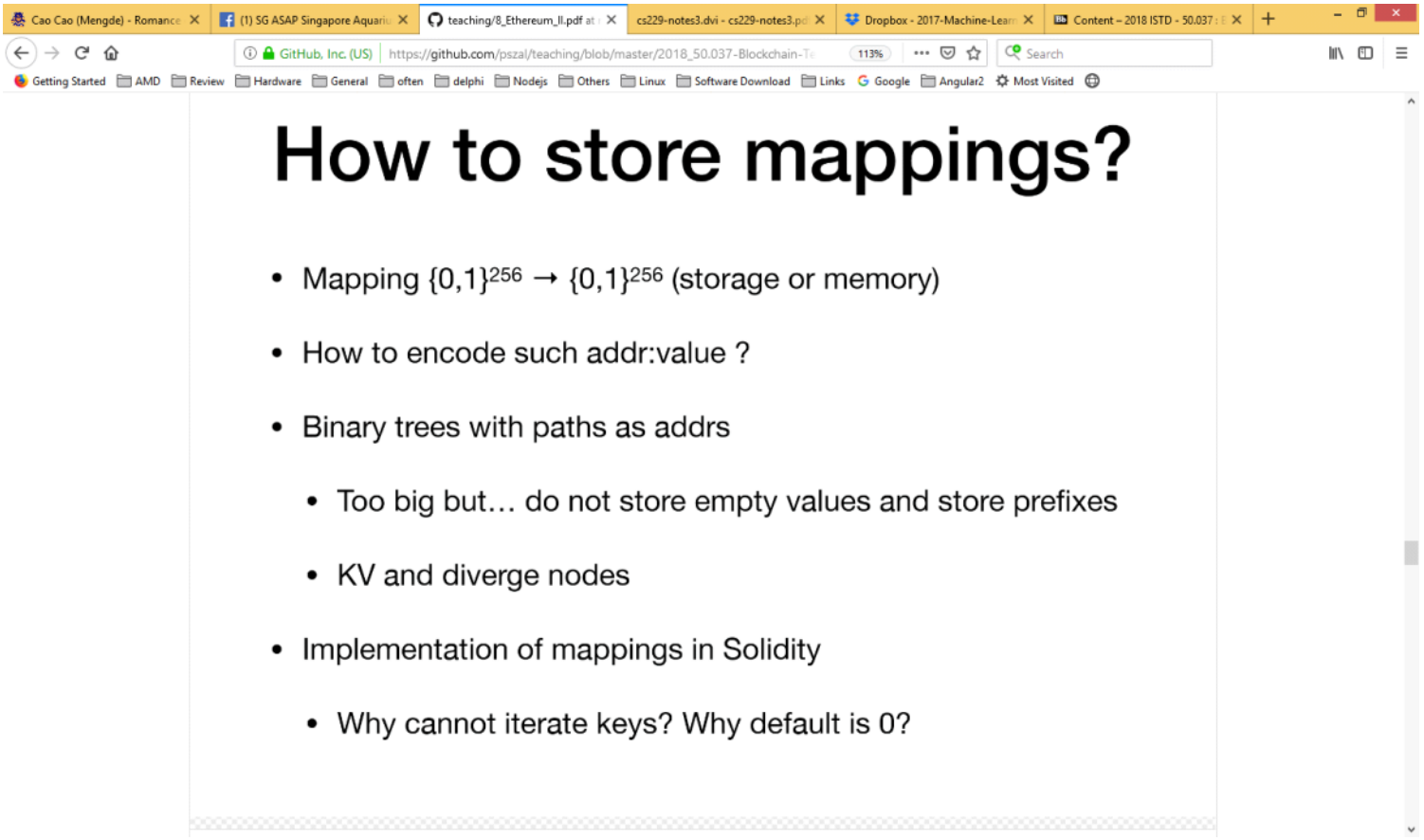
We have bloom filters which contains all the logs of the blocks. Logs of the receipt which aggregate for a given transaction. Can check if false positive or not.

In the block, we have a bloom filter for all these blocks.

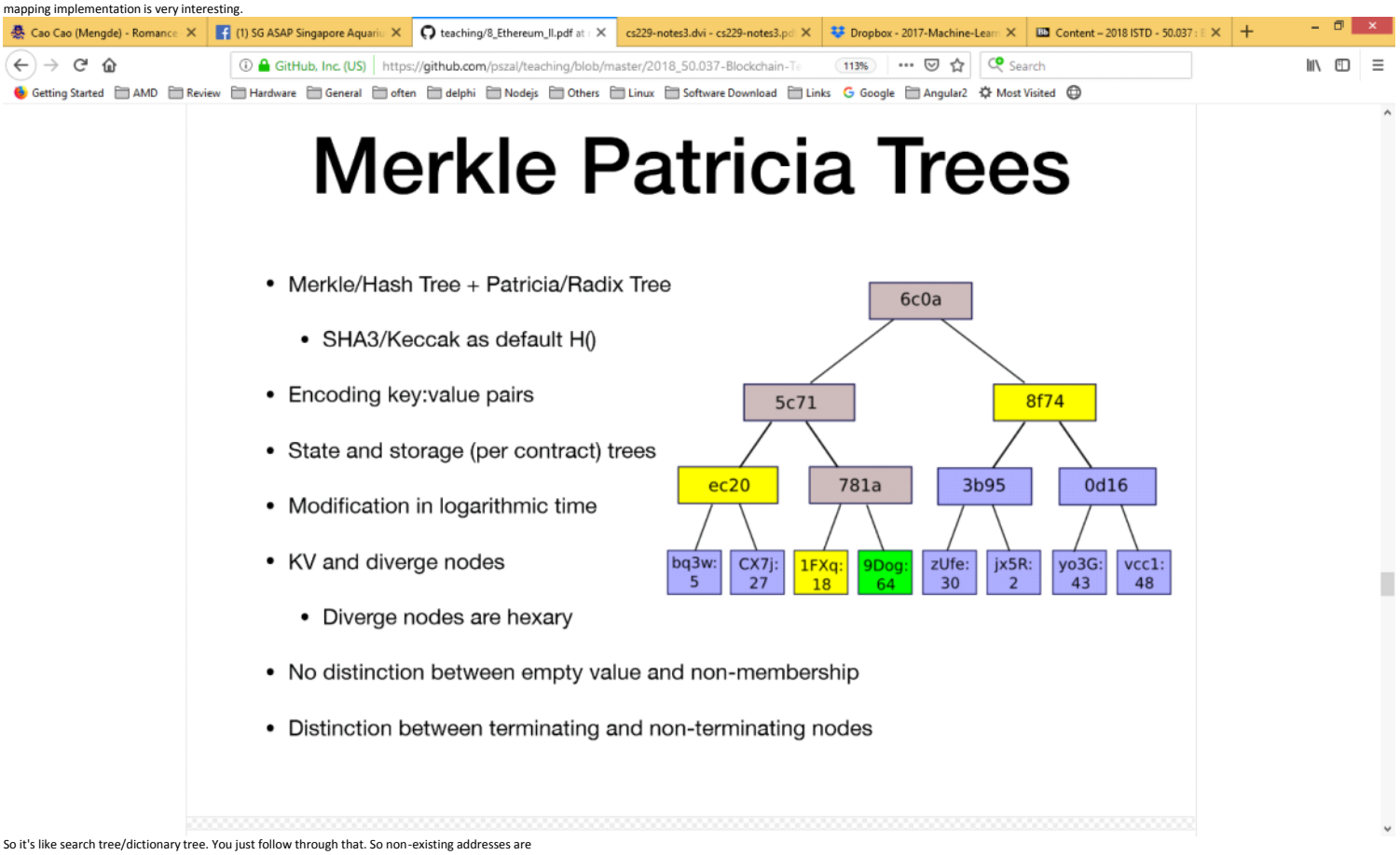


In the tree, we can find any address used.
With this address of this account, we can see many associations. If you create a smart contract, you can get the nonce for this smart contract. It's like nonce concatenated.
How much ether this circumstance. For this account, you can see from this node what code the smart contract writes.
The most interesting this is like storage unit. You have associated storage tree. Remember that the storage is permanent. Keep the storage in the tree. You can access every storage here which is kept by your given smart contract.

So this tree stores all addresses. All accounts. This account will be associated with balance, node and storage root. If this smart contract has some storage, this will be associated here. Each account will have this storage tree, which will keep strorage of the contracts.
This tree is for each account with each storage.



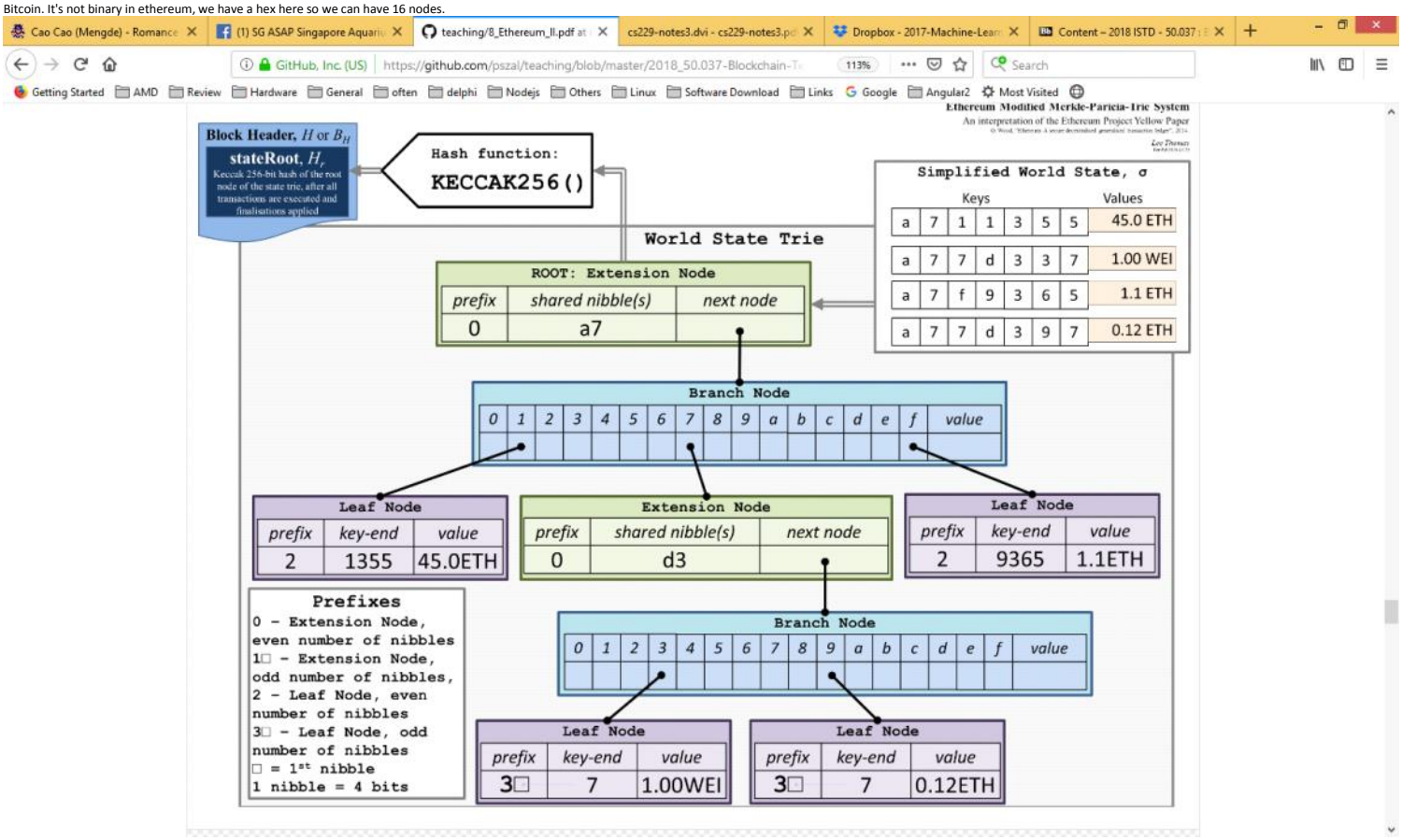
How to efficiently store the mappings? Obviously, not in EVM. How do they encode all these addresses and values. Encode something efficiently, so think about the trees. You can build a binary tree.
The trick is that you make the most of the mappings will be zero values. So what this observation is that you don't store zero nodes. If zero is concatenated with zero, you can just remove the zero. So another tree that they use is that they
Remember prefixes. So they remember 1 1 1 , and they cull the branch. If it is just chained, you can just encode that C has prefix 0,1,1 and they cull the branch.
They have Key Value nodes and Diverge nodes. KV are leaves, and diverge are branches. So this mapping implementation is very interesting.

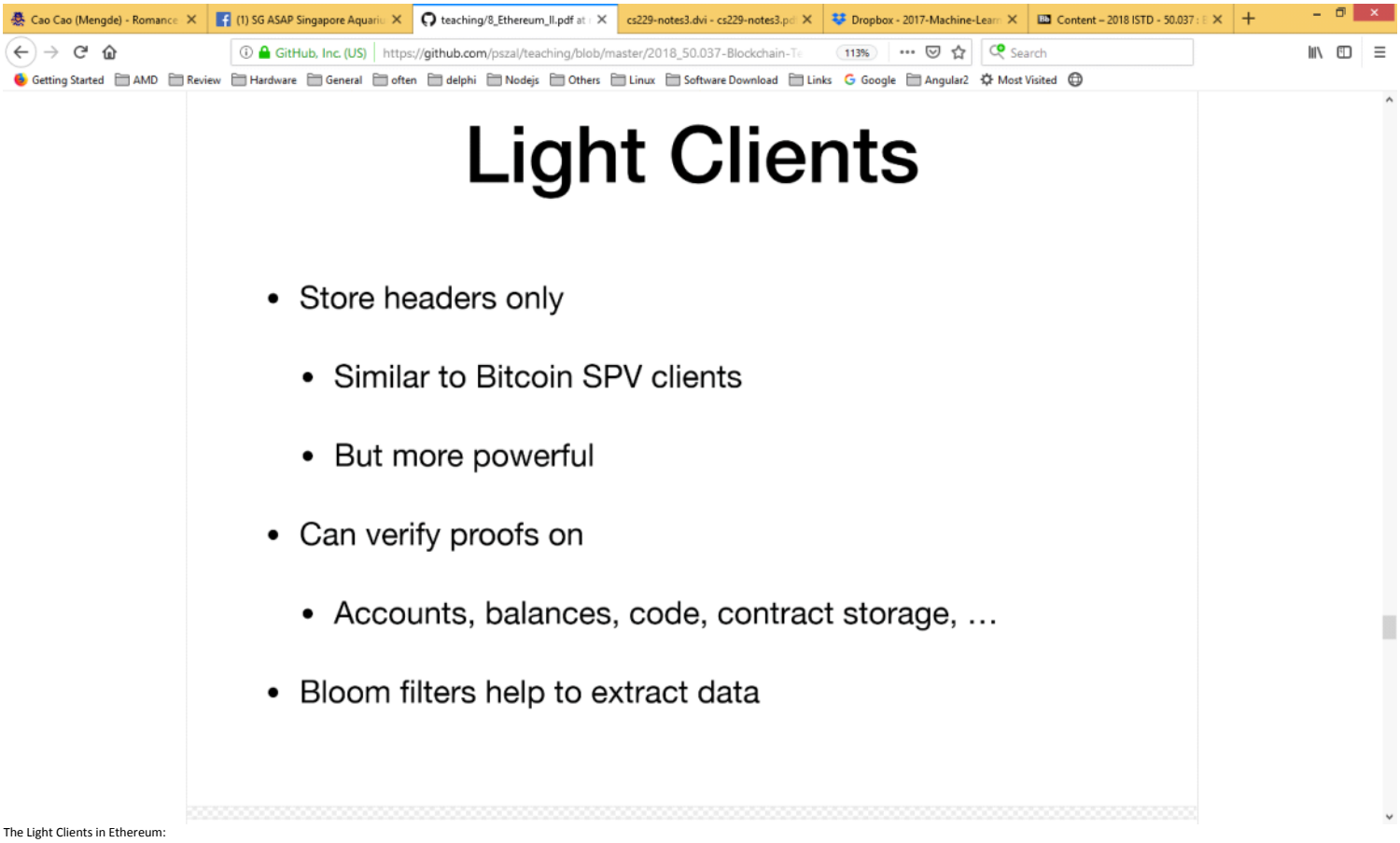


So it's like search tree/dictionary tree. You just follow through that. So non-existing addresses are not in this tree. You can find very easily that a given address is not existing and you construct this prefix tree with the hash. This leaf will aggregate hash of the leaves. The properties of the Merkle Tree is that with the root is given the key value in the tree.

It's used for keeping statement storages. It is used for keeping the state of accounts. All storage is

encoded there too.
There is no distinction between empty values and non-membership. So if something is empty, we don't store it. If something is not existing, it's the same. It's like the same as non-existing.
What they distinguish is terminating and non-terminating. We don't know if the leaf is terminating in Bitcoin. It's not binary in ethereum, we have a hex here so we can have 16 nodes.

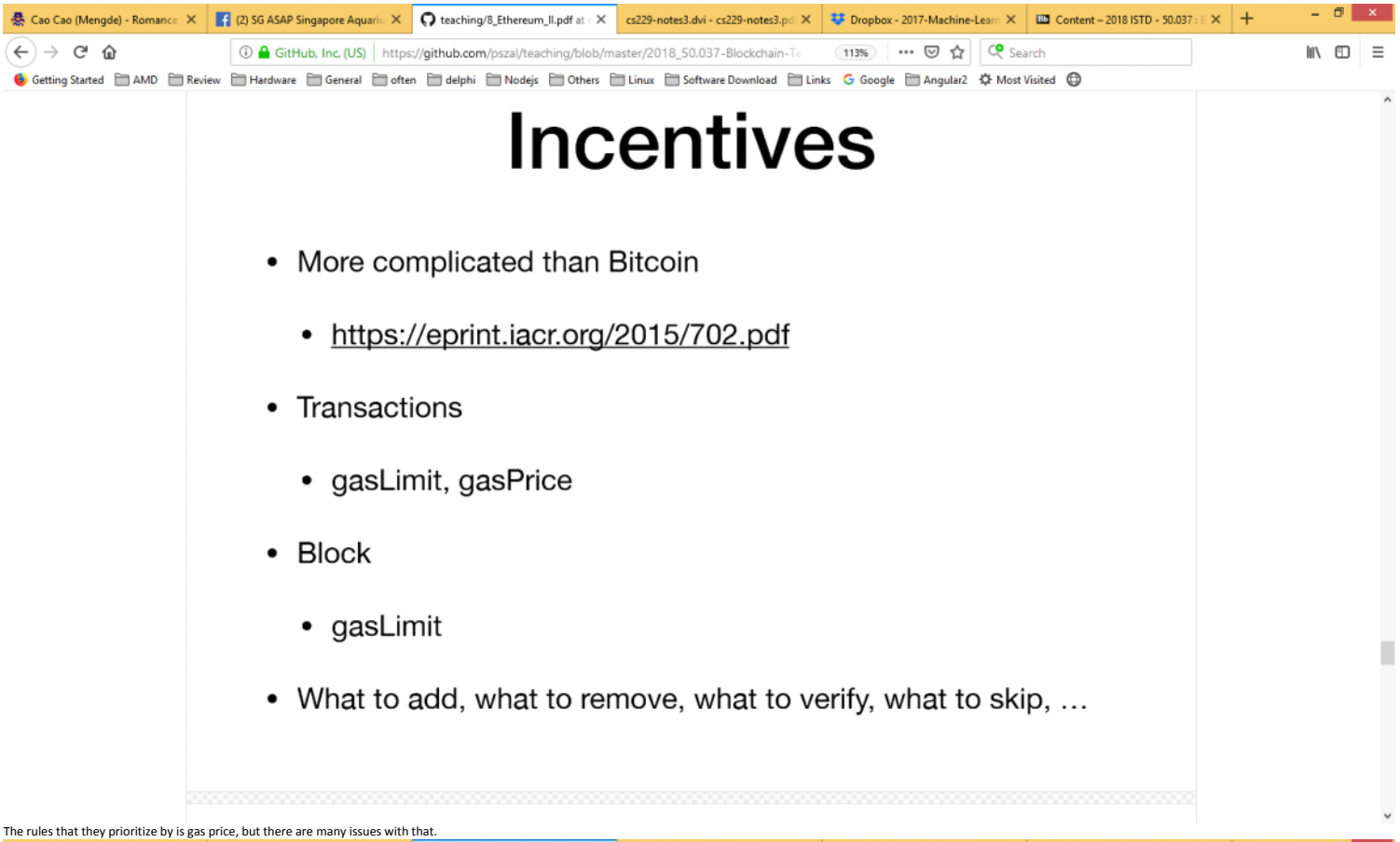




The Light Clients in Ethereum:
Sender is similar to my key. So the smart contract will make sure that only I can give you break.
So this student's grades is stored in the dictionary. We can deploy the smart contract, and if the instructor has sent this grade, then we can update the student's marks. I don't want to pay for putting your grades on anyway.

So this is like the storage of this contract.
In the state group, I can say, do this function. So the miner will pick up this transaction, execute this code. With this new block, which appends my transaction, there will be state tree. This state tree will have address of this smart contract instance. Where I send this, I can send smart contract. Here, they will find from address. This account has also like the storage root. So this is like a state tree and this

is storage.
You are able to find mappings between grades and students.
You can get the proof what is your grade. You can ask the miner what is the proof for your storage.
Give me the value for storage 100 for this address. So the address of the smart contract has persistent storage. Any miner will provide you this proof.
100 is debus, and it is associated with correct smart contract.
This correct state for smart contract instance for this block.
You will be able to verify what is your grade. We don't run this transaction. You can just get the output of that. Because you are sure that the code is executed correctly. You are sure that I executed this because the value is non zero.
Only myself is kicking.
Otherwise the grade function will terminate. The execution here will modify this state. You only with headers can verify. You only need to make sure that your headers are correct. You can get proofs about accounts balances. Codes, storages.
Bloom filters get some data, but you can skip for now.



The rules that they prioritize by is gas price, but there are many issues with that.

These cryptokitties could multiply. It was money laundering? It was the successful application on Ethereum, and it killed it. There are 15 transactions per second. You can see two pools consume like 50% of hashing power.
The project has many issues with privacy.

Lesson 9

Wednesday, October 31, 2018 10:34 AM

How you store mappings?
{0,1} ^256 -> bit stream of 256, and how to store this mapping. Do you know why this is always keyed like that? They would use hash as a key.
So how to store it? There are many ways of storing this things. Usually we go with binary trees.
{0,1}^3 ->

```
      /  \
    / \   / \
  / \ /\ /\ /\
     H     AB
```

010-> Hello
100 -> A
110 -> B

You will never create such a large tree. The observation is that whenever you are using this mapping you will have dominated the number of empty leaves. 99.99% would have empty leaves. Storing something empty doesn't make much sense. So we don't store the null values

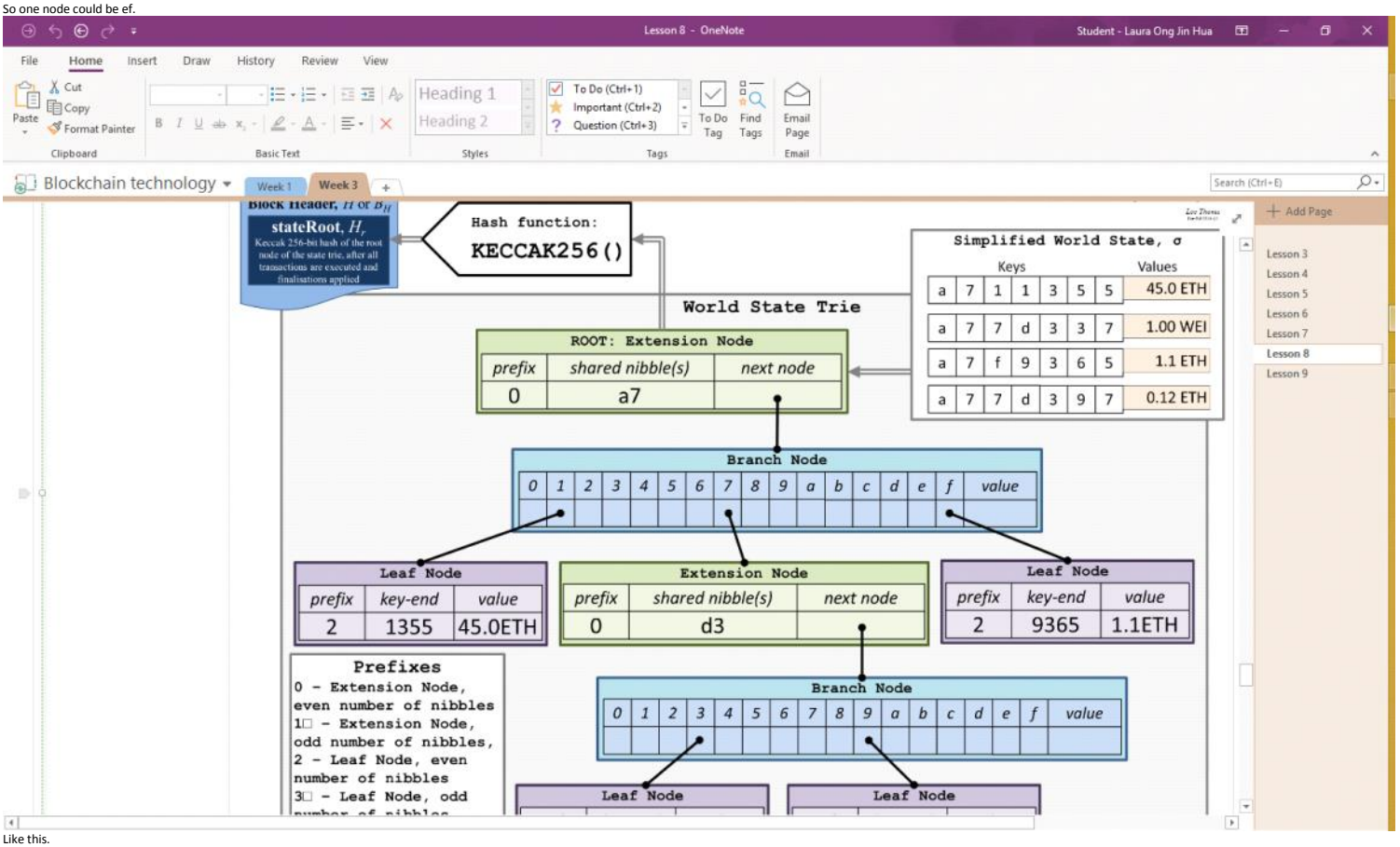
So we remove the null values from the tree

```
      /\
      \ /\
      / \ /
      H  AB
```

But why store three nodes for that? Put the longest prefix where you can branch. So we store three leaves. For that. (H)
We have one value. So we go down and we are looking for the longest prefix that we can branch.
So for H: it's 010
And we associate the data with this leavel

```
  /          \
010          D
KV           /   \
          01A 10B
```

So these nodes, are called key value nodes. And these nodes are called divergent nodes
Each of the 16 values are branching to another one. So they would be branching here, the tree is pretty flat, since the nodes are pretty far, there are 16 pointers and no value. If you are going to address like 2,1,e,f something something. This one we are looking for 1, for f, and one on.
So one node could be ef.



Like this.
Hash trees are combined with this tree.

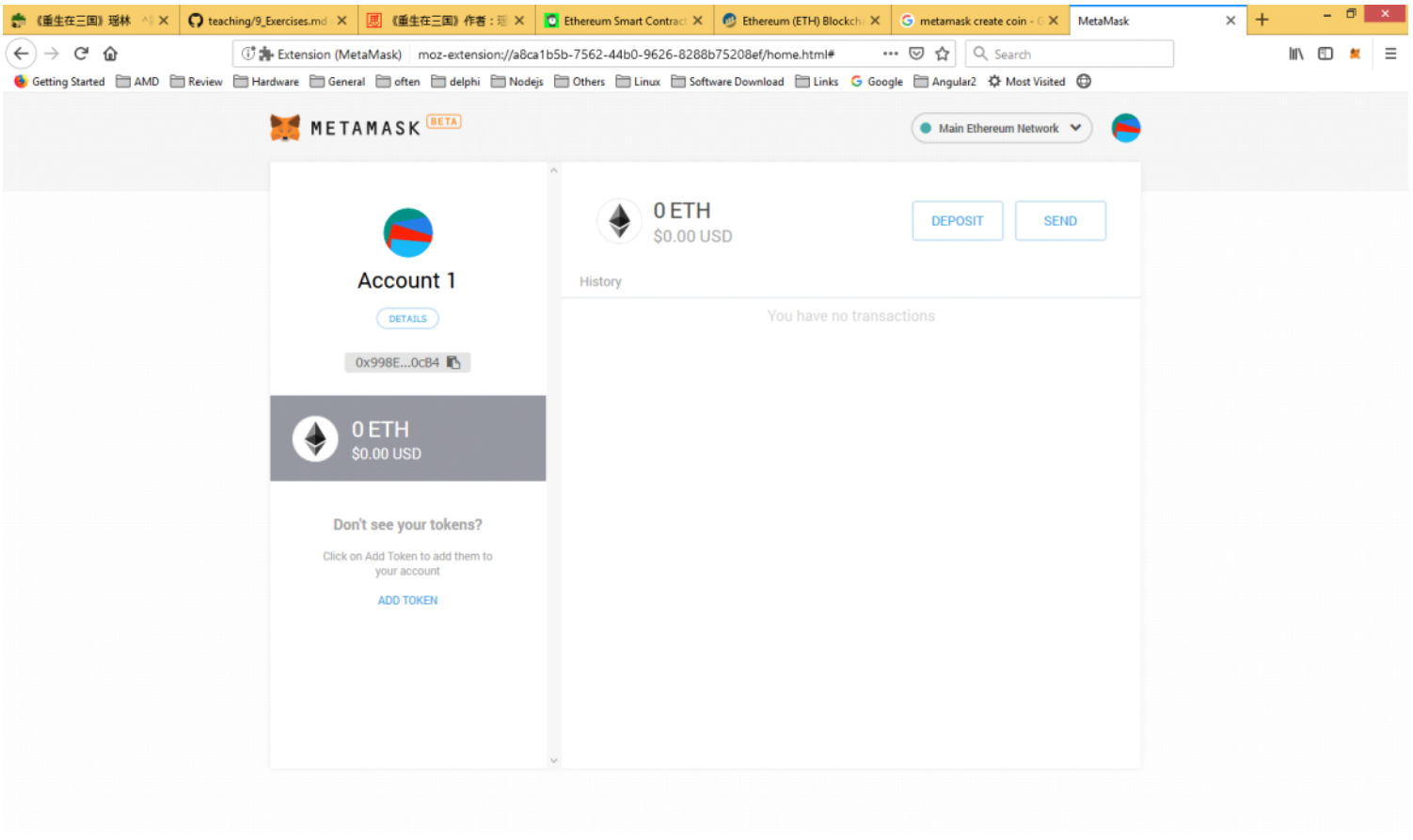
So the hash of the intermediate is like a prefix, and child node concatenated with another child node. This is what hash is associated. It computes like prefix and value. You can do the same stuff with Merkle tree in terms of root, you can prove that something is within this tree. So this is what is used for the state tree and the storage tree.

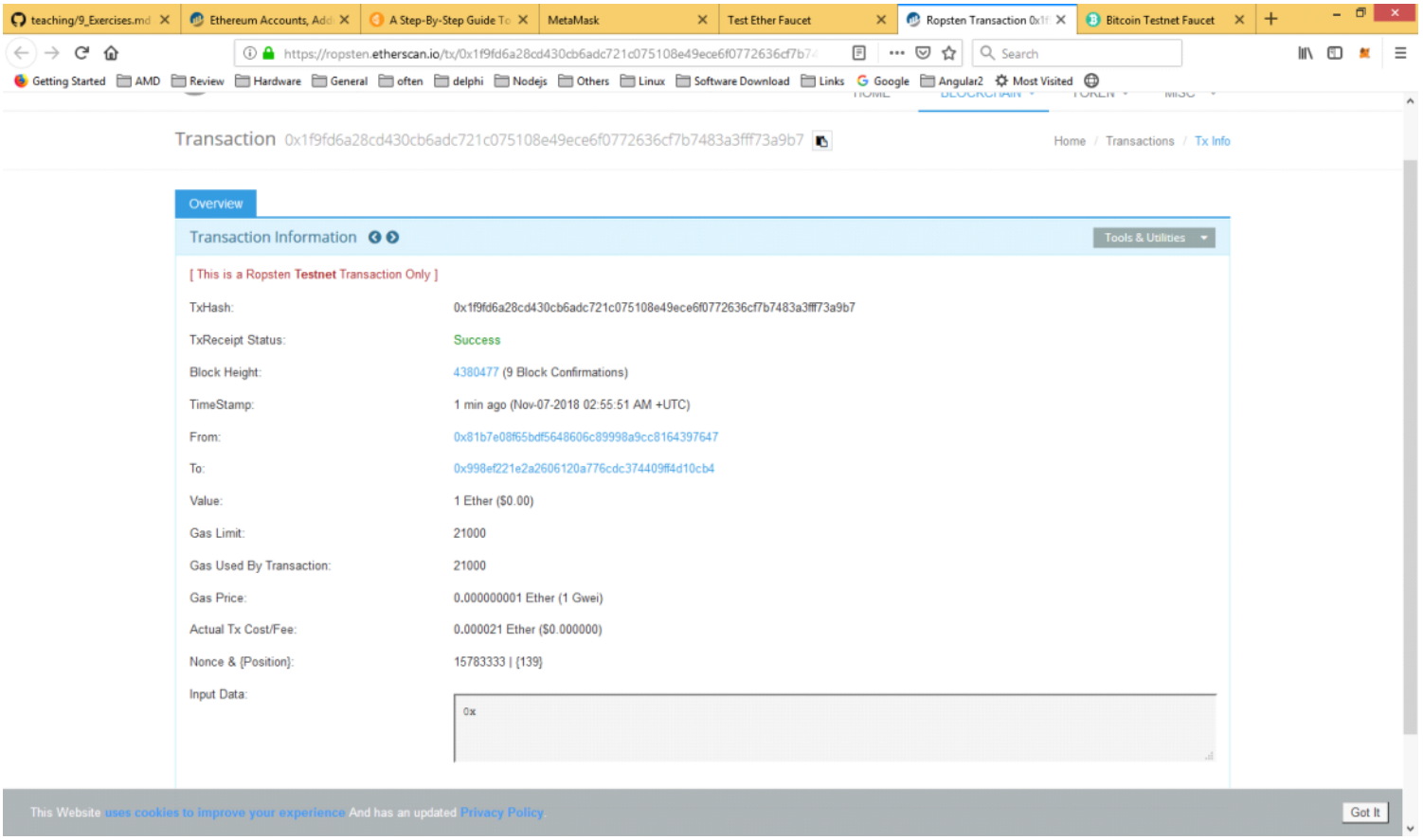
How do they combine it with mappings in Solidity?
In solidity, you have mapping(int256 ->String) test;
How to map it? What solidity is doing, which is clever, is the following.
The hashing function is collision resistant, you can use the entire space for your application, you do not need space for your mapping. You cannot iterate over keys, as they are like hashed.

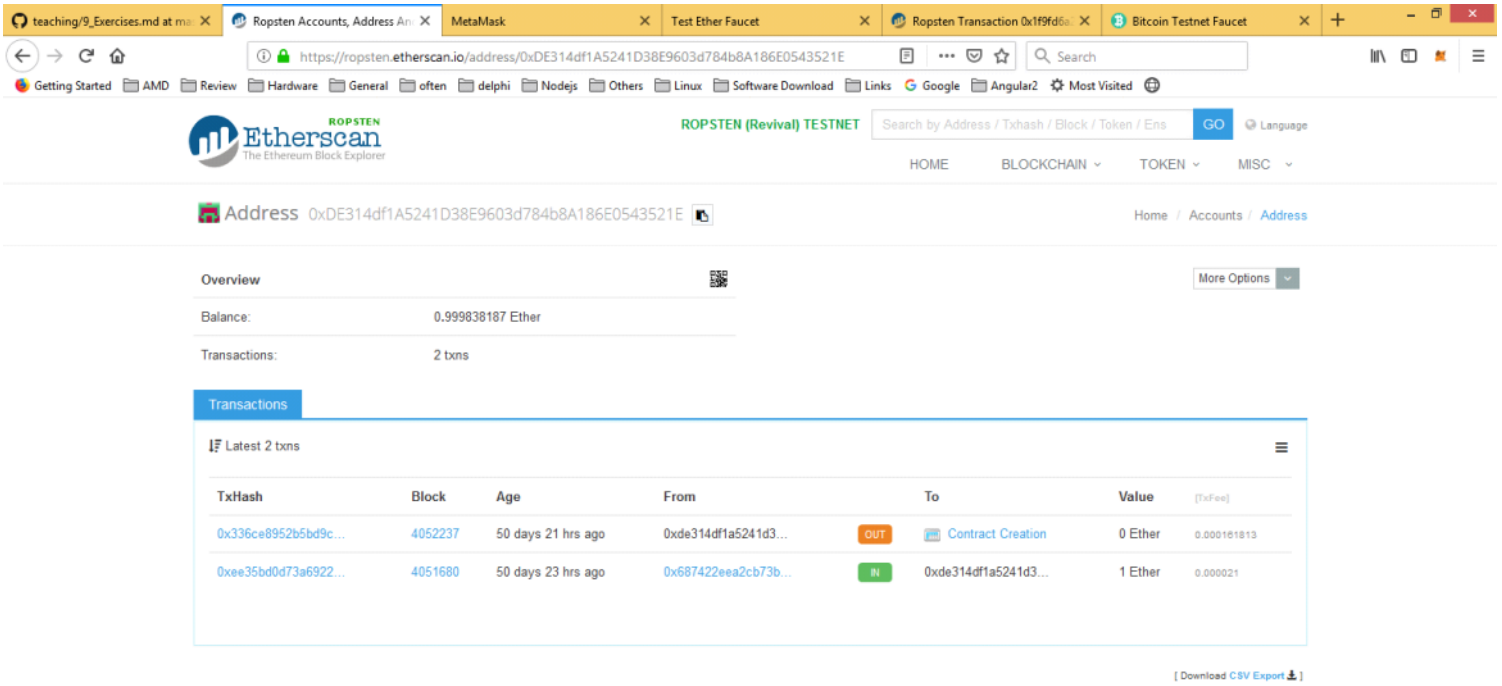
This mapping, this huge address space is encoded as this tree. This tree is pretty short. We take like hash of this object, and we know the address, and we can access value, going through the tree. This is like obstruction and how it is implemented.

Smart Contract
Clients not want to show
It's very difficult to submit cases. Someone has to be trusted in this lottery.
Very difficult to implement a lottery. With other entrances and smart contracts.

inform nice derive situate excite wheat afraid learn warrior inhale switch obey

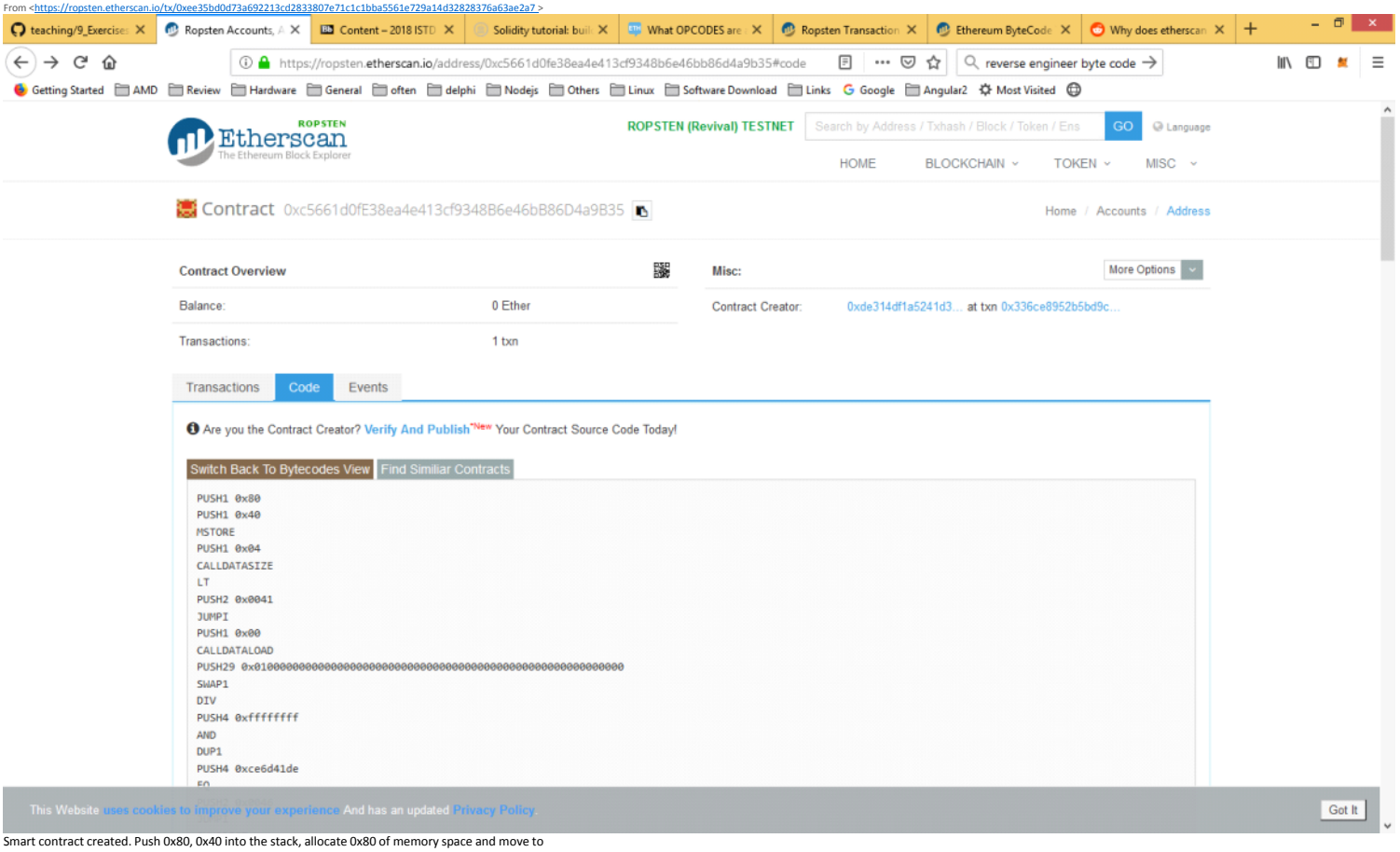

MetaMask
Extension (MetaMask)
moz-extension://a8ca1b5b-7562-44b0-9626-8288b75208ef/home.html#
METAMASK BETA
Main Ethereum Network
Account 1
DETAILS
0x998E...0cB4
0 ETH
$0.00 USD
DEPOSIT
SEND
History
You have no transactions
Don't see your tokens?
Click on Add Token to add them to your account
ADD TOKEN


Transaction 0x1f9fd6a28cd430cb6adc721c075108e49ece6f0772636cf7b7483a3fff73a9b7
Home / Transactions / Tx Info
Overview
Transaction Information
Tools & Utilities
[This is a Ropsten Testnet Transaction Only]
TxHash: 0x1f9fd6a28cd430cb6adc721c075108e49ece6f0772636cf7b7483a3fff73a9b7
TxReceipt Status: Success
Block Height: 4380477 (9 Block Confirmations)
TimeStamp: 1 min ago (Nov-07-2018 02:55:51 AM +UTC)
From: 0x81b7e08f65bdf5648606c89998a9cc8164397647
To: 0x998ef221e2a2606120a776cdc374409ff4d10cb4
Value: 1 Ether ($0.00)
Gas Limit: 21000
Gas Used By Transaction: 21000
Gas Price: 0.000000001 Ether (1 Gwei)
Actual Tx Cost/Fee: 0.000021 Ether ($0.000000)
Nonce & {Position}: 15783333 | {139}
Input Data:
0x
This Website uses cookies to improve your experience And has an updated Privacy Policy
Got It

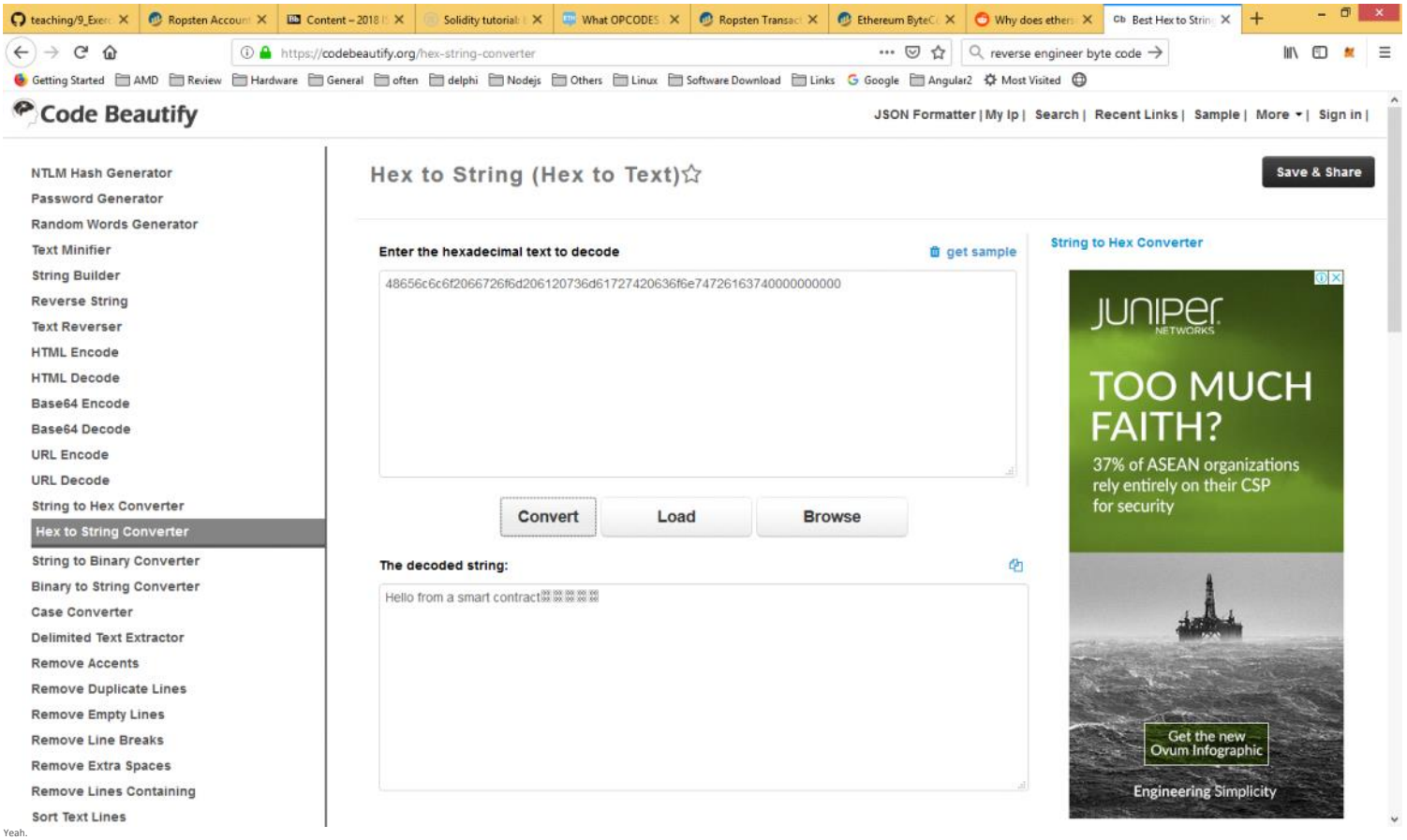


Account Balance =0.999838187 Ether
When was last transaction performed?
(Sep-17-2018 03:37:33 AM +UTC)

From <https://ropsten.etherscan.io/tx/0xee35bd0d73a692213cd2833807e71c1c1bba5561e729a14d32828376a63ae2a7>



Smart contract created. Push 0x80, 0x40 into the stack, allocate 0x80 of memory space and move to the 0x40 position.
So in this case, this chain is storing 16*8 = 128 bytes of memory and moving the pointer to the beginning of the 64th byte. We now have 64 bytes for scratch space and 64 bytes for temporary memory storage.
This entire string of opcode contains 0 Ether, meaning that only the gas price was paid for this transaction to be confirmed. This is to store the data encoded in byte form
0x48656c6c6f2066726f6d206120736d61727420636f6e747261637400000000

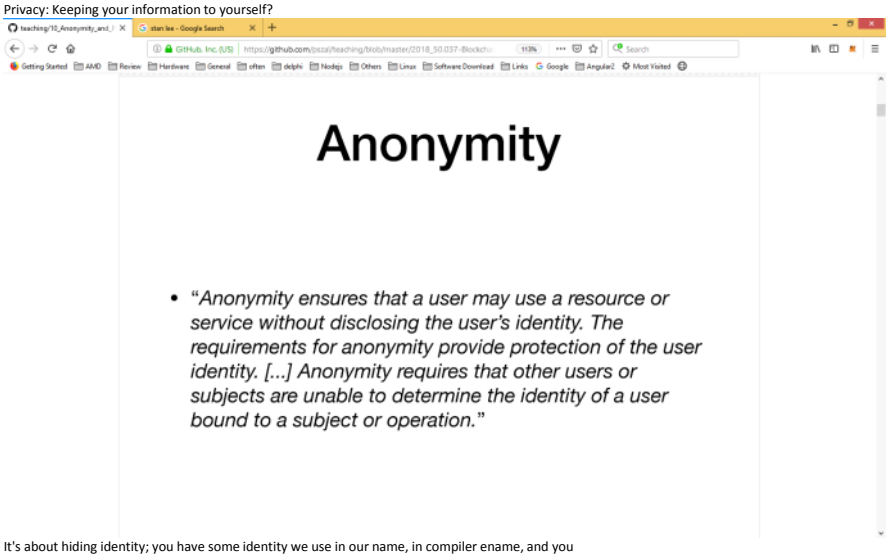

Code Beautify
JSON Formatter | My Ip | Search | Recent Links | Sample | More | Sign in |
NTLM Hash Generator
Password Generator
Random Words Generator
Text Minifier
String Builder
Reverse String
Text Reverser
HTML Encode
HTML Decode
Base64 Encode
Base64 Decode
URL Encode
URL Decode
String to Hex Converter
Hex to String Converter
String to Binary Converter
Binary to String Converter
Case Converter
Delimited Text Extractor
Remove Accents
Remove Duplicate Lines
Remove Empty Lines
Remove Line Breaks
Remove Extra Spaces
Remove Lines Containing
Sort Text Lines
Hex to String (Hex to Text)
Save & Share
Enter the hexadecimal text to decode
get sample
48656c6c6f2066726f6d206120736d61727420636f6e7472616374000000000
Convert
Load
Browse
The decoded string:
Hello from a smart contract
String to Hex Converter
JUNIPER NETWORKS
TOO MUCH FAITH?
37% of ASEAN organizations rely entirely on their CSP for security
Get the new Ovum Infographic
Engineering Simplicity

Yeah.

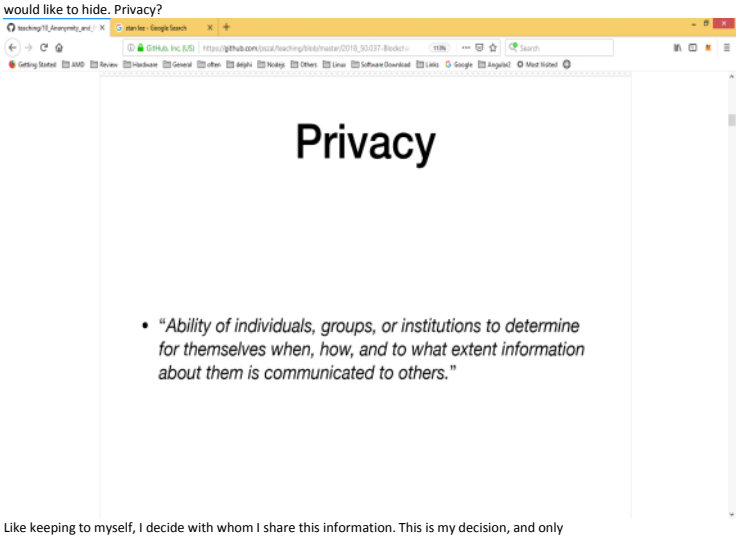
Lesson 10

Tuesday, November 13, 2018 8:34 AM

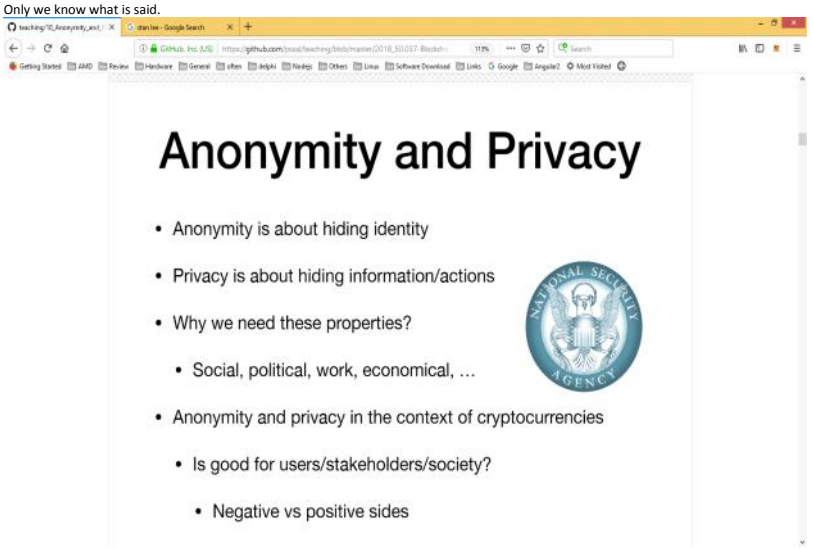
What's privacy? What's anonymity?
Anon: Not having an identity
Privacy: Keeping your information to yourself?



It's about hiding identity; you have some identity we use in our name, in compiler ename, and you would like to hide. Privacy?



Like keeping to myself, I decide with whom I share this information. This is my decision, and only that the second party can seek this information. What are ways of assuring privacy?
Better mine which information is available to others.
Only we know what is said.



Do we need this properties in general?
You never want to hide your identity? Why do you have this? Do many things illegal! Immediately; if you provide this properties, no law enforcement could stop us. Another motivation, with unlimited privacy, people tends to be more honest. Who doesn't like this course? Just raise their hands. Voting system are anonymous. It doesn't make much sense.

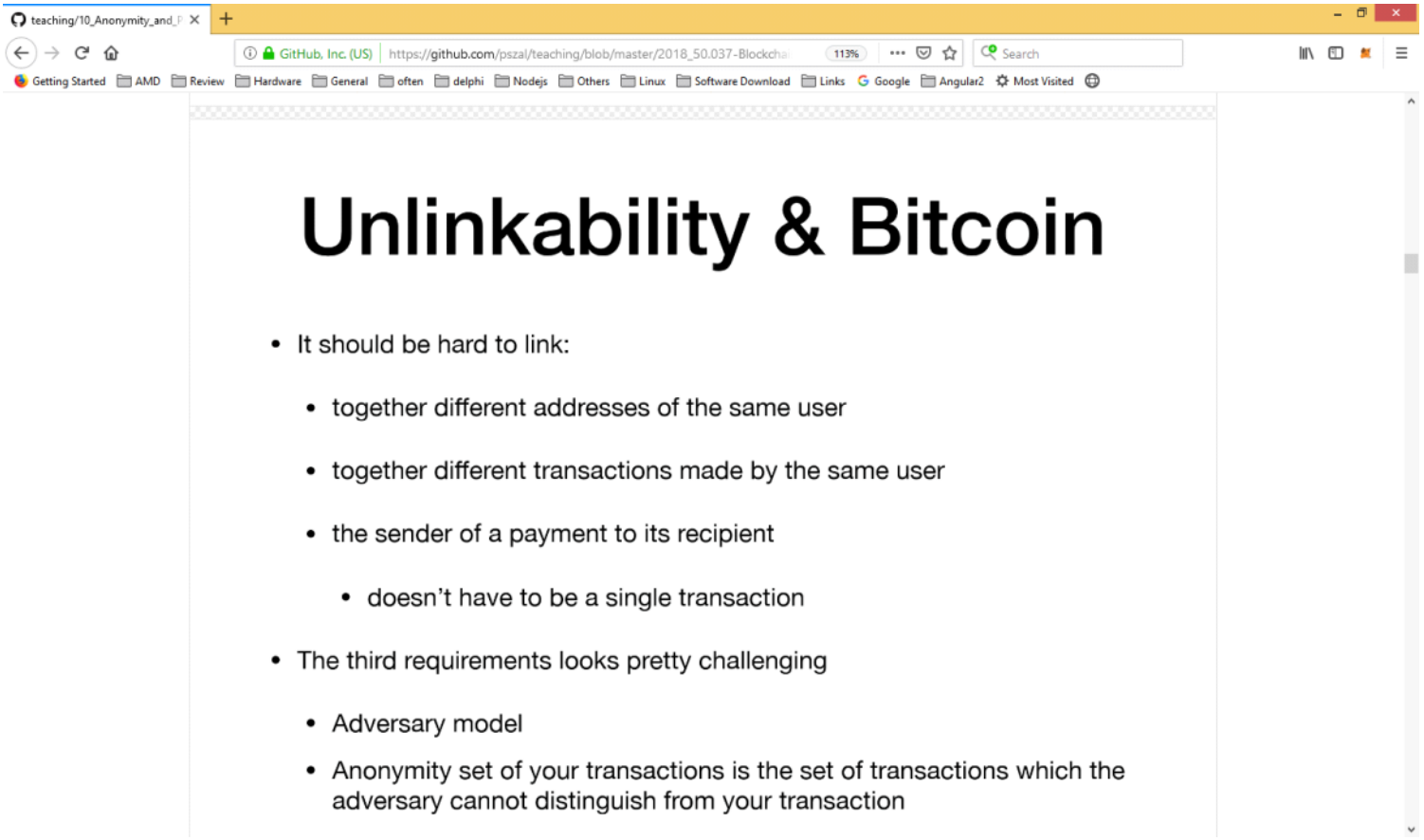
There are examples when you need it. Like some departments, like HR or Law, they just do some espionage to see what other companies are doing. They would like not to reveal that they are connecting from company A or B. Companies don't want to reveal too much info about what is going on.

Is it good to be anonymous or not? We can or cannot, but should we? Should we provide anonymity or privacy? It has two sides: if you provide anonymity, but such anonymity that the government can de-anon it? The other governments can de-anon it. If something bad happens to us, the strong

anonymity, we would like to fight this guy, and strong anon would make our lives very hard. There's a debate what anon should we provide to cryptocurrencies.

Bitcoin and Anonymity
An address is seen on the Bitcoin. What is Blockchain? A block? A collection of transactions. It's encoded as a hash or script. It's the transfer of some asset to another place. You can see all transactions, but you don't see the person behind it!

We can create many bitcoin addresses. We play with this addresses and they are like pseudonyms.
On the other hand, you might see some issues.
Pseudonymity: it is weaker. For 4chan, you get a new fresh identity. We all use reddit, we keep our username, and we just use this username. Properties are not as strong.
In the blockchain, you have all transactions visible. They are signed. So there is non-repudiation, they are authentic. You have strong proof that the given transaction was created by the user.
Linking addresses to identities is pretty easy. Like you are using some wallets, communicate with some websites. You basically manage your cryptocurrencies. This wallet knows it. You have med services, you can do very similar things. You register and just send money. Exchanges know your identity; they are obligated to do it.
If you have a payment processor and your browser, someone can track you.

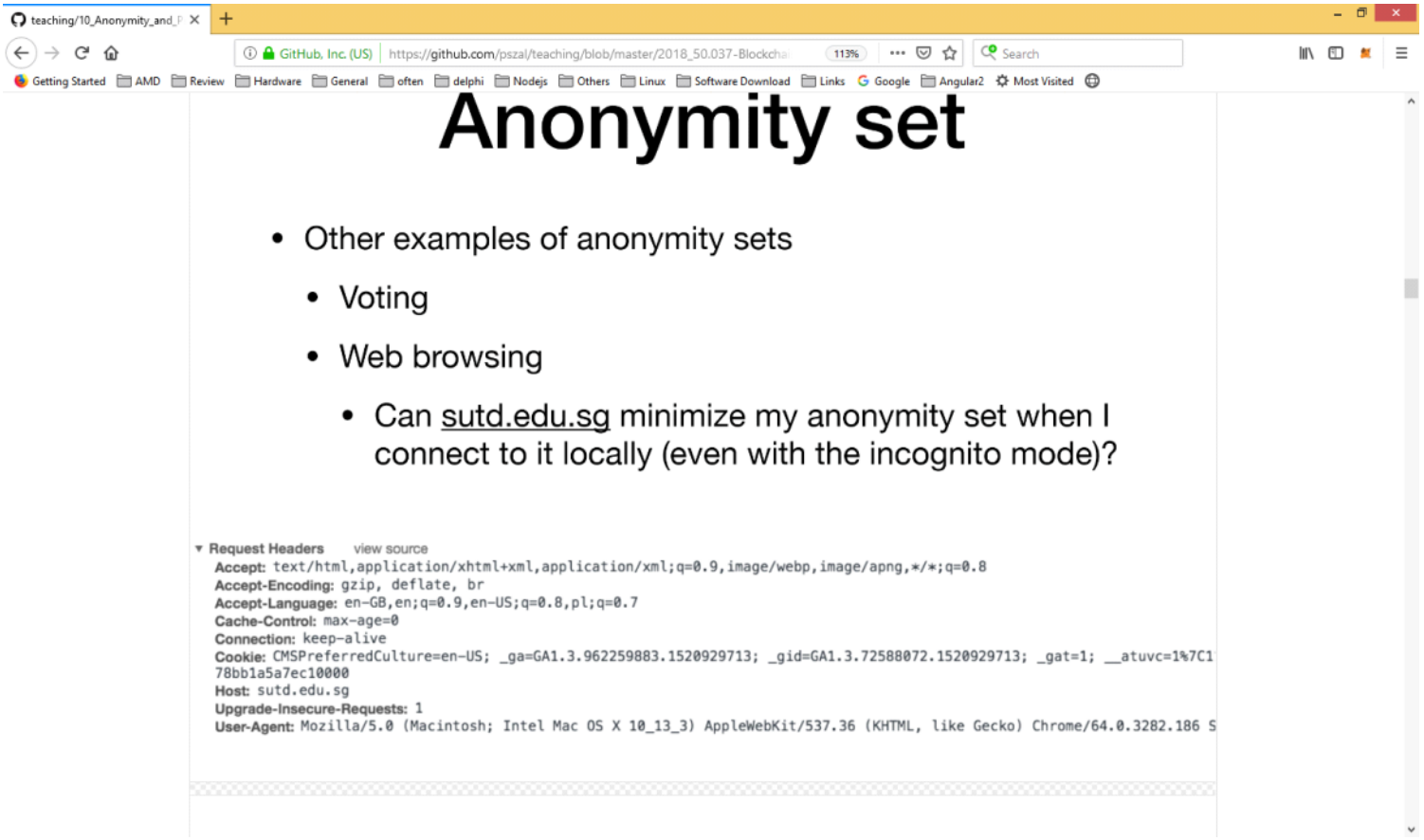


? For any two addresses, it should be impossible to link that they belong to the same user.
It should be difficult to link different transactions. If Alice is making some transactions; all of these transactions cannot be attributed to the given user.
What is the third requirement? If there is Alice, and she pays to Bob, It should be difficult to link sender to receiver. Difficult to associate these two guys. It is easy to link sender and receiver or not? It is very easy because it is a transaction. In real life, it is easy to attribute. Like a bank transfer, you know the sender and receiver

How are you going to achieve it? This payment can be sent in many transactions; so it is impossible to associate user A and user B. It is counterintuitive because you expect transaction systems are linked like sender and receiver. From our perspective, it would be good to have because no one can track your user by your transaction.

We need to define our adversary model, what it can do and what it cannot.
It's not a strong adversary, it's a weak adversary. We assume that adversary can provoke some user to commit transaction. If your system can provide interface in response, it's stronger.

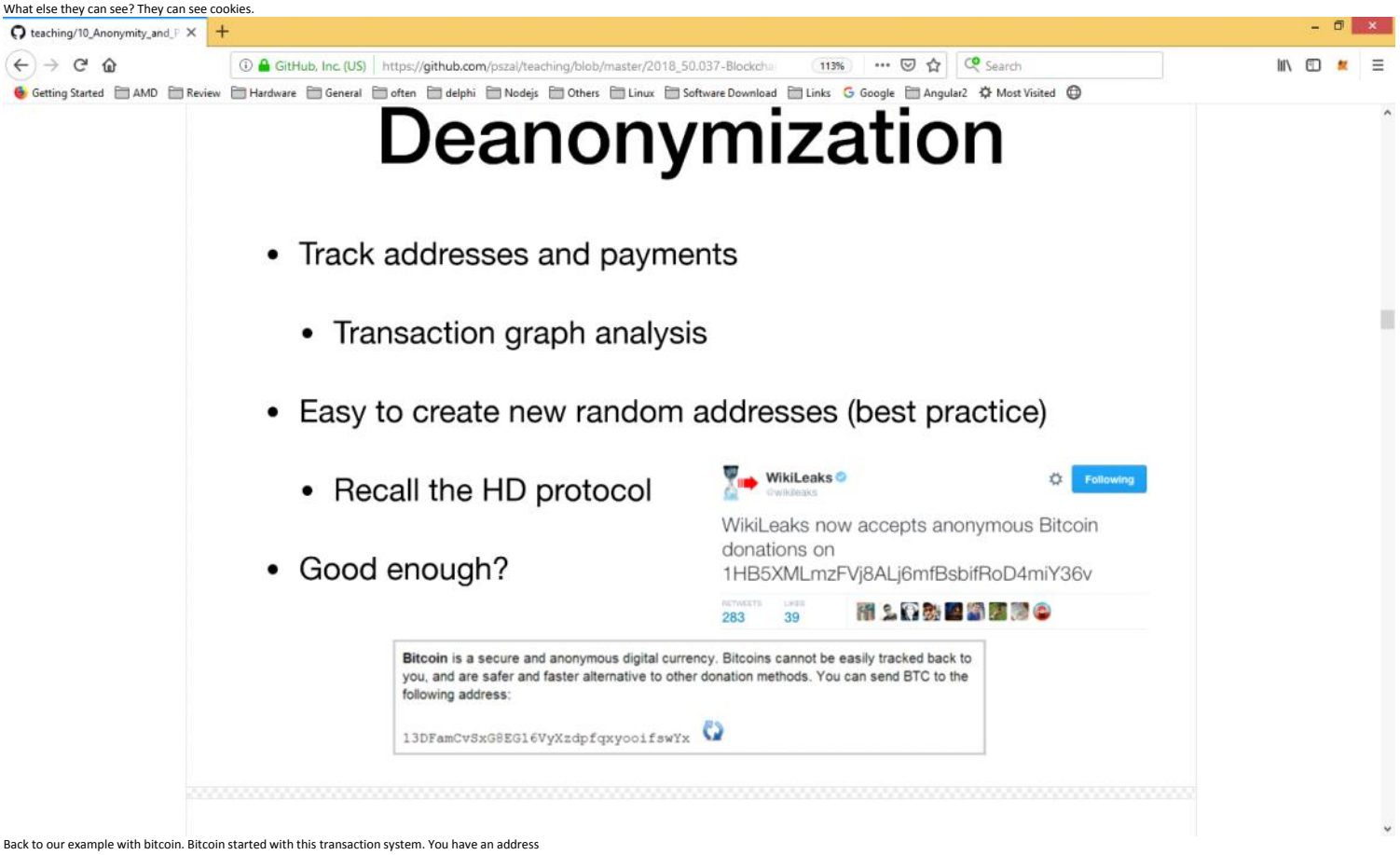
Anonymity set: set of transactions which the adversary cannot distinguish from your transactions. In the beginning, if you are a malicious student, who doesn't like this course, what's your anonymity set? What is anonymity set? Everyone who raised hand, in the birthday example, in an anon survey, are your anonymity set. Because you cannot distinguish between those who voted and a particular person.



Since the counter of the vote know what is your vote, then there is no anonymity set.
If two are voting, you cannot distinguish between the two, so the counter only has a 50% chance of guessing.
So you would try to increase your anonymity set.

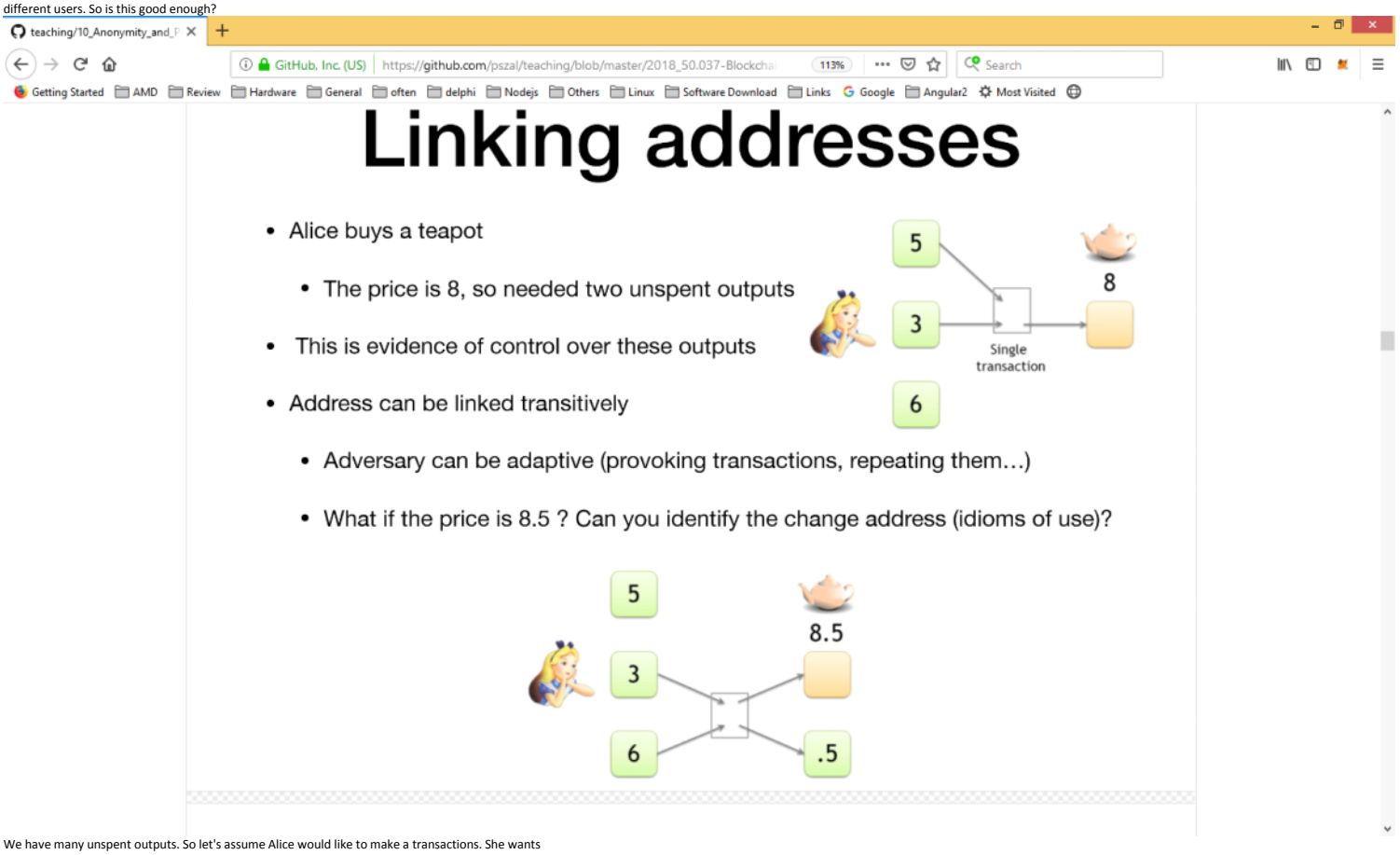
It was done by people who have access to network, someone with the network did it. Connecting to the website, you would leave some information in the header. So the set is first all users with access to the network.

What else they can see? They can see cookies.



Back to our example with bitcoin. Bitcoin started with this transaction system. You have an address and send transactions. You can see what's going on in the network.
Wikileaks was disconnected from payment networks, like Visa and Mastercard. So they just publish the address on twitter. Everyone knew that this was the address of Wikileaks. You know that this user is supporting Wikileaks. You can see the transactions of this user. So this is a big issue. Then they started to think if they can improve it.

The Deterministic wallet protocol; you can use them for many transactions with different wallets. Wikileaks implemented a random address space. It's like Wikileaks has different addresses for different users. So is this good enough?



We have many unspent outputs. So let's assume Alice would like to make a transactions. She wants privacy; so she sends money from two different wallets. They are spent as one transaction too! So the two addresses belong to the same person. It means that someone control these two addresses, and have a single transaction which has one output. 5 and 3 would be combined into A, and A would be used to buy this item. It's an issue, because if you are buying something, having many addresses.

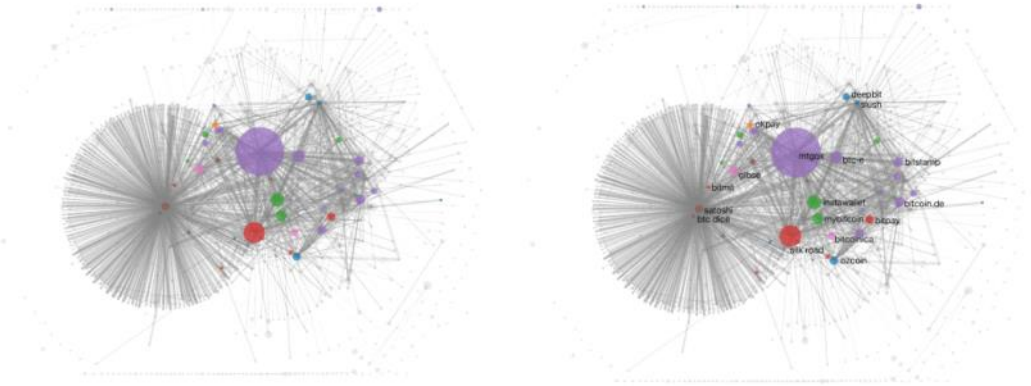
This would happen. The common space, is someone making transaction with one item. This might give some end. The addresses can be linked transitively. The adversary can be adaptive (provoking transactions, repeating them.)

Adversary can be provoking some transaction. So we need to find a combination of unspent outputs. So basically, let's buy this. So we are sending nine, and then something is 8.5. We don't want to pay more when needed. What we'll do is that one output will be 8.5, and another output would be .5. Let's say we are using the best practice, so this address is a fresh address.

Usually, the wallet is created with a fresh address. If you analyze blockchain, you see that with this transaction, you could assume that this is changeable. So usually, there will be a static address. If it is completely new, there would be a change of address. It would be very successful against every wallet systems which have hard coded to change their address.

Clustering Addresses

- https://arxiv.org/pdf/1107.4524.pdf
- https://cseweb.ucsd.edu/~smeiklejohn/files/imc13.pdf
  - Real world identities to clusters?



Satoshi dice: gambling service.

Identifying individuals

- Can we link “little” clusters (of individuals) to real-life identities?
- Directly transacting: a transaction party often knows at least one address belonging to other party(ies)
- Via service providers: exchanges or another centralized service provide typically ask users for their identities (Know Your Customer — KYC)
- Carelessness: people often post their Bitcoin addresses in public forums (donations, payments, …)
- Things get worse over time: deanonymization algorithms usually improve over time (data is publicly available!)

Clustering works when you have huge identities. The problem is that you have individuals. You don't send too many transactions. Can you identify them. Something to be more interesting. The easiest case is directly transacting. If I have business with Alice, she sends me some money, I know her identity because it is easy to figure out who you do business with.

With service providers; currency exchanges or centralized services to take your cryptocurrency, they would need KYC. If someone is colluding through the service providers, easy to de-anon them

Network-layer Deanonymization

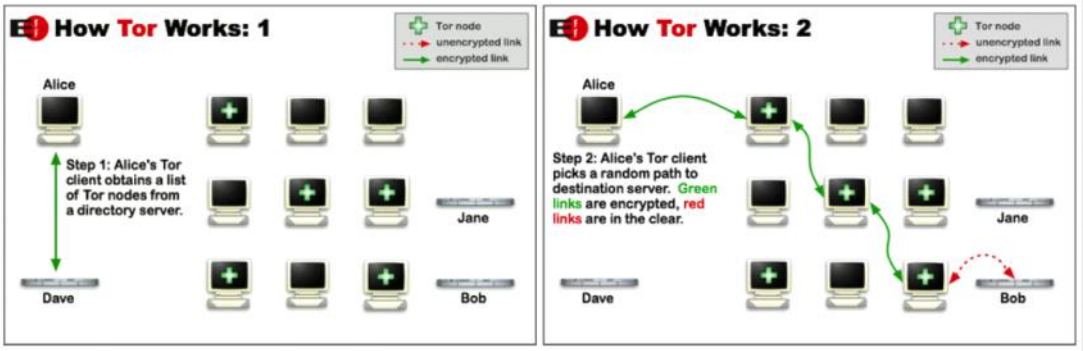
- Can we use the underlying p2p network?
  - Network identities (IP addresses) can be very precise
- Tracking source
  - Observation: “the first node to inform you of a transaction is probably the source of it.” - D. Kaminsky
- No encryption
- Prevention: network-layer anonymization

Now we assume that we see a blockchain. If you control many points, see that the transaction is propagated over this link, you can see the adversary. That is pretty bad because the network layer can reveal your IP address. This is some kind of bound to your vocation. So if your address is very static, the de-anon can be localized to a given location.

There is no encryption in a p2p. If someone is eavesdropping here, then it would be found here.

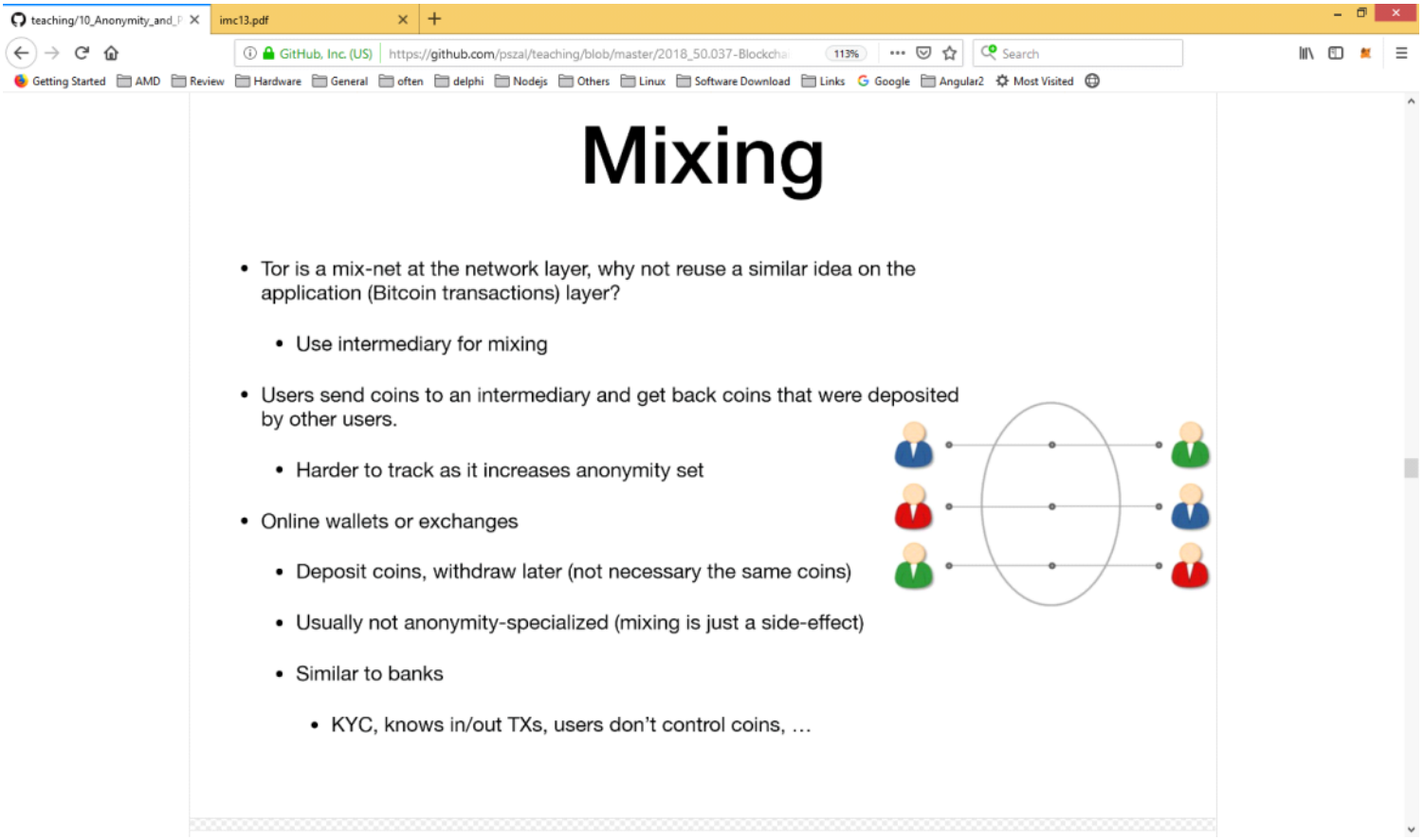
teaching/10_Anonymity_and_P | imc13.pdf
GitHub, Inc. (US) https://github.com/pszal/teaching/blob/master/2018_50.037-Blockcha 113% Search
Getting Started AMD Review Hardware General often delphi Nodejs Others Linux Software Download Links Google Angular2 Most Visited

Tor Network



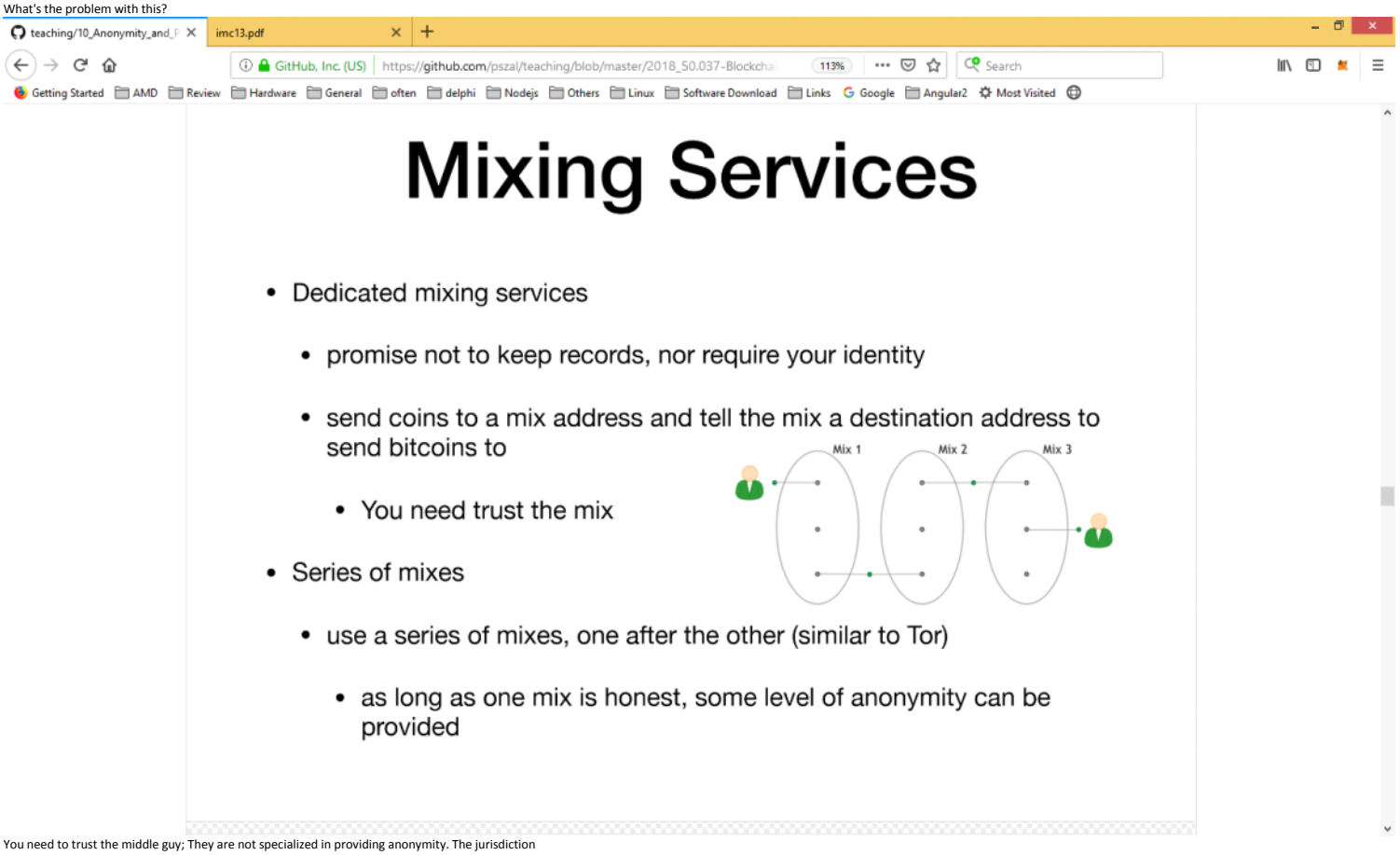
The first guy only knows Alice and the middle man. The exit node only knows the destination and the middle node. So the links between Alice and the destination is impossible to detect because these guys don't pollute. It's a pretty strong property. Let's use a TOR, but not popular.

We have some infrastructure that mixes our packets. It just sends somewhere else. Natural question is can we do something like that> On the bitcoin layer.



We can assume that they can talk through this intermediary, we can use the transaction through the intermediary. We increase our anonymity set and make our transactions. You don't know to pay whom. It is very difficult to find out who is aggregating the addresses. All the users can use the address through the intermediary. It's actually what happens in an exchange.

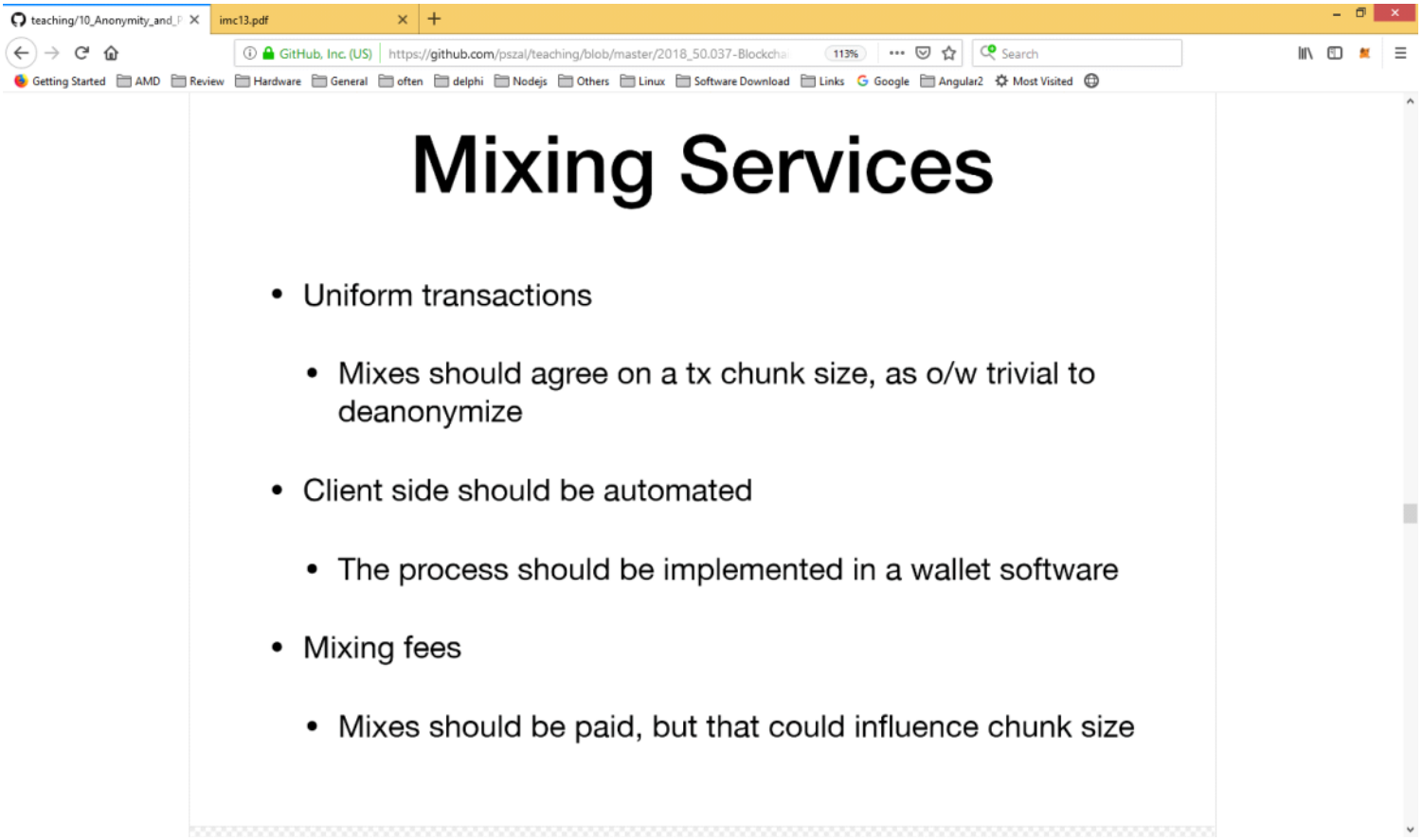
This happens with online wallets and exchanges. Through online interfaces, we can make transactions on behalf of this intermediary.
What's the problem with this?



You need to trust the middle guy; They are not specialized in providing anonymity. The jurisdiction would force you to get a lot of info, the government need to know which your customer is. They know all inputs outputs, they know identities and so on. It's like intermediary. You need to trust to ensure that you know your points.

We have some mixing services. These are services that are dedicated to increase your anonymity. It's not very popular because due to jurisdiction and so on, they are not entirely legal? Basically, they promise this intermediary, they won't keep track of inputs and outputs. So they are also not legal

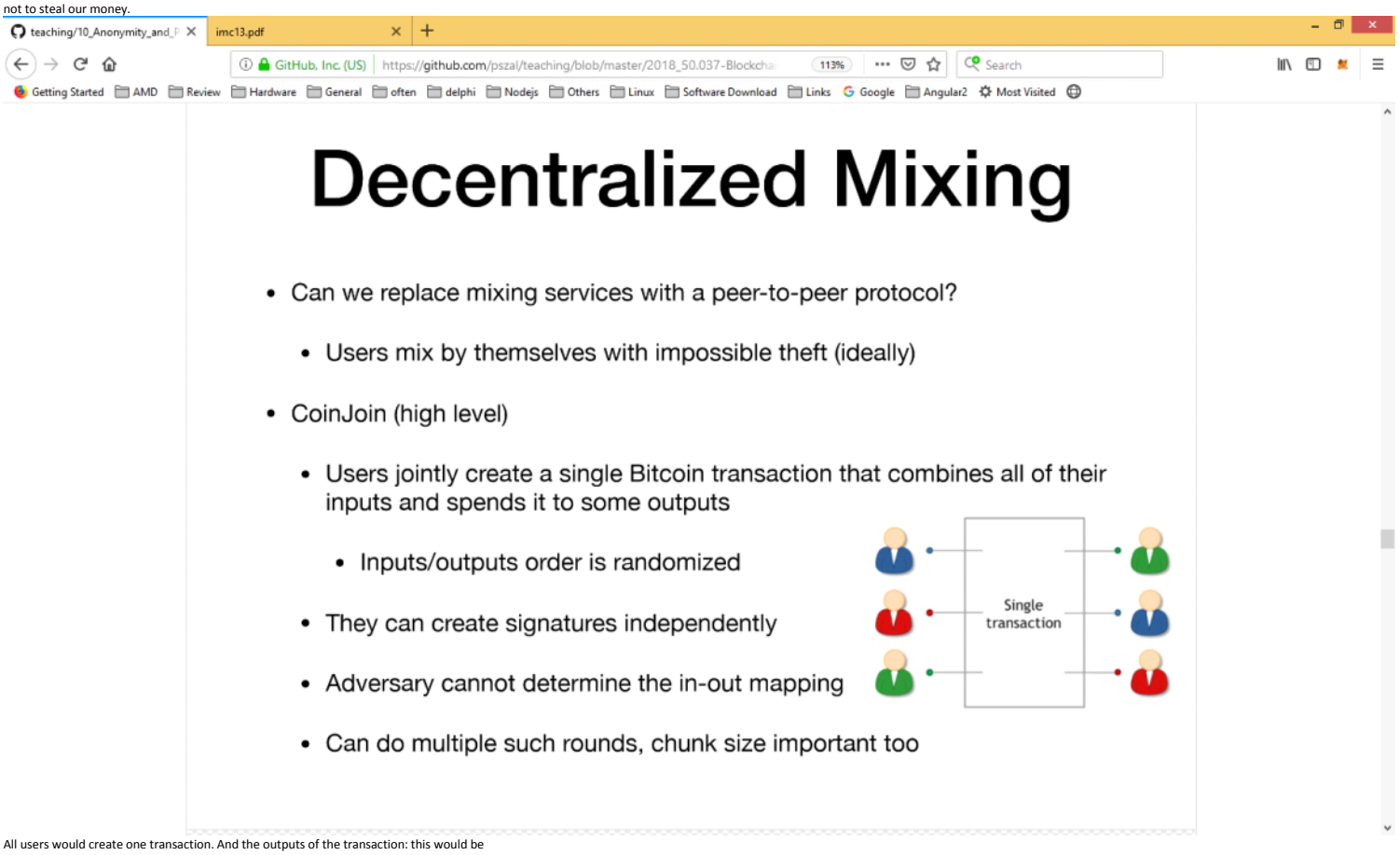
entities. Usually, they are just hacks.



If they are honest, it's pretty okay.
What happens here? Let's say everyone knows what is a mixer? If Alice is sending bitcoin, and this 1.1 bitcoin is sent through another mix, and send to Bob. It's just trivial to mix it. At the end, if the amount is unique in the mixed network, then you can very easily de-anon it. They need to operate on the same chance of bitcoins. So they need to divide.

So the client side should be automated. Send a fix amount of units.
They would like to charge for the infrastructure used by the mixer.

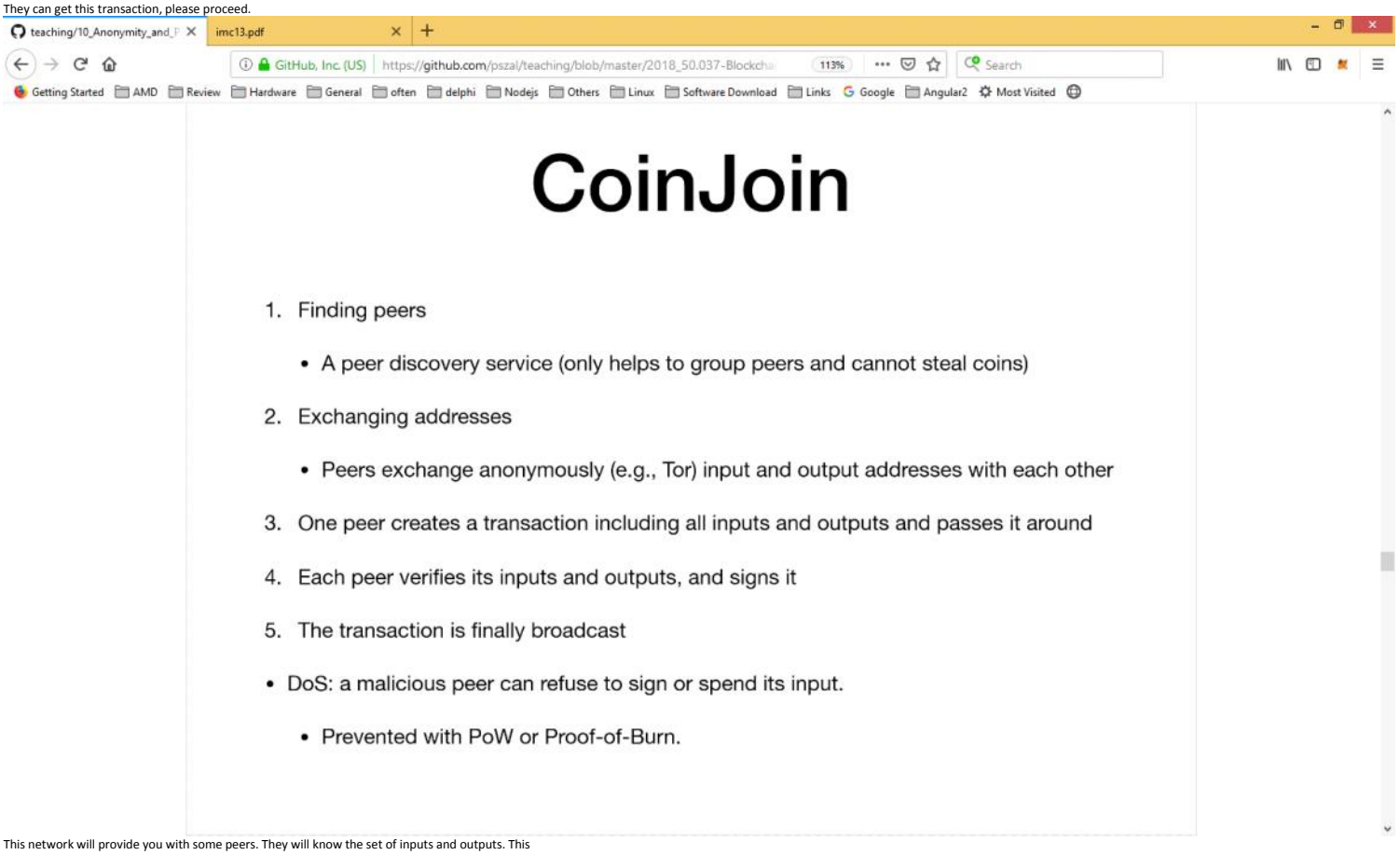
The mixers know incoming transactions, so they know what are the outgoing transactions.
So it would be slow?
Disadvantages? Expensive?
All securities are about minimizing trust; but we are trusting them in providing de-anonymizing; and not to steal our money.



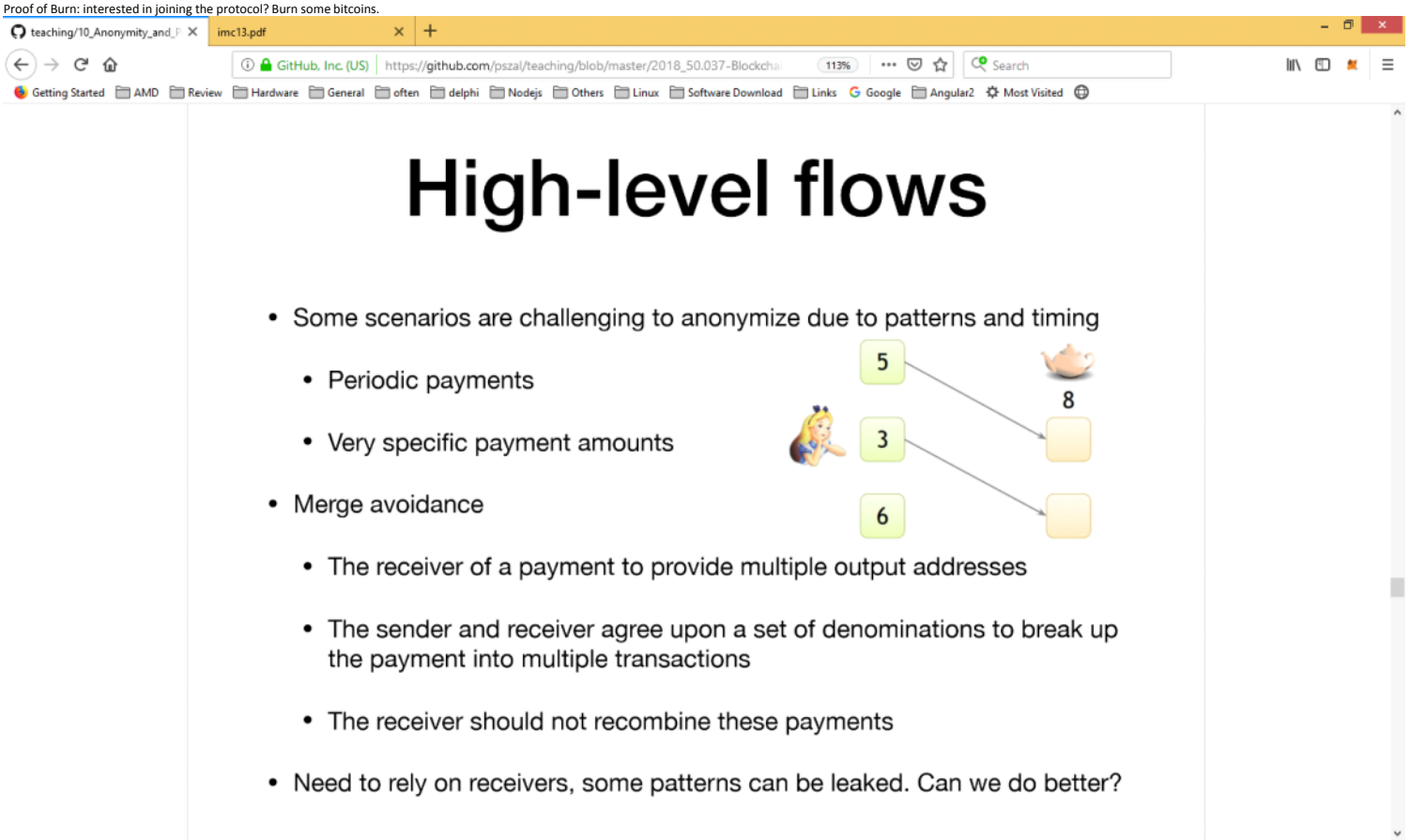
All users would create one transaction. And the outputs of the transaction: this would be

randomized. So you would not be assigning input to output, but randomnizing. Here would be just input and outputs, and then you would mix it. The order will not be preserved. One observation would be that it is impossible. This input corresponds to the output. If you have a pretty large anonymity, it would be difficult to figure out who is spending what.

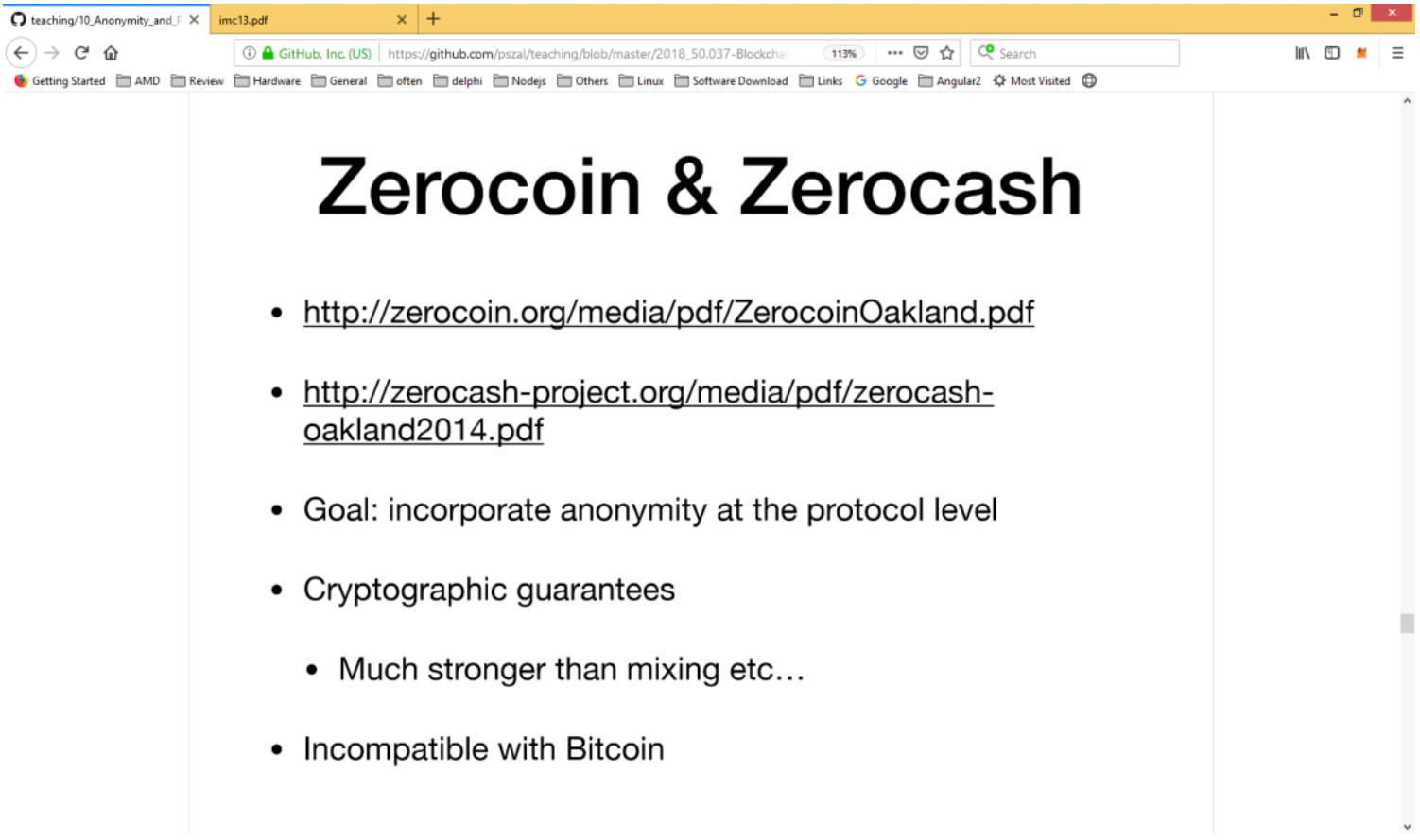
They can get this transaction, please proceed.

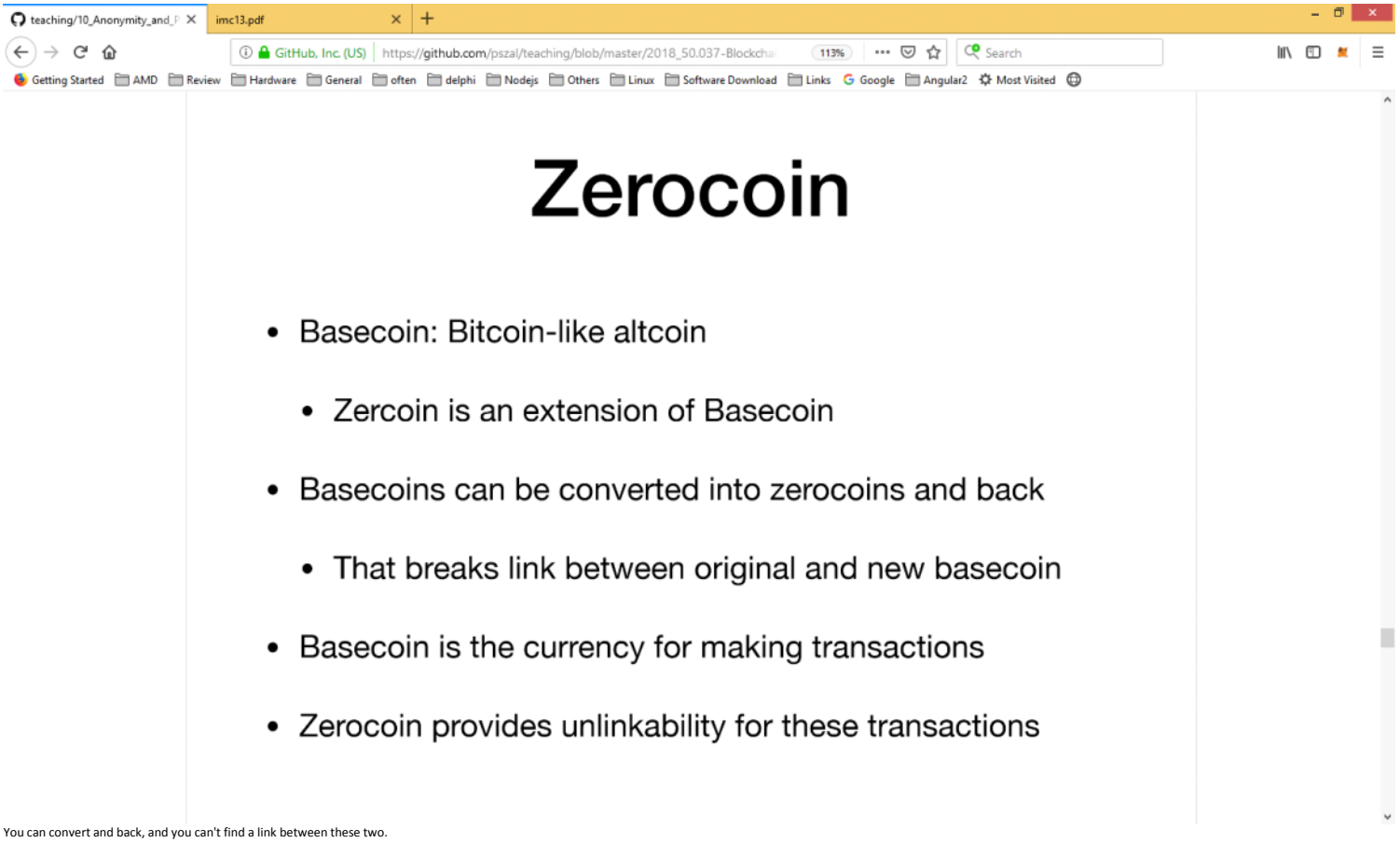


This network will provide you with some peers. They will know the set of inputs and outputs. This peer will start spending to other peers.
Proof of Burn: interested in joining the protocol? Burn some bitcoins.



You need to rely on the receiver not to recombine this.
To never merge transactions. Because this leads to the control being handed over.
Why can't I build a bank using this system? You need to make a lot of transactions?
Can you build a currency with privacy and anonymity built in?
You need to avoid double-spend attacks. So the user would not spend in money. Could have some kind of ledger to check that it is correct. That is very challenging. There are two systems that will achieve it.





You can convert and back, and you can't find a link between these two.

Zerocoin

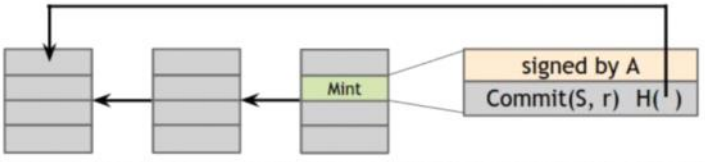
- A Zerocoin is a cryptographic proof that you owned a Basecoin and made it unspendable
- Miners can verify these proofs
- How?
- Zero-knowledge (zk) proofs: proof on a statement w/o revealing any other information that leads to that statement being true
  - Examples:
    - I known x s.t. H(x) = f730ae…
    - I know x s.t. H(x) is in {0349e…, 330ea…, f850ed…, …}

It's a mathematical statement: that you own the base coin and made it unspendable. I had one basecoin, and now I just spent it; can't use it anymore. Miners can verify this; they don't know which coin she used, but they can verify that it is correct.

Minting zerocoins

- Anyone can mint a zerocoin (let's assume 1 basecoin denomination)
  - Put a zerocoin on blockchain requires 1 basecoin

1. Generate serial number S and a random secret r
2. Compute Commit(S,r), the commitment to the serial number
3. Publish the commitment onto the block chain. This burns a basecoin, making it unspendable, and creates a Zerocoin. Keep S and r secret for now.



Anyone can create a zerocoin, but need to proof that you have 1 basecoin.

Spending zerocoins

- At any point, there will be many commitments on the blockchain c_1, c_2, …, c_n

1. Create a special "spend" transaction that contains S, along with a zk proof of the statement:

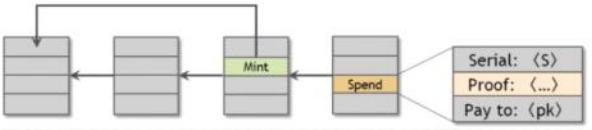
   "I know r such that Commit(S, r) is in the set {c_1, c_2, …, c_n}".

2. Miners will verify the zk proof which establishes an ability to open one of the zerocoin commitments on the blockchain, without actually opening it

3. Miners will also check that the serial number S has never been used in any previous spend transaction (since that would be a double-spend)

4. The output of your spend transaction will now act as a new basecoin. For the output address, you should use an address that you own.

- Once a zerocoin is spent, the serial number becomes public, and no one will ever be able to redeem this serial number again



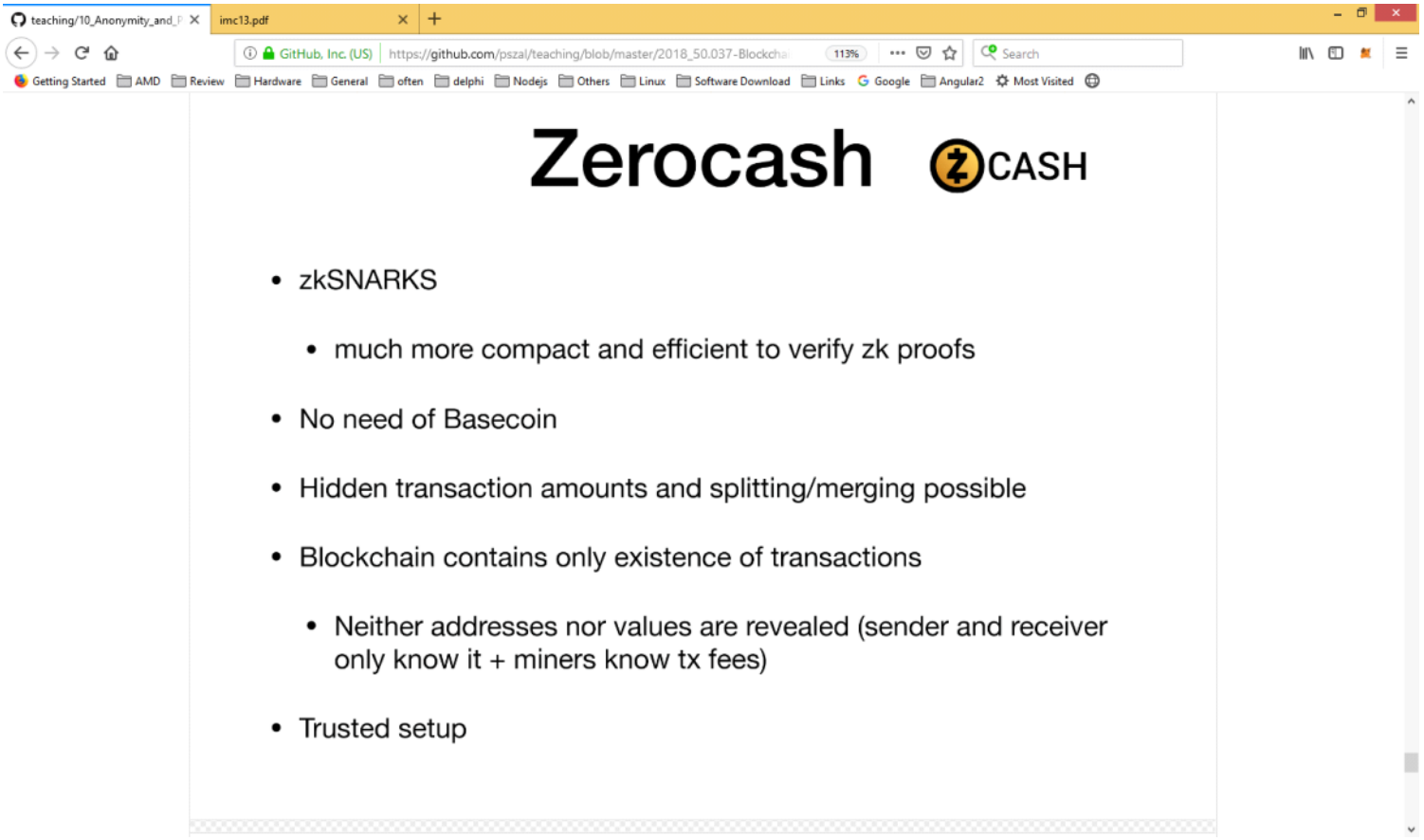
We reveal the Serial number, but not r. We know that such r, that commitment of this s and r is in the set of all active zerocoins/commitments. We prove that we know r, that the commitment of the serial number is in the set of all unspent commitments.
Miners can now verify it. They are sure we are spending something that we are allowed to. They know that only this commitment in the set. They check whether the serial number is unique, check that it cannot be spent twice. Every time you create the unique serial number, you want to spend the zero coin twice. This is in the set of all commitments, and it was never used before. They don't know this r.
In the output, you can say where to spend it. You create in the output of this transaction, another basecoin2.
The serial number is public, everyone is sure that you have created one. For miners, this zerocoin, is just the number of the commitment. They know that you could do it; but you don't know which commitment is yours.

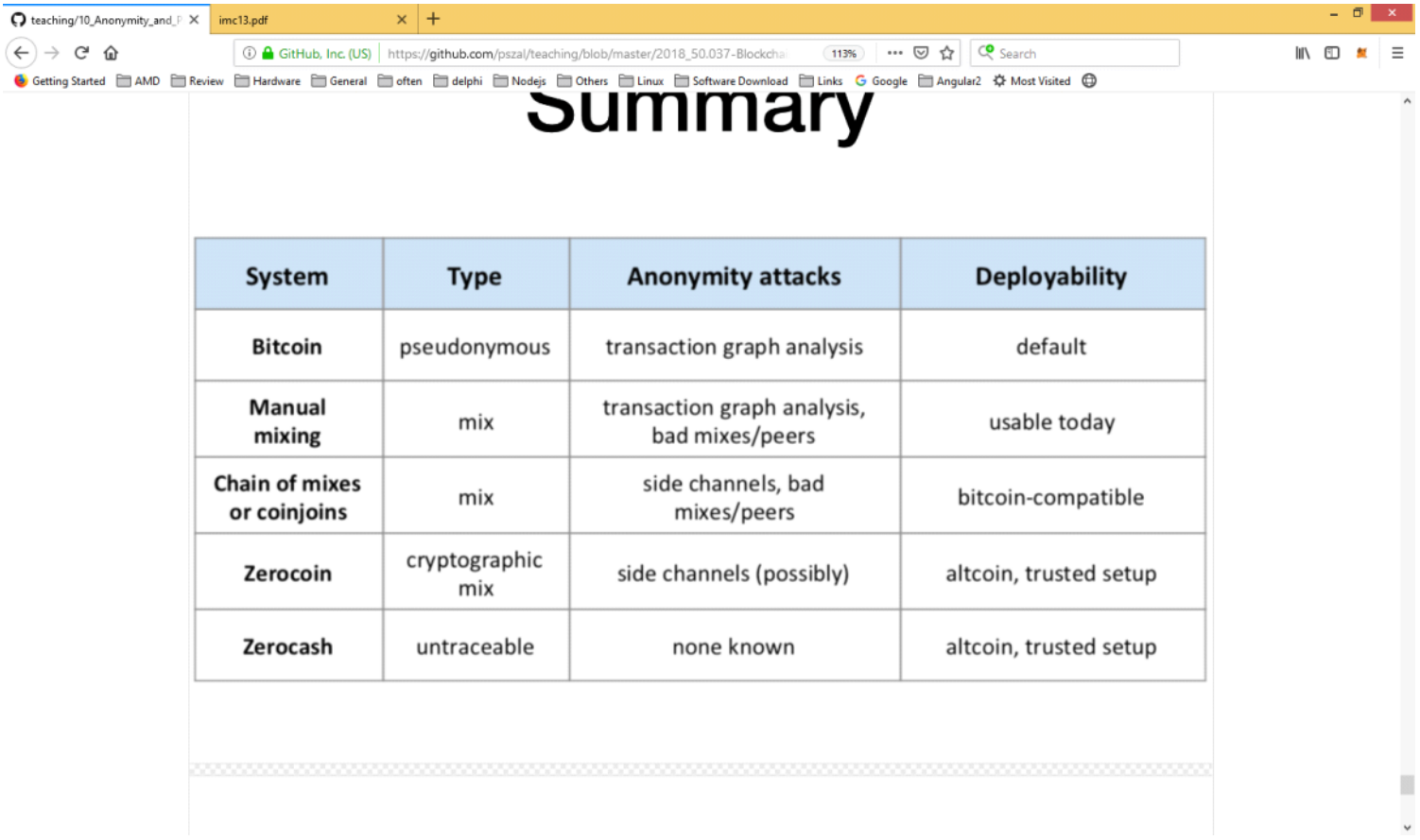
teaching/10_Anonymity_and_P | imc13.pdf
GitHub, Inc. (US) https://github.com/pszal/teaching/blob/master/2018_50.037-Blockcha 113% Search
Getting Started AMD Review Hardware General often delphi Nodejs Others Linux Software Download Links Google Angular2 Most Visited

Zerocoin Properties

- Anonymity
  - r is kept secret, nobody knows which serial number corresponds to which zerocoin
  - No link between mint and spend transactions
- Efficiency
  - Size of zk proofs is logarithmic in n (but still ~50kB)
- Trusted setup
  - Some security-critical parameters have to be generated prior deployment (can be removed after the setup)

Efficiency: Not great! But it works. Zerocoins decide how basecoins are spent. But semantics are like bitcoin. This trusted setup, if someone has this control, that may be difficult to deal with.
Improvements?





System	Type	Anonymity attacks	Deployability
Bitcoin	pseudonymous	transaction graph analysis	default
Manual mixing	mix	transaction graph analysis, bad mixes/peers	usable today
Chain of mixes or coinjoins	mix	side channels, bad mixes/peers	bitcoin-compatible
Zerocoin	cryptographic mix	side channels (possibly)	altcoin, trusted setup
Zerocash	untraceable	none known	altcoin, trusted setup

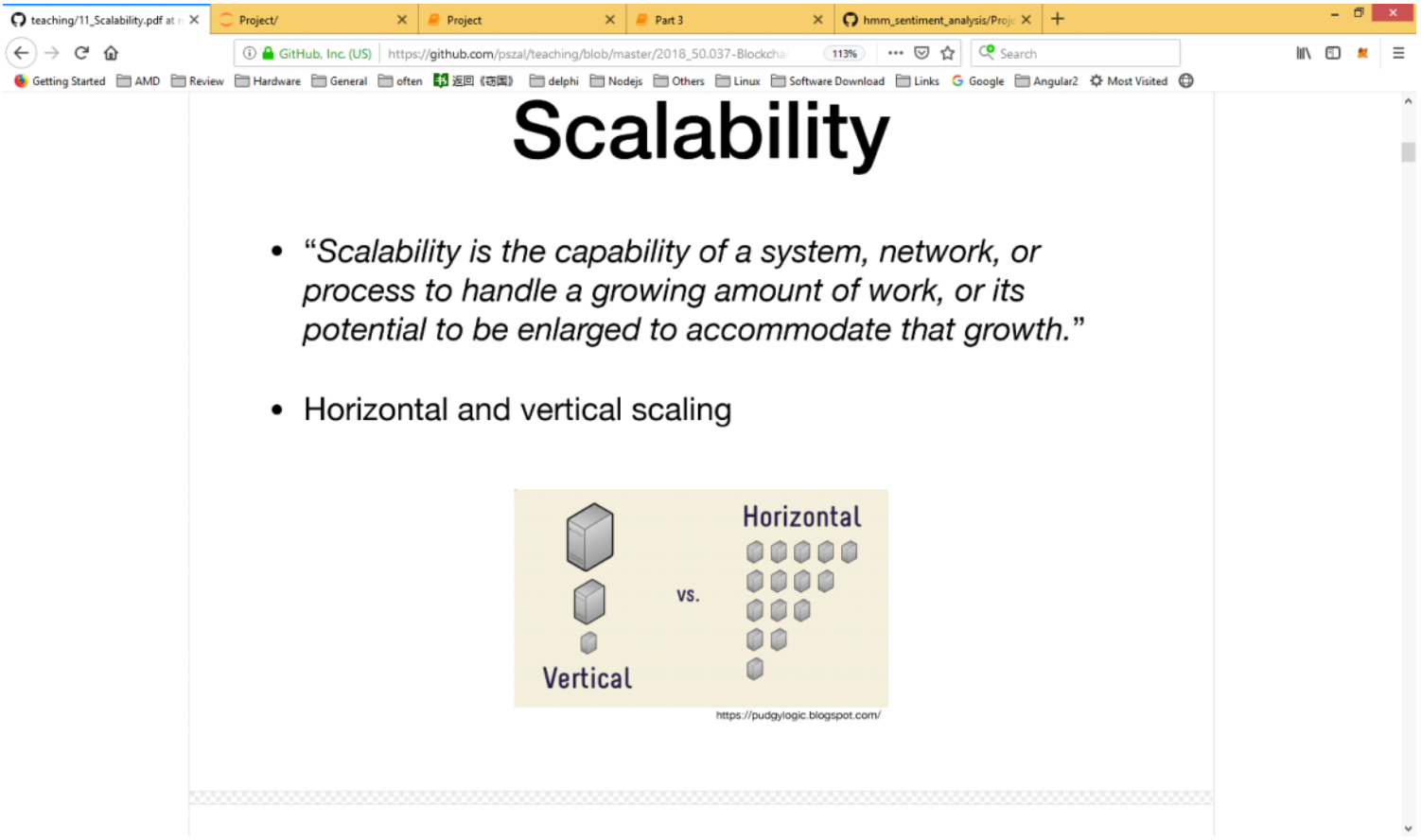
Lesson 11

Tuesday, November 20, 2018 8:33 AM

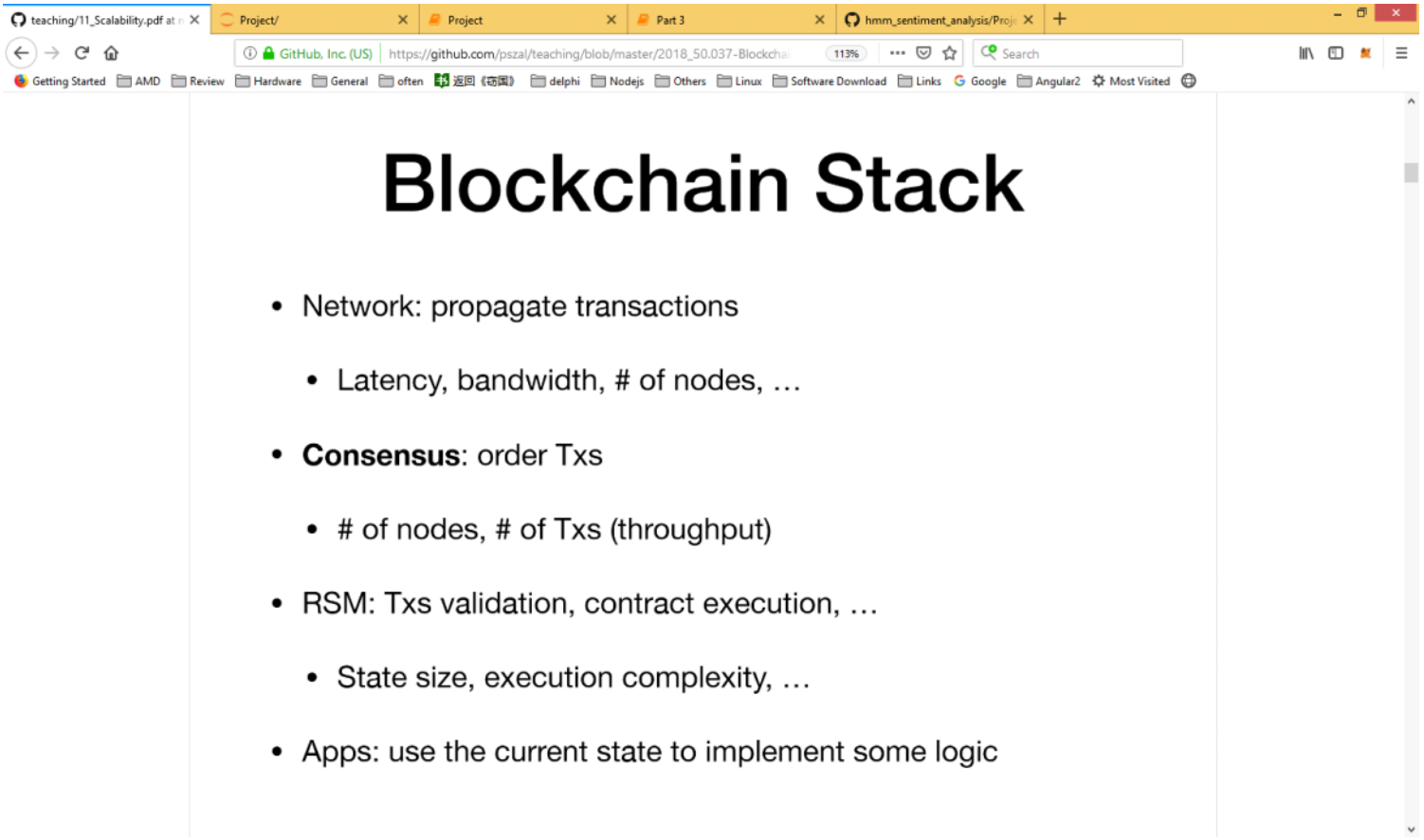
Scalability: If you think of all of your algorithms, we think about how they behave. It's the complexity of algorithms. If we go to infinity with n. You have function which is bounded in infinity by some function cm**2. So you can put any constant. If your execution time in infinity is bounded by some n**2, then it has exactly the same complexity.

This way of describing scalability? It's meaningless. It will not happen from your point of view. This is the problem with this notation, can prove that the algorithm has some properties, but there are asymptotic, they go to infinity with n, but in some cases, if your function starts behaving nicely only after 100 steps, you shouldn't discuss about that.

Scalability in algorithms: we have notation, this function is bounded by this function. In distributed systems, "Scalability is the capability of a system, network or process to handle a growing amount of work, or its potential to be enlarged to accommodate that growth."
In a web server, what can go wrong? One is traffic. So bandwidth. Or Memory: your process is taking up more memory? Put in more. At some point, you notice you cannot scale your machine all the time. So you need to figure a way putting your loads into multiple machines.



Vertical is easier: Because you don't have to change your algorithm, if you put another 16GB of RAM. If horizontal is more complex, why are we using horizontal? You can't go better than the strongest machine in the world. It's also availability. Not only scalability, but availability: if you have one powerful machine going down.



Latency:You compare how long it takes to disseminate this transaction from this point to this point.
Bandwidth: How many transactions the network can handle.
How the network is behaving if the number of nodes is growing: This is when we use the big O notation.
Each node to each node: if it's growing, o(n^2).
If we have one million nodes, then it will not work. Conceptually, first they should order the transactions. Then they validate the transactions: checking headers/ or unspent outputs. So if you mean validation of header, then that's fine. But in general, you should have order of transactions. You need to agree on which one is first, which one is second.

So we build a chain. This is a consensus protocol. If you describe how it scales, what is critical about the scaling consensus? Does it matter on the month? Yes. What does the throughput depend on? If you put a lot of transactions, it may not be able to handle that. We can assume that network has some constant delay. So why didn't it work so well? Because each message goes to other nodes. So they had small number of nodes, they were sending messages, so you cannot have more than a thousand. So it depends on the number of nodes.

If we have an order of transactions, then what do we have then? We have Replicated State Machine. It's the simple logic in bitcoin, that you can only spend unspent transactions. In reality, these layers are a bit confused. They are mixed and overlapping. In the bitcoin, the consensus is a bit like: you'll never validate something that is incorrect. Why in bitcoin do we not accept transactions that are invalid? It's not sure as to which transactions they put them in. So why at the consensus layer, do we care what transactions are valid? We need to know the logic of those transactions?

How do lite clients work? They prove that transactions are equivalent. Conflicting transactions here. For full nodes, they can tell it is invalid. But lite clients only know it is in the blockchain, so they can be deceived that it is valid. You need to prove that there is no other transaction there is output in this train.

Another reason why it is not included because it gives mining fee for conflicting transactions. But this RSM in ethereum is much more common. So they order all these calls and contracts, and just execute here. Which is EVM in ethereum.
What about scalability of the stuff? We care about state size. It's the state size which is growing, and we have problems with just executing it. If you have this replicated state machine, then we have apps here. These are applications that we can use.

With networks, we know how to deal with them. With applications, we can deal with them. But we don't have consensus that can validate. We just don't know how to do it fast; without compromising all the properties. It is inherently slow.

If you look at the current state and throughput

Current State

- Throughput
  - Bitcoin: 7 tx/s
  - Ethereum: 10 tx/s
  - Visa: 50k tx/s
- Strategy
  - Faster tx processing
  - Faster consensus
  - Parallel execution

What can we do about this? We want to put here at least 2 zeros for Bitcoin. Whenever we have this chain, how to put more transactions there. So this is one thing. Another thing is how do you have faster consensus? We have more of these blocks, the blocks are more frequent. This is not complete, because we are bounded by processing in one machine. How to process this in parallel. We don't have just one miner committing this transaction, but many miners dividing this work among themselves and then merging.

teaching/11_Scalability.pdf at m | Project/ | Project | Part 3 | hmm_sentiment_analysis/Proje
GitHub, Inc. (US) | https://github.com/pszal/teaching/blob/master/2018_50.037-Blockcha | 113% | Search
Getting Started | AMD | Review | Hardware | General | often | 返回 《动画》 | delphi | Nodejs | Others | Linux | Software Download | Links | Google | Angular2 | Most Visited

Why needed?

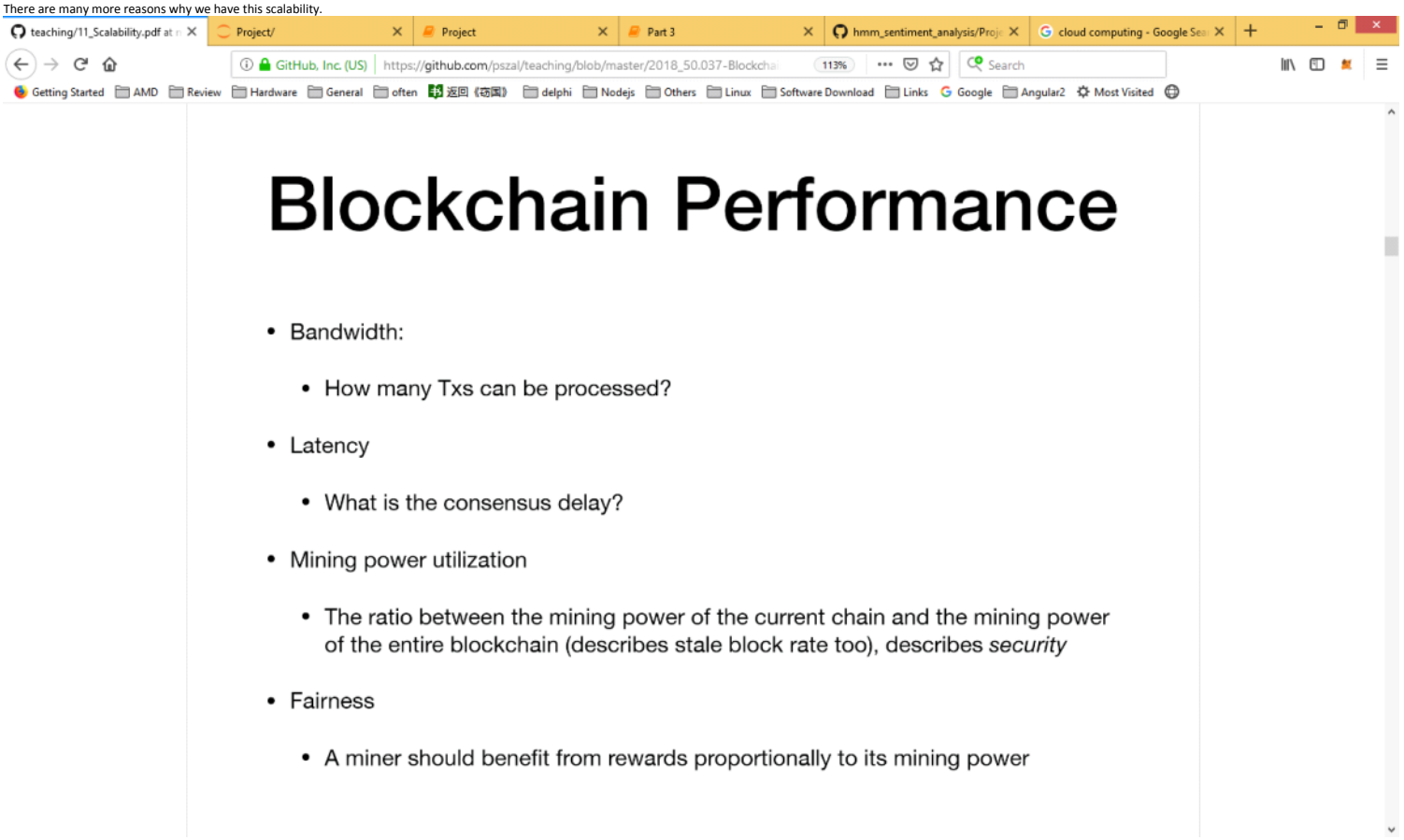
- Adoption
  - Real-world apps require high throughput and low latency
- Inverse scale effect
  - Fees

There are a few things;
For adoption of this technology: it's critical. To have real world applications, it has to be faster than 6 transactions per second.
I mention about cryptokitties. All the system: in a very strange way.
Why is cloud so successful? Because it increasing the computation? You can compute more stuff? You can put more servers too? It's cheap: you are sharing your server space with others, you can manage it very cheaply. It's like everything works a bit differently. There is this huge scale if you are using cloud computing.

The more people using this, the more expensive it gets, because we have some inherent limits. If

there are more transactions, then the fee goes up. They cannot buy another data center. So we need to increase transaction fees. Graph one year ago, there were 50 dollars per transaction; nobody is using such a system.

There are many more reasons why we have this scalability.

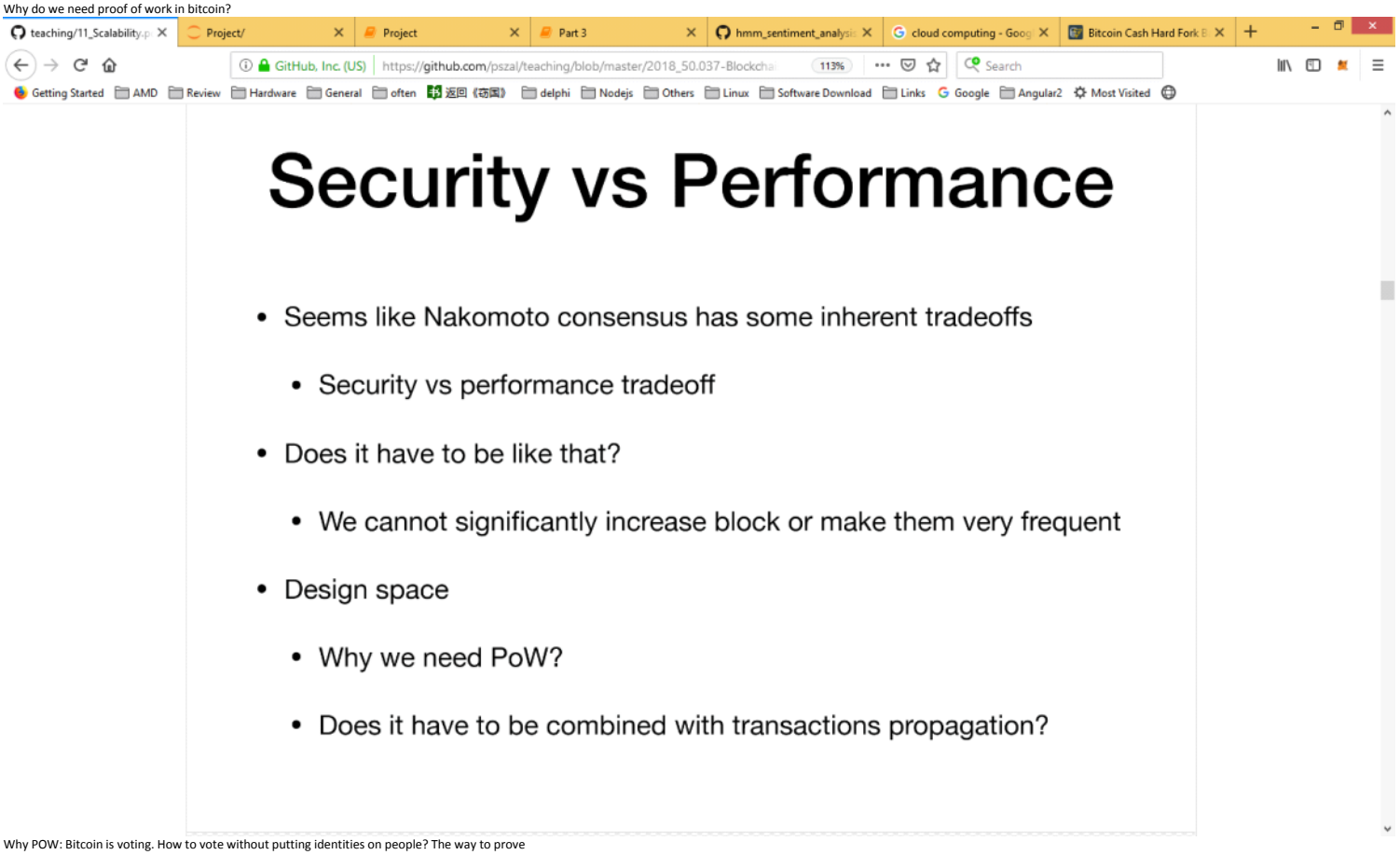


We measure using four properties:
From this point to this point: how many transactions can you pass by yourself? It's the most important, usually with the replicated state machine. How many transactions can we go through?
How long do you need to wait to get your confirmation.
With Visa: how long does it take for visa? Like four seconds. You get approval. Something like that, we need to have. So FinTech industry: they hate this latency and this proof things.
What is the ratio between the mining power of the current train and the mining power of the entire blockchain. Like 50%. How many resources is basically wasted. Why is it important to have it close to 1? Why is this chain less secure then having 10 blocks one by one? Less variability, can't tell which chain is the true chain.

Fairness: a miner present in the blockchain should be proportional to her mining power. In general, it seems that there is tendency to centralization. If they collaborate more, they will get more likelihood of mining.

If you want to increase the scalability, you can increase block size.

Bitcoin ABC: conservative. SV: greater size.
Why do we need proof of work in bitcoin?



Why POW: Bitcoin is voting. How to vote without putting identities on people? The way to prove that you are not creating hundred of identities is this proof of work. This is to prevent sybil attacks> Creating many mining blocks.
You have this proof: I'm not Bob or the other identities, this combined with transaction propagation.

Here, we have some epoch, with a new block forming.
They get these transactions, and at some point, they create their own block with a transaction. Now I'm a leader, and I can just put this transaction there. You don't control the selection process and when it would be next. You are bounded; you allow it to propagate.

And these are my transactions: one data structure.

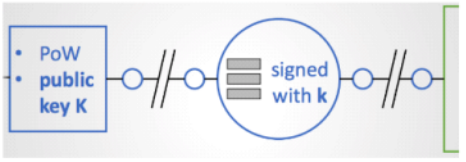
Bitcoin-NG

- https://www.usenix.org/node/194907 (paper, slides, talk)
- Insights
  - In Bitcoin, leader election and transaction serialization is combined
  - Why do not try to decouple it?
    - Elect leader via PoW and let her commit transactions
    - (Different order than in Bitcoin)

First you are running proof of work, I'm the guy allowed to add it, and then you are sending transactions.
How it works.
You have two things: keyblocks.

Bitcoin-NG

- Key blocks
  - Used for PoW-based leader election, i.e., H(header) < T
  - Point to the previous block (key or microblock)
  - The strongest-chain rule
- Microblocks
  - Generated by leader, at a defined rate
  - Contains header (with PrevHash) and a set of transactions



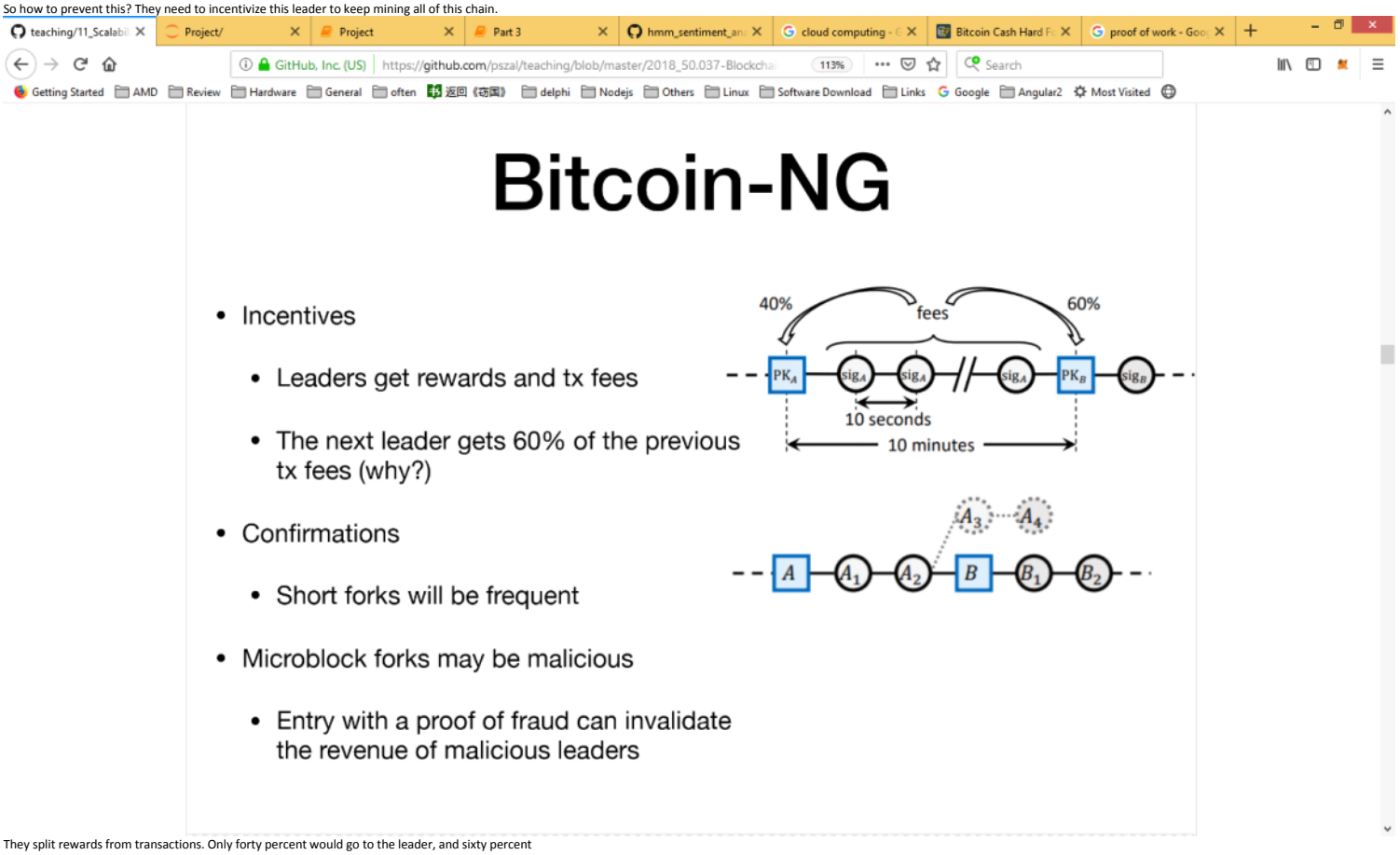
It contains this proof of work. In addition to bitcoin header, it has a public key.
It has a pointer to the previous block.
We have the same rule:
We have another thing: microblocks.

Here are blocks that contain all of these transactions, which are signed by the public key of the leader. They have no proof of work, but link to the previous block. So basically here, the leader would be sending transactions ever 10 seconds, and these transactions would happen to the blockchain. This guy is allowed to do it. You are changing proof of latency.

You will need to get confirmation, but in bitcoin, for 10 minutes, the network is like that. They are collecting transactions, and see who is first. We know who the leader is, let's send transactions there.

So leader action, send all transactions to the network, have another election that confirms all the transaction.
When to look for a new leader? This guy can step out any point of time. He is just ignoring all these transactions. I start running a new leader action, because I want to be a new leader.

So how to prevent this? They need to incentivize this leader to keep mining all of this chain.

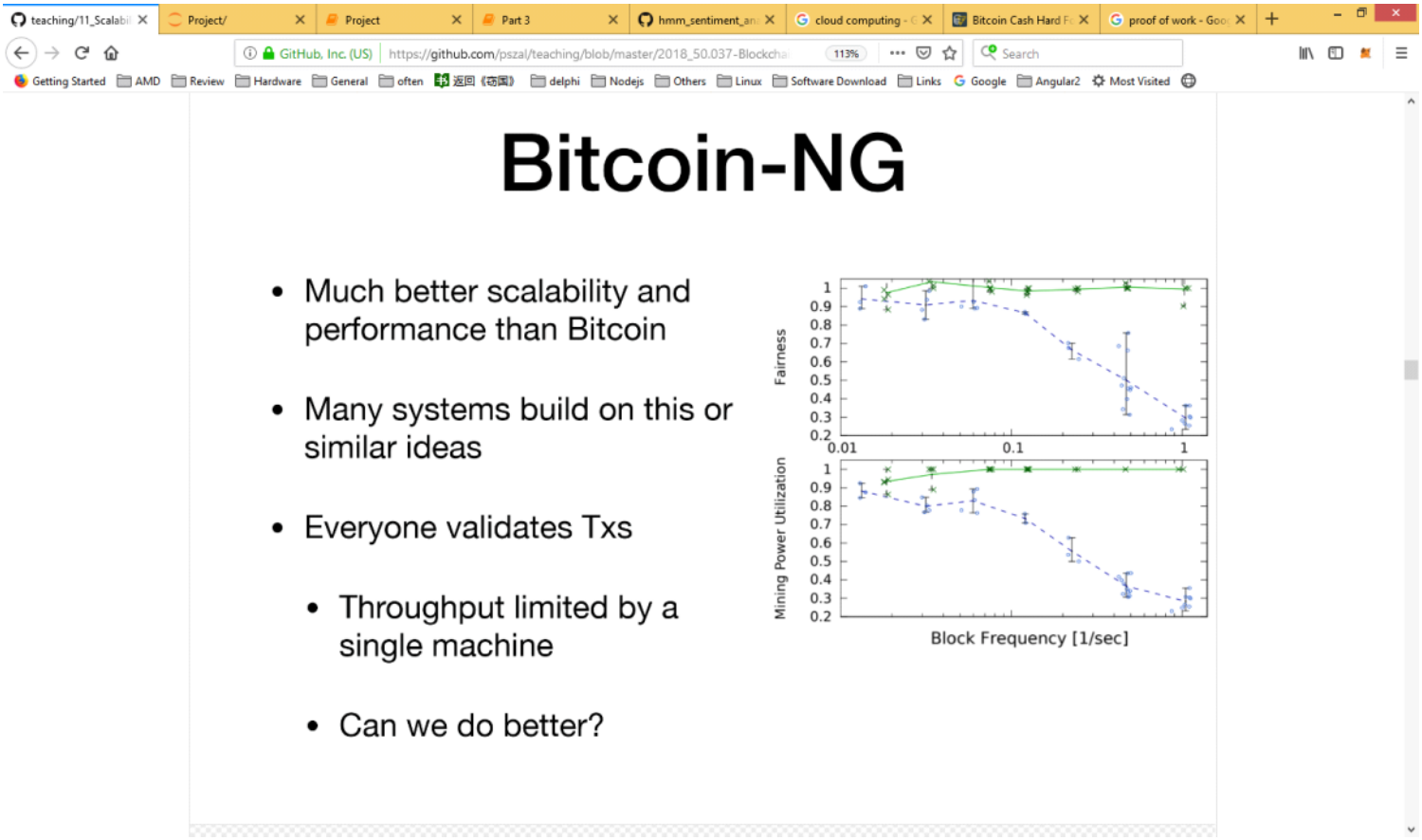


They split rewards from transactions. Only forty percent would go to the leader, and sixty percent goes to the next one. Miners are incentivized to mine on the most recent microblock. If they are here, starting a new fork, there is no microblock, no transaction fee. But if they keep mining on the microblock, so it increases the revenue. It's a mechanism to keep them mining on the chain.

What will happen is the forks on microblocks. If new leader started, the microblocks get cancelled, he will have this transaction, there is this latency issue. If he notices the microblock, so he benefits from it, and now he gains from them. Because this will be pretty fast and centralized. So you wait like ten minutes. However, you need to remember, this block is like confirmation of your transaction. You need to wait for another leader action, and you need to wait 10 confirmations, and every confirmation is like this key block.

So what is more interesting is that when you generate this blocks, it's very costly to create conflicts by miner. If miner would like to create a fork in constant transactions, miners need to invest twice. Because he needs to generate more than one block. But if you have a leader, you can fork. You have one transaction that conflicts with another, create two transactions that are invalid. What they do? They have punishment mechanism. So this miner, if he notices that there is a fork in microblocks introduced by a leader. This one can create a proof of fraud transaction.

So this transaction would prove that this and this are inconsistent. So this proof will take rewards from the 40%, and send it to the next leader. They are incentivized to control themselves. Miners cannot spend coins in the next 100 blocks. If you mine something, you cannot spend it, but the network is observing you, so if they find something, you will get this reward.



It has much better scalability, because not waiting 10 minutes. If you see like block frequency, in both fairness and mining power utilization, it's close to 1.
It has much better scalability.

Summary
Proving that you have identity is pretty slow: proves that you are not a bot.
If it takes 10 minutes, it will be okay. But your transactions are bound by this process. Every time, I'm proving not a bot. You need to wait 10 minutes before confirmation. There is no new transactions to be validated in that time. They are not appended in the blockchain. They wait until someone else is finding a new proof of work.

This proof of work is tied to transaction serialization. I spent a lot of time, this is my 1000 of transactions. No one can add transactions to the leader, because the new leader is elected. In Bitcoin, the leader election is tied to the transaction. Usually, now you are sending transactions there, and he is sending it there. In bitcoin, who is lucky, who will make transactions. No new transactions is added, so the throughput goes down.

The idea here is have this leader election first, and then let leader add transactions. We have this header. Which has proof of work inside. He has nonce, and private key. Let's say I'm Miner A, and want to use BITCOIN NG. I look for a header that has hash less than this target. I put into this header my public key.

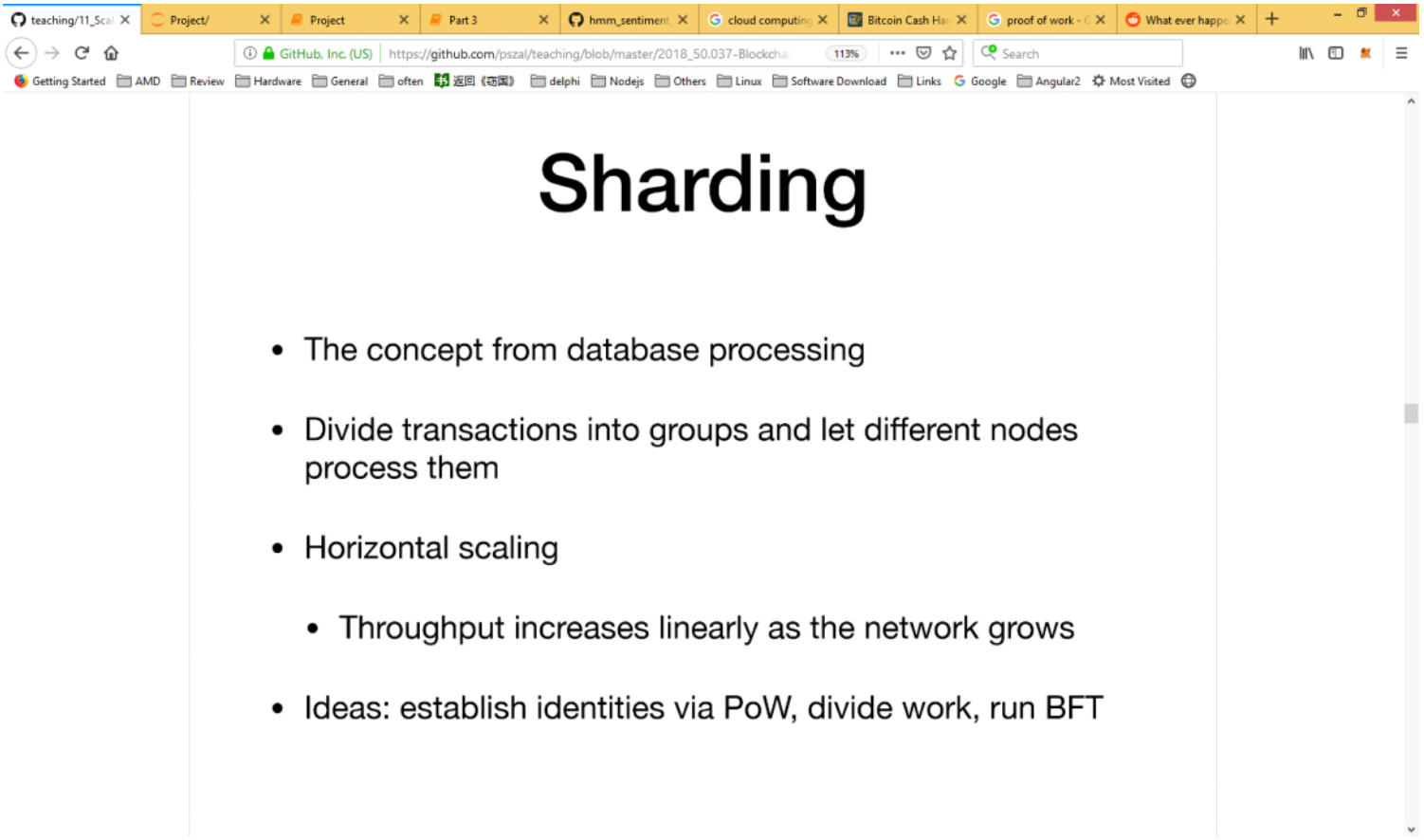
I have hash of this nonce, previous header, transaction and so on, and I have my public key. I'm trying to bruteforce nodes, and at some point, I shall find it. So when I find it, I announce it. This is called a keyblock. So this has my public key inside, link to the previous block. All of these stemming from bitcoin. I'm the lucky guy adding these transactions. This block has no transactions inside. What happens here: I'm creating this microblocks. This microblocks in every 10 seconds which contain all of these transactions, and they are assigned with my public key.

So this basically a microblock, with transactions. I'm just listening to transactions and put everything to the ledger immediately. I sign it with my key, and everyone just trusts the leader because he won the race, he is authorized to add to the transaction. This microblocks contain some meta data, but no proof of work. He can just keep adding this forever.

It is faster because there is no proof of work, the transaction is added before another leader to add them. So the proof is not a problem. You don't have huge blocks, just small packages of data, and everyone is able to get them. Then every miner tries to become a leader.
Because he will benefit from all of these transactions. He is giving all of these microblocks. Checks whether this transaction is not double spending. The consensus mechanism is different. You have leader election, serialization.

Someone tries to bruteforce, they keep pointing to the latest microblock. Each chain is kept to the previous block. So if they put in, the previous leader can't add any more transactions, he will keep adding and so on. He is incentivized to point to the previous block because 60% go to him. If he's not benefiting from this fees, everyone will continue mining for the previous miner, they will ignore his fork.

TLDR: More transactions being added.



What it gives us? In chain, everyone has to validate it. Now, we can somehow divide it, and we can give a portion of work to one node and the other. So it's really nice throughput because your network is growing, and your throughput is growing. What's your main blocks for running this kind of thing.

This Proof of Work is necessary to have the permission in this set. We have identities with this proof of work. We just elect leader in NG. So we establish identities with POW. It is only needed for identity.

Then we want to divide work among them.
And to agree (group them) and then, among these groups, you would like to run some classic consensus to achieve order of the events.

You establish some identities, divide them into groups and get some work. Within the group, they will run this classic protocol to agree what our view is in this committee. If they have results of this group, they have bft among these results. This is one layer of direction for this BFT protocol; aggregating results and so on. This is challenging in adversarial set. You want ot make sure they don't misbehave.

We can tolerate in BFT 30% of malicious entities. This selection into this group should not be malicious, such as 50% of a group is malicious.

The first sharding protocol:

Sharding: Elastico

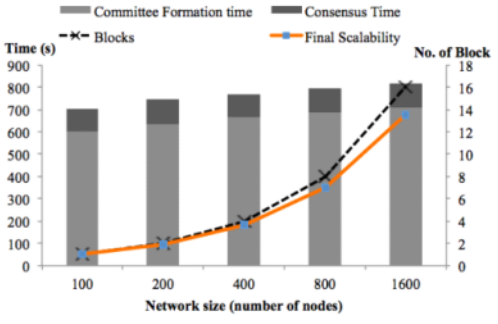
https://dl.acm.org/citation.cfm?id=2978389

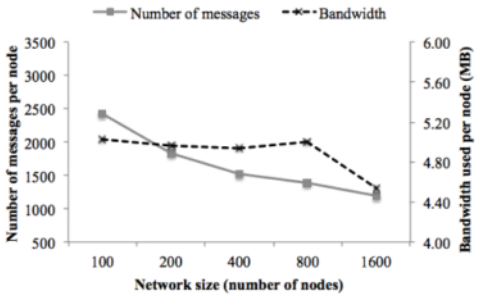
1. Use PoW to establish identities
   - ID = H(R, IP, PubKey, Nonce) < T
   - R is security-critical, see below
2. Assign committees (use randomness of IDs)
   - Each committee has C members and a directory server (w/ members)
3. Propose a block within a committee
   - Run BFT agreement, valid blocks have 2C/3 + 1 signatures
4. Final committee to union all data blocks
   - Run BFT to produce a final block, that is then broadcast to everyone
   - R generated using R_x of final committee members

They are assigned to the committees. This is random, no malicious nodes are congregated. How are they assigned? They use the randomness of the last bits of the proof of work. If it is less than target, then there would be many zeros in front. This is how we assign them to committees.

Each have a c node, and a security parameter, so C members in one committee.
It's fast because we are running classical fast protocol. We are using trick of establishing proof of work. You need to know what are participants. If you know participants, then you can run to create results. Need to decide what is it with the final block. You need a final committee which is selected randomly, and decide which is the block. The work of this committee is independent.

Sharding: Elastico





- Multiple improvements (ongoing research)
  - ZILLIQA, OmniLedger, Chainspace, Saber, …

In that 10 minutes of committee being formed, no transactions are being added. Before they start the round, wouldn't they have to ensure they join? As long as you have honest majority, like sixty percent, it would be fine. They are treated as Byzantine nodes. Honest nodes would be able to vote, can get 2C/3 + 1 signatures.

Lesson 12

Tuesday, November 27, 2018 8:34 AM

Forks are bad because a lot of mining power is wasted. Let's talk about this wasted thing, the bud of a chain. Let's say we have blocks with transactions. Let's say that this block has transaction 5, 6,7 and this block has transaction 1,2,3. Let's assume that they are non-conflicting, no transactions that try to double spend. What is bad about going with one. We can go with bitcoin either with this or that. Why is it not optimal? Observation is: you don't have many conflicting/double spending transactions. It limits us, it's less efficient. We could accept these two blocks, they can be overlapping, but the main point is that they are not conflicting, so why drop one?

DAG:
They store the relation between two nodes. This relation is like direction. We use arrows.
So the Nakamoto-like consensus is very easy, and the chain structure is simple. Can we introduce a more efficient data structure? For graphs, it is more powerful, as blocks can encode their worldviews.

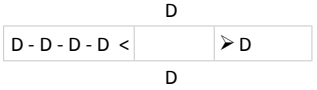
- Instead of having his list, we have a graph that would tell us something about blockchain. About this global mean. In the bitcoin, the miners might wish to drop the block that got wasted. How graphs are more powerful, is that you can capture the block, and point it to the whole view of the data structure. It's much more powerful, because not only are you attaching to the list of the previous block, you can see forgotten forks as well.

In our example, why acyclic? Otherwise: the block can link back to itself? Cryptography will ensure that it is acyclic due to the headers.

- One pointer that points to the header is needed, and that's competitively unfeasible because you don't know which node would occur.

SPECTRE
One of the previous properties, mainly for payments, not smart contracts.

- DAG can be used to encode the longest chain-rule
- Bitcoin is related to voting
  - Every block votes for its chain
    - Another node can say that I'm accepting this transaction of this fork, as well as another. A node that is pointing backwards. This is something you can't do in Nakamoto Chain.



    - In Nakamoto, you can only give your vote to one. So what's the point?
      - E.g. In a voting system
        - 60% and 40%. Republic and Democrat
        - Even if you have some of these blocks, you are only voting for one candidate. E.g. Voter A votes for Trump, Voter B votes for Clinton…
        - But there are some who vote for both? You can split your edges, and Voter C could be attached to Voter B in the Clinton chain if he thinks Clinton is good but still remain a member of the Trump chain because he still likes Trump.
  - Cloning in Bitcoin
- Vote amplification (miners strengthen the majority decision)

We have blocks in transactions and a proof of work system

- Miners create PoW-protected blocks with Transactions
- It's okay if the blocks point to all known *tips* of the DAG.
- It points to all other ends of the chain. The DAG will be growing and growing; we are putting this hash. Can we say that which is before which?
  - They link to the same blocks, you only know that they are created after this block. If we can give a list, we know that that they are preserving the order of the DAGs.
  - Blocks can have conflicting transactions, but miners are putting some transactions and they are doing proof of work with transactions pointing to all known ends of this.
  - Every miner has his own copy of the DAG. Every block can have one of three states
    - Robust Reject
    - Pending
    - Robust Accept

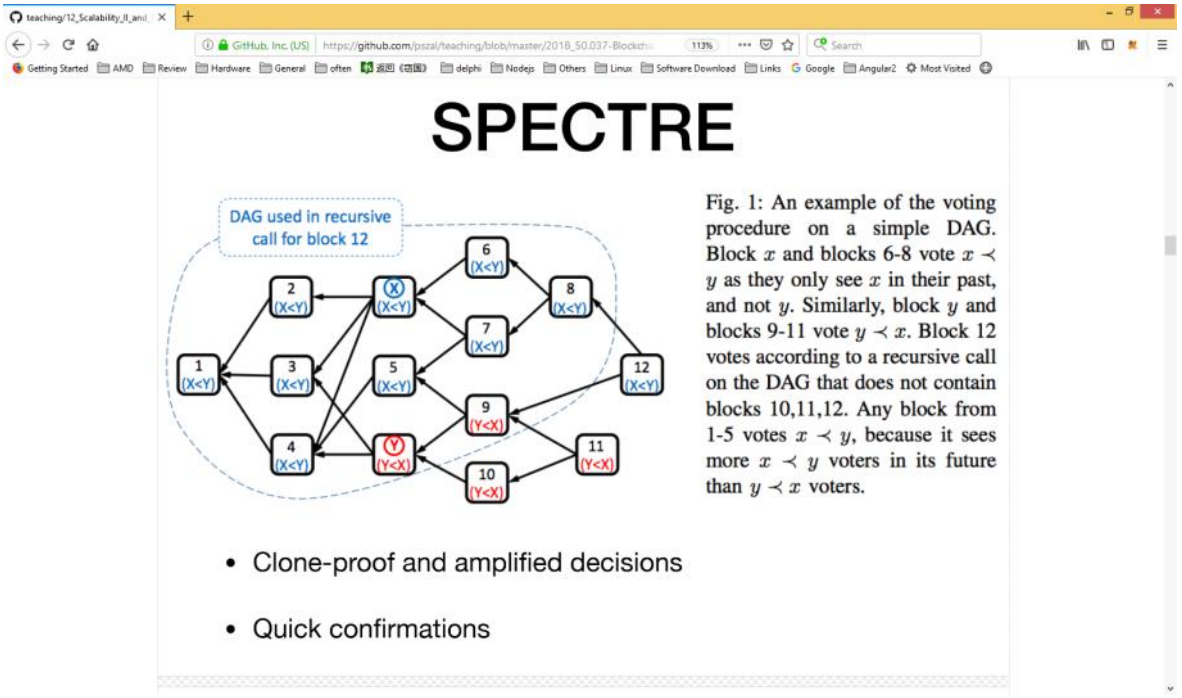
A	B	C	D	E	F	G
		D	E			

    - D and E (highlighted) are rejected. Rule Confirmation for Bitcoin is six blocks, but we can say it's like three blocks. It can change because some fork can happen, so the state of E,F,G is pending. ABC are accepted

If we are attaching a new block, this block actually votes for the state of this graph

- From the graph itself, we can decide what is the set of transactions which is accepted
  - The problem is that the structure should decide which is correct. There has to be some kind of voting. Either voting for this block, or that.
  - We cannot decide if D or E are the latest tips. We don't care which transaction is right, we just need to pick one to make a consistent view.
- So, transactions are accepted if all inputs are accepted

Only if the DAG decided that if the DAG is in a legitimate view.



You have two conflicting transactions, X and Y.

Now we need to decide whether X is before Y, whether X will be accepted.
12 has seen that more blocks in the DAG believes that X is less than Y, then we accept X. How many blocks believe in Y. Miner is just voting by adding this node. At this point, X will win. Because 12 didn't point to 9, we still need to weigh. The more nodes are added, the more decisions are amplified.

We have some rules as to which is accepted. Those in red are rejected.

By design, SPECTRE emphasizes the time-based nature of transactions in its voting scheme. If block ***a*** precedes block ***b*** and they conflict, then the protocol will ensure that ***a*** gets voted on the most, the fastest. By ensuring that every new block references all leaf blocks it sees, honest participants swarm to build on honest blocks. Attackers, revealing hidden blocks at some later date, do not benefit from receiving connections from honest blocks; this reduces the rate and amount that these blocks get voted on, i.e. SPECTRE prioritizes *quickness* over *powerfulness* in terms of sheer mining power.

From <https://medium.com/@drstone/an-overview-of-spectre-a-blockdag-consensus-protocol-part-2-36d3d2bd33fc>

Vote in favour of visible blocks. Honest miners build on honest blocks.
Majority amplification. Honest blocks will side with earlier (honest) blocks in conflicts.
Referencing recent blocks is beneficial. Building on recent, honest blocks gives more votes to honest blocks versus attackers mining hidden blocks who lose out on votes.
Votes from the past counter pre-mining attacks. Concurrent block creation usually means honest blocks will outnumber attacker blocks and thus vote accordingly if an attacker withholds block propagation.

From <https://medium.com/@drstone/an-overview-of-spectre-a-blockdag-consensus-protocol-part-2-36d3d2bd33fc>

Any ideas why is it not for smart contracts? The order is not as critical as for smart contracts.
12 can't see what 11's contract is.

Intel's Proof of Elapsed Time (PoET)

They claim that SPECTRE is scalable, but it's unclear? Because past confirmations, and some good properties.
You don't want someone to create many identities

You create a random number, and then you sleep
After you wake up, you check if no new block has propagated.
Like bitcoin, you are doing this proof of work, except by sleeping. I'm lucky in generating a random number, because if it is low, I will wake up quickly, and propagate
But I can't prove that I'm sleeping!!! I'm always getting 0 TD.
But if we could do something like this, it would be nice.

Intel Software Guard Extensions

Here, our OS can be compromised, VM is compromised, user space compromised, but we can still run safe applications
User code can allocate private regions of memory, protected from other processes (even those

running at higher privilege levels)

This is running code in isolated from the rest of the system

Main novelty:

Our DRAM isn't trusted. Our memory in the system isn't trusted. Only the CPU

CPU is encrypting and decrypting the memory on the fly.

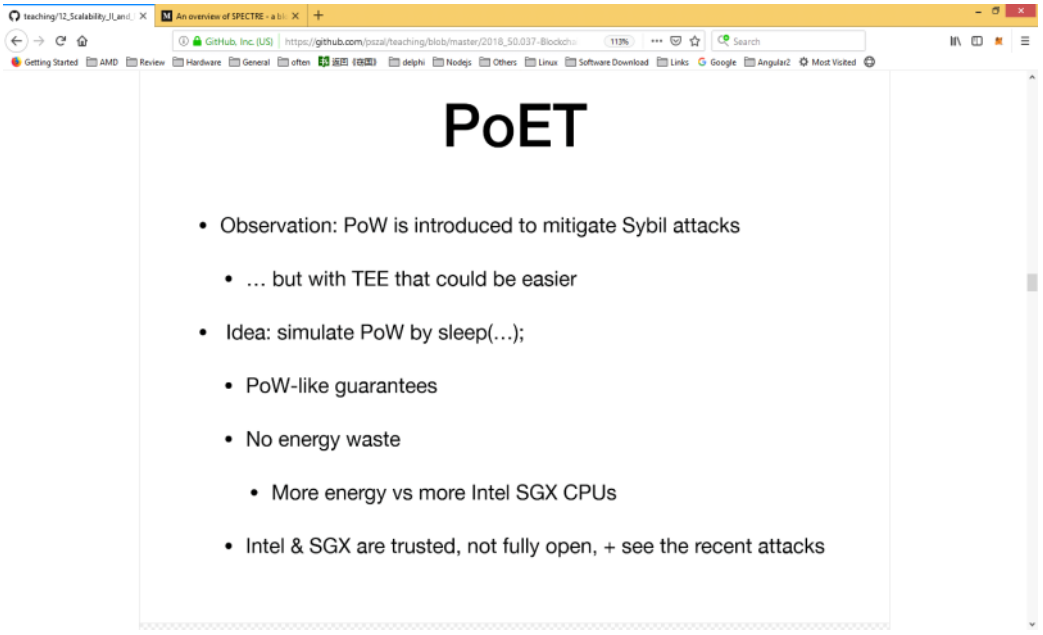
Tamper proof module.

The system can prove what code enclave is running on the system

I can verify that this server is running the correct game, mail server.

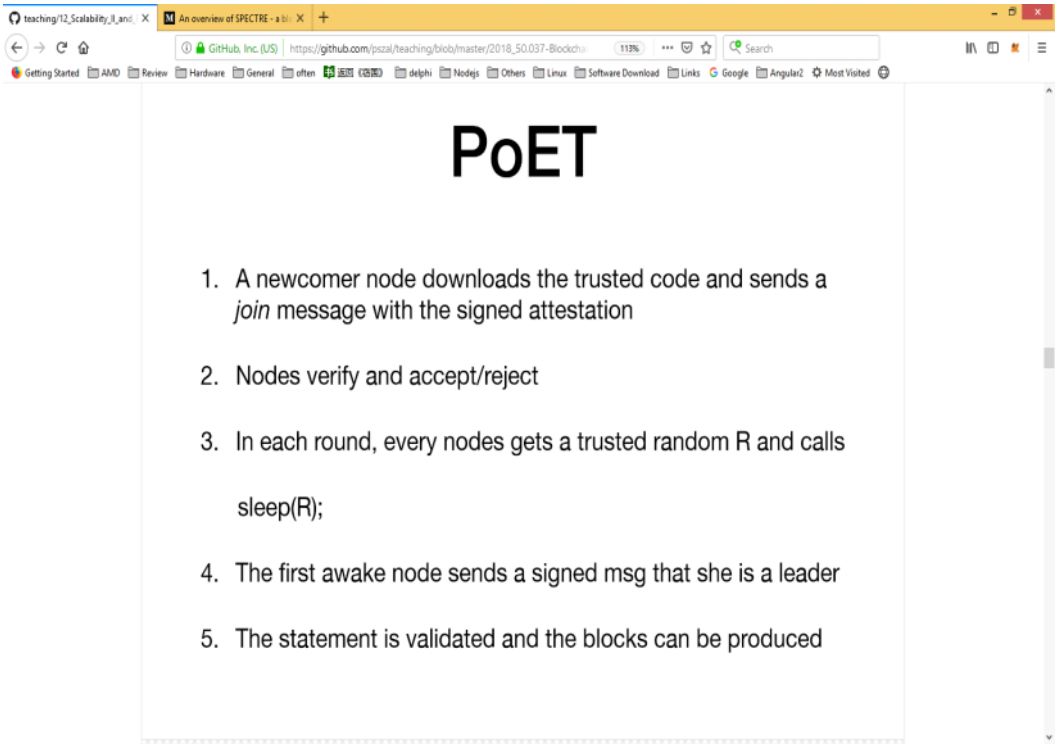
So we can say, run this instead of hash. I can verify that the miner is running the sleep function, because the remote that the station is constructed is assuming that everything else is realized.

In a trusted execution environment, you can mitigate it faster. Given the node is running this code, I can verify that it's doing proof of work.



So now the competition is: I want to buy more ASICS. If we have this, we buy more Intel CPUs. They would buy old CPUs.

Only Intel CPUs can be used for mining, then miners can buy more and more CPUs.



What's bad about this protocol? The huge trusted party is Intel, the statement that this is trusted is signed by CPU. The key is that you should buy Intel, which could create Sybil Attacks. Bitcoins where the bitcoin can reject the miner?

That new node is a petition. If we have currency, this node is like a competition. Let's say each miner agrees to letting the new node get a little more profit at their own expense?

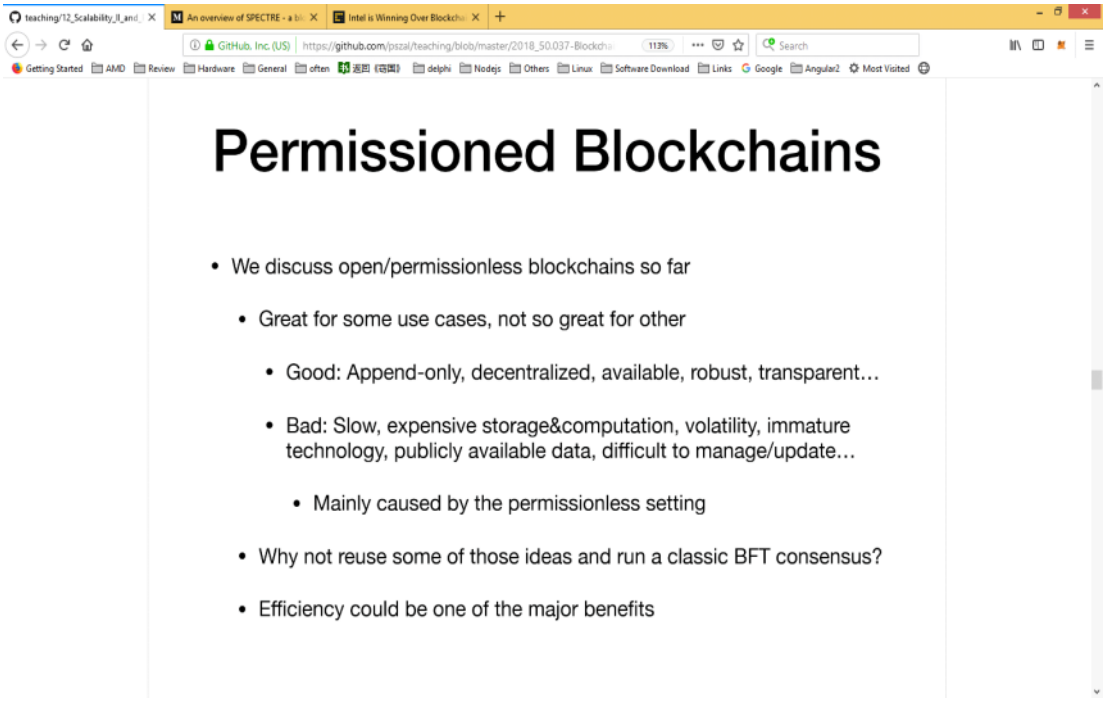
Permissionless blockchain: No one can stop you from going there. Here, someone needs to be refurbished

Has an Intel CPU

Other nodes will have to verify and accept or reject you. Why would they decide

If you have an environment where you don't have a block reward, then accepting the node is okay, because he's helping you run this protocol.

Permissioned Blockchains.



How you can vote without identities: it's great for some use cases.
It's decentralized. It's robust: fraction of nodes die, it can still run. Transparent: Censorship resistance

Volatility: Random>
Can we build applications on top of that?
Data is visible for everyone else
Difficult to manage, because they don't control something.

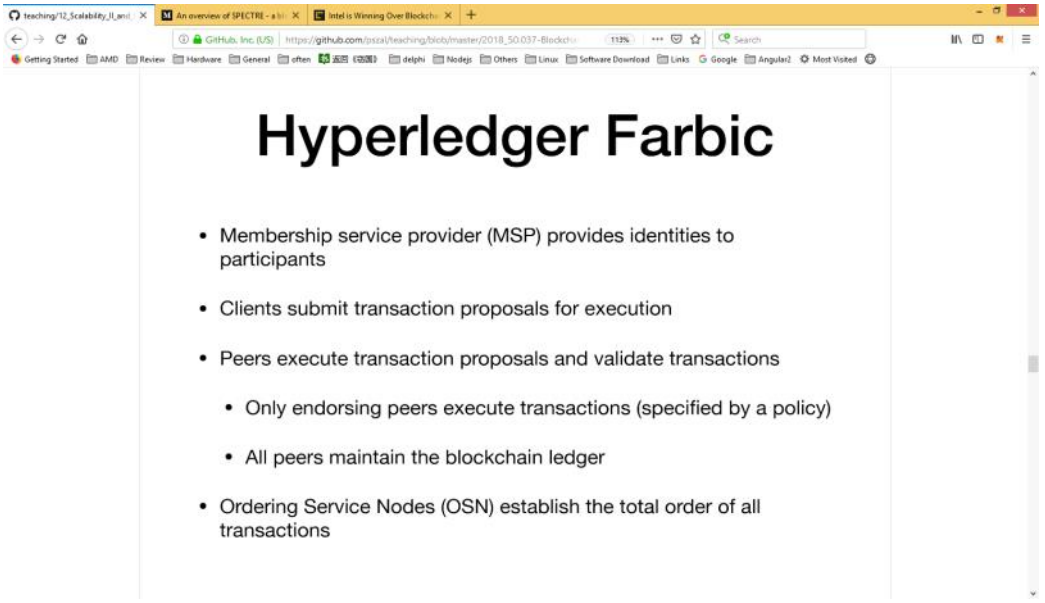
People can join, can vote, can do whatever they want.

Business people started using blockchain, rather than cryptocurrency to become more mainstream.
Can decide who to join the blockchain. There are some entities who can decide about the system.
You cannot implement the cryptocurrency.

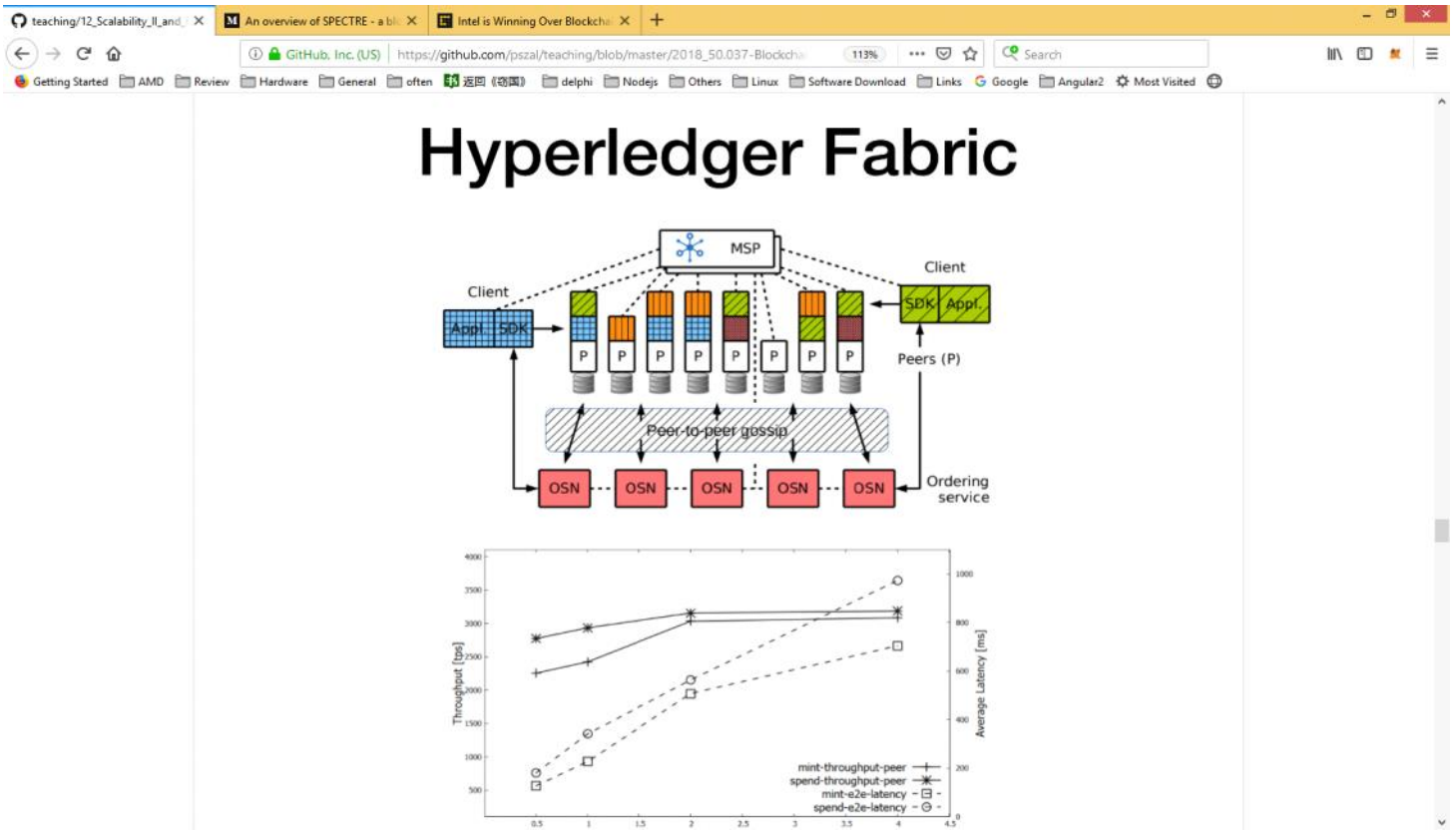
Main project is Hyperledger, who consists of all these companies who jumped into the space.
Consortium of companies investigating how the blockchain can help their business
We can have this available, a bit decentralized system. Let's try to investigate if that can be for our builds.

It's business oriented; so they run different protocols. One protocol was Proof of Elapsed Time.
There is no built-in Cryptocurrency. It has a powerful smart contract engine. Can use GoLang, and throughput is really nice

What's the architecture?



Entity that decides who can run/participate in the network.
Let's say that client runs some application. To peers in the network. So peers take the transaction proposals and validate them. This validate could be an application; closer to Ethereum.



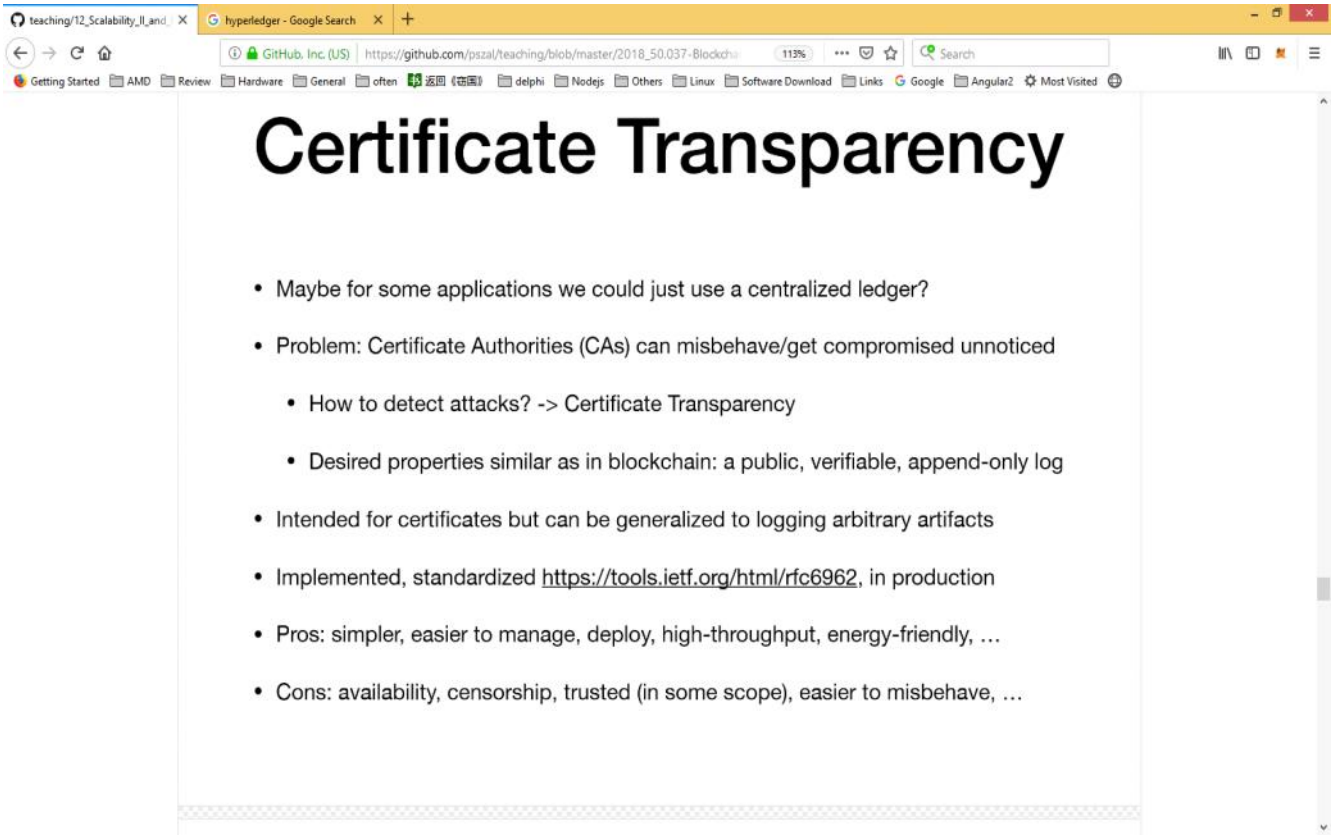
You can have a blockchain developing on different transactions for different applications. It's scalable. If there's more peers, you can add more peers, take on this transaction. So they talk to the peers; they don't run a consensus protocol, but talk to OSN to decide which transactions are first.

These OSNs, they run the consensus protocol. The number of these nodes could be up to 100, and on the otherhand, it's really fast. The transactions per second is insanely fast.

Censorship is a problem: Because they can just reject the client. It's static and so on.

So now we have this open consensus with no boundaries. Everyone who wants to join, joins. You have this permission thing. These clear boundaries are decided by one guy.

What about centralized ledger. We have a system of this class.



Say you visit this website, you see that the information of the website is signed by a CA. We have like hundreds of the CA. Compromise one, can issue certificate for any website.

It's hard to prevent, but maybe we can detect.

Updates to the phone: I want to create an application: to hack to any apple device. To any device in the world. The process is pretty public. US FBI investigates the tourist. If they have just one short string of data, they could do really bad things they are not authorized to.

Idea was to introduce certificate transparency, make all certificates visible. If some certificate is issued by the CA, then this certificate is visible to everyone.

At one point, if it's visible for everyone, The attacker will not be able to launch attacks because it would be visible.

Like Iran attack: Microsoft update that was prepared by Taiwanese hacked CA. If we make all certificates visible, then everyone would be able to detect this attack. If they know that this is not visible. (Worm Nuclear Plant in Teheran.)

You need most of the good properties: public, verifiable, append-only log.

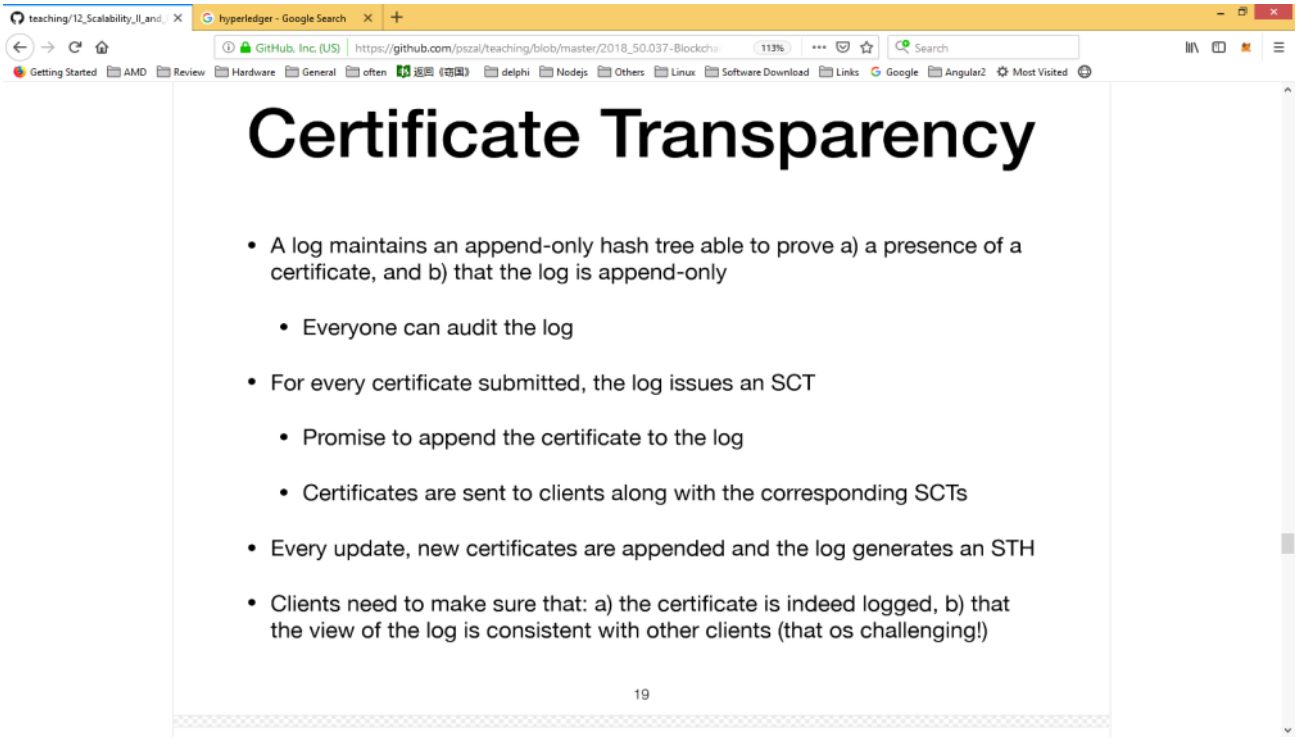
It is intended for certificates, but can be logging arbitrary artifacts.

What would be good about running this centralized log

- Much simpler: Building the tree, and then running the server to add entries
- Easier to manage and deploy; no argument with miners
- High-throughput
- Energy friendly

Bad:

- Problems with availability
- Censorship
- Sending some transaction to the block? Could be rejected
- Has to be trusted in some scope (they have easier way to misbehave)



A log server has a list of the certificates. If a new cert is added to the log, the log returns a promise, a SCT. The certificate was sent to me, at this timestamp, and I signed it.
Every log update, the certificates added: Like submitted, would be appended to this list. This log would be building hash tree over those. It would append only hash tree. So this is what the log is maintained.

What are the properties of the hash tree? If we have some root, which would be here in the hash tree, we can get proof that some entries are in there. Every certificate has to be sent with this promise from the user. The certificate will be sent with this SCD. The log has seen that, the log has appended this, and the user would accept this certificate.

This log would give you two kind of proofs: every update period, it would update its stake by updating new cert, and generate new rules

The rules are called in the SCT, the root of the tree stores the sign

The client accepts the connection, because log registers that this certificate will be added soon, so accept the connection.

What are the things that the user can check? It wants to check the Cert is added. The user will go to the log and ask that this proof that Cert A is added to log. So log will be returned

What's the problem? Privacy. They know which website I connected to, but let's leave that.

The user would know he has the certificate.
What's another problem? If you think that the log may not be honest. Because we cannot assume that we are having trusted entities. We need to make sure that the log is honest.

So first, it'll just reject the cert.
The statement that the log promises to append it.
What's another thing? Trains its field for other clients. It's to generate different views. It's like centralized entity; it's very easy to generate its quiz. The log would show that this cert is here, but the log would create the view of its tree without Cert 8. So if adversary colluding with the log and the CA, it would attack the user.

It would say that the SCT is here, and then the user can be attacked.

If it misbehaves, it wants to create two versions of its content, it's very easy.

So it's creating a fork, and it's cheap to do a fork, unlike bitcoin.

It's assumed that tree is append-only. What you can do with append only trees is to prove that one tree is an extension of another. So let's say that you have this root. Which is called R1. And you have two new leaves. So there would be new root trees. If the tree is constructed in a specific way, you can prove that this version of the tree was before this version of a tree.

I can verify that R2 is an extension of R1, because the tree is extending.

This Certificate Transparency log, can provide a certificate. Can prove that my root is consistent with this root. So after one year, prove me that my latest root is consistent with root number 1001. So

this is how we need to keep this log like append-only. You want to make sure that it's append only.

This is not enough; because if the user is attacked, the malicious user could maintain this log for this particular user. I can maintain that the CA is growing, it's fine, don't worry. But for other users, it could show something different. So the users which use SCT, which use the browser. In some way, it should exchange info/ this view of the log. And then, it would make sure that we see all the same.

In some sense, the ledgers are really nice. For many reasons, they are worse than permissionless blockchains.

In bitcoin, you never know that you are on the right chain. It's easier to misbehave.

Malicious log could be no response, could introduce censorship.

These CA's need to have this promise, because they would go out of business. They need to get this promise from the log. If the log operator is biased, can start rejecting certificates from this company. How to prove that you are rejecteding from here.

Censorship; provide the same properties of decentralized to centralized blockchain, it's not there.

inform nice derive situate excite wheat afraid learn warrior inhale switch obey

Lesson 13

Tuesday, December 4, 2018 8:33 AM

What do you not like about bitcoin? We don't want to wait a few seconds longer. The energy/resource consumption.
This running consensus is somehow tied to burning energy. If your document of supporters of proof of work. It's in the network, or it wouldn't work and so on and so forth. Half of its bullshit.

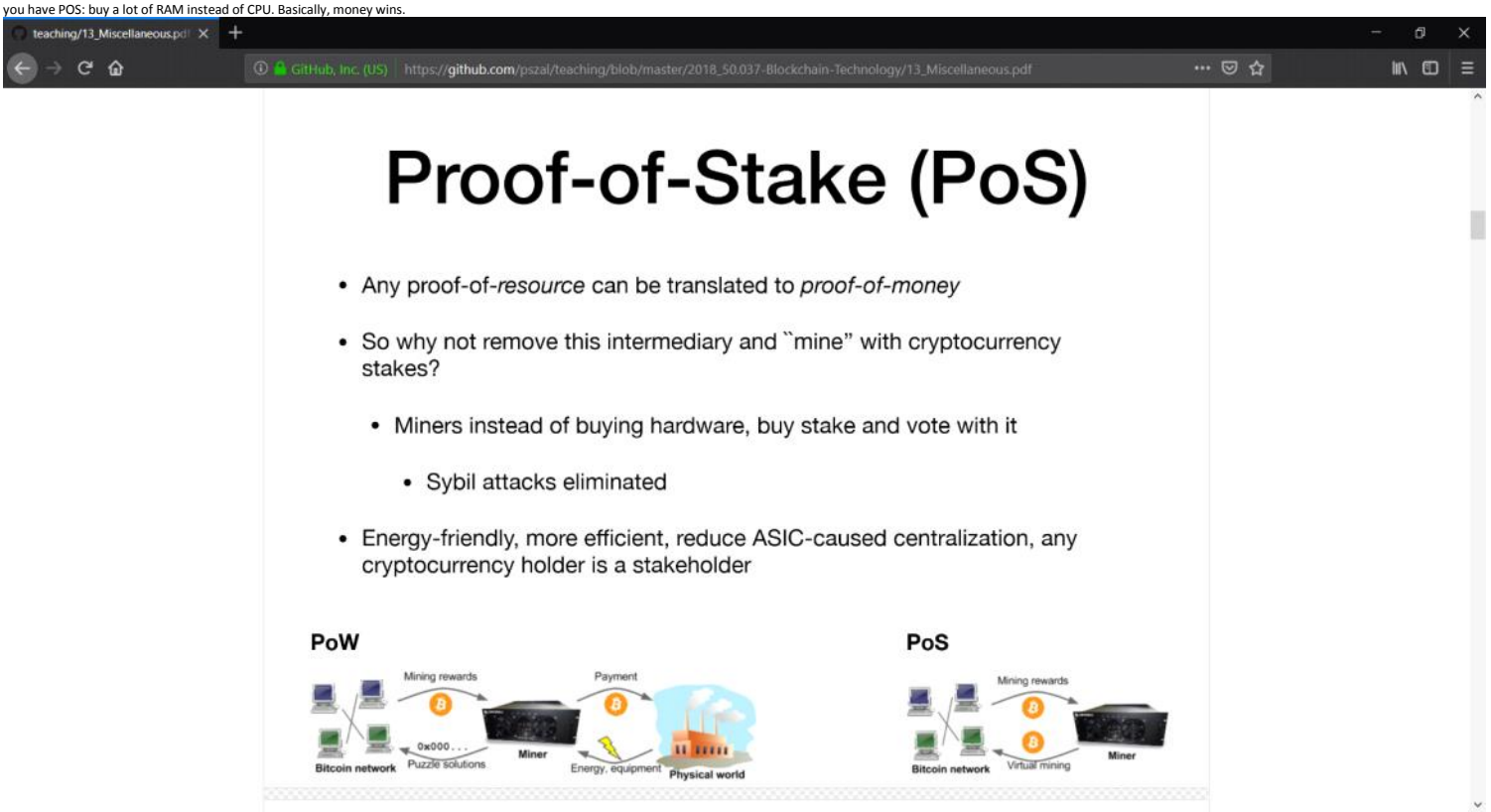
It's just silly, we are wasting energy, trying consensus. We don't see a different way of doing it efficiently. We don't know how to provide the same without having energy. This is just silly, not the way it should be.

Ones like transaction throughput. The transaction should be low. But in a smart contract, is there something that you don't like? How basically Smart contracts can talk to external infrastructures. Sometimes, your smart contract needs some info from outside world.

We'll start with the Proof of Stake (Virtual Mining).
Motivation is what we already like mentioned. This energy consumption: it's pretty clear. In orange, you have countries that consume less than bitcoin. Now that mining power went down, pretty safe how much energy it takes. To prevent Sybil attacks; we have proof of work.

We have this proof of lapsed time (INTEL SGX): The thing is sleeping, have some time. If you have this one, what would be the consequence of that. If we replaced proof of work with like a hash function, what would be the consequence of that. Still, what would miners do? They would buy a lot of machines with a lot of CPUs. They will optimize Intel SGX by buying a lot of CPU because it gives you a vote in the system.

If we have this proof of stake, you need to compute something, prove that you have a lot of RAM. If you have POS: buy a lot of RAM instead of CPU. Basically, money wins.



Any system can be translated to fighting with money. Buying more equipment, buying more CPUs, more clouds, resources whatever. That's all you can translate very easily to your money. Obviously, the energy consumption in different places is different. Most of the mining happens in China.

In the end, you translate proof of whatever you had to proof of money.

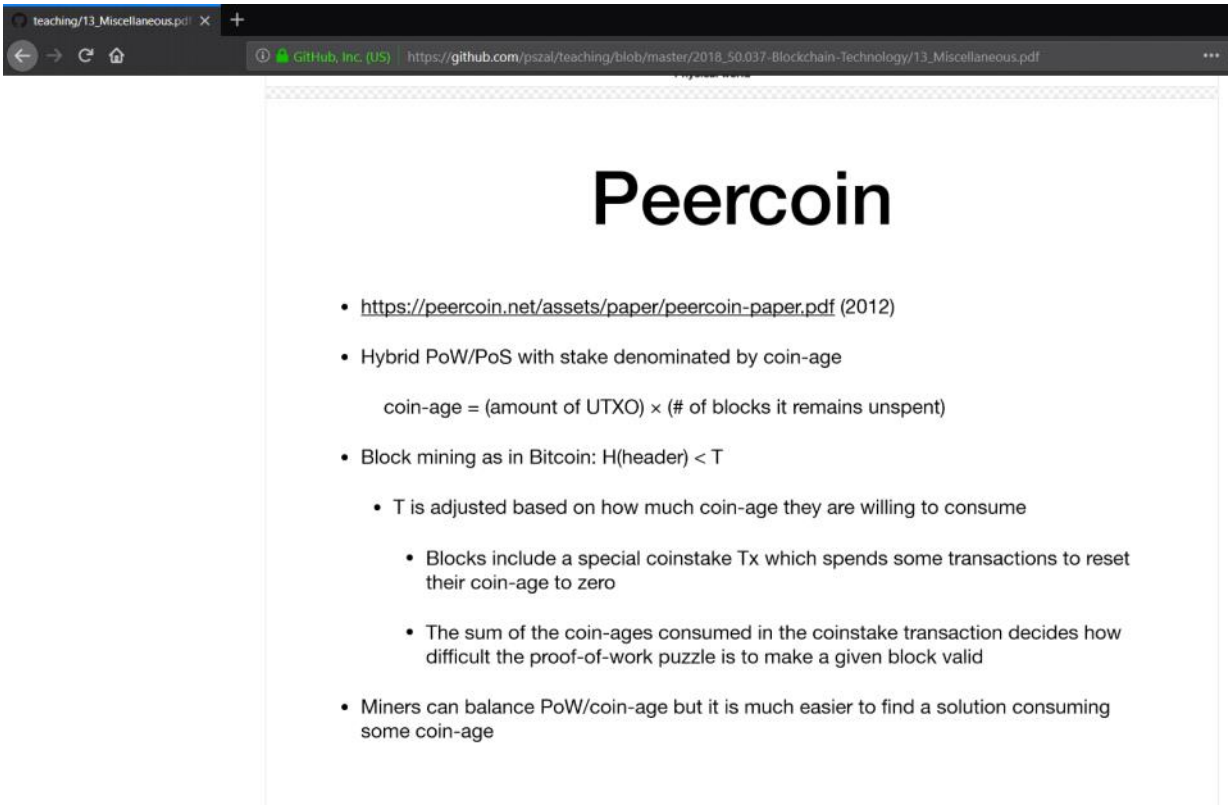
Here we have the miners. What the miner is doing, is citing this hash function to find a solution. Here you have a blockchain network, giving this code, and whatever he gets from the network is bitcoin. The miner needs to operate because resource consumption. He miner is translating this bitcoin to some dollars and he is buying energy. Paying for facilities, paying for the stuff. So that's pretty strange.

You get some money, translate this money, and you do this step to actually operate. This is where you consume energy. This is why it is very efficient. Why don't you remove this reliance on the Physical World. Here, you don't have any supplier with any energy, any Intel SGX. The idea is that you vote in your cryptocurrency state. Instead of buying GPU, you buy 2K of Bitcoin. You vote with that.

This happens, where you have stakeholders and vote on some platform. The activity is gone, money comes in. That's how you vote. This is basically the idea behind this proof of stake. You don't have this intermediary things, you buy stake and vote with this stake. If we can figure out a protocol to what Bitcoin gives us, that would be perfect. There would be no energy waste, you have this scale with voting every 25 seconds. Actually, we can have many nodes.

Benefits? Energy friendly. Proof of Resource, show that we have stake in the system. It would be more efficient. It would basically have this voting like every 30 seconds here. Because it costs nothing. We can adjust to something smarter. Another one that could get rid of ASIC, and centralization caused by ASIC, this very fast dedicated hardware that only mines bitcoins, because you can't have own a large number of ASIC, you have little mining power.

Another side effect: if you are a coder, you are a stakeholder. So you can vote. Another argument is that most likely that you care about the currency if you have a lot of stake in it. Let's assume you have the stake in the system? Divided into miners. So you don't have this problem of allocation. But somehow, you vote with the state. That's the tricky part. You start having this problems, with Standard Byzantine Fault Tolerance. It doesn't scale for any number of users. The first thing in the class which was 2012, and it was peercoin.



What this works:

Let's assume you have a bitcoin-like blockchain. You have headers and transactions, and you have unspent output model. They define something like: Coin H.

Here, we have like transactions, and here in one transaction, we have output. Transaction, one peercoin. Here we have another transaction, and this transaction output is 2 peercoins.

So now the system introduces something like coin H. Coin H for every coin, we can actually have an amount, how much you have in this coin, multiplies by the number of blocks it remains unspent.

So what would be the coin H of tx1? Well, five blocks in front of it, and 1 coin, and becomes 5*1 = 5 . Coin Age for tx2? 10.

Coin H is like meta data. We don't dervie it, what's the block number, and see what's going on. Obviously, if you don't spend it, your coin just grows old.

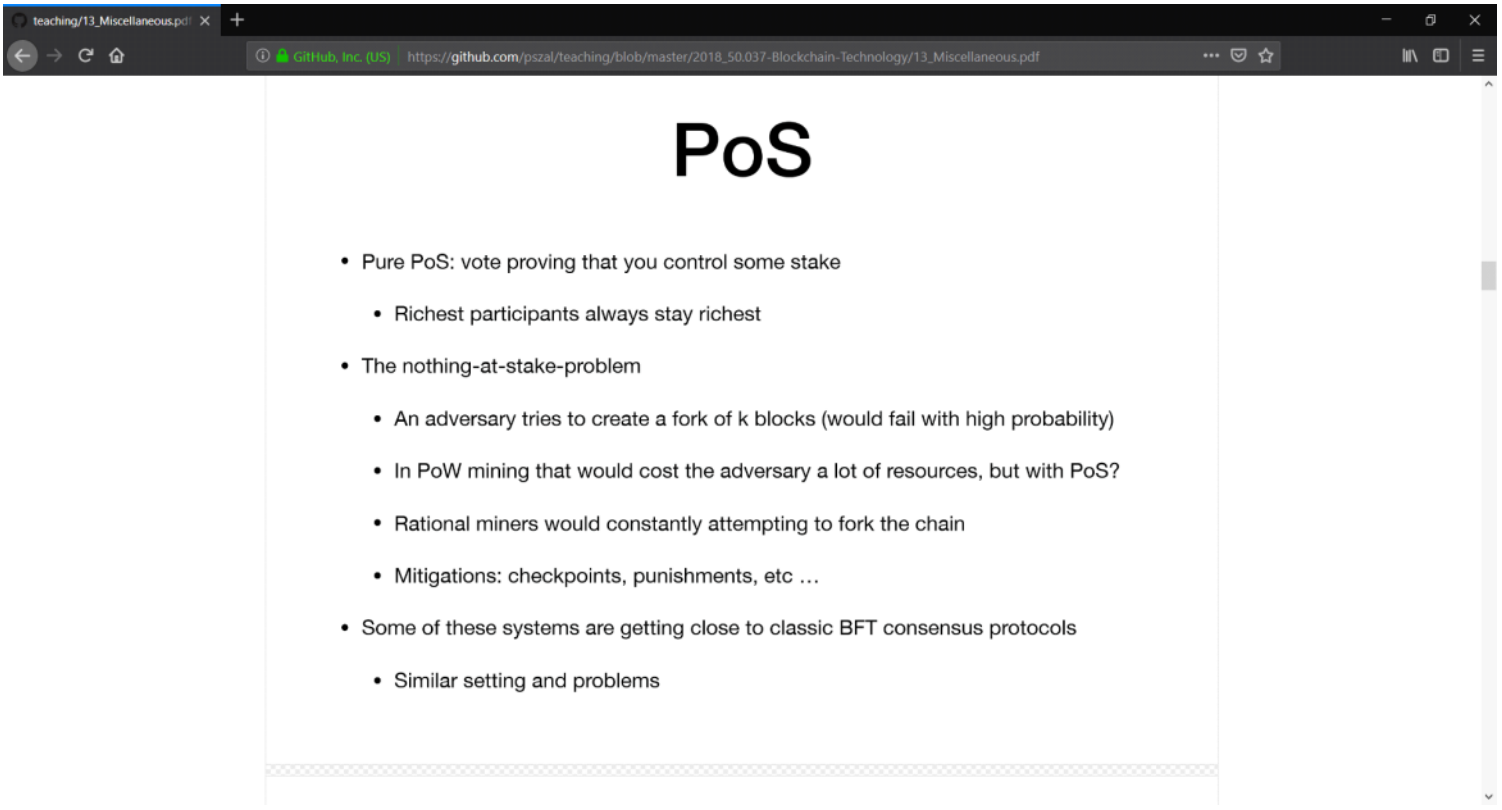
Miners try to find the block/header and transaction. There is no change so far, but the catch is that this target, this T is adjusted based on how much coin age they are willing to consume. Basically, these blocks have some coin spent transaction. Coin state transaction. These transactions would be spending one or two of these coins. The miner trying to find the new block, will spend these two transactions.

Miner will link his old transaction with new transaction. The sum of this coin age of transactions you are willing to spend decides how easy for you to find a solution. Basically, you can visualize this: So combine the coin ages, like 15 in this example. And this miner is trying to find this new block, it's basically spending this 15 coin age transaction. The higher the T, the lower the difficulty. It would be easier to find this header.

You are proving that you have owned old coins, that you have stakes in the system. You can still mine without wanting to stake transactions. It would be very difficult. Someone with very old coins. By spending, you would reset those transactions, those coin ages.

This system is much better in terms of consumption, go much easier. So you can vote with FPGA and stuff. If you imagine the miner trying to come in with a lot of ASICs, he can't compete with your FPGA. So there are like bars with your age. It's still proof of work.

How to remove this hash(header) less than target? We don't know still how to do it some nice way. To give like similar properties.



What we would like in a pure Proof of Stake system is to vote with your stake. Your stake is like 10 coins, this is like the next ether and so on. If you think about the system. You have five percent of this currency. According to your contribution, then there would be no matter use. There's no newcomer who can compete with you, and there's a lot of stake. All control of the system is regulated by this stake.

Another problem is the nothing at stake problem. Vote with stake. The point is that:

Actually, rational miner will always try to create a fork. Why? If you need to vote on one branch, how would you vote. You have five percent, you need to pick one with proof of work. But if you have proof of stake, you can vote for both and not lose anything.

Moreover, if you have a chain, you would like to fork it. With proof of work, you need to invest resources into creating this fork. You can vote and create a fork with every block. It would cost you nothing. This is my vote for this block, just go ahead. In this situation, you have incentive to vote.

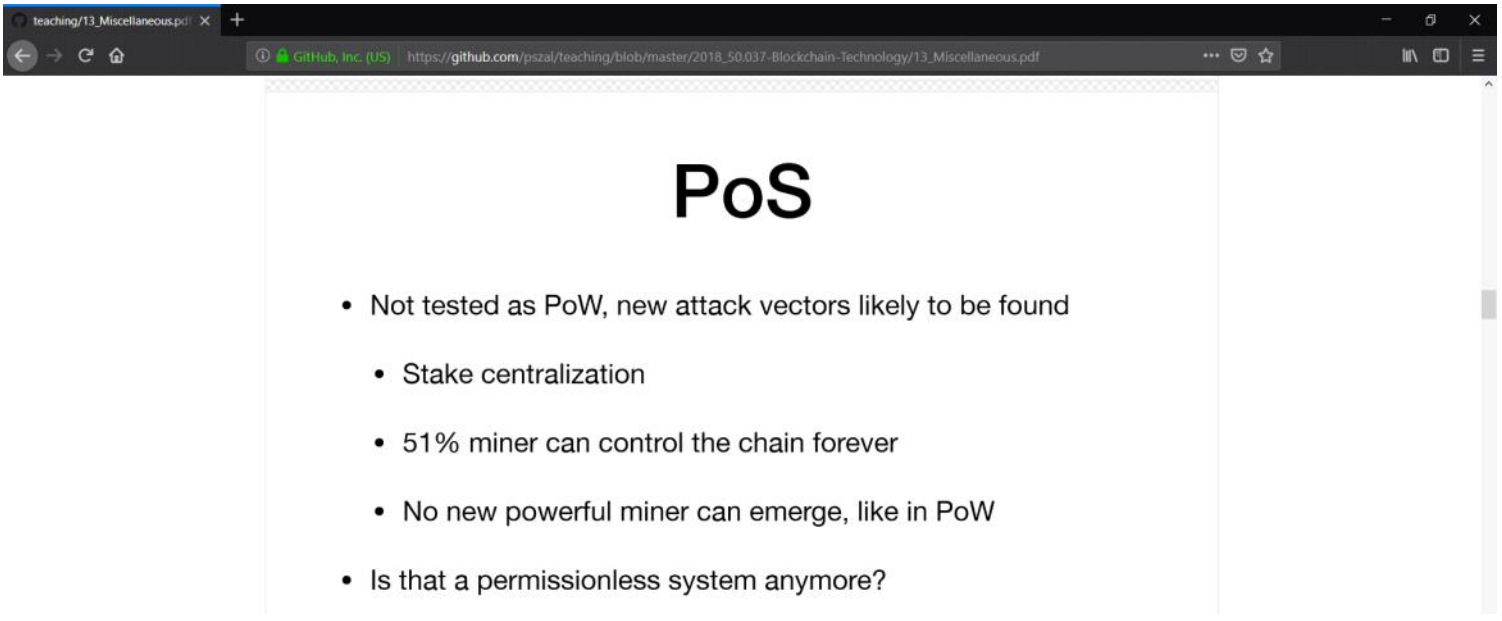
Few mitigations:

If you vote for both, the protocol can encode that if we can catch it, you would be punished. You first deposit some stake and you can vote. If we can catch you here, the protocol would broadcast that you voted in different branches , and the protocol would give your money to somewhere else.

There would be checkpoints. The biggest threat in this is selfish mining. You are never sure you are in the correct branch. The malicious adversary can create this fork very easily. Can create the voting of this stake, and over time create a stake. After like 20 blocks, we do a check point. We will not change it. IF there is a longer fork than 20 blocks, never sense for some adversary.

Problem: After 20 blocks, the rule that you will never revert the 20 blocks back. The two branches will never converge! The two parties who have stayed at these 20 blocks, they will never change their own branch. This is going from the longest chain rule, which is the base for bitcoin, and that's protocol deviation. They are having the same problems that Byzantine Fault Tolerance consensus protocols.

You have again voting, In the BFT> we didn't have stake, but identities. One identity, one vote. Another identity will have more stake than one identity.



- Is that a permissionless system anymore?
  - Someone needs to give us stake
- Active research: Ethereum's Casper, Algorand, …

We are convinced this system is tested, we roughly know that attacks, we can create some trust, some sense of consensus. Every few moments, there would be new attack vectors. These systems are very different from bitcoin like systems.
It can be like Green, but a huge decentralization issue.

If you have new system, where they basically vote with this stake, then 51% miner can control the chain forever. He can keep mining his chain, and he would be luckier in generating new blocks. He can control the chain 100%.

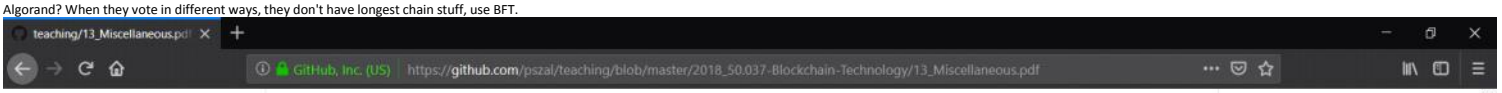
Another issue: if we have such a large miner with some stake. In bitcoin, it's like the powerful thing that you can just enter the system any time. It's not the case in proof of stake. In proof of stake, someone has to be in stake. Someone needs to give you share. It's not as closed as the permission systems.

Someone has to give you systems. Here, the entry barriers is pretty low. But someone can just join.

If someone gives you stake in this BFT, someone gives you identity. How do you proof of stake, he needs to give you from this voting power.

Casper Project: want to have the longest chain, with punishments and checkpoints. This is one of the main research project, objective.

Algorand? When they vote in different ways, they don't have longest chain stuff, use BFT.



Atomic Swaps

- In Bitcoin, it is easy to move tokens controlled by different entities
  - Txs are confirmed on the blockchain
- Can we swap one type of coin for another?
  - Alice wants to sell a quantity **a** of altcoin to Bob in exchange for a quantity **b** of his bitcoin
  - Tx should be atomic and without a trusted intermediary
    - Easy to solve w/ a trusted intermediary (exchange), but w/o?

Alice has a bitcoin, she wants to move this transaction 1 to transaction 2. And this is like moving the bitcoin from one output to another input. We know that transaction happen, we have on chain confirmation. How can we move like bitcoins from one cryptocurrency to another? How can we swap them? We have lite coins and bitcoins. Trade my one bitcoin for 10 litecoins.

What happens, usually, is that we change using fiat currency. And we need a trusted third party. It's malicious, and sometimes that a lot of this fiat to currency, and they disappear. What can we do without this guy? How can I trade with you my bitcoins to lite coins without trusting some exchange? Without someone in that intermediary.

This is called like swaps. Swapping one cryptocurrency to another.

The atomic step: either ends, or causes no effect. It either executes, or reverts to the previous step. So if you write something on your harddrive, you put your entry, or return error. You have no intermediate. You have nothing in between. If you write like 10 KB, or 0. You can arrive like 100 B. Atomic means that if you executes, or does not happen.

IT means that Bob receives altcoins and must send some bitcoins. There's no just receiving or just sending; if it is interrupted, all changes would revert. The protocol is pretty tricky.

1. Alice generates a refundable deposit of a altcoins as follows:
   - 1.1 Alice generates a random string x and computes the hash $h=H(x)$
   - 1.2 Alice generates **DepositA** as shown below, but doesn't publish it yet
   - 1.3 Alice generates **RefundA**, and gets Bob's signature on it
   - 1.4 Once Bob signs **RefundA**, she publishes DepositA (but doesn't publish **RefundA**)
2. Bob generates a refundable deposit of b bitcoins as follows:
   - 2.1 Bob generates **DepositB** as shown below, but doesn't publish it yet
   - 2.2 Bob generates **RefundB**, and gets Alice's signature on it
   - 2.2 Once Alice signs **RefundB**, he publishes **DepositB** (but doesn't publish **RefundB**)
3. Case 1: Alice goes through with the swap
   - 3.1 Alice claims the bitcoins by time T_1, revealing x to Bob (and everyone) in the process
   - 3.2 Bob claims the altcoins by time T_2

   Case 2: Alice changes her mind, does not claim the altcoins, does not reveal x to Bob
   - 3.1 Bob claims his altcoin refund at time T_1
   - 3.2 Alice claims her Bitcoin refund at time T_2

DepositA [Altcoin block chain]

Input:	Alice's coins of value a
ScriptPubkey:	Redeemable by providing either (*sigA* and *sigB*) or *sigB* and x s.t. $H(x) = <h>$

→

RefundA [Altcoin block chain]

Input:	DepositA
Output:	AddrA
Timelock:	T_2
ScriptSig:	*sigA, sigB*

DepositB [Bitcoin block chain]

Input:	Bob's coins of value b
ScriptPubkey:	Redeemable by providing either (*sigA* and *sigB*) or *sigA* and x s.t. $H(x) = <h>$

→

RefundB [Bitcoin block chain]

Input:	DepositB
Output:	AddrB
Timelock:	T_1
ScriptSig:	*sigA, sigB*

If they both agree how would they transfer it?
If the two transactions are there, it would be either by Alice and Bob. So they have this refund, if someone doesn't want to execute in the end. If Bob stops, he gets transaction where he gets the money back. If Alice changes her mind, she can publish this transaction, and say that this transaction goes back to her.

The nice thing that would happen: Alice would claim this coins. She would sign this transaction and reveal x, her hash. But by revealing this x, she's also revealing the x that is used in his hash.

So if Alice is moving this coin, she would reveal this x, and Bob would learn this x, sign this transaction, and reveal his own x. They would be able to move these transactions. These refunds are time locked. If Alice wants to spend deposit B, sending and revealing X and be fast in publishing this to get her deposit back, Bob has this one week to claim it.

teaching/13_Miscellaneous.pdf
GitHub, Inc. (US) https://github.com/pszal/teaching/blob/master/2018_50.037-Blockchain-Technology/13_Miscellaneous.pdf

Payments in Bitcoin

- Issues
  - Scalability and Tx latency
  - Micropayments
    - High fees if done "on-chain" w/o trusted parties
- Idea: 2nd layer scalability
  - So far we discussed "on-chain" scalability (1st layer)
  - Consider micropayments: do we need to settle all Txs on chain?
  - Payment channels: "off-chain" Txs with the final balance recorded on-chain
    - Can be generalized to any state (state channels), e.g., smart contracts

They are like slow, and with micropayments, it's pretty annoying. When you pay with some fraction of bitcoin, you need to pay for a transaction fee. This transaction has fee and so on. How now are micropayments implemented? Give me examples of micropayments you are paying.

Prepaid phones? Per second, they will pay. Singtel will just charge your deposit by whatever price it is.

After one month, when this party or this party would cash out, they would publish the last transaction. This transaction would be published, and moved from deposit of A to deposit of B.

Payment Channel (unidirectional)

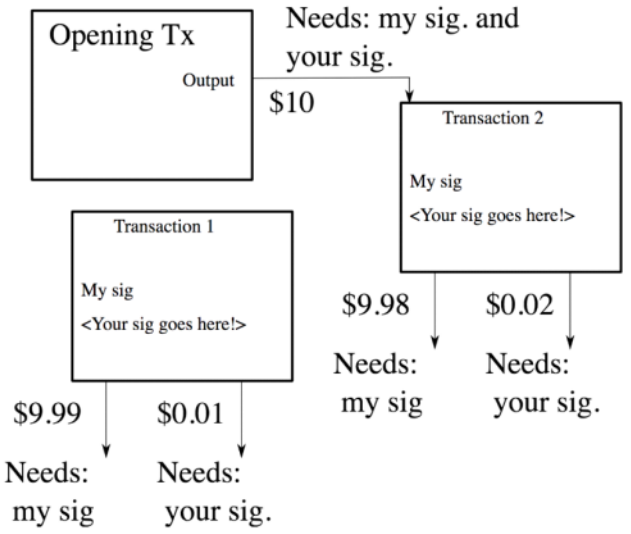


Fig src: https://rusty.ozlabs.org/

They would send this transactions, tx1 and tx2 in a private channel. Only opening TX is published at first to establish the channel. So afterwards, they close, and publish the last of their transactions.

Alice can close sending the first transaction. But Alice can try to misbehave by sending the first one. So she can benefit by having more currency in the first transaction than the last transaction when she lost her money.

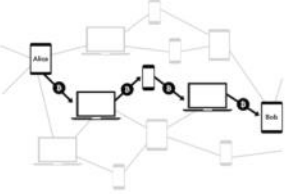
There is some way to prevent this by preimage. That you can revoke any previous transaction. With the last transaction you can revoke any previous one.
It gives time to Bob that you can close the current balance. So I'm just balancing the transaction that would revoke the previous one. Alice cannot revoke this one. So if you think about this channel, it's pretty easy to implement this in Ethereum. Message is signed by two sides. There is some sequence number.

In Bitcoin, it's more tricky, but you can revoke any previous transaction. If you think about that more, what's another disadvantage? This is like channel from A to B. Actually, it's in one direction; only Alice can pay to Bob. You can extend in a bidirectional channel. You need to encode more options if someone is misbehaving.

teaching/13_Miscellaneous.pdf
GitHub, Inc. (US) https://github.com/pszal/teaching/blob/master/2018_50.037-Blockchain-Technology/13_Miscellaneous.pdf

Payment Networks

- Payment Networks (e.g., Lightning https://lightning.network/)
  - Do we need to establish many p2p channels? Multiple “hops”
  - Low fees, fast, high throughput, buyers are protected, sellers need to watch the blockchain
- Many off-chain solutions
  - Not only payments, smart contracts too
    - Plasma, Arbitrum, …



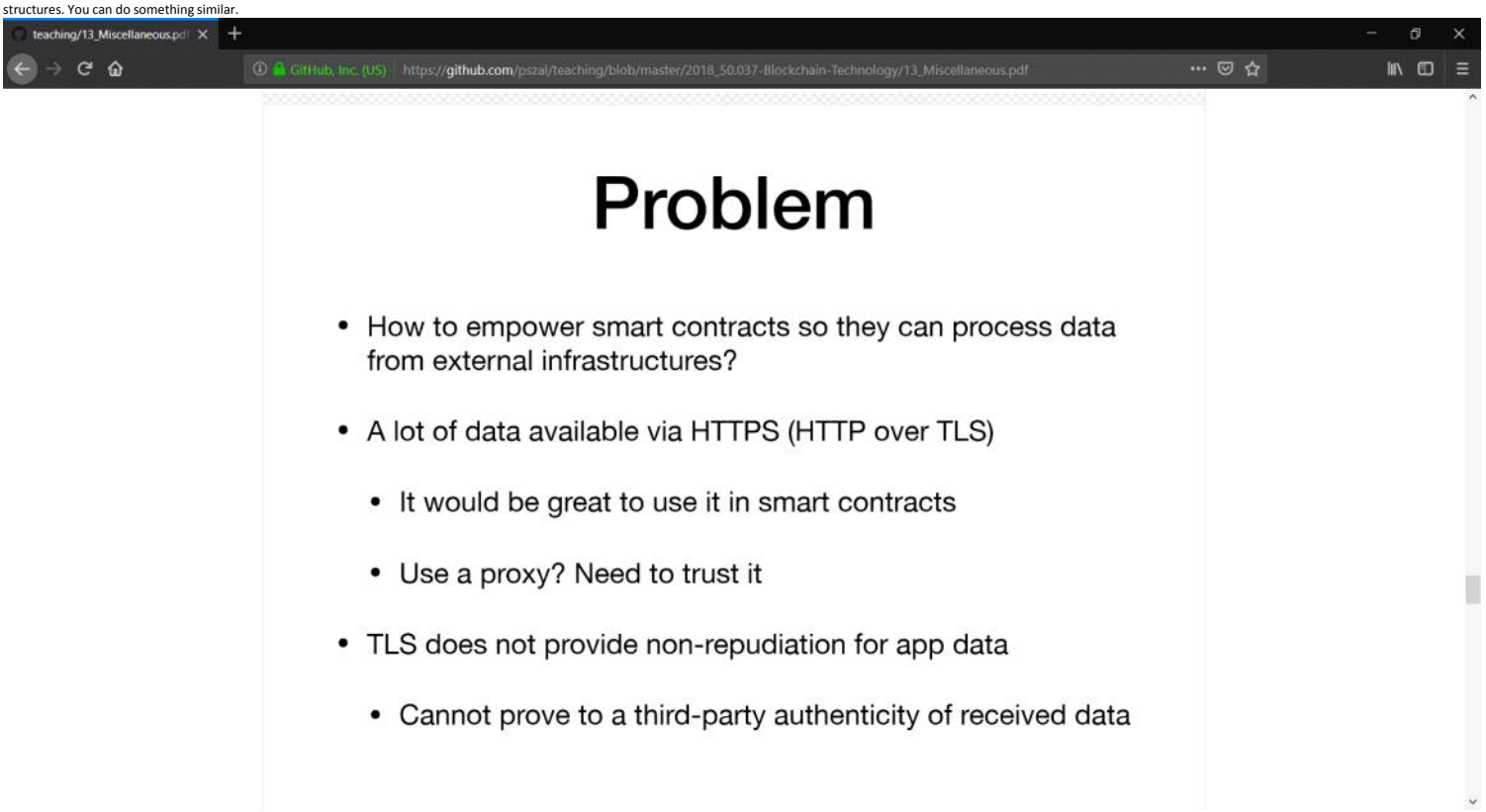
We can transfer from peer to peer, throughput is almost infinite, the annoying point is that Alice has to open many channels. She has to create this funding transaction.
They notice that if more people use it? Like A B C D E F. We actually could create a network of people. Do we need to establish peer to peer channels. Connect people in some infrastructure. Everyone can send money to send money to everyone without creating channels. We have this nice scalar effect. This is like network effect. Then they fought like Lightning.

This kind of signing, revoking transaction. This is a base layer.

What lightning is a network. It's more meshed. If they want to send money, they have a pretty tricky protocol that sending money from A to E will go through D. It will go through a few transactions. Obviously, you have parallelized problem sets. They transact like from peer to peer, and have protocol that the intermediary doesn't stop. The protocol handles by ensuring that B does not handle this transaction.

You have really low fees, very fast/ higher than Visa, and so many packets. The throughput: even it would be faster. There are a few problems. The shortest way for instance. Going from G to A. If you want to have short links. You can pay very fast, you can join networks very easily by creating one deposit to the lightning node and this can be done via the network.

It's very good for buyers. For the sellers: Someone doesn't try to just publish the first. It's not behaving like publishing previous transactions. There are many chains solutions similar in their structures. You can do something similar.



Data Feeds Smart Contract: Grab some authentic information from the outside world. We have insurance, I go to some next vacation. It's not raining. I buy this insurance, it was raining in the week. It was raining in Bali, please send money back. Let's say I trust Yahoo.com, so it would be great of Yahoo gives this information. We have this like secular connection.

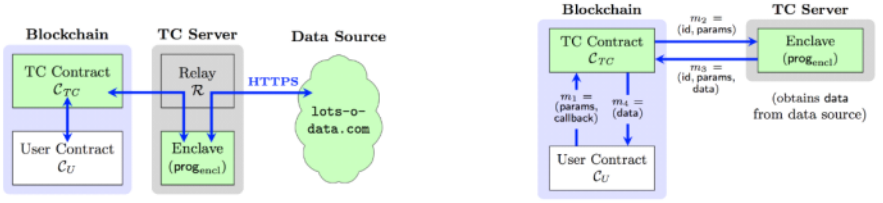
Where we know that this information is used by Facebook.com. Get data from externally.

However, there are many problems. Basically, if you are establishing connection, talking to Facebook.com, tell me that's it's being produced by Facebook.com, I cannot trust you. It is more authentic for Facebook users than me. For a third party, you cannot prove that this is produced by Facebook.com. For TLS, you cannot prove to your smart contract that it was produced by Facebook.com

We have this smart contract here. You are pulling from a proxy, that is pulling from HTTPS. This proxy misbehaves, can push whatever it wants.

TownCrier

- https://eprint.iacr.org/2016/168.pdf
- A proxy between a blockchain and HTTPS websites
  - but implemented with Intel's SGX
- Pros: easy integration, website operators do not have to deploy
- Cons: needs to trust Intel (single point of failure) and SGX (recent attacks)

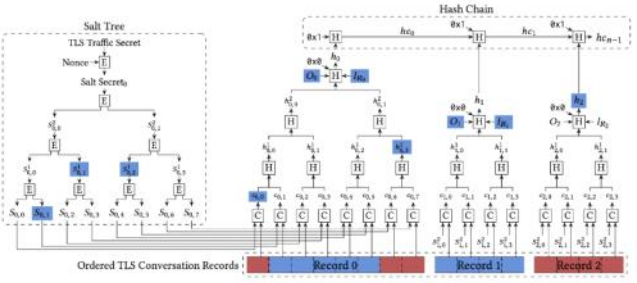


It's like proxy, however this proxy is implemented with Intel's SGX.
User contract goes to some town crier contract. And this end plate gets some info from IntelSGX. It's like moving data to smart contract.

What about other alternatives?

TLS-N

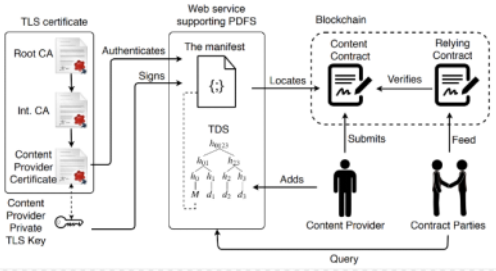
- https://eprint.iacr.org/2017/578.pdf
- TLS with non-repudiation
  - Efficient Merkle-tree-based authentication on TLS layer
- Pros: more general and powerful solution, extra features (like privacy)
- Cons: significant changes to the TLS spec, TLS servers have to be updated, difficult/expensive integration with smart contracts (TLS records are signed)



Not only for smart contracts, you can do other powerful stuff with that.

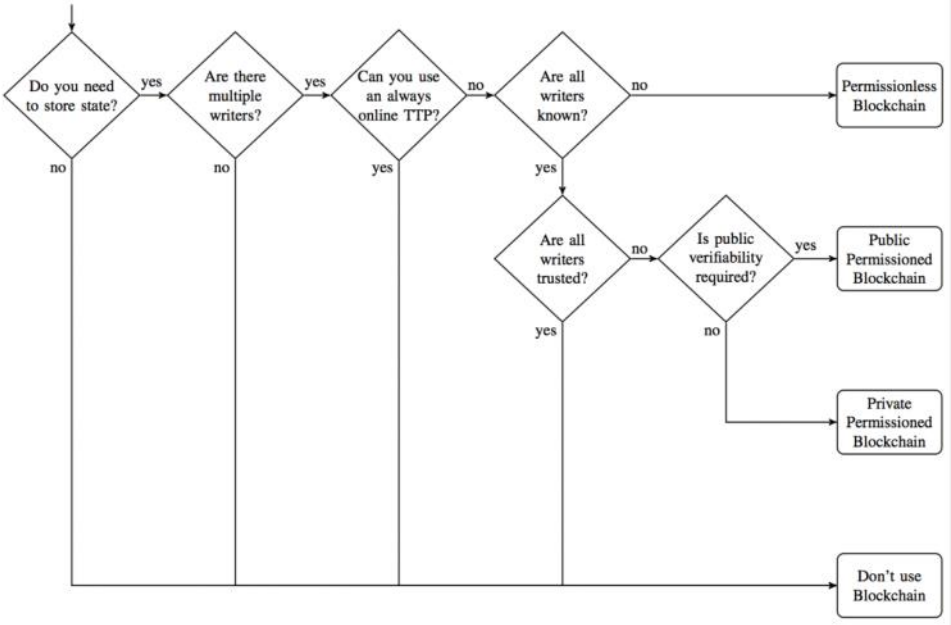
PDFS

- https://arxiv.org/pdf/1808.06641.pdf
- Data providers sign (via TLS) their contract locations with API allowing:
  A. Update the contract with a new root of a database (append-only)
  B. Prove that given data is authentic (is in the database), can be paid
  C. Query for data if provider seems unavailable
- Pros: direct TLS authentication, easy integration and data format, extra features
- Cons: website operators have to deploy



Many servers with secret connections. For the producer of the content, authenticate this contract with a TLS certificate.
We can sign this contract with Yahoo.com. And this contract goes to Yahoo.com because it is signed by Yahoo.com. You aggregate data in the Merkle tree and publish the root of the entry in this contract.

teaching/13_Miscellaneous.pdf
GitHub, Inc. (US) https://github.com/pszal/teaching/blob/master/2018_50.037-Blockchain-Technology/13_Miscellaneous.pdf



Do you need a blockchain?
You don't need to store state: It stores everything. But the answer is blockchain
If you have just one writing the blockchain, you can just use SQL
If you can also assume that you have multiple writers, you can trust online trusted third party, you trust someone to run the database.

You have multiple writers, and you trust them: it's called cloud computing. How they manage it is some problem.
You have multiple writers, you are unknown, and they need to write to make your application happen. That's the only case to get a permissionless blockchain

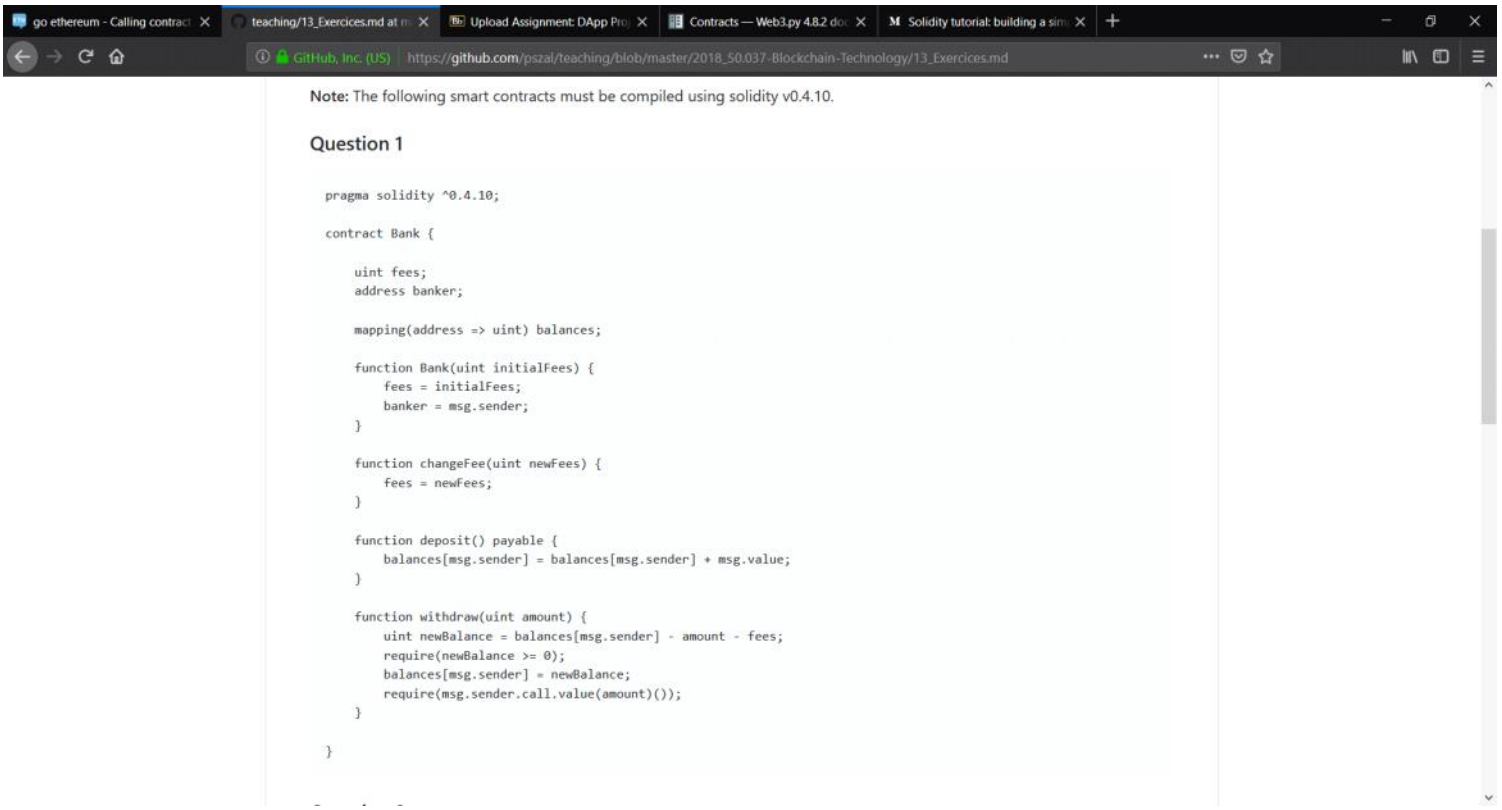
If you know the writers, you can ask if they are trusted. I mean, you don't need blockchain, just replicate your database, and have some cloudcomputing.

If you need to publicly verify something, You want to prove to the people they misbehave, you need ot have a public permissioned blockchain.

You can run a private blockchain since you know the writers, you don't trust them but you don't need a signee.

Lesson 14

Wednesday, December 5, 2018 12:11 PM

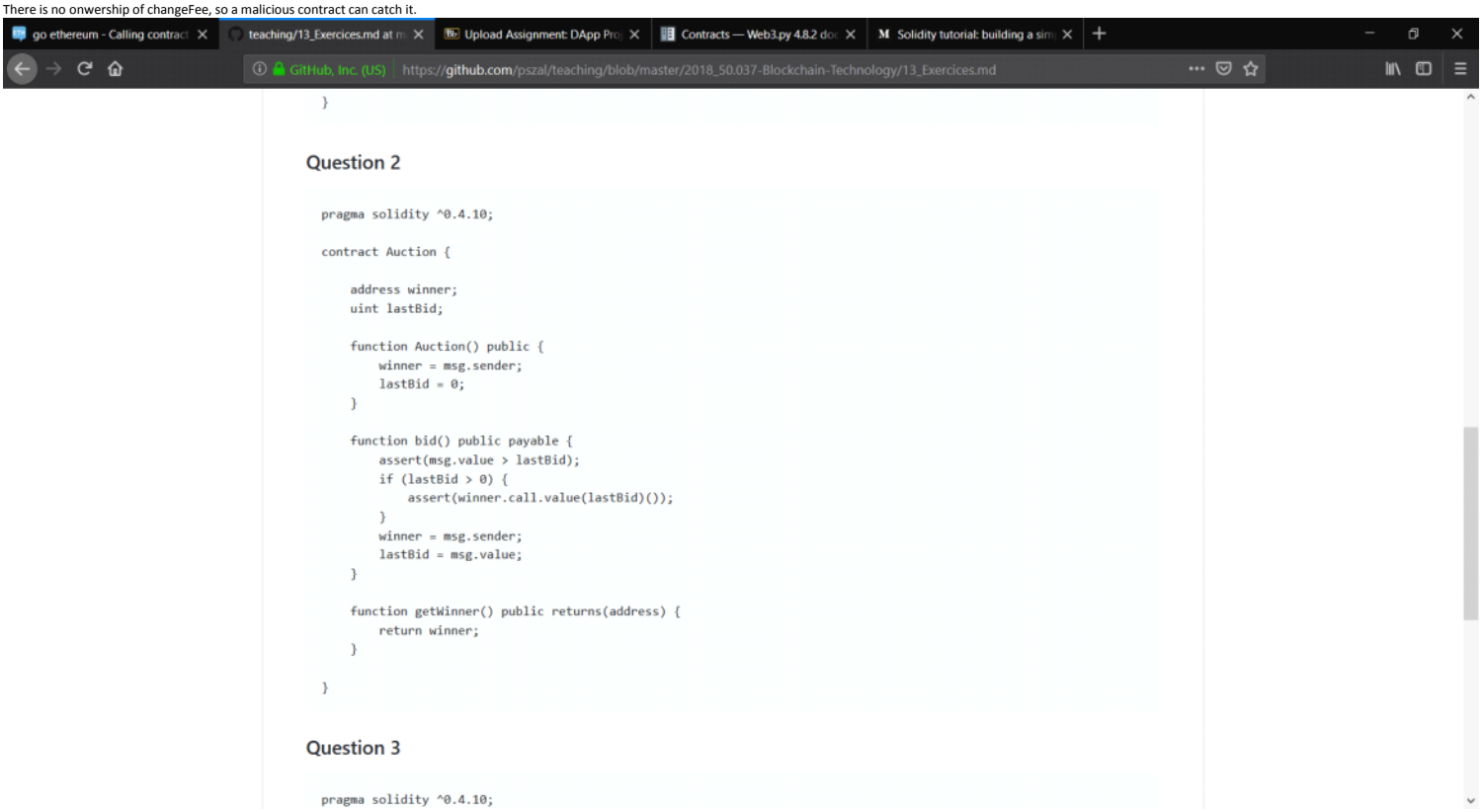


Note: The following smart contracts must be compiled using solidity v0.4.10.

Question 1

```
pragma solidity ^0.4.10;

contract Bank {

    uint fees;
    address banker;

    mapping(address => uint) balances;

    function Bank(uint initialFees) {
        fees = initialFees;
        banker = msg.sender;
    }

    function changeFee(uint newFees) {
        fees = newFees;
    }

    function deposit() payable {
        balances[msg.sender] = balances[msg.sender] + msg.value;
    }

    function withdraw(uint amount) {
        uint newBalance = balances[msg.sender] - amount - fees;
        require(newBalance >= 0);
        balances[msg.sender] = newBalance;
        require(msg.sender.call.value(amount)());
    }

}
```

If you are not the bank, anyone can change the fee. So you can have very high fees or low fees.
There is no onwership of changeFee, so a malicious contract can catch it.



```
}
```

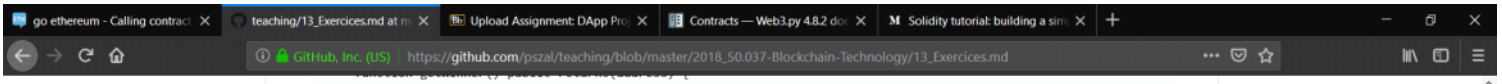
Question 2

```
pragma solidity ^0.4.10;

contract Auction {

    address winner;
    uint lastBid;

    function Auction() public {
        winner = msg.sender;
        lastBid = 0;
    }

    function bid() public payable {
        assert(msg.value > lastBid);
        if (lastBid > 0) {
            assert(winner.call.value(lastBid)());
        }
        winner = msg.sender;
        lastBid = msg.value;
    }

    function getWinner() public returns(address) {
        return winner;
    }

}
```

Question 3

```
pragma solidity ^0.4.10;
```

What's wrong with this?
It can be that the assert function is blocked by another smart contract, such that assert is not allowed to occur. So that any attempt at returning the bid to someone else, I can throw an error and say "Oh, I never received." And the auction ends with the last bid set by the malicious smart contract




```
        return winner;
    }

}
```

Question 3

```
pragma solidity ^0.4.10;

contract Bank {

    mapping(address => uint) balances;

    function deposit() public payable {
        balances[msg.sender] = balances[msg.sender] + msg.value;
    }

    function withdrawAll() public {
        uint amount = balances[msg.sender];
        assert(msg.sender.call.value(amount)());
        balances[msg.sender] = 0;
    }

    function getBalance(address _address) public returns(uint) {
        return(balances[_address]);
    }

}
```

We are giving control of different smart contracts.
Re-entrancy attack: the smart contract can also go back something. Smart contract can recurse
Famous with the DAO attack. Something about 50 billion was exploited.

What to study for slides

Wednesday, December 5, 2018 12:29 PM

Introduction: I don't expect questions like properties of hash functions. I don't expect questions like Birthday Attack.

Proof of Work important: expect how it works

Don't expect questions like properties of digital signatures
But it's good if you understand.

You should understand how hash chain works, how merkle tree works and how they are combined.
Don't expect bloom functions

Second: Distributed Functions
Don't expect CAP theorem or about naming. You need to understand what is Ledger, what's the important in implementing public ledger, open ledgers.

Don't expect system models. There would not be questions about bounds. It's important how the Byzantine problem works, what are the bounds for that.

Broadcast? I don't expect that. PBFT no? But you need to understand the distributed keys: You need to remember the DAOs, and how it is derived.

Distributed properties, open consensus, sybil attacks. You need to understand Nakamoto consensus. Properties, everything here.

Third: Bitcoin:
For Bitcoin… Low-level overview need to understand. No question like what is length of given header, that would not be questioned. How blocks are composed, how they are validated and so forth. What was proof of work in the bitcoin. This you need to understand more Block and header validation. What's not full node.
Transaction model , format you need to understand. How you can send transactions and so on.

You have the script validation? I don't expect this question but you need to understand
how it works.
Coin Minting, Genesis blocks, forks, double spending very important. Consensus Rule changes: You should know it but I don't expect it.

Fourth: Bitcoin II:
SPV is important, Wallets, Yes that's still important. HD? Not really?
P2P Network: not so important. That's probably. You should know but I don't expect. Everything in Bitcoin is important. But if I change my mind…

Incentives: Everything is important, Mining Evolution, selfish mining… Mining pools is important. Mining pools attack: Block withholding: That's important. I think we described> We have example with this one. Block reforming is important. Kay. Misc we didn't discuss in very much detail, so not happening?

Time in Bitcoin: Let's skip it.

You should understand consequences of time: It's good to know,

That's all important. But will not be there.

Fifth: We started with Ethereum, smart contracts?
Smart contracts are very important, how it is, how we run it, here is some example of bitcoin smart contracts. Ethereum: obviously, not asking when it started, history of that and so on. It's important how it works, high-level but also gas is consumed, what is VM and how it is executed. To Gas it's important? There will be a question of g code that was deposit? I give you this table. Obviously ont

Calls you need to understand. I don't expect programming question, you have this code, what is bug? I don't expect this question.

There are some examples, memory, but this is part of your project. I will not ask about the function of this project. That's pretty important. Security is still important, but don't expect to ask you any of these questions.

Sixth: Ethereum II
Important, important, transactions and Message impoortant, consensus important but not details of GHOST because we don't understand some well, but you should have insight how it is important

Ethash? I don't have strong opinion of that

I think it's pretty detailed for a question for describe Ethash, but I don't expect completely.

GAS is imporant, Miners is important. Don't expect block header in ethereum, but you should understand trees, how it's linked together. What functionalities there are. State trees and so on.

I don't expect questions about how to store mappings. But understanding Merkle Patricia Trees will be good. This is complicated, skip it/

Light clients important, incentive too high-level, don't see questions. No No.

You should know issues.

Seventh: Alternative Mining, Bitcoin Applications and Alt coins

Requirements important, Asic , memory, scrypt: good to know scrypt for instance.

Proof of useful work? Important. Primecoin? It's not very popular, let's skip. Permacoin: more interesting.
Bitcoin applications? What question I will give you? There will be some exercise kind of thing.

Other Applications you should know but I probably would not know

Altcoin is pretty important to understand. But not too much out there. Merchmining> that's interesting. (Last slide)

Eighth: Anonymity and Privacy
Important all the way: I don't know clustering address. Everything here is pretty much important. I will not ask bout Tor. Important mixing services, coin join, flows, everything here is important.

Zero coin? That's actually nice. But it is also difficult. Which it means that it is good for exam. Zero coin is important.

Zerocash? Extension: You should what elvel of type, anonymity attacks, deployability.

Ninth: Scalability:
That's probably important: That's all pretty much important. Bitcoin NG is important, sharding is still but important.

Tenth: Scalability 2:
SPECTRE is important. SGX is good to know. Permissioned Blockchains don't expect. Certificate Transparency not expected.

Eleventh: Peercoin is pretty important, PoS is important. Maybe except datafeeds for smart contract, town crier, no that's important.

I need to avoid storing as a list.
So I can have a mapping of bidders to highest bid
So that once I withdraw
I return the withdrawal amount to this guy,
    If highest, I remove the withdrawal