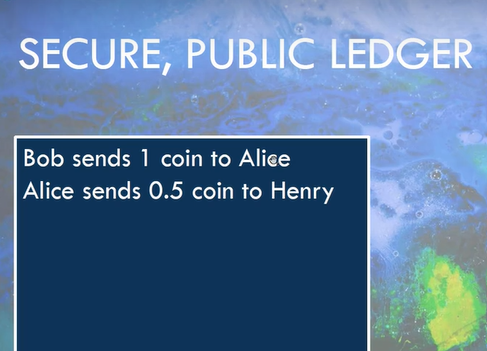
**Vid 29 – Securing a Public Transaction Ledger**

Ledger should be secured in 2 ways

1. Anyone should be able to read this ledger and quickly verify the digital signatures on each of these transactions and in this way be reassured that all of these transactions were added to the ledger with the permission of the coin owner.
   1. When bob sends coins to Alice the only way for those coins to move out of Alice’s account is for her to sign using her secret private key – a transaction sending them somewhere else.
2. Should also be tamper proof – once a transaction is committed to the ledger, it should be very hard for bob to go back and delete this transaction.



In a crypto blockchain transactions are collected into a block then the hash of the previous block is computed and added to the current block, then the entire block, including the hash of the previous block, is published to the network.

Graphical user interface, text, application, chat or text message

Description automatically generated

Each new block will include new transaction and the hash of the block before it making it impossible to change the contents of any block without changing the contents of every block following

Text

Description automatically generated

Most crypto blockchains begin with the first block called the Genesis block. Both the contents of this block and its hash are typically stored in the software itself

In this section well create a genesis block for our coin and will declare it valid by def and create some coins in it out of thin air. Other blocks will not be allowed to do this until we add whats called a block reward in the next section.

**Vid 30 – Review of Previous Work**

Mostly about assembling the pieces that we’ve already created into a tamper-proof transaction ledger.

Key loading and serialization 🡪 how well be able to save and load our private key. Key serialization takes our private or public key and then takes your private and public key and creates some text out of it that can be then rolled back into a public and private key object using the functionality in key loading

<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/>

passwords 🡪 should it ever require a password a public key is known (everybody should know it) but a private key is very different, and storing it on your hard drive is a very bad idea so the private key is usually serialized with a password

loading SSH public keys

<https://cryptography.io/en/latest/hazmat/primitives/asymmetric/serialization/#cryptography.hazmat.primitives.serialization.load_ssh_public_key>

Graphical user interface, text, application, website

Description automatically generated

Graphical user interface, text, application

Description automatically generated3 trues

* the transaction was valid,
* sig valid before the saving and loading
* sig valid after the verifying and loading

These steps can allow us to DUMP our private key, only problem is the Tx right now is storing the public key as a class instance not a serialized object that can be pickled

Going back to the signature class 🡪 instead of returning a public object which is the public key class. Serialize it as we pass it out of generate\_keys and then load it as it is passed into verify

Graphical user interface, text, application

Description automatically generated

**Vid 33 – the TxBlock Class**

10:20

Text

Description automatically generated with medium confidence

**Vid 33 – Assignment 2 Solution**

**Graphical user interface, text, application

Description automatically generated**used the super functionality to call the parent constructor for our particular instance of TXblock

its going to run back and find this constructor in CBlock and its gonna make an empty list (data) and then look at the previousBlock, compute its hash and store that as self.previousHash – 00:41

we don’t have to do it in 2 different places, our TxBlock will just do as CBlock is instructed to do with an empty list

Text

Description automatically generated

Graphical user interface, application

Description automatically generatedNeed to figure out why load\_B1 is not valid even before the tampering

**Vid 34 – Assignment 2 Solution**

When we call this str on class instances we always get these funny strs that just tell us the name of the class and its location. Instead, wed like this str to be all the data inside the particular transaction.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Right now we have a centralized tamper proof blockchain