Introduction to Blockchain Technology and Applications

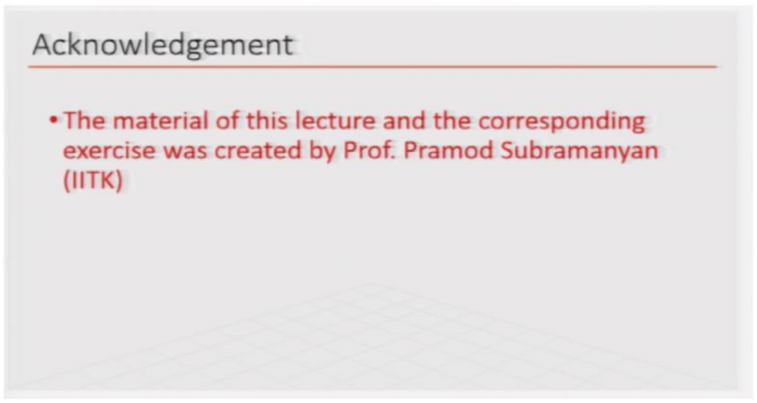
Prof. Sandeep Shukla
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture No. 03
Blockchain Technology and Applications

Welcome to the lecture number 3 for blockchain technology and applications. So, what we will do today is announced a homework, it will not be graded but it is something that you could do yourself so that you can actually familiarize yourself with a number of concepts that are very important in blockchain. So, you will be designing your own blockchain platform implementation. This is going to be a little bit you know, simple blockchain application.

So, it is not really how the bitcoin or ethereum or other blockchains work, but it will have to come concepts that will allow you to understand the other blockchains better. Now, if you did not know C++, then this homework may not be something you could do, unless you actually learned how to program in C++ at least the minimal concepts and the data structures. But even if you even if you did not know C++, the concepts still are pretty general and you should be able to understand those.

(Refer Slide Time: 01:36)

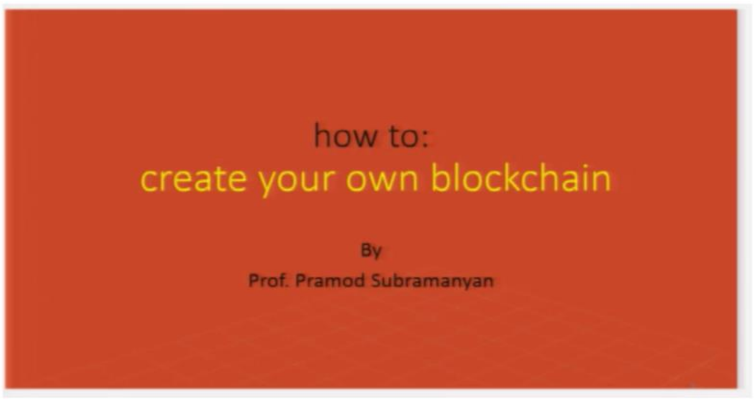


Acknowledgement

- The material of this lecture and the corresponding exercise was created by Prof. Pramod Subramanyan (IITK)

So, the material for this lecture and the exercise was created by my colleague, Professor Pramod Subramanyan from IIT Kanpur. So, we should acknowledge his help in creating this exercise.

(Refer Slide Time: 01:48)



how to:

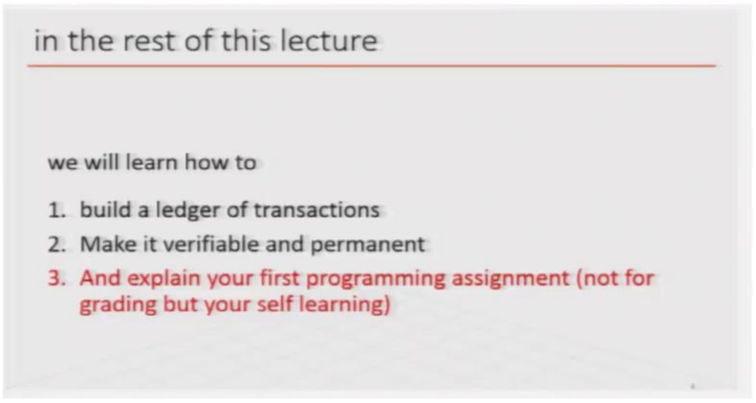
create your own blockchain

By

Prof. Pramod Subramanyan

So, the question is how to create your own blockchain.

(Refer Slide Time: 01:53)



in the rest of this lecture

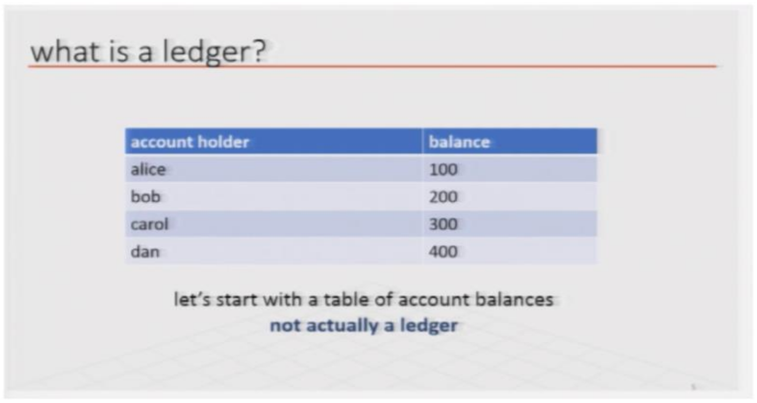
we will learn how to

1. build a ledger of transactions
2. Make it verifiable and permanent
3. And explain your first programming assignment (not for grading but your self learning)

So, what we will do in this lecture is we will build a ledger of transactions and once you build a ledger of transactions, you have to make sure, that I can or anyone can verify that it is a valid, the transactions are valid, and therefore, the blockchain has everything valid. And also the how to make this thing permanent. That is how to ensure that nobody can tamper with data once it is put in the blockchain. And then we will explain the first programming exercise is for your self-learning so we will not grade them.

If you did not know C++ you may not be able to do this however, the concepts are very important. So you should pay attention to what we are doing maybe you will have a little difficulty understanding, how we are using pointers and various kinds of data structures. But you can probably, you know, guess what we are doing, and that will be enough to understand.

(Refer Slide Time: 03:02)



what is a ledger?

account holder	balance
alice	100
bob	200
carol	300
dan	400

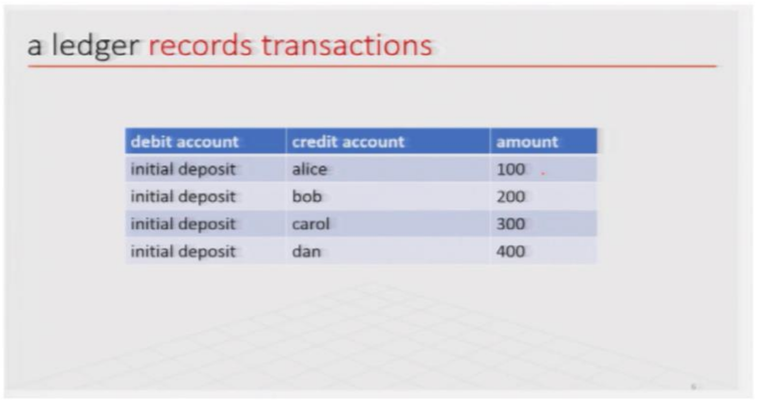
let's start with a table of account balances

not actually a ledger

So the question is what is a ledger? So a ledger could be many things. But let us look at this. So let us say this is a bank's balance sheet. And here we have 4 account holders, and they have their balances. The question is, is this balanced table? A ledger? The answer is not really, because it is a balanced table. So it does not remember the transactions. It only remembers after a transaction, what the new balances for each of the users who are part of that transaction.

But we want to remember all transactions and we want to make sure all transactions are made by legitimate people. And we want to also make sure that the transaction actually is a valid transaction.

(Refer Slide Time: 03:54)

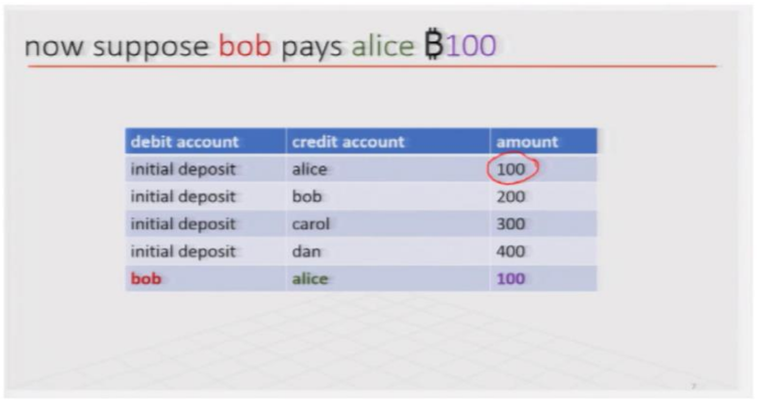


a ledger records transactions

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400

So now if we instead keep a record of transactions, so you should here that we have a record of transactions. So when Alice opens her account, she makes an initial deposit of amount 100, Bob makes an initial deposit of 200, Carol makes an initial deposit of 300, and Dan makes initial deposit of 400. So, initial deposit is one of the transactions. So, it has to be put in the ledger.

(Refer Slide Time: 04:24)

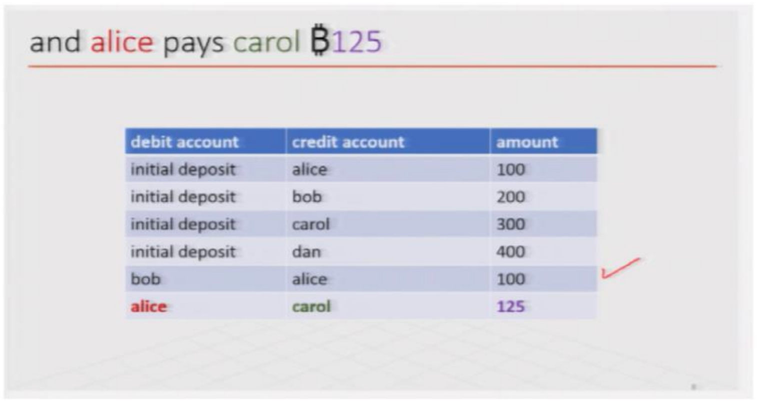


now suppose bob pays alice ₿100

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100

Suppose now real transactions starts happening. So let us say Bob now, for some reason, maybe Alice sold something to Bob and Bob makes a transaction maybe over the internet or by cheque, and Bob gives Alice 100 amount and when that happens, then earlier Alice used to have 200 now, the Alice's account should be debited to become 200, because Bob is now giving money to Alice.

(Refer Slide Time: 04:55)

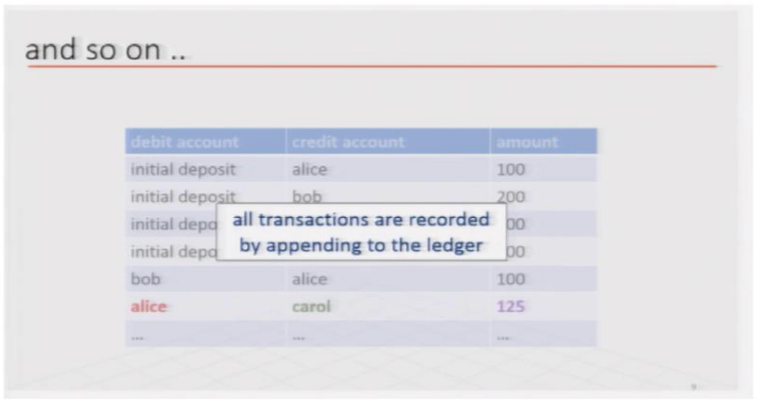


and alice pays carol ₿125

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125

So at this point, Alice also now gives some to Carol, and in the amount of 125. So we have 2 new transactions where after the initial transactions initial deposit transactions, this is when Bob gives Alice 100 let us a bitcoins, and Alice gives Carol 125 bit transactions.

(Refer Slide Time: 05:19)



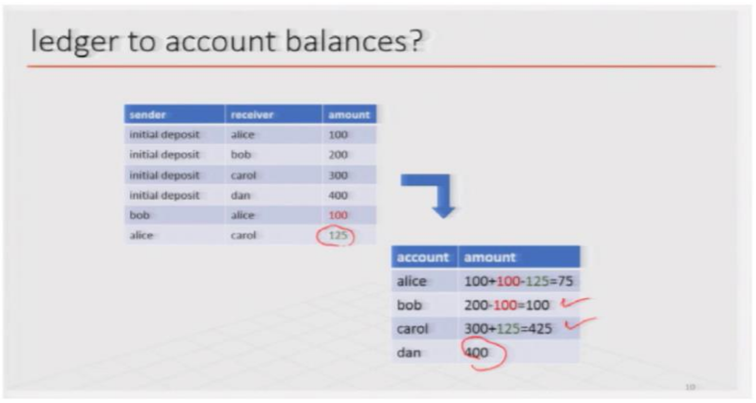
and so on ..

debit account	credit account	amount
initial deposit	alice	100
initial deposit	bob	200
initial depo		00
initial depo		00
bob	alice	100
alice	carol	125
...	...	...

all transactions are recorded by appending to the ledger

And so on so as the transactions come in, we want to add them to the ledger. And the ledger is always up and did nothing. You did not we did not normally want to, we did not go back to previous transactions and make any changes to them. So it is an append only ledger, right. And we want to ensure that this append only property is enforced and nobody can go and change the previous transactions once they have been committed.

(Refer Slide Time: 05:54)



ledger to account balances?

sender	receiver	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125

account	amount
alice	100+100-125=75
bob	200-100=100 ✓
carol	300+125=425 ✓
dan	400

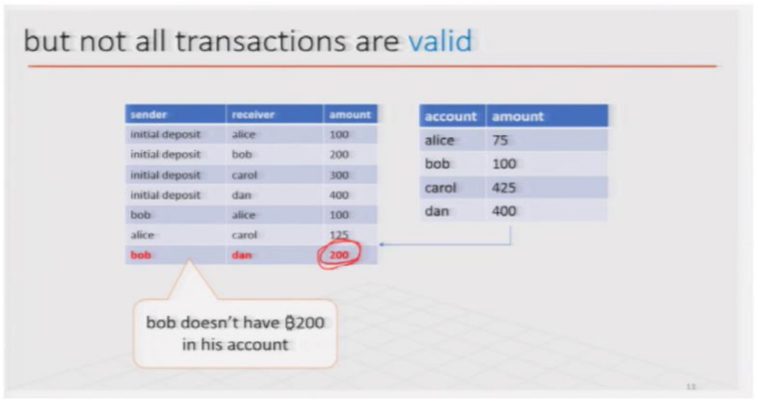
10

And then I can construct the account balances by going through the transaction so earlier we were we started with an account balance table. But what we are doing here is that if you keep track of all the transactions, that is enough, because at any point in time, you can actually reconstruct that table of account balances. So, I can actually see that Alice initially deposited 100, then she got from Bob another 100, then she gave 125 to Carol.

So her final balance at the end of this transaction is going to be 75. Whereas here, if you see Bob, he has given 100 to Alice. And initially he had 200. So now he has 100. Carol actually got money first by initially depositing herself 300 + 125 years so she has now 425 and Dan did not get any money or give any money to anybody. So he is balanced out that 400 so, this balance table can always be reconstructed from the ledger of transactions.

So ledger of transactions is very, you know fundamental. Whereas from the balance table you cannot reconstruct the transactions. So it is better to keep the transactions in the ledger, then the account balance, because account balance is going to lose information.

(Refer Slide Time: 07:23)



but not all transactions are valid

sender	receiver	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125
bob	dan	200

account	amount
alice	75
bob	100
carol	425
dan	400

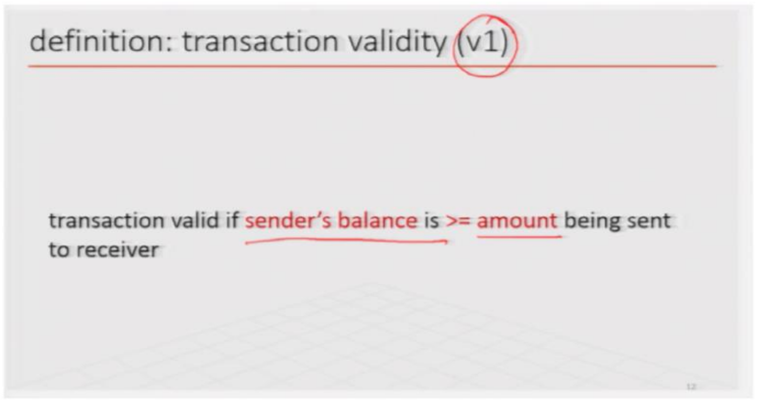
bob doesn't have ₿200 in his account

Now, when I try to do a transaction, so we spoke about that in the earlier lectures, that not all transactions are valid. For example, suppose at this point, Bob wants to give Dan 200 but Bob has an amount of 100 after all the transactions he has made earlier. So if he tries to give 200 to somebody like Dan, then that transaction will not be valid because he does not have 200 at this point. So therefore, this transaction will be rejected or should be rejected.

By the, whoever is keeping track of validity of transactions. So, in case of a bank, the bank keeps track of the validity of the transaction. But in a blockchain will see that such attempt to make invalid transactions will be checked by everybody who is part of the blockchain or whoever is a part of the blockchain and is interested in validating transaction and why would I be interested in validating other people's transaction because I want to be part of the mining system.

And so, if I validate transactions and then somehow after validating transactions, if my validation results are actually kept in the blockchain, and many others are validating transactions, and theirs is also trying to be on the blockchain and if mine goes through as part of the blockchain, then I get rewards. I get there are so many bitcoins or whatever the currency is, and therefore, I have an incentive to do the validation. This is how a decentralized kind of crowdsource validation of all transactions can be met without a central authority like the bank to do this validation.

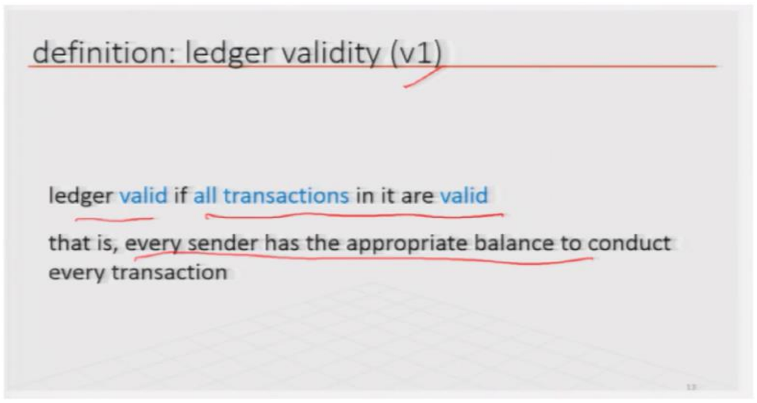
(Refer Slide Time: 09:20)


definition: transaction validity (v1)

transaction valid if sender's balance is >= amount being sent to receiver

So, definition of transaction validity is now then if the sender's balance is greater than equal to the amount that the sender is trying to send, then we say that the transaction is valid, otherwise the transaction is invalid. And we call it transaction validity definition version one, because as you will see, this is not this is necessary, but this is not sufficient to declare a transaction to be valid. So we will come back and redefine or add more into this definition and get version 2 of the transaction validity.

(Refer Slide Time: 09:56)

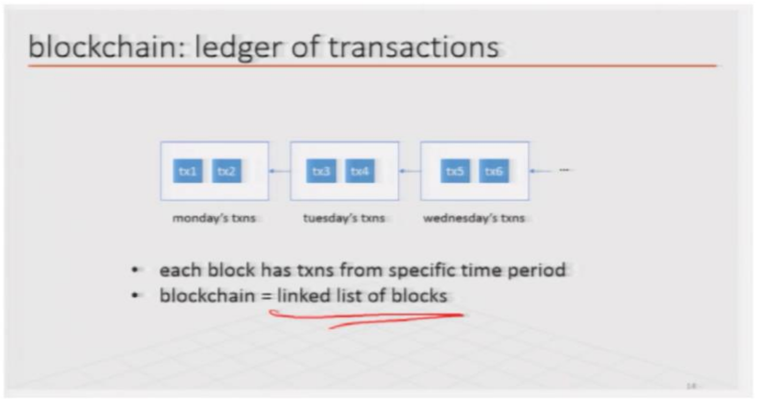

definition: ledger validity (v1)

ledger valid if all transactions in it are valid

that is, every sender has the appropriate balance to conduct every transaction

And the ledger validity is that if all transactions that are recorded in the ledger. If they are all valid transactions, then we say that the ledger is valid. And this is also version 1, because it is actually based on the version one of transaction validity. So that is a valid ledger definition. So in other words, every sender has an appropriate balance to conduct every transaction. So that is C, and they have been recorded, and that is a valid ledger.

(Refer Slide Time: 10:27)



blockchain: ledger of transactions

- each block has txns from specific time period
- blockchain = linked list of blocks

Now, what happens in a blockchain is that there is some further optimization. So if you start keeping every transaction, you know as a separate entity in the ledger, then and there are many, many transactions, then the ledger grows very, fast. And remember, the ledger is has to become a linked list. So you will have too many nodes in the linked list. So what people do is that they wait for some time and bundle all the transactions that happens within that time check their validity and make a block.

So, and then these blocks are connected as nodes in a linked list. So, therefore, now how you can decide how you want to make this you know how much time window you have to wait until you put the transactions in a block. So, it could be on a daily basis it could be on a on every 1 hour it could be every 10 minutes, it could be some other criteria by which you decide when it is time to put all the transactions that you have seen into a block.

Now, interesting part here to understand is that if bank is doing it, then it has a fixed rule that I will only wait for 1 hour and every hour, whatever transactions has happened. I will put them in a block I will validate them and then I put at that block into the linked list, but if it is being done by everybody independently, so, there are let us say 10,000 people, each of which are looking at transactions and these transactions as they come at some point they have to decide.

Now it is time for creating a block with the transactions that I have seen after validating them. Now, remember this users or this node this participants in the blockchain are distributed geographically, maybe some in the US or in India, some are in China, some in Africa. So, therefore, their notion of time will be very because the clock synchronization is not being applied here. So, therefore, what is when I think that it is one hour the other participant from another place may not think it is 1 hour maybe a few minutes here.

And there also, the other problem is that if I make a transaction in India through the network to reach to receive that transaction happening may take more time to the US user and a transaction happening in the US, the Indian observer may actually see them later. So, what may happen is that even if you have a standard notion of 1 hour, but the transactions you have seen at this point this by observers in US and observer in India would be little different.

To the reach of the transaction information over the internet may be a little bit delayed based on the proximity of the transaction happening and the observer. Therefore, everybody when they start creating blocks, that blocks will differ. Also, be user may say, I did not like this particular transaction. So I may not include it in the block right. So therefore every user say 10,000 users are all trying to make this valid blocks of transactions.

The reason they are doing it is because they will get some reward, but they the blocks they create may differ from each other. So, then the question is, how do you decide how the system decides whose block will be the next block in the linked list. And that is where the whole mining process or the last time we talked about the hash puzzles. The hash puzzle solving, or mining process takes place. And that is how a competition starts among all the users.

And this competition is won by solving the hash puzzle first, and when the hash puzzle is solved by somebody, then the person can claim that my block is the winner and this block goes into the link less. Now, what that means is that depending on who wins B, which transactions go into a block will be different. However, in this setup in in our particular programming exercise, we would not have this kind of uncertainty, because we will be doing it in from one place. Why? Because it is just an illustrative example.

This is not how the blockchain works in terms of creating blocks of transaction and validating blocks of production, or adding them to the list. What we are doing here is just to get the concept of blocks and concept of linking them together, rather than completely mimicking what happens in a realistic blockchain. Therefore, for us, which transactions come into a block would be much easier to decide.

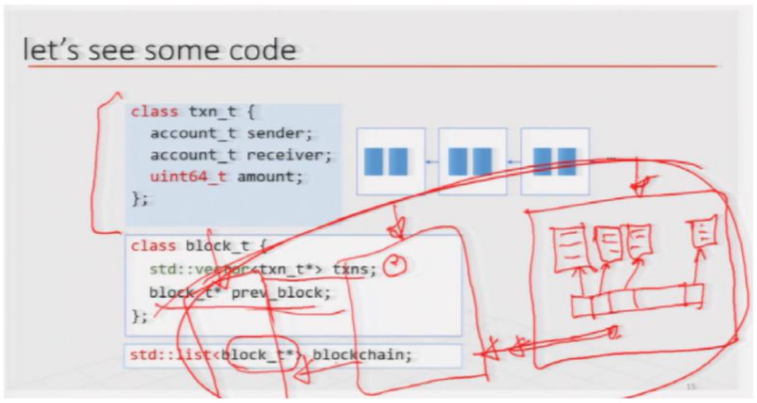
And there will be only one block created because only one entity in which case your program will be creating the blocks and adding them to the blockchain. But I just want to ensure that you understand that the difference between what we are doing in this program and what actually happens in the real blockchain. So, the concept of mining is absent in this exercise or in this example, only the concept of creating blocks, validating the blocks.

And validating the transactions and putting them in a list is what we are going to do here, which is sufficient to understand some of the concepts, but not all the concepts. Also, as you will remember, when I am doing it from one program and no other participants, I can create a real C++ linked list right. So, we spoke about that in the previous classes that in C++ linked list, a link between from one block to the other is actually a pointer, which is a memory address.

But we also said in the previous lectures that in case of a distributed and replicated blockchain, this memory address concept is not applicable. And then we do hash pointers here also we will do some form of hash pointer. But that is also different from how the hash pointers are used in the real blockchain. So those are the differences. So we will in this case, we will actually create a real C++ linked list or C linked list just illustrate the concept of a linked list of valid blocks.

And this linking is done through memory addresses. But with a little bit of more idea thrown into it that we will also have the hash of the previous block stored in the current block, and therefore it becomes kind of hash pointer.

(Refer Slide Time: 17:37)



let’s see some code

```
class txn_t {
  account_t sender;
  account_t receiver;
  uint64_t amount;
};

class block_t {
  std::vector<txn_t*> txns;
  block_t* prev_block;
};

std::list<block_t*> blockchain;
```

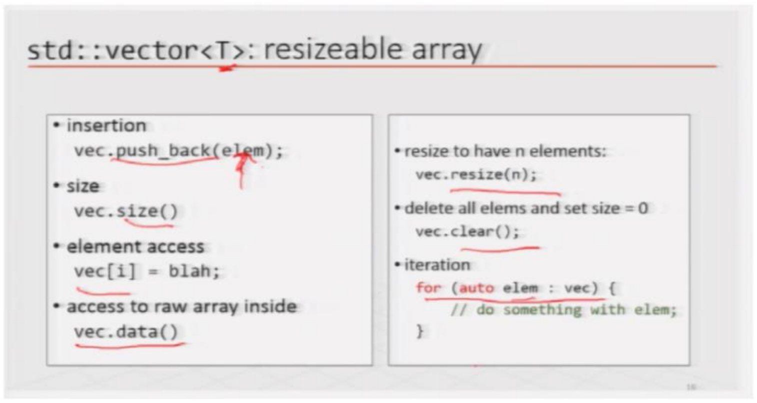
So let us see some code then in C++ we have this notion of a class is basically a structure and this class will call transaction underscore T. And this class is basically has 3 elements in the class 3 data items. One is the account information of the sender. One is the account information of the receiver and another is a 64 bit unsigned integer amount because we are not going to deal with negative amounts in transactions we are doing monetary transactions here.

So, this is a class, any instance of this class that is an object created in this prototype of this class will be kind of representing a transaction so, every time a transaction happens, we will be considering this, kind of information structure that will have sender information receiver information and transaction information. So, Bob gives Carol 100 coins. So, that is information that we want to put to that way the classes 3 elements.

Now, we have to also represent blocks. So what is a block is actually a collection of transactions and then it has to have a pointer to link the previous block. So it has 2 elements, 1 is vector of transaction. So vector is basically kind of you can think of is an array of transactions, although it has a lot more interesting properties than arrays, but it is a transaction. So it is a, it is actually a list of pointers to transactions of this type.

So, you will have a vector of transactions, which will be the all the transactions that you want to put in the block. And then you have a pointer to the previous block. So that is how the block is defined. And then the linked list then will be a list of blocks, or in this case, a list of pointers to the blocks. So we keep pointers to the blocks, and the blockchain will be a list of blocks. So that is the concept here.

(Refer Slide Time: 19:44)



std::vector<T>: resizeable array

- insertion
```
vec.push_back(elem);
```
- size
```
vec.size()
```
- element access
```
vec[i] = blah;
```
- access to raw array inside
```
vec.data()
```
- resize to have n elements:
```
vec.resize(n);
```
- delete all elems and set size = 0
```
vec.clear();
```
- iteration
```
for (auto elem : vec) {
    // do something with elem;
}
```

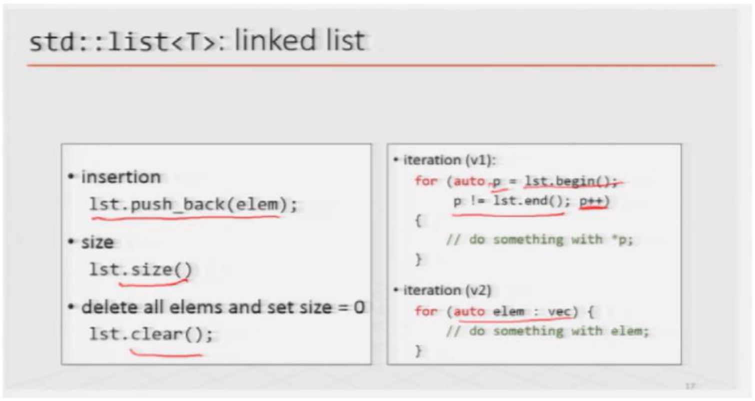
Now let us look at this data structures from the standard C++ Library. So vector is actually a template. So it can take any type T and it will create vectors of that type and if you want to add elements to this vector, so you use this function push back and whatever you are trying to push into that vector. So, this elem or element should be of the type, whatever type T you have defined. Similarly, you can check the size of the vector with the size function.

You can also access, say i th element of this vector using an indexing, like an array, so vector of i equals something would put that thing into the vectors if position. And if you want to extract the array of information inside the vector, then you can call the data function and it will give you access to that array inside the vector. Now, sometimes when you add things in the vector, so, let us say initially you define the vector of a certain size.

And then you want to add more stuff into the vector in that case because you realize that you know, initial size that you thought of is not enough, then you can call a resize function. And then it will resize it to have another in number of elements in the vector. You can also clear out a vector with a clear function or delete all the elements and set the size to 0. And then finally, if you want to iterate, if you go through the list to do some action on the elements of the vector.

For example, reading them, or changing them or playing a function on them, then this is the new way of doing it in the C++ recent standards that you basically say that elem is the element of a vector. And what this means is that you automatically iterate through the vector Zeroth element, first element, second element, third element, all that stuff inside this loop. So this for loop basically allows you to go through all the elements of the vector and do something with the element.

(Refer Slide Time: 21:57)



std::list<T>: linked list

- insertion
  `lst.push_back(elem);`
- size
  `lst.size()`
- delete all elems and set size = 0
  `lst.clear();`

- iteration (v1):

```
for (auto p = lst.begin();
     p != lst.end(); p++)
{
    // do something with *p;
}
```

- iteration (v2)

```
for (auto elem : vec) {
    // do something with elem;
}
```

17

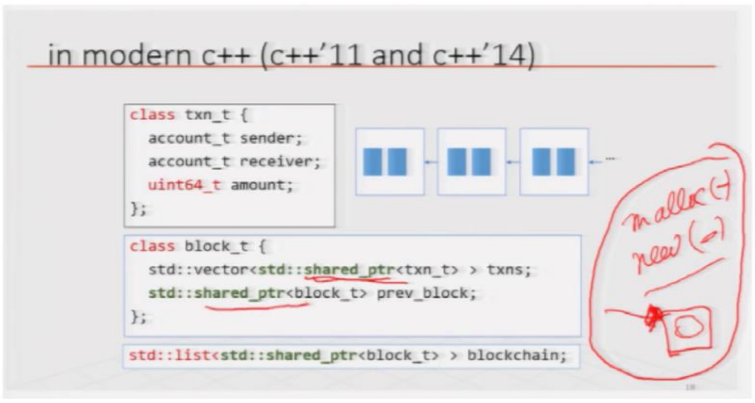
Now the other structure that you saw so vector was actually used to store transactions. And then we other thing that we saw is a linked list of blocks. So list structure was used to do that. So list has, again, certain manipulations. So in case of, you want to add an element to the list at the end, then you basically do a push back function. If you want to know the size of the list, you can call the size function, you can delete all the elements of the list by clear function, you can also iterate by various methods.

So one is that you do like you know i = 0; i0 = n, i + kind of thing. So here, it is a pointer iteration. So, you take the first pointer, and then you iterate over it, and then until you reach the last pointer in the list, so, that is one way of doing that. And the other is doing how you were doing it on the vector. You do how to do vector and do something with the elements. So for you to understand what is going on here again.

If you did not know C++, you can think of the blocks. Let us look at this so, you can think of a transaction has 3 elements the sender receiver and amount and they are put the pointer to them is put in a vector right so a vector which has first location, second location, here is another transaction, third, location here is another transaction and so on. So this is a vector of pointers to transactions. So each of these transactions has 3 elements now is that this whole thing along with the pointer is a block.

So then when the next block comes, a new block is created, this pointer is set to the previous memory rest of this block. So this is how this is done. So and then another one here. And then this is a list. So this is the list of. So this, these are the pointers to the blocks. And remember this pointer, and this pointer is pointing to the same thing, this pointer and this pointer is pointing to the same thing, but this is a list of pointers to blocks and that is your Linked List of blocks. So, that is how we are representing the entire blockchain.

(Refer Slide Time: 24:29)



in modern c++ (c++'11 and c++'14)

```
class txn_t {
  account_t sender;
  account_t receiver;
  uint64_t amount;
};
```

```
class block_t {
  std::vector<std::shared_ptr<txn_t> > txns;
  std::shared_ptr<block_t> prev_block;
};
```

```
std::list<std::shared_ptr<block_t> > blockchain;
```

Now, another thing that the modern C++ gives you now pointed management so those of you who know C++ know that pointer management is a big headache. For example, if you define a pointer, and then you allocate some memory to be pointed to by that pointer, like using a malloc, or a function or a new in case of C++, then you are pointer is now pointing to a chunk of memory depending on what you have put here as size and everything.

Now, if you stop use of this particular chunk of memory later in your program, then you are supposed to deallocate. So, you have to deallocate this amount of memory otherwise your program will start running out of memory. Now, that deallocation is a lot of concerned, because many times people forget to do allocate, and then we have this problem of memory leak. So the memory that is allocated to be pointed to by a pointer, it has been used for a while.

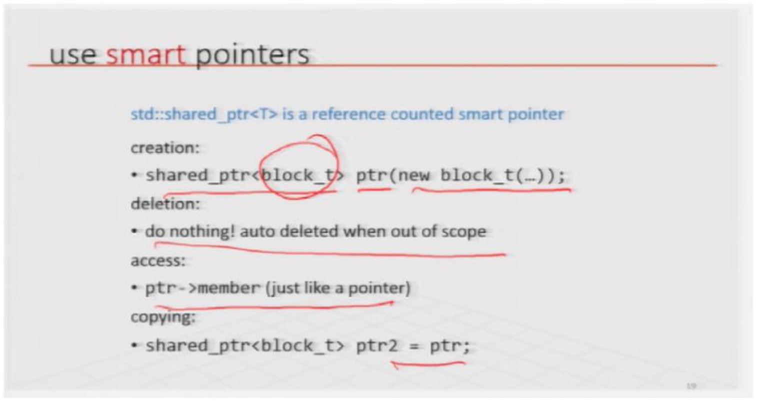
And then after a while, it is of no use, but the user, the programmer, forgot to say dealloc or the, you know, destroy or something. And therefore, the memory remains they are sitting there with nobody using them. And if it happens in a lot of such pointers, for example, in a loop, then it keeps accumulating and your memory space gets you know, gets used up and your program then will crash. So, therefore, that is a memory leak problem.

The other problem is the dangling pointer like you have a pointer for which the memory has been de allocated, but you have in your program, you start using that pointer for referencing something with that pointer and then you have a problem of null pointer exception. So, therefore, people are quite worried about using pointers. So in modern C++, there is a smart data structure called shared pointers.

Shared pointers actually are what we call reference counted pointers, which means it keeps track of whether it is under use whether the memory allocated for it is underused or not. And based on that it deallocates automatically as a programmer once you have created the pointer, you did not have to worry about deallocating the pointer, or, you know, worry about null pointer exception, because as soon as the pointer gets out of scope, or not being used, it is deallocated.

So that is why in this program, shared pointers are being used. So the only advantage here for you in the shared pointer usage is that when you are looking at you did not have to worry about deallocate.

(Refer Slide Time: 27:25)



use smart pointers

std::shared_ptr<T> is a reference counted smart pointer

creation:

- `shared_ptr<block_t> ptr(new block_t(…));`

deletion:

- do nothing! auto deleted when out of scope

access:

- `ptr->member` (just like a pointer)

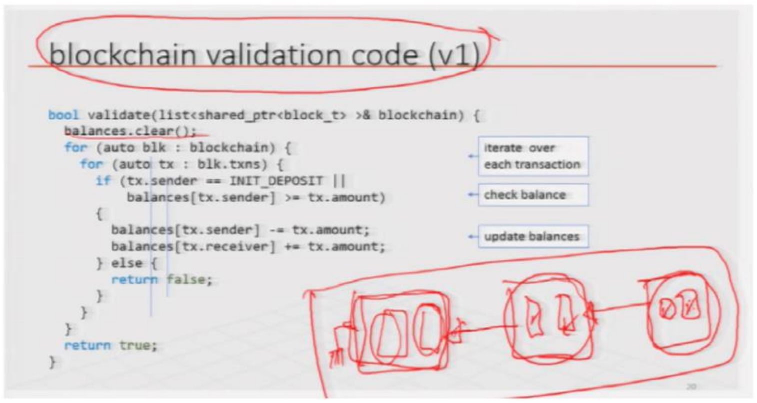
copying:

- `shared_ptr<block_t> ptr2 = ptr;`

So that is the other thing. So it is a, as I said, is a reference counted pointer. So when you define a pointer, you are, going to actually define it like this. So it is a shared pointer, which points to a block, in our case, the block we defined, and then you have to define it like this for deletion. You did not have to do anything. It is auto deleted when out of scope. So that is the reason for using shared pointer.

And the usage is like a pointer right pointer arrow member name will give you access to that member of that structure which is a pointer is pointed to so, if the structure has 3 members, member 1, member 2, member 3 then you need to know the name of the pointer that is pointing to that and then you reference them as member 1, member 2, and member 3. So, that is the use of smart pointers and you can also copy a pointer to another pointer to point to the same thing and so on.

(Refer Slide Time: 28:24)



blockchain validation code (v1)

```
bool validate(list<shared_ptr<block_t> >& blockchain) {
  balances.clear();
  for (auto blk : blockchain) {
    for (auto tx : blk.txns) {
      if (tx.sender == INIT_DEPOSIT ||
          balances[tx.sender] >= tx.amount)
      {
        balances[tx.sender] -= tx.amount;
        balances[tx.receiver] += tx.amount;
      } else {
        return false;
      }
    }
  }
  return true;
}
```

So, now, the question is, how do you validate a blockchain So, what is the validated blockchain do you have transactions etc in one block and you have next block pointing to it with transactions in them, next block pointing to it with transaction to them. And then when I say valid, I have to first check that in each of the blocks, your transactions are valid. And when I say transactions are valid, I have only defined what it means for a transaction to be valid partially as version 1.

So, therefore, this validation code is also version 1. So, first I go and validate all transactions in a block and then we declare that this block is valid. Similarly, I do this for this and then I declare this block is valid. Similarly, I do for this and I declared this block as valid. And suppose this is the first block or the genesis block. In that case, this is not pointing to anything else. So, therefore, I declared that the entire chain is valid in the sense of the chain we have defined in this program.

So, let us see how the code works. So, first of all, in this case, we actually use a balances table, right so balances table as I argued earlier, is that I can create the balances table from the transaction ledger. Now, why do I then recreate the balances table within the program? The reason is that to validate that a transaction is valid, I have to check whether the sender's balance is greater than the amount he or she is trying to send. So I need to quickly calculate the balance of the sender.

Now if I have to calculate the balance of the sender, every time I validate a transaction, then you will be redoing the same work again and again, I have to go through all transactions and add and add and subtract, or any transactions, any amounts that he received and subtract the ones that he has transferred to somebody else. So therefore, it is like a real kind of like a caching technique. So we keep our balances stable on the site.

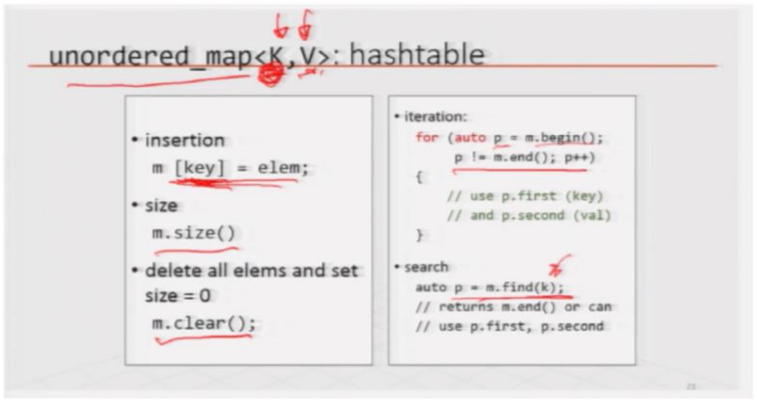
And therefore, first what we do is that we create a balances table. And then what you do is that we go through the transaction and check whether there is a real sender or it is an initial deposit. If it is an initial deposit, then all you need to do is that you have to add that amount to the receivers balance if it is not initial deposit, which means somebody sending money, so that the sender's balance should be greater than the transaction amount.

And that transaction amount should be subtracted from the sender's balances and added to the receivers balances. And if this transaction is, neither initial deposit or the sender's balance is greater than equal to transaction amount, then we return false. That means is that this code is for validating transactions. So we are basically looking at all transactions. So here you see that we are doing an auto blk blockchain in auto tx block dot transactions.

So I am picking one block, and I am going through all these transactions. Then I am picking the next block. Going through all its transaction I am picking the next block and I am going through all each transaction and for each transaction I am checking whether it is an initial deposit or the balance of the sender is greater than the amount then I do the appropriate changes in the balances and then I return false.

So, in this I have to one side effect which is the balances table at the at this current condition of the blockchain the blockchain is a growing structure, but at any point when the blockchain is at this stage, maybe later on there will be more here, but right now, I have up to this. So, at this point, if I validate or not only I validate all the transactions, but also I have constructed the balances table. So, in the beginning of this validate call, I cleared out the balances if there was any other previous information and then we start reconstructing the balances field. So, this is how this is done.

(Refer Slide Time: 32:49)



unordered_map<K,V>: hashtable

- insertion
```
m [key] = elem;
```
- size
```
m.size()
```
- delete all elems and set size = 0
```
m.clear();
```

- iteration:
```
for (auto p = m.begin();
     p != m.end(); p++)
{
    // use p.first (key)
    // and p.second (val)
}
```
- search
```
auto p = m.find(k);
// returns m.end() or can
// use p.first, p.second
```

Now, how are we keeping the balances table? So, what we have done is that we create what is called a hash table. So, and what is a hash table? Hash table is a table, which is indexed by hash. So let us say it is a table with 2 columns. In one column, you have some element. In the second column, you have the other element. And then to access the second element, like, let us say, Bob, this, Alice this so in order to know what is there for Alice.

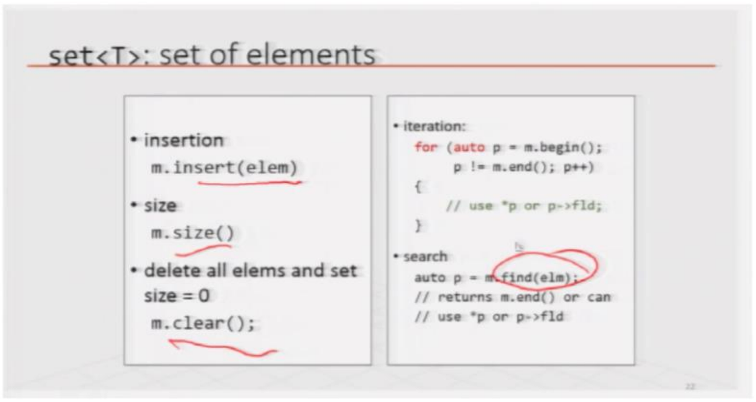
I have to index directly to Alice's name. And then I should be able to read what Alice has so in this case, if it is a balance table, it will be Alice and her balance, it will be Bob and his balance. So this is how the hash table is formed. So in this case, called an unordered map, and it has 2 different types. So k is the type of the first element or first column and v is the type of the second column. So if the first column is a string, and second column is an integer.

Like Bob and his amount, then we will have integer and will have the string and the integer. If this is a more complex structure, then let us say an address, then it will be in that structure, and it will be amount. That is how we are going to define. Now the way we map is accessed is by indexing the map with this elements. So you can think of this kind of an array, where index is used to access the element here, instead of array index, we can have our type k thing to index the array. So that is the idea.

So here also again, to insert something, you have to just say what key and what the element that you are inserting the size can be obtained by this call. You can clear the map on the table to make it reset. And you can iterate through it like if you want to go from the first elements of the table, the first row of the table to the all the way to the end. You have to do something like this like a point after that begins here, these are called iterators.

Actually, they are not really not necessarily pointers, and then you do this iteration or you can also search by calling a find function. So search will allow you to say, what is there for Alice, so usually find Alice as a string, and then it should tell you what the amount of Alice has.

(Refer Slide Time: 35:22)



set<T>: set of elements

- insertion
  `m.insert(elem)`
- size
  `m.size()`
- delete all elems and set size = 0
  `m.clear();`

- iteration:

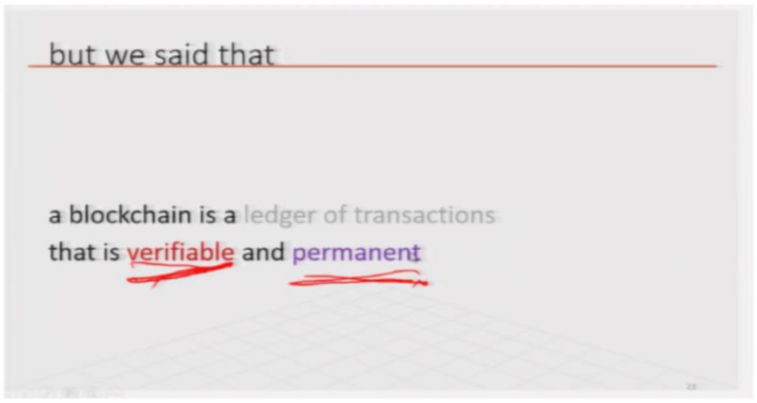
```
for (auto p = m.begin();
      p != m.end(); p++)
{
    // use *p or p->fld;
}
```

- search

```
auto p = m.find(elm);
// returns m.end() or can
// use *p or p->fld
```

And also we have a set data structure or a template C++ template. So set as you know, is a set of elements. So, if you want to represent it in a data structure, you have to have a way to insert an element you have to know the size that is how many elements there are in the set, you can reset the set so it does not have anything anymore. And you can iterate over this set to go from the one element to the next to the next and so on. And you can also search in the set. So, that C set data structure.

(Refer Slide Time: 35:54)

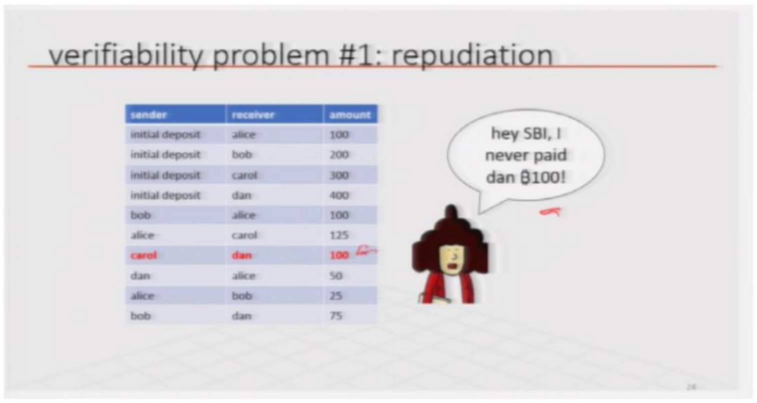


but we said that

a blockchain is a ledger of transactions
that is verifiable and permanent

Now, blockchain is a ledger of transactions. That is verifiable and permanent. So now, we only focused on the verifiable part that the transactions have some validity. But permanence is something that we have to now worry about.

(Refer Slide Time: 36:14)



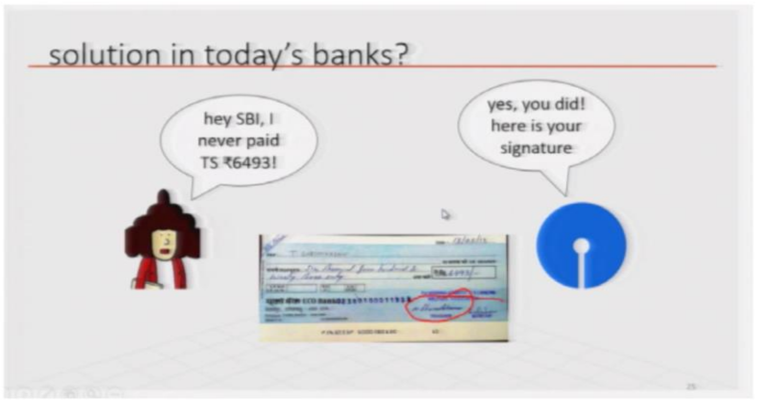
verifiability problem #1: repudiation

sender	receiver	amount
initial deposit	alice	100
initial deposit	bob	200
initial deposit	carol	300
initial deposit	dan	400
bob	alice	100
alice	carol	125
carol	dan	100
dan	alice	50
alice	bob	25
bob	dan	75

hey SBI, I never paid dan B100!

So one issue is that we talked about repudiation a lot in the previous lectures. So let us say in the transaction found that Carol has given Dan 100 coins. Now, Carol comes back and says, I never paid Dan that money. So but why is it in your transaction, it is affecting my balance in the balance table. And so I repeat yet I did not agree that I have done this transaction.

(Refer Slide Time: 36:42)

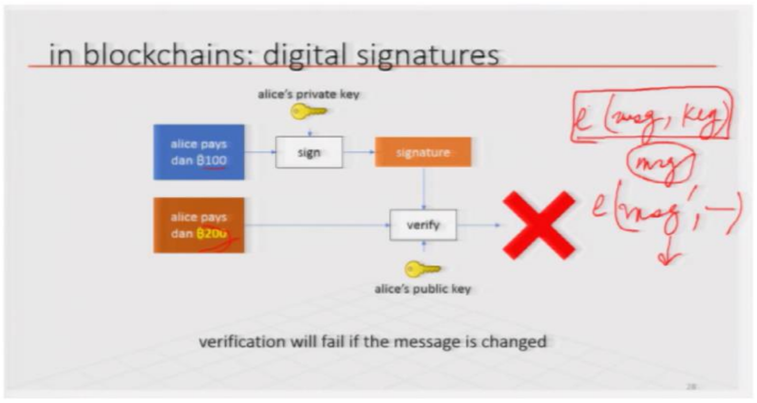

solution in today's banks?

hey SBI, I never paid TS ₹6493!
yes, you did! here is your signature

What do you do in the today's bank case? That bank comes back to you and says that look, here is the check you signed? This is your signature, we verified your signature and therefore you cannot repeat here in case of an internet banking. What you do is that you say, look, Carol, this was done from your account, you use the signature, you use the username, password and OTP and therefore it is you right? So that is how the banks handle repudiation cases.

But in case of blockchain as I said that blockchain is there is no central authority. So you cannot just go and complain to 10,000 people who are all part of the blockchain, they are all checking validity of transactions, they are all trying to make blocks of transactions and hoping that their block makes it into the blockchain. So, there is nobody particular to listen to your reputation. So, therefore, this system should be non-repeatable. So any transaction that has been validated should be non-repudiation.

(Refer Slide Time: 37:46)



And this is where digital signatures come in handy. That is when Alice says that I am paying Dan 100 coins, Alice must sign it with her private key, and then Alice's public key should be available to everybody. So therefore anybody who wants to check the validity of the transaction, now, not only they check the amount of the transaction being less than the balance Alice has, that is one part of the validation. But the second part of the validation would be that they use Alice's public key to check it is indeed Alice's signature.

If it is Alice's signature, then only they declared the transaction is valid. Therefore, that transaction once it is there, becomes non repeatable because without Alice's private key nobody could have signed for Alice. So therefore, the signature verification has to be done. Now if the chase signature does not check out, so they know Alice's public key and they check and the transaction is saying that, it is Alice is paying Bob, but is signed by somebody else, then we did not have any reason to validate that transaction.

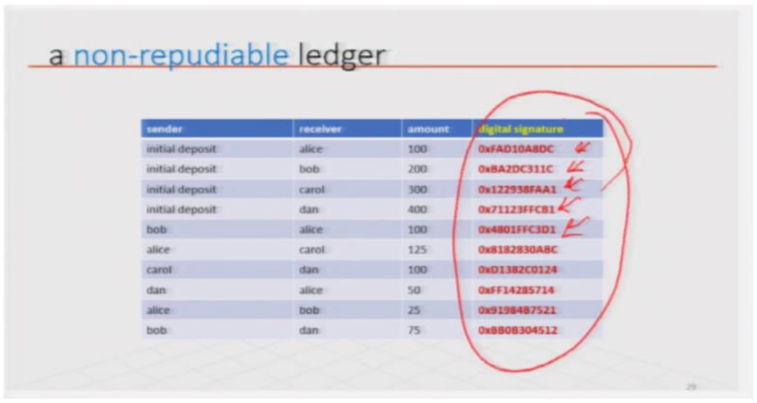
So therefore nobody should be able to forge signature without Alice's private key. That is the assumption now if Alice's private key gets stolen, then that is a different story. But we assume that Alice's private key is Alice's responsibility. And also, if Alice said that I want to pay 100. But somebody in the middle intercept, see this message is going to the broadcast to the entire blockchain. That is how the blockchain works. Everything is sent to everybody.

So if Alice pays Dan, let us say, somebody changes it to say that Alice pays Dan 200 coins, Alice said 100 coins and each sign the transaction, remember and how they sign the transaction, they basically take the transaction message and then apply the private key and then they encrypt it. And this encryption is basically with the private key of Alice. So with the private key, somebody should be able to open it and then they will get the message back.

So, therefore, if message is forged by somebody, they have to also forge Alice's signature. Otherwise, if the message is changed to message prime when somebody encrypts it with some other private key, then that will not be able to check out with the Alice's public key right. So, the when you try to decrypt it, it will come out as garbage. So, therefore, if the transaction message comes through with Alice's signature.

And I am able to decrypt it with Alice's public key that means that Alice must have not only done this transaction, her transaction amount or any information she has put in the message is also not tampered with it so, that basically gives us the not only authenticity, also the integrity of the transaction.

(Refer Slide Time: 40:51)

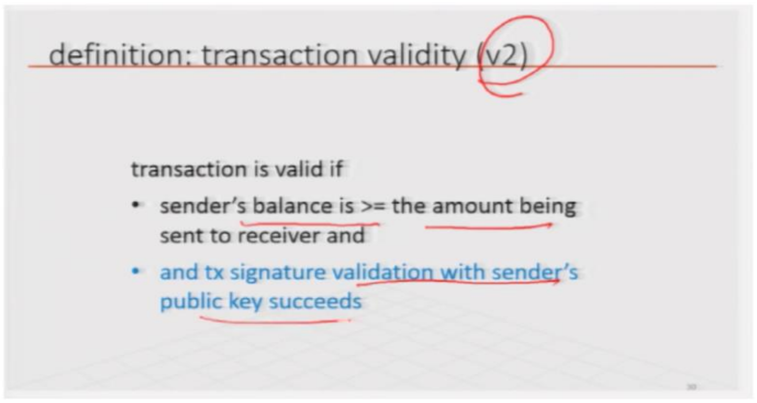


a non-repudiable ledger

sender	receiver	amount	digital signature
initial deposit	alice	100	0xFAD10A8DC
initial deposit	bob	200	0xBA2DC311C
initial deposit	carol	300	0x122938FAA1
initial deposit	dan	400	0x71123FFCB1
bob	alice	100	0x4801FFC3D1
alice	carol	125	0x8182830ABC
carol	dan	100	0xD1382C0124
dan	alice	50	0xFF14285714
alice	bob	25	0x9198487521
bob	dan	75	0x8808304512

So, therefore, now, in the ledger, we have to not only store the transactions message or the information like who gets how much we have to also put the digital signature that came with it. So that later on if I again want to validate all transactions, I should be able to validate it by using the public key of the person who put the transaction in the sender and then use that to verify that the transaction amount and the whoever is saying the transaction is from are genuine. So, we have to also add the digital signature as part of the transaction information.

(Refer Slide Time: 41:30)



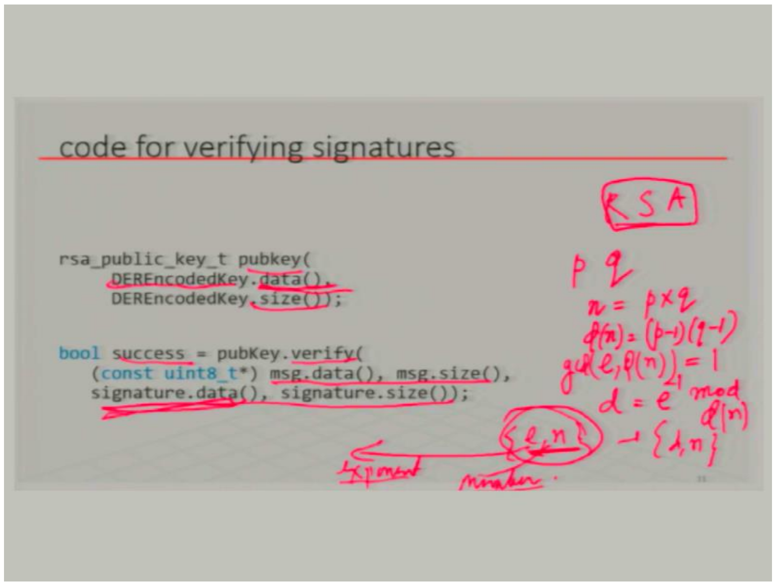
definition: transaction validity $\bigcirc$(v2)

transaction is valid if

- sender's balance is >= the amount being sent to receiver and
- and tx signature validation with sender's public key succeeds

So now we have the transaction validity redefined or extended the definition. So not only the balance of the sender should be greater than the amount. Also transaction signature, validation succeeds. So the sender's public key should be used to validate the signature and it should succeed.

(Refer Slide Time: 41:52)



code for verifying signatures

```
rsa_public_key_t pubkey(
     DEREncodedKey.data(),
     DEREncodedKey.size());

bool success = pubKey.verify(
   (const uint8_t*) msg.data(), msg.size(),
   signature.data(), signature.size());
```

$\boxed{RSA}$

$p \quad q$

$n = p \times q$

$\phi(n) = (p-1)(q-1)$

$\gcd(e, \phi(n)) = 1$

$d = e^{-1} \bmod \phi(n)$

$\{e, n\} \rightarrow \{d, n\}$

Exponent
modulus

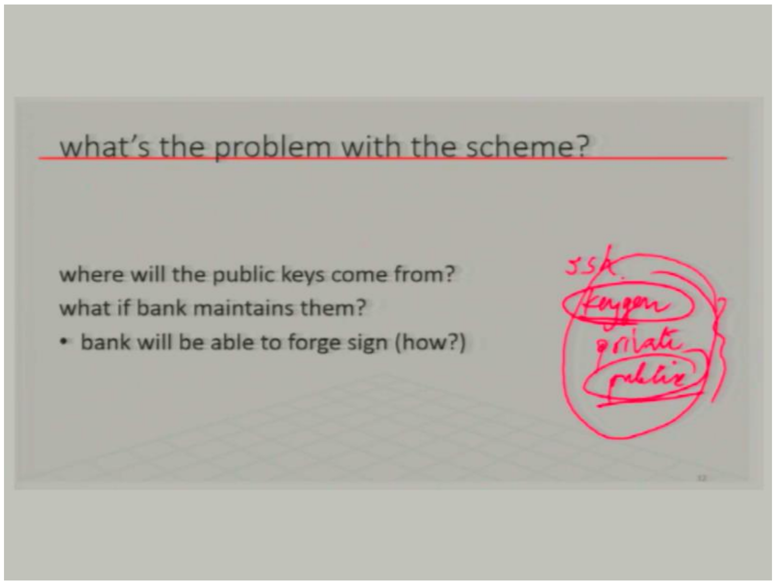
So now how do we do the signatures in programmatically? So there is this RSA public key type So, RSA we talked about the Rivest, Shamir, Adleman. So, here we are assuming that we are going to use RSA public key cryptography, and how a public key is represented is actually there are multiple different ways of representing a public key. Remember we said that in RSA as an aside, in the previous some previous lecture.

We said that we take 2 very large numbers prime numbers p and q, we compute n = p times q and we compute phi n = p – 1 times q - 1. Then we choose a number and then we choose a number e which is mute, which was GCD, which phi n is 1 and then we choose d which is a multiplicative inverse of e module of phi n and then we publish e, n is public key and we keep d and n is private key.

So, in this case, the public key is represented by the, this is called the exponent and this is the number. So, this is e and n has to be represented and how do you represent that you actually do sometimes you do these are very large integers. So, if you will represent them in binary and then convert them to hex or you can return them to binary and then you can do a base 64 encoding. So, based on that there are many encoding schemes, multiple encoding schemes.

So, here we are talking about the, our encoding, which is one of the standard encoding of the key. So, the data of the public key will read this information and then the size of the public key then the size information. Now, so you have this public key has a verify function and verify function basically is success when the signature, verifies with that public key. So the public key information is here, the signature information is here, you send it to the verify function. And if it is success, then we say that we have verified that the signature is genuine.

(Refer Slide Time: 44:33)



what's the problem with the scheme?

where will the public keys come from?
what if bank maintains them?
- bank will be able to forge sign (how?)

ssh
keygen
private
public

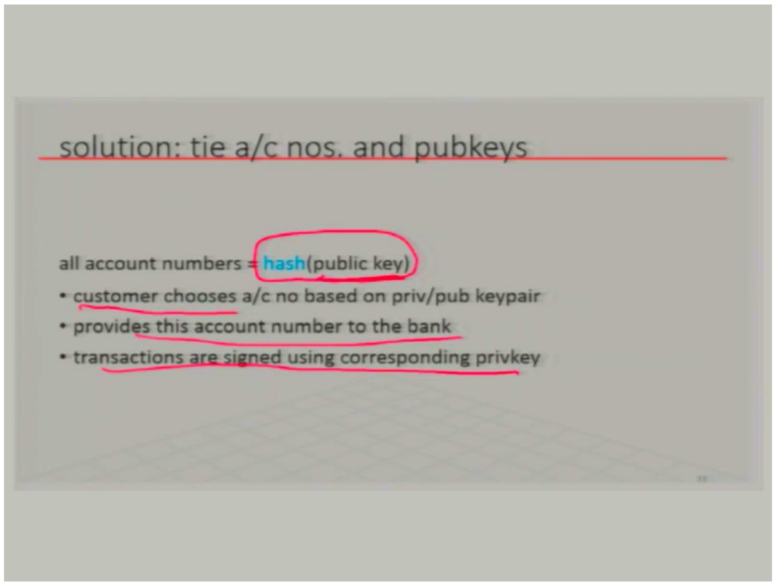
So, that is how the signature is going to be represented. Now, the question is, where do you get the public key? So, you know, in ssh, if you have used ssh, then there is a key Gen function that allows you to create private key and public key, so when you use a key Gen function, that you can create your own private key and public key, then you can keep the private key guarded by some passcode. So that nobody else who actually who accesses your computer cannot easily read it unless they know your passcode the public key.

You have to publish or you have to upload to let us say, github or something. So, so the so this is how we actually generate the keys. But that is not the point we are discussing here generation of the key is not difficult. What we are discussing here is what we discussed in a previous we also is, which is, how do you tell other people what your public key is? Now, in our setting, for example, in this particular example, we can tell the bank, our public key and the bank can store them.

But the question is, if that happens, then the bank can misuse them, especially if the bank has the private key also, like if the bank gives us a public key and private key, then bank generated the private key also, and therefore, the bank will know the private key for public key, so, then bank can actually forge messages transactions on our behalf, which is not good. So, similarly, if I have a blockchain which is not maintained by a bank.

But maintained by let us say a whole crowd like of 10,000 people who are all participating in the blockchain, then and I generate my private public key pair, then I can announce that this is my public key. Now, the problem with that is that what if somebody else announces a public key as mine. So, that problem is still there.

(Refer Slide Time: 46:40)



solution: tie a/c nos. and pubkeys

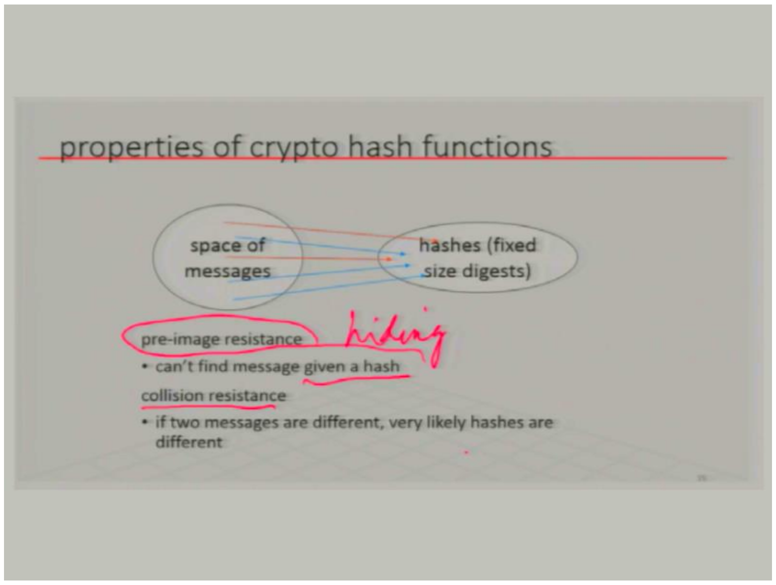
all account numbers = hash(public key)

- customer chooses a/c no based on priv/pub keypair
- provides this account number to the bank
- transactions are signed using corresponding privkey

So, in our case, in our single controlled blockchain case, how we solve it here in this example, is that instead of bank giving us account number see now, when you open an account in a bank tells you this is your account number. In this system, what we assume is that I generate my public key private key pair, keep the private key and hash the public key right and send bank the hash and say that this is my public key, this is my make this my account number.

So, the customer is now choosing his account number banks may or may not like that, but in this case that since this is a made up bank, we assume that the bank is okay with that. So, bank gets the account number for me. And when I sign transactions with the public with the private key, then my account number can be come handy in very validating that that my signature is genuine.

(Refer Slide Time: 47:50)



properties of crypto hash functions

space of messages
hashes (fixed size digests)
hiding

pre-image resistance
- can't find message given a hash

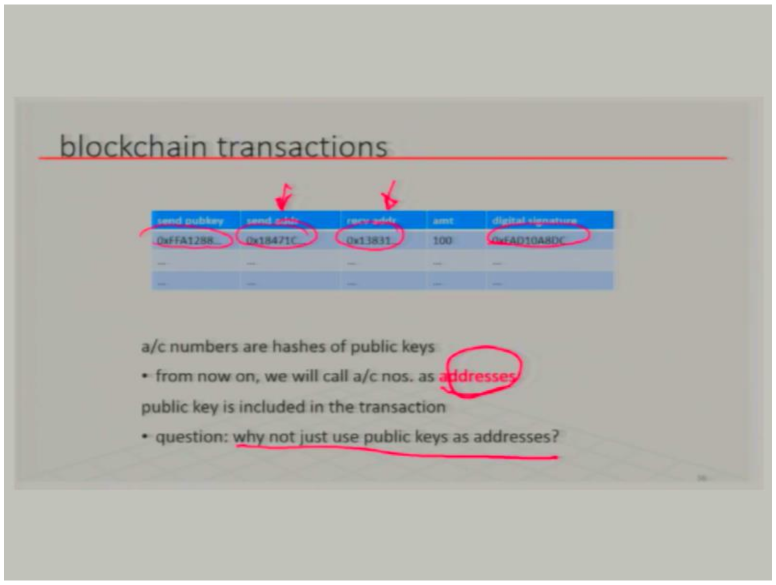
collision resistance
- if two messages are different, very likely hashes are different

So why do we want to hash first of all, the public key if you generate a public key using key gen, you will see that the public key usually very, long string of numbers and therefore, nobody can remember or it is going to take like, you know, many you know letters or characters length string to represent the account number, but if you hash it, then you can map it to a fixed length digest and then you are going to use that maybe 16 or 32 byte digest and then use that as your account number.

So, we will revisit some of the hash function properties, although we have already done this, that the number of real keys because the keys have much, larger size, if you hash them, many of the keys will become the same hash, the hash function should be such that that should not be common at all and not only that people should not be able to find that in case they want to and so, that see the 2 properties one is that hiding property or what is called this is the hiding property.

We talked about before that given a hash, you should not be able to find what the original was and then coalition resistance that is, you cannot create 2 messages yourself whose hashes are going to be colliding.

(Refer Slide Time: 49:27)



blockchain transactions

send pubkey	send addr	recv addr	amt	digital signature
0xFFA1288...	0x18471C...	0x13831...	100	0xFAD10A8DC...
...	...	...	...	...
...	...	...	...	...

a/c numbers are hashes of public keys

- from now on, we will call a/c nos. as addresses

public key is included in the transaction

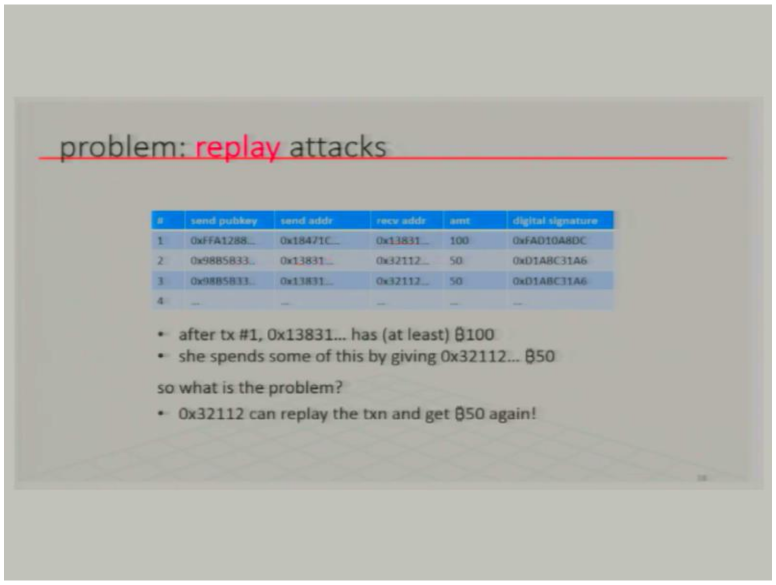
- question: why not just use public keys as addresses?

So, now, what you do is that a transaction will have sender's public key senders address, receiver address amount and a digital signature. So, the address is the hash of the public key. For receiver you only need the hash of the public key as the account number because that is the account number. Whereas for the sender, you need to also tell the public key because otherwise how would you use their digital signature.

So, this is account numbers are also called addresses in line with how Bitcoin calls them. So in Bitcoin also, the addresses of each person is actually a hash of a public key. So each person who wants to participate in the Bitcoin ecosystem, they actually create this public private key pair, keep the private key, the public key is hashed. And then that is announced as the address of the account. And that is the also called a wallet. So that is the wallet address.

So public key is included in the transaction. And you can check whether the public key is genuine by checking the sender's address by hashing the public key. So the question is, and we already discussed this, that why are not we using public keys as addresses because public keys are too long and too cumbersome to deal with.

(Refer Slide Time: 50:52)



problem: replay attacks

#	send pubkey	send addr	recv addr	amt	digital signature
1	0xFFA1288...	0x18471C...	0x13831...	100	0xFAD10A8DC
2	0x98B5833...	0x13831...	0x32112...	50	0xD1A8C31A6
3	0x98B5833...	0x13831...	0x32112...	50	0xD1A8C31A6
4	...	...	...	...	...

- after tx #1, 0x13831... has (at least) Ƀ100
- she spends some of this by giving 0x32112... Ƀ50

so what is the problem?

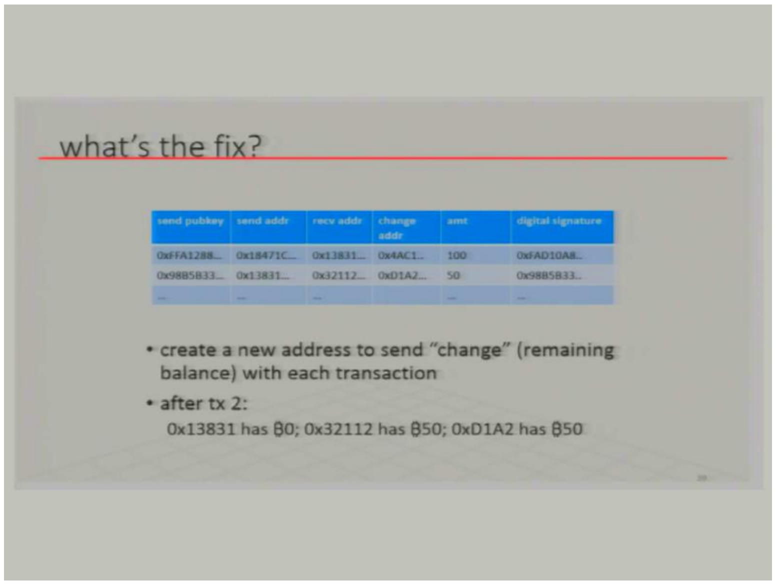
- 0x32112 can replay the txn and get Ƀ50 again!

So now one problem here that can happen is called a replay attack. So let us say this sender sending some 100 amount to this address. And then this sender. Now out of this 100 sends an software transaction one, this address has 100. And then to buy something she uses 50. Now, somebody who is listening to this transaction actually can actually change this and for example this this person can broadcast these transactions in transactions that broadcast.

Because everybody needs to see all the transactions because everybody is keeping track of the transactions into blocks and so on. So the this person then broadcast another message with the same with all the same thing like the public key of the sender, the sender's address, his or her own address, and under 50 and nothing changes because message integrity cannot be sending the same thing again. So there is no message integrity problem.

So now, if others believe that this is another 50 coin transaction, then at the end of that this address will now have 0, although, she intended to actually spend only 50. So that is somebody replaying an old transaction again. So, this is one problem that one has to worry about. So therefore, what you do is you have to make some tricky solution.

(Refer Slide Time: 52:32)



what's the fix?

send pubkey	send addr	recv addr	change addr	amt	digital signature
0xFFA1288...	0x18471C...	0x13831...	0x4AC1...	100	0xFAD10A8...
0x98B5B33...	0x13831...	0x32112...	0xD1A2...	50	0x98B5B33...
...	...	...		...	...

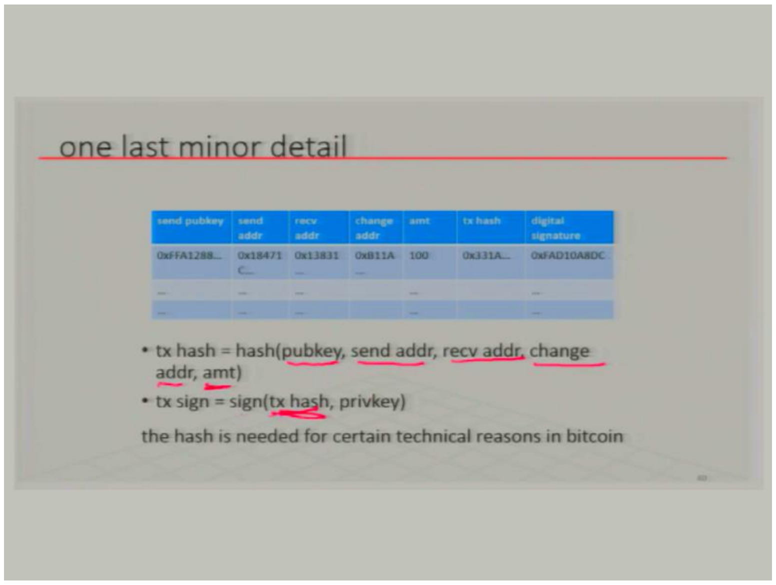
- create a new address to send "change" (remaining balance) with each transaction
- after tx 2:
  0x13831 has Ƀ0; 0x32112 has Ƀ50; 0xD1A2 has Ƀ50

So every time you make a transaction in this kind of setup, you did not keep the balance or whatever the rest of the balance in the same account. So you create another account with a different address. And then the part of the transaction you say, that give 50 to this address, and give the rest of the other like, in this case, 50 to a new address. So now the original address no longer has 50 left original address will have 0 left.

Whereas, the receiver of the transaction will get 50 and rest of the 50 will come to my own a different address or to call an alias address. So, therefore, replaying the transaction will not help because when somebody tries to replay this transaction, the sending address no longer has any money. So, this is something that is very common, I mean, that is done in Bitcoin that every time you make a transaction.

You broadcast a transaction, if it is the full amount that is your only 100 and you are giving out 100, then that is fine, then in that case, you did not have to do anything. But if you are giving part of the, what you have, like in this case 50 out of 100, then the rest 50 has to go to a new address, it cannot remain in the old address because replay attack can happen. So this is something important to understand.

(Refer Slide Time: 53:59)



one last minor detail

send pubkey	send addr	recv addr	change addr	amt	tx hash	digital signature
0xFFA1288...	0x18471 C...	0x13831 ...	0xB11A ...	100	0x331A...	0xFAD10A8DC
...	...	...		...		...
...	...	...		...		...

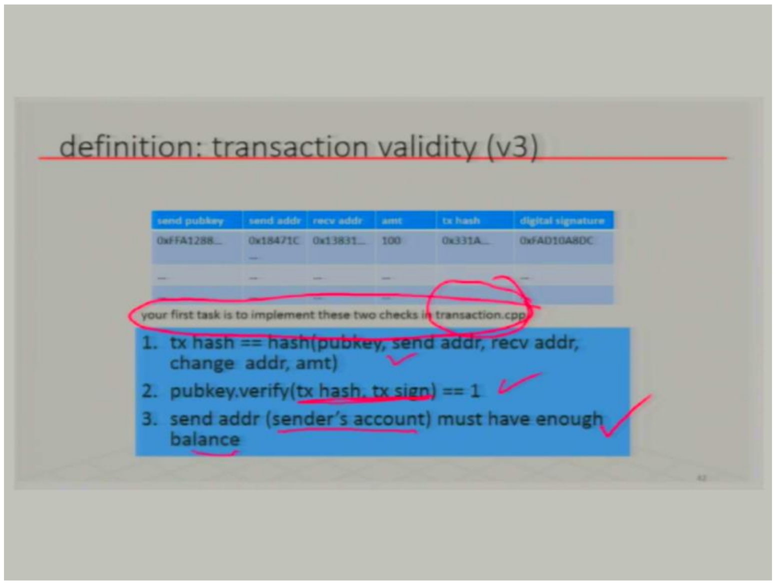
- tx hash = hash(pubkey, send addr, recv addr, change addr, amt)
- tx sign = sign(tx hash, privkey)

the hash is needed for certain technical reasons in bitcoin

So one last minor detail for every transaction, we also add the transaction hash. Now remember the transaction contains the public key, the sender address the receiver address, the change address that is the new address to which you want to send the change left after the transaction and the amount of the transaction. So this is hashed. And then this hash can prove that your transaction is not tampered with.

And then you sign the transaction on the transaction hash rather than the entire transaction. Of course, that makes the signing easier because the entire transaction has a lot of information. And then if you create a digest, and then encrypt the digest with your private key that will be faster. So therefore it makes sense also to do the hash before you do the transaction signing.

(Refer Slide Time: 54:57)



definition: transaction validity (v3)

send pubkey	send addr	recv addr	amt	tx hash	digital signature
0xFFA1288…	0x18471C…	0x13831…	100	0x331A…	0xFAD10A8DC
…	…	…	…		…
	…	…	…		

your first task is to implement these two checks in transaction.cpp

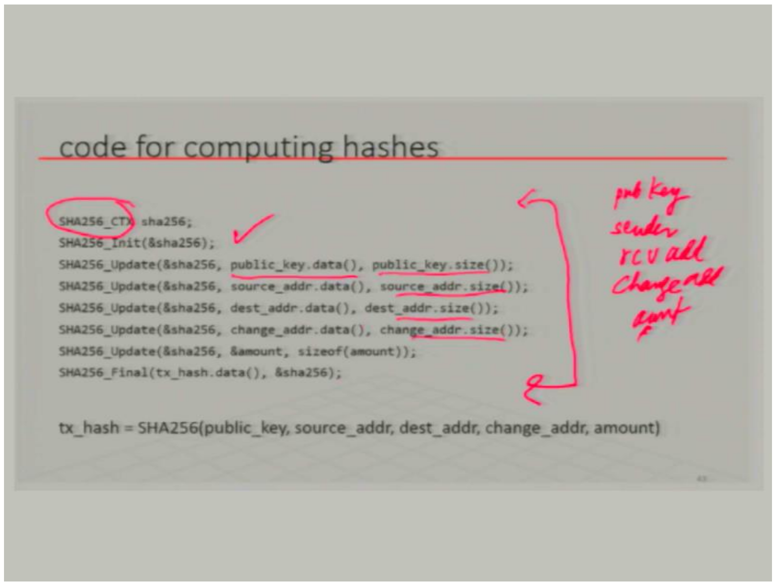
1. tx hash == hash(pubkey, send addr, recv addr, change addr, amt) ✓
2. pubkey.verify(tx hash, tx sign) == 1 ✓
3. send addr (sender's account) must have enough balance ✓

So the final transaction structure will then be a public key, source address, destination address, change address, amount, transaction hash and transaction signature. Remember that source address is result of a hash. Similarly, destination is also a result of a hash. And change addresses result of a hash all addresses are hash of the public key. But you also need to give the public key so you have to use the public key as a vector.

And then at the end, you will actually add another vector of the signature of the transaction. So this is basically c it is a sequence of bytes. Because it is a 8 bit unsigned integer where they are basically byte so you represent the public key as a sequence of bytes. Similarly, use represent the signature also as a sequence of bytes. So now we have a third definition of transaction validity. First, the transaction hash is there then you should be able to verify the signature on the hash. And then your sender's account must have enough balance.

So this we already had and we also had this signature thing, but the new thing is that we have to take the hash of the transaction and then verify the signature on the hash in the exercise that we are proposing. So we will give you a set of already almost implemented blockchain as part of today's discussion. However, the there are places where you have to fill the gap, otherwise, this will not work. So there is this function, this file called transaction dot cpp, where you have to implement these checks, write the checks of the signature verification, and take up the balance of being enough.

(Refer Slide Time: 56:49)



code for computing hashes

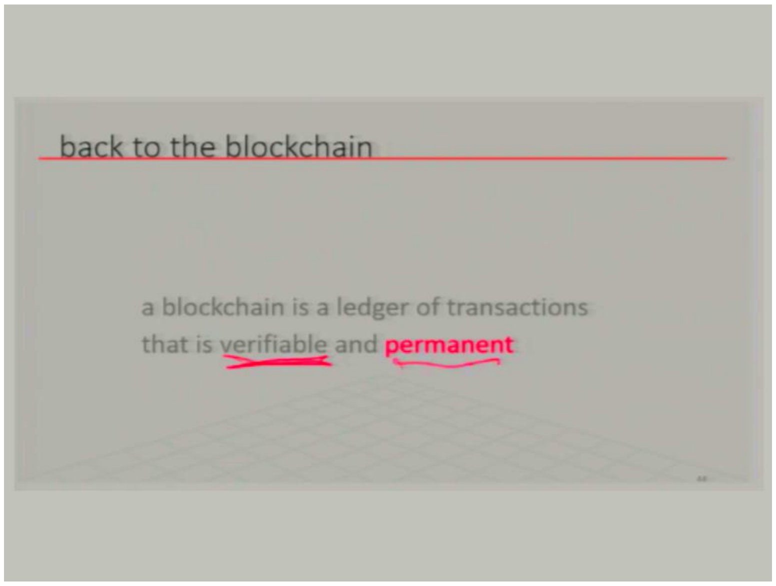
```
SHA256_CTX sha256;
SHA256_Init(&sha256);
SHA256_Update(&sha256, public_key.data(), public_key.size());
SHA256_Update(&sha256, source_addr.data(), source_addr.size());
SHA256_Update(&sha256, dest_addr.data(), dest_addr.size());
SHA256_Update(&sha256, change_addr.data(), change_addr.size());
SHA256_Update(&sha256, &amount, sizeof(amount));
SHA256_Final(tx_hash.data(), &sha256);
```

tx_hash = SHA256(public_key, source_addr, dest_addr, change_addr, amount)

pub key
sender
rcv add
change add
amt

So this is the code for computing hashes. So this is a little bit airy, but this is the type of the SHA 256 hashing. So you initialize it and then you keep adding, what is the next piece of data that you want to add into the hash your data structure has this public key, you have the sender address, you have the receiver address, you have the change address, you have the amount and you have the signature, while signature is not part of the hash. So, what you are doing is that you are telling the size also and then you are adding one by one and updating the SHA hashing process. And eventually what you get is the transaction hash.

(Refer Slide Time: 57:37)

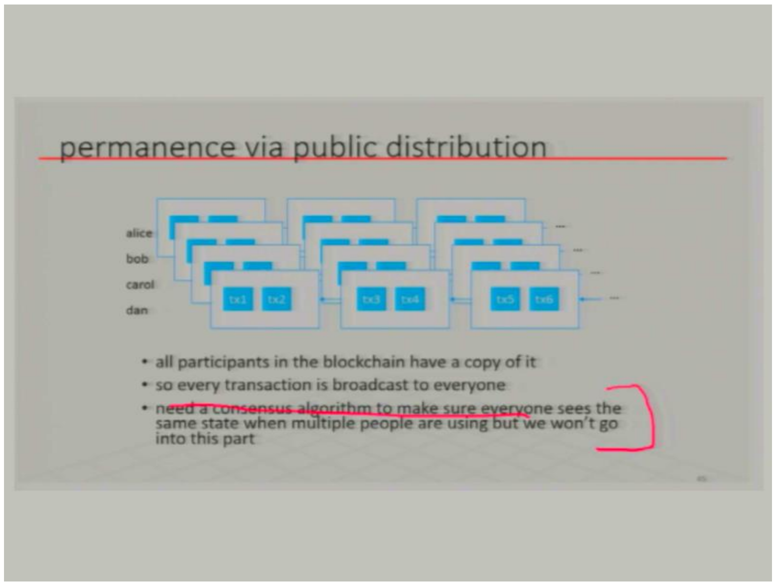


back to the blockchain

a blockchain is a ledger of transactions
that is verifiable and **permanent**

Back to the discussion on blockchain. Again, that verification part is done the question of permanence.

(Refer Slide Time: 57:45)



permanence via public distribution

alice
bob
carol
dan
tx1
tx2
tx3
tx4
tx5
tx6

- all participants in the blockchain have a copy of it
- so every transaction is broadcast to everyone
- need a consensus algorithm to make sure everyone sees the same state when multiple people are using but we won't go into this part

So, now, one way to have the permanence is that it is a highly tolerant to failure of storage. So, for example, if the transactions are all stored in 1 central location, and that location fails, then you are going to have you have lost the data? But blockchain is, one thing that happens is that chain is, as I said, is a dynamic structure. It keeps growing. But at any point, everybody has a copy of the current blockchain.

Now, they may differ slightly from each other because as I said before, because of network delay, and everybody is far from each other, my copy may be slightly behind your copy. I have not gotten the latest block added to my copy yet, but we have an eventual consistency that eventually everybody will have the same blockchain. So all participants having a copy of it. So we have the permanence in the sense.

That if I lose 1 copy, or 3 copies, I still have a lot of copies around. Second is that every transaction is broadcast to everyone. And that is how everybody independently creates the chain. So this is an interesting thing is that nobody is telling them what goes into the blockchain what is the next block, when a next block is created, multiple people will create next block. And I discussed before that this multiple people created next block is actually going to be slightly different from each other.

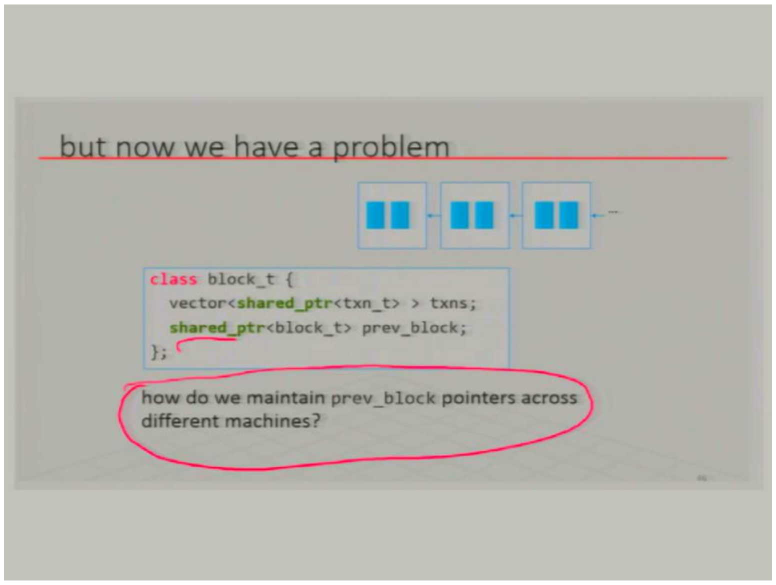
Because the last transactions they have seen their choices of transactions that they put into the block, all that stuff will define what each individual creates this block. Then they compete once they compete. And then some one of them will solve the hash puzzle first. And then he will declare himself winner and then he will broadcast the block with the proof of his win to everybody. So everybody is now getting a copy of the block that has one.

So even though I have I have been sitting there, myself and I have created my own block also but unfortunately, I did not win. So once a winner block is broadcast to everybody, adds that copy of that block to the latest blockchain. So that is how we distributed, eventually everybody has the same chain. Now there are some race conditions and some periods of time when multiple copies may differ from each other.

But we will see when we discuss Bitcoin blockchain, that eventually everything becomes convergent again. So there is a, transience that may actually make some difference, but overall, it works. But in our case, actually, that is not a problem because we have only one copy of the blockchain that is being maintained by our program, but that is not real situation. So what I described as you know, this block, being winning, and then broadcast everybody, this winning process is called a consensus process.

And so there has to be a consensus algorithm. So what the hash puzzle solving is the consensus process, that for is followed by Bitcoin and aetherium, and few other blockchains also, for example, whereas many other block chains, there are other consensus algorithms, and we will see them like Byzantine fault tolerant consensus algorithms of various flavors. So, consensus algorithm is not necessarily always by hash puzzle solving. But the first few blockchains that we will see that is aetherium and Bitcoin will see that the consensus algorithm is indeed based on hash puzzle solving and therefore, based on mining.

(Refer Slide Time: 01:01:57)



but now we have a problem

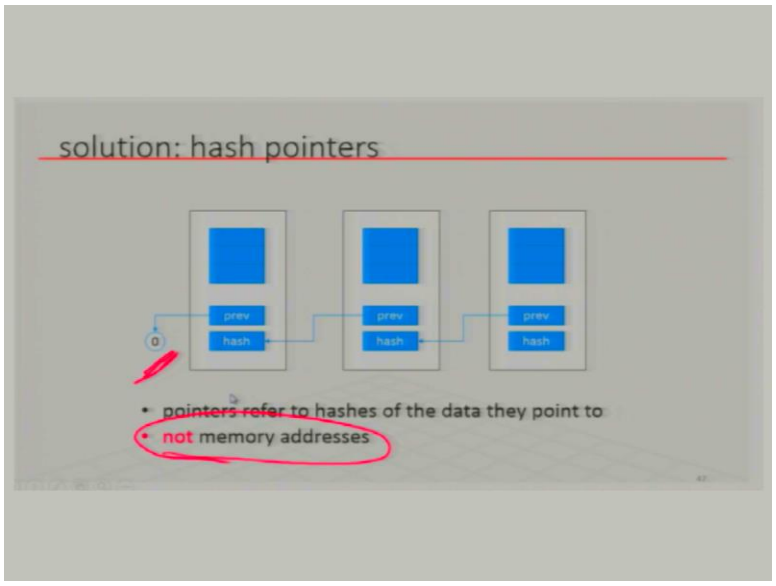
```
class block_t {
  vector<shared_ptr<txn_t> > txns;
  shared_ptr<block_t> prev_block;
};
```

how do we maintain prev_block pointers across different machines?

Now, as at this, one, this is where we actually are having an advantage in our particular example programmatic example is that we are only keeping one block copy of the chain in one within one program. So, we are able to create like shared pointers, which are memory addresses and so on. But in case of real blockchain we have the copies all across and in there on different systems they might be on Linux system, they might be on Windows system and so on.

So, therefore, the pointer sizes will be different based on 64 bit machine 32 bit machine and so on. So, therefore, this kind of pointers are not what is used in the real blockchain.

(Refer Slide Time: 01:02:46)



solution: hash pointers

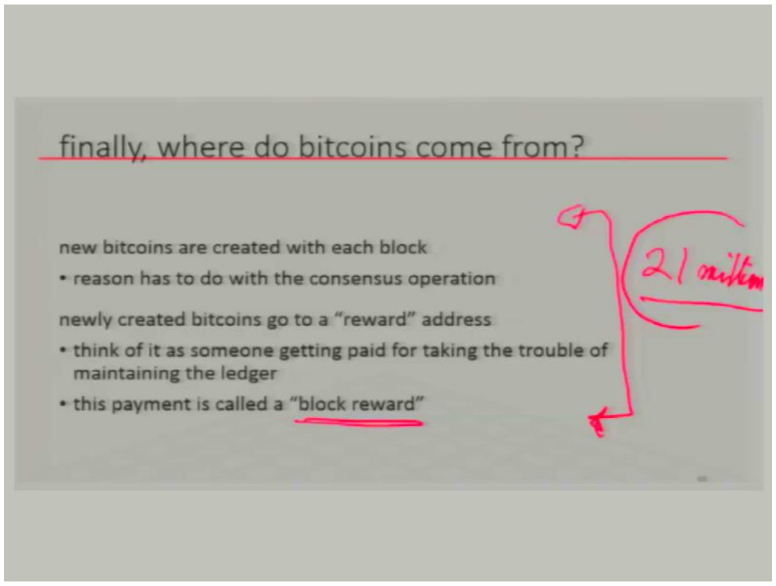
- pointers refer to hashes of the data they point to
- not memory addresses

So, the previous pointer actually across different machines work slightly differently. So, the previous pointer in this case refers to the hash of the previous block. So, as I said in a previous lectures also, that, that is enough actually if you think about it that I have many blocks, I have the hash of some block, which I considered my previous block. So all I have to do to check which one is my previous block, I have to go through all the blocks and compute their hashes.

And whichever guy's hash matches with the hash that I am carrying as my pointer is my previous blocks. So this way I can keep building the, the previous to previous and so on. And I can actually cache it. So at some point, I will not want to, every time I want to find who is my previous block, I did not want to do all the hashing and computation, I might because once the blocks have become permanent, their hashes also become permanent, so I can actually cache that information.

But the point here is that the having the hash of previous block in yours it is stored inside your block is good enough for finding the previous block and that is how the chain of blocks happen and the first block or the genesis block does not point to anybody before it because there is none before. In the real blockchain, they are not memory addresses. They are actually hashes and they are called hash pointers.

(Refer Slide Time: 01:04:11)



finally, where do bitcoins come from?

new bitcoins are created with each block
- reason has to do with the consensus operation

newly created bitcoins go to a "reward" address
- think of it as someone getting paid for taking the trouble of maintaining the ledger
- this payment is called a "block reward"

The last thing that where do the bitcoins come from so the bitcoins are created whenever a block is mined, that is when a block is created, everybody tries to solve the consensus problem and can discuss in case of Bitcoin hash puzzle, and then as a reward of the winner with a winner is given some new bitcoins and that only way Bitcoin is created. So the reason why you were not is there is that because otherwise, why would I spend my computing resources, which requests money to validate transactions to put them in a block.

And then solve a hash puzzle to be part of the blockchain? I need some reason to do it. So there has to be an incentive. So this incentive or reward actually makes the system keep going otherwise everybody will stop mining because it is expensive to mine. So, the reward should be also enough so that the amount of money they spend in paying for electricity bill and paying for the hardware and paying for the cooling of the very large hardware that they use to solve the hash puzzles should be surpassed by the reward.

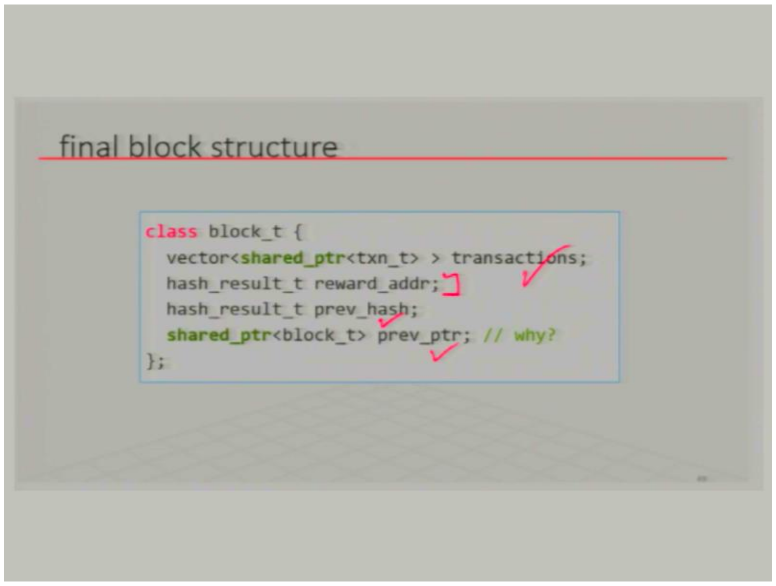
So the reward should be big enough and this is called a block reward. So the block reward initially in Bitcoin was 50 bitcoins, then after 4 years, it became 25. And after 4 years, it has now become 12.5. And in another couple of years, it will become 6.75 and so on, and at the end, it will become 0. And the reason is that the Bitcoin ecosystem also has this thing that there will never be more than 21 million bitcoins unless the Bitcoin changes its strategy and policy and everything.

So and the only way to create bitcoins is through the block rewards. So as we, if we keep the block reward the same, always, then the, then you have to start, you know, past the 21 million mark. So therefore, they are tapering it off, it is like kind of like a geometric distribution so that the eventual sum becomes 21 million. So then you might ask, why would, somebody be interested in modeling, mining, spending, you know, money to do the mining solving hash puzzles.

So there is something also that every transaction that you make, you are also has to give some transaction fee. So in the beginning, the transaction fee was in there, and we have not included in our example here. But if the transaction fees there, which means that whoever wins them hash puzzle can keep the can transfer the transaction fee from each transaction to his account. So therefore the, if there, you know, 100 transactions in a block, and each gives.

Let us say, some small amount of transaction fees, then together, they will become a pretty good amount, and therefore, the miners will still have incentive to gain new bitcoins through this process even though the rewards have stopped. So this is how this thing is planned to happen.

(Refer Slide Time: 01:07:34)

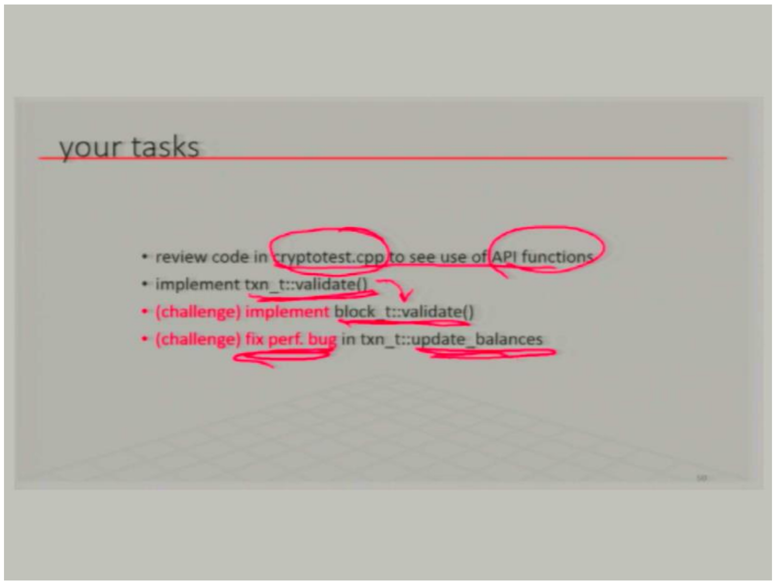


final block structure

```
class block_t {
  vector<shared_ptr<txn_t> > transactions;
  hash_result_t reward_addr;
  hash_result_t prev_hash;
  shared_ptr<block_t> prev_ptr; // why?
};
```

So the final block structure is that you need to add also the reward address. Because whenever there is a reward, in the block we should know where to send the reward. Rest is the same as before. It is a trust list of transactions, previous hash, and in our case, it is a previous question. Also because we are doing it inside the program otherwise that party's not necessary to be there.

(Refer Slide Time: 01:08:08)



your tasks

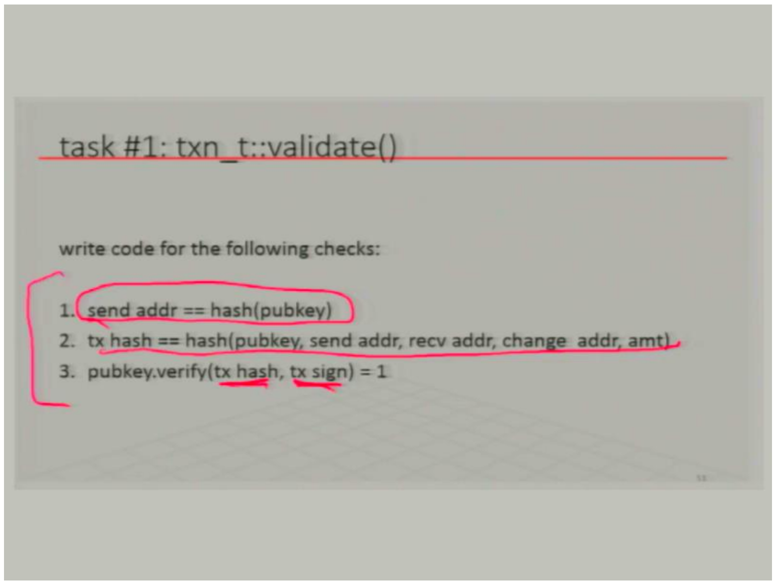
- review code in cryptotest.cpp to see use of API functions
- implement txn_t::validate()
- (challenge) implement block_t::validate()
- (challenge) fix perf. bug in txn_t::update_balances

So, now coming to the, your optional exercise based on this particular lecture. So, first of all, we will give you a zip file containing the code and it will have some instructions you have to actually use ubuntu linux to do this. So, you if you did not have ubuntu linux, you can actually use virtual box which you can download from oracle and virtual box is actually a virtualization software and then you can download an Ubuntu Linux which is free.

And then you can have a built a virtual machine using Virtual box which runs Ubuntu 18.4 or letter, because we will also will need you to get some of the libraries that are not usually coming with the Linux distribution. And those libraries are working only with the latest Ubuntu. And then you look at the code in this file crypto test dot cpp to see how the various API functions from this libraries that that you have to use for example, for generating keys and for hashing, SHA256 and so on.

And then you have to implement this validate function. So, the final version 3 validate of transaction and then you have to implement the block validation that is how to implement the validation of all the transactions by calling this function actually you can validate and then you can also, there is a performance bug in the code that is given in the update balances function. So that you can find then you will have some good exercise in you know performance debugging.

(Refer Slide Time: 01:10:08)



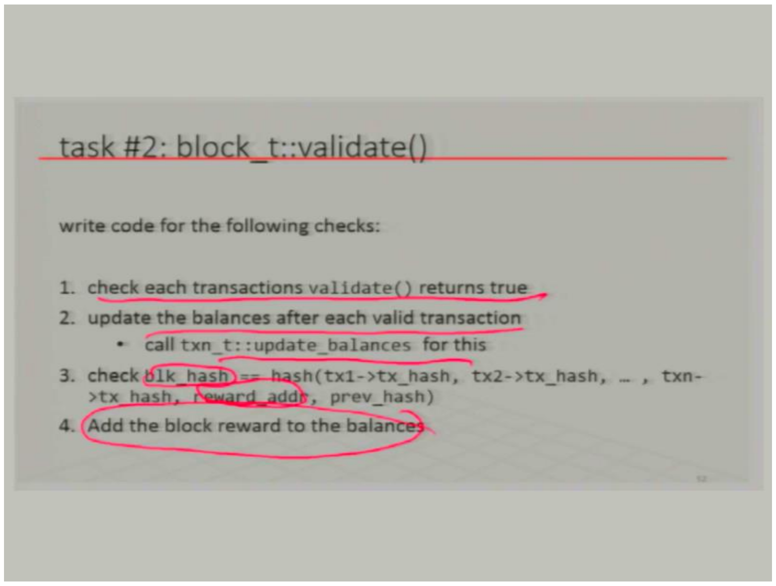
task #1: txn_t::validate()

write code for the following checks:

1. send addr == hash(pubkey)
2. tx hash == hash(pubkey, send addr, recv addr, change addr, amt)
3. pubkey.verify(tx hash, tx sign) = 1

So, for transaction validate, you have to write code for the checking that the send address is the hash of the public key that is part of the transaction also you have to compute the transaction hash and check that the hash that that has been put part of the structure matches and then you will also have to use this transaction hash and check the signature that is there. So, these are the things that you have to check for validating a transaction other than the balance check which actually we are assuming.

(Refer Slide Time: 01:10:45)



task #2: block_t::validate()

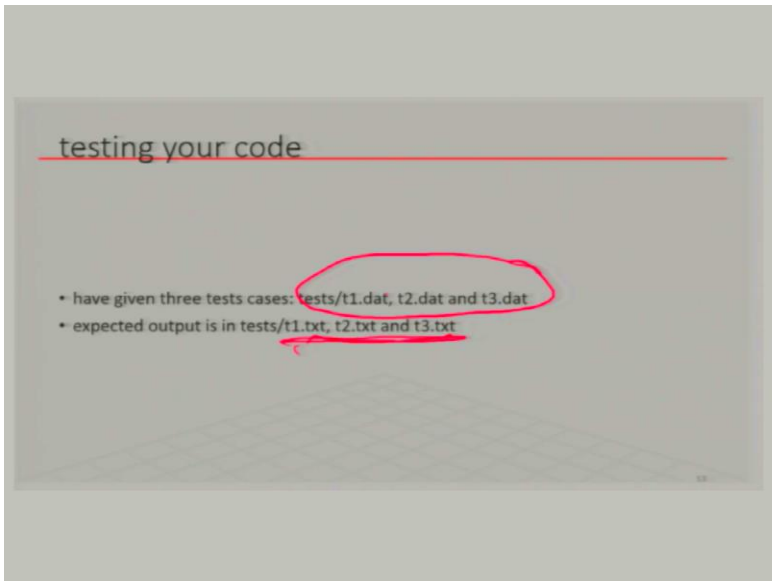
write code for the following checks:

1. check each transactions `validate()` returns true
2. update the balances after each valid transaction
   - call `txn_t::update_balances` for this
3. check `blk_hash == hash(tx1->tx_hash, tx2->tx_hash, … , txn->tx_hash, reward_addr, prev_hash)`
4. Add the block reward to the balances

You will do for block validate, you have to go through the C block is a vector of transactions. So, you have to iterate over the vector and apply the transaction validate to each of those, and if any of them returned false then the block is invalid. If all of them returns true, then the block is valid, then you do update the balances after each valid transaction. So you can call existing update balances for this, then you check the block hash.

And then you add the block reward to the balances, wherever in this case, whatever the reward addresses, you just put it there, in case of a distributed computational situation like in bitcoin, every person who creates a block puts his own addresses the reward address, but eventually only one of them will get reward.

(Refer Slide Time: 01:11:38)

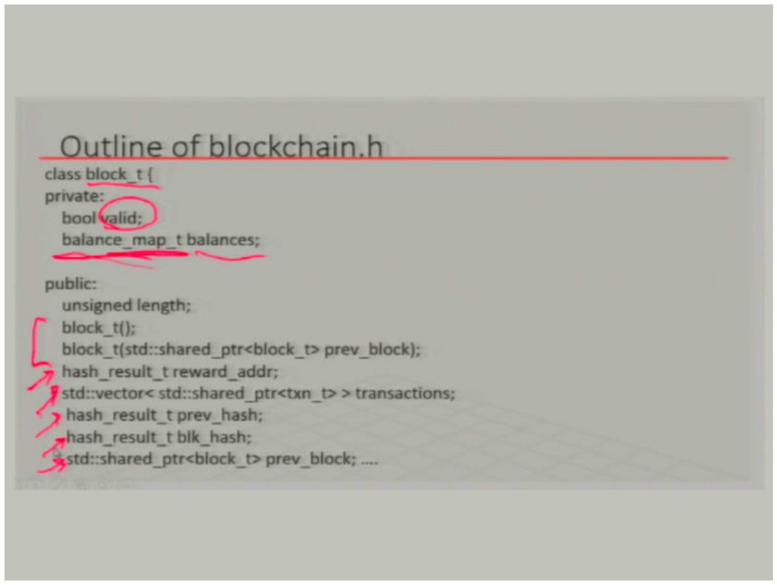


testing your code

- have given three tests cases: tests/t1.dat, t2.dat and t3.dat
- expected output is in tests/t1.txt, t2.txt and t3.txt

Now in the same zip file, you will get a number of test files, number of data files, and if you use them, then you should get the output so you should match them against these outputs by doing a comparison of the this file against the output of this running your program on this data and that will give you a satisfaction of having done it correctly.

(Refer Slide Time: 01:12:02)



Outline of blockchain.h

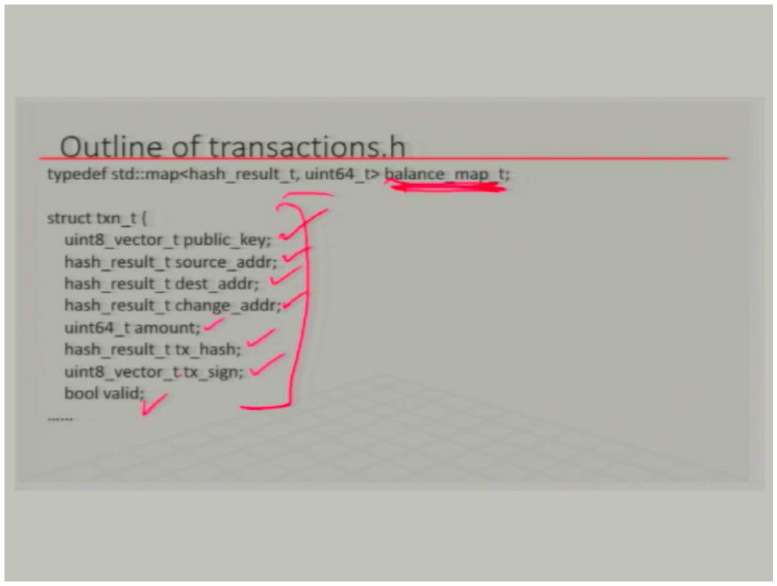
```
class block_t {
private:
  bool valid;
  balance_map_t balances;

public:
  unsigned length;
  block_t();
  block_t(std::shared_ptr<block_t> prev_block);
  hash_result_t reward_addr;
  std::vector< std::shared_ptr<txn_t> > transactions;
  hash_result_t prev_hash;
  hash_result_t blk_hash;
  std::shared_ptr<block_t> prev_block; ....
```

So this shows the outline of the classes. So earlier when we were actually going through this, we had lesser, you know, we only showed the snippets. So this shows a little more like you know, this is the block class whether it is valid or not, after the validation process goes through, it will be set to true or false. And then you have the, remember this is a map, this map is a hash table for balances. And then you have all these functions like these are constructed functions.

This is actually a data item. This is also a data item, which is a set of transactions as a vector, this is the previous hash, the block hash, and then this pointer is there for our program, because our program is based on a single process. Therefore, we can use real pointers.

(Refer Slide Time: 01:12:53)



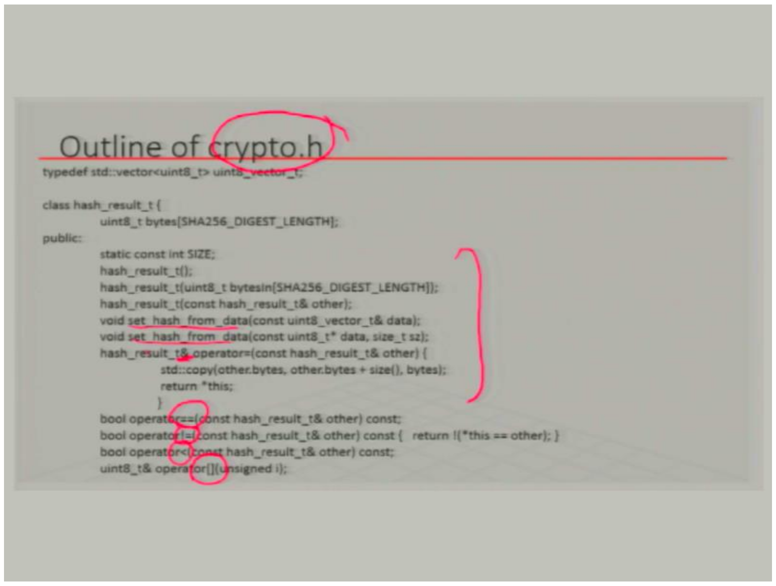
Outline of transactions.h

```
typedef std::map<hash_result_t, uint64_t> balance_map_t;

struct txn_t {
  uint8_vector_t public_key;
  hash_result_t source_addr;
  hash_result_t dest_addr;
  hash_result_t change_addr;
  uint64_t amount;
  hash_result_t tx_hash;
  uint8_vector_t tx_sign;
  bool valid;
......
```

Transactions are also actually having this structure and this balance map hash table type is actually defined in this transactions in the h and this is what are the different parts content of the transaction we already saw have the public key the source address, destination address, change address amount hash sign and with a direct transaction is valid.

(Refer Slide Time: 01:13:20)



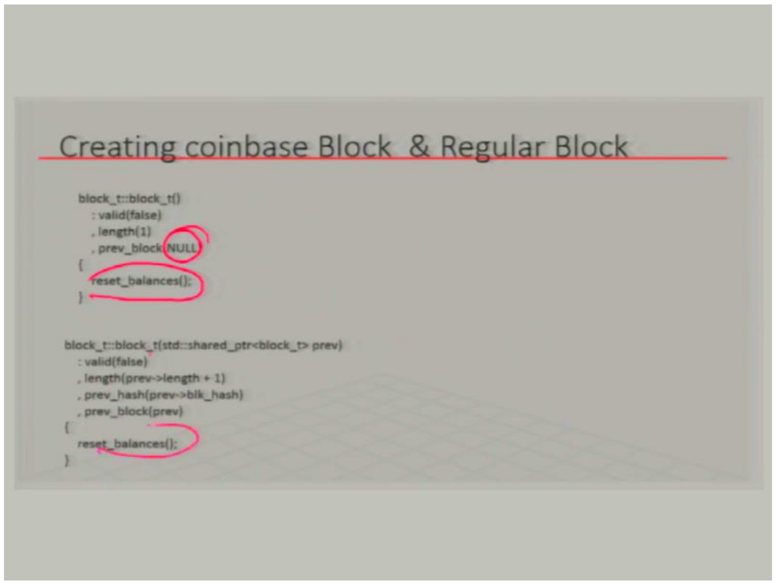
Outline of crypto.h

```
typedef std::vector<uint8_t> uint8_vector_t;

class hash_result_t {
            uint8_t bytes[SHA256_DIGEST_LENGTH];
public:
            static const int SIZE;
            hash_result_t();
            hash_result_t(uint8_t bytesin[SHA256_DIGEST_LENGTH]);
            hash_result_t(const hash_result_t& other);
            void set_hash_from_data(const uint8_vector_t& data);
            void set_hash_from_data(const uint8_t* data, size_t sz);
            hash_result_t& operator=(const hash_result_t& other) {
                        std::copy(other.bytes, other.bytes + size(), bytes);
                        return *this;
                        }
            bool operator==(const hash_result_t& other) const;
            bool operator!=(const hash_result_t& other) const {   return !(*this == other); }
            bool operator<(const hash_result_t& other) const;
            uint8_t& operator[](unsigned i);
```

Hash result is the so, the crypto dot h has prototypes of the crypto functions and the classes in this case the hash result class is defined here and it has all the constructors it has the way to set hash from data and also it has a hash result. This is the operator overloading. This is the competition overloading this is not equal overloading this is less overloading. So, these are the kind of things and this is how you access a particular part of the past results. So, not all of them are required for this particular program, but this is how the hash result classes defined.

(Refer Slide Time: 01:14:08)



Creating coinbase Block & Regular Block

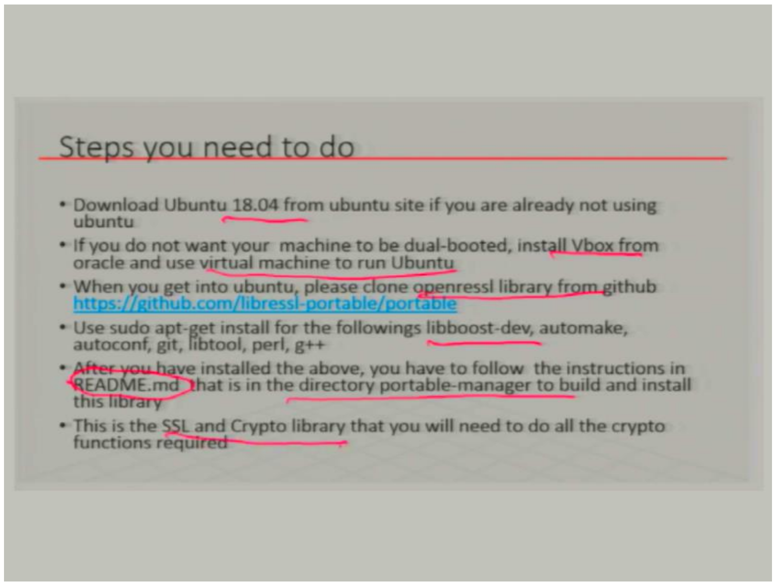
```
block_t::block_t()
    : valid(false)
    , length(1)
    , prev_block(NULL)
{
    reset_balances();
}

block_t::block_t(std::shared_ptr<block_t> prev)
    : valid(false)
    , length(prev->length + 1)
    , prev_hash(prev->blk_hash)
    , prev_block(prev)
{
    reset_balances();
}
```

So, the first block is called also called the coin base block. So, coin base block can be created using this constructor. And when you say that the previous block points to null and it is linked is one it is validities set to false initially, later on if it is validated, then it will be done and in the beginning you reset balances for regular block initially you create you initialize all these different data items like the validity is set to false length is updated.

To the previous length plus one previous hash is the previous blocks hash and previous block is you know the pointer to the previous and you reset balances after that with different functions you keep adding transactions and so on in the block.

(Refer Slide Time: 01:14:58)

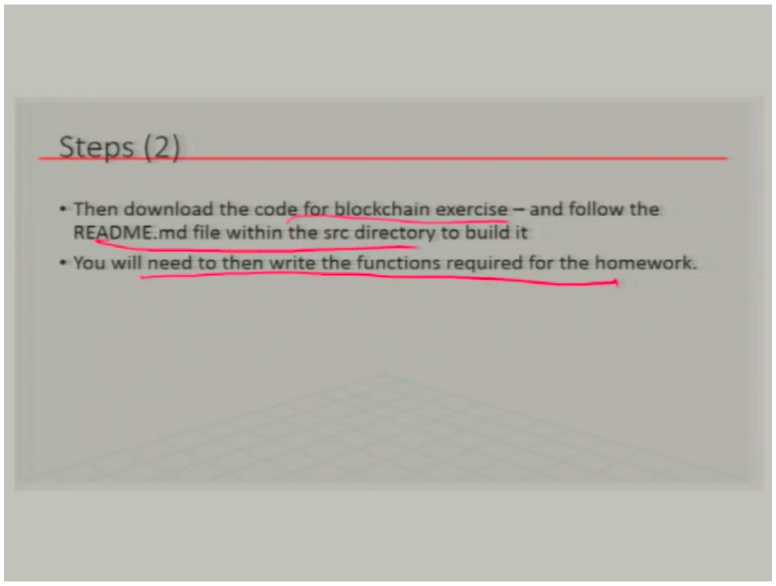


Steps you need to do

- Download Ubuntu 18.04 from ubuntu site if you are already not using ubuntu
- If you do not want your machine to be dual-booted, install Vbox from oracle and use virtual machine to run Ubuntu
- When you get into ubuntu, please clone openressl library from github https://github.com/libressl-portable/portable
- Use sudo apt-get install for the followings libboost-dev, automake, autoconf, git, libtool, perl, g++
- After you have installed the above, you have to follow the instructions in README.md that is in the directory portable-manager to build and install this library
- This is the SSL and Crypto library that you will need to do all the crypto functions required

So, the steps you need to do is you have to download the latest Ubuntu. And if your machine is not dual booted, install Virtual box from Oracle and use virtual machine to run Ubuntu. Then you go to clone the open ressl library from github. And then you do the installation of the various libraries boost library, automatic auto configure, etc. Once you have installed all this, then there is a README.md file, which you have to view to see how to build and install the portable manager library. And this is the where you get all the crypto functions that are required.

(Refer Slide Time: 01:15:42)



Steps (2)

- Then download the code for blockchain exercise – and follow the README.md file within the src directory to build it
- You will need to then write the functions required for the homework.

And then you download the code for blockchain exercise from the sherm platform that we will be putting in there and follow the README file within the sociality to build it. And then and then you write the specific functions that you require in the homework. So the writing the functions is not a lot of work here. But what is happening, what is going to happen is that you have to understand the entire code to actually make the changes, small changes.

And that is where you will get to see the entire code and understand how it works and so on, which is the goal of this exercise. So with that, this lecture 3 comes to an end. So, next time we will actually look at the first real blockchain you know, how it is done. You know, it is, you know, techniques and structure and you know, how the distributed maintenance of it is done for Bitcoin blockchain. And we will go from there thank you.