Introduction to Blockchain Technology and Applications

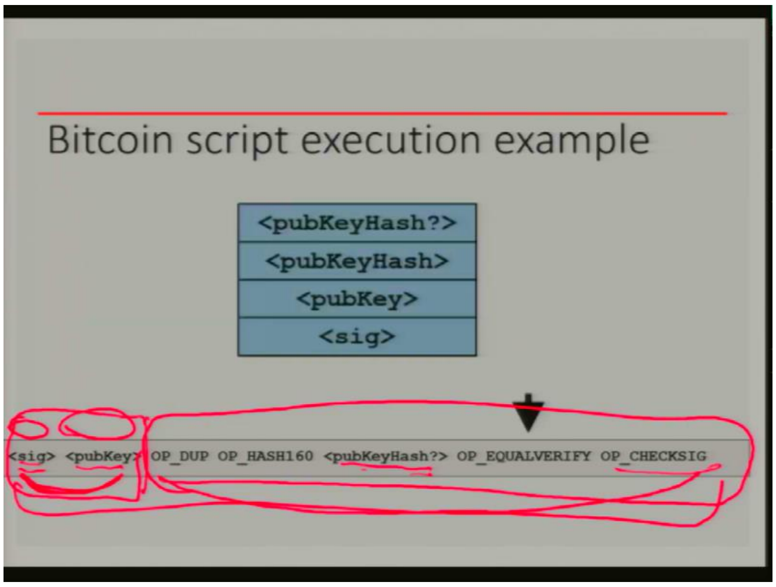
Prof. Sandeep Shukla
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture No. 10
Blockchain Technology and Applications

Welcome back. So we were discussing bitcoin scripting language and we ended by saying that bitcoin scripting language is not turing complete. And what that means is that we cannot express arbitrary functionality with bitcoin scripting language, very, simple things can be only expressed with such a, you know, restricted language. Now, what happened is that we were when the users of bitcoin saw that, the limitations of this.

Then the Ethereum guys came in, and they said that we need a much more expressive language. So they created a new language called solidity. And in athereum, they started using solidity to write this kind of complex functions. And those are called smart contracts. So smart contract, the bitcoin scripts are actually precursors of smart contract.

(Refer Slide Time: 01:13)



Bitcoin script execution example

<pubKeyHash?>
<pubKeyHash>
<pubKey>
<sig>

```
<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG
```

And we, will see in a few within a few slides, that the bitcoin script, even though it is very restricted, still can express a number of things that are very useful and interesting. But before that, let us see what the bitcoin script execution looks like. So let us say I have this after concatenating, the CIG script. And then there is a script pub key, I have this so this part is actually part of the script SIG.

And this is the script pub key and we have concatenated them so this is my complete script and I want to execute this in a stack based machine. So what happens is that the, this now one things that are expressed with the angular brackets are data because this is just an just a prototype example. So we did not have real data here in the actual case, there will be a signature there will be a public key which will be actually data.

And therefore, and this will also be data and register instructions. So, whenever we are seeing these angular brackets, those are data. So as I said that when I encountered data while executing, I push them on top of a stack. So let us look at the stack. So when I encountered stack, I will put that on the stack. Now next, I see that there is another piece of data so I have to put that on top of that. So I will put that also on top of the stack, the stack grows.

Now I encountered an instruction OPT up, so when I see OPT up, what it does is duplicates the top of the stack, so when it duplicates the top of the stack without removing what was on top of the stack, some instructions remove the entity on top of the stack, but the OPT up basically keeps the entity on top of the stack. So it creates another copy of this key and we will understand very soon why it did that.

Now, the next one is another instruction this instruction is saying that hash whatever is on the top of the stack. So, it is going to hash this top of the stack. So, and this one replaces what was on top of the stack. So now we have the hash of the public key on the top of the stack. Now, I encountered another data. So I put it on top of the stack. So I have now this data and this data. Then I have another instruction which says equivalence verify.

So which means that take the top 2 elements of the stack this instruction semantics is that you take the 2 elements on the top of the stack and verify if they are equal. If they are equal, then you will it will return true, which means those 2 things will disappear from the stack, but nothing will be pushed on the stack. So, now, I have done that. So now I have only the public key and the signature.

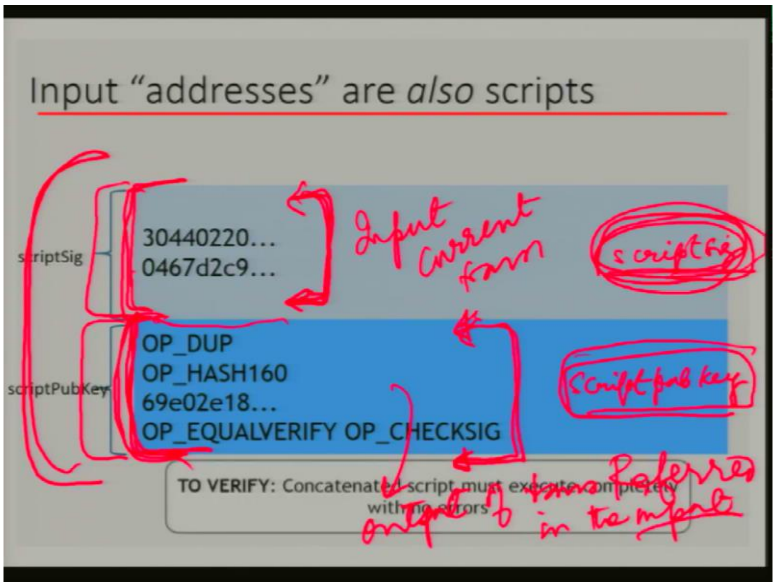
And then I encountered the next instruction which says, check the signature. The signature is done with the private key so if this public key is the correct public key for this private key, then we will be getting a verification otherwise the verification will fail. So if assuming that the verification passes, then it is going to be empty stack or it will be all true. And then I have succeeded in executing this entire script.

Now, let us have a little look at this a little bit more closely. So the guy who is spending the money he does a signature and he gives us public key because without the public key one cannot change, check the signature. So he gives his public key. Now if the users want, they can check from other places, because public key is a public information that this is indeed his public key. So he puts the public key, and he signs with the private key.

So this is the input part of the transaction gives the output part of the transaction is supposed to only give you the public key hash, of the person who is supposed to get the money. What it does is that it I am going to check whether this public key hash matches with the public key of the recipient. So for this, what I am going to do is that I am going to duplicate the public key, and then I will make a hash and I am going to compare it.

And then I am going to check verification of the equality and if they match then we know that this is the correct hash, and then I will check the signature, I have the public key of the signature, so I can check the signature. So, this is how this thing actually make sure that the spender of the money is indeed the correct spender has the authority to spend that money.

(Refer Slide Time: 06:26)



Input "addresses" are *also* scripts

scriptSig

30440220...
0467d2c9...

scriptPubKey

OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

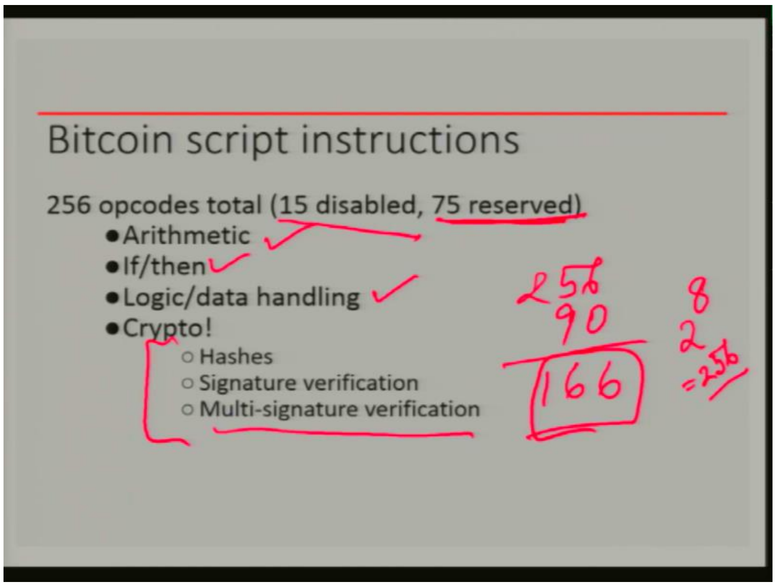
TO VERIFY: Concatenated script must execute completely with no errors

Now, remember, one thing I forgot to tell you is that you actually take the, this part so when you put together this, you take this from your current transaction and this part you take from the previous transaction. Why? Because see, this is the part current transaction in the input part I put this I signed it and this is the in the output part of the previous transaction, which I am trying to use.

So this is not the output of the current transaction. This is the output of the previous transaction. And this is an the transaction which is being used for as input and this part is from the input so in the input I put my signature saying that I am referring to so answer transaction on which so and so output 0 or 1 or 2 whatever is I am going to redeem. So, what this has to do is that it has to take that corresponding outputs from the previous transaction corresponding outputs, output script pub key and then concatenate them together.

So, it is not that the input of this and the output of the current one our input and output of the current transaction are concatenated it is the input part of the current transaction and this part in this part is from the output of the transaction referred in the input. So, that is very important otherwise you will be confused when we when you try to understand this part. So, this is the how the execution happens. So it is completely stack based.

(Refer Slide Time: 08:03)



Bitcoin script instructions

256 opcodes total (15 disabled, 75 reserved)

- Arithmetic ✓
- If/then ✓
- Logic/data handling ✓
- Crypto!
  - Hashes
  - Signature verification
  - Multi-signature verification

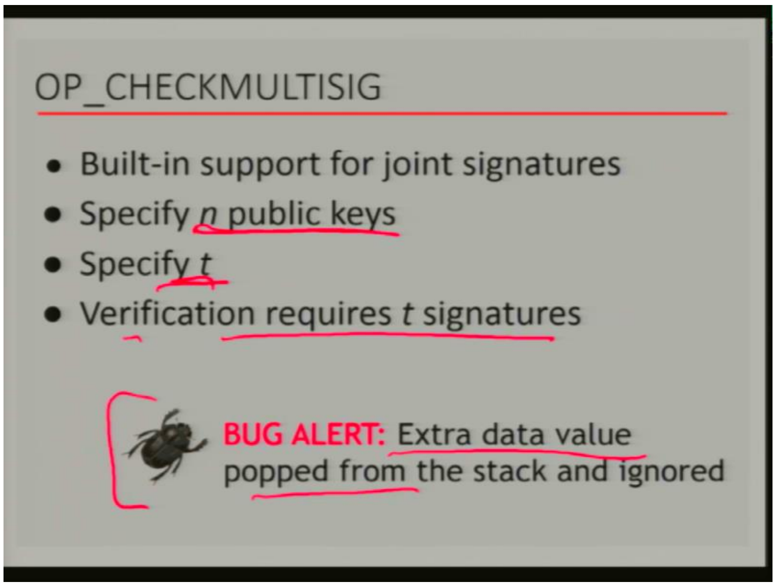
$$\begin{array}{r} 256 \\ 90 \\ \hline \boxed{166} \end{array}$$

$2^8 = 256$

And the script bitcoin script instructions 8 bits long. And so there are 2 to the 8 that is 256 different possible opcodes. But now there are 15. Of course, they are not used, they are disabled, and 75 are reserved, but those opcodes have not given any functionality yet. So, out of 256, you subtract 90, so you have 166, possible opcodes. So some of the opcodes are just arithmetic. In our example, it did not see any arithmetic.

Also some if then some logical data handling and crypto functions like hash computing hash, verifying signature, and verifying multiple signature, things like that. So that is what the very, simple, and small language.

(Refer Slide Time: 08:55)



OP_CHECKMULTISIG

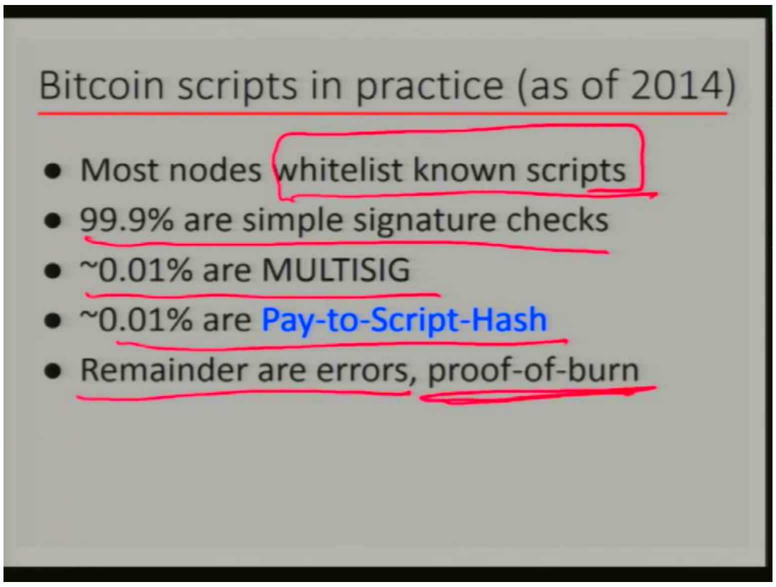
- Built-in support for joint signatures
- Specify *n* public keys
- Specify *t*
- Verification requires *t* signatures

BUG ALERT: Extra data value popped from the stack and ignored

So multi SIG is an important opcode so, what it does is that it specifies n public keys and specify a number T and if you want to verify joint signatures, you record that t of them should verify. So, not all in necessarily have to be signing it to verify it but at least 80 of them should verify. Now if you specify TSN then you would require all the in entities to sign the transaction for the transaction to be valid.

Now, there is a bug which was there and in the beginning that there was an extra data value that is popped from the stack and it is ignored when the up multi SIG is executed. Remember, each opcode when we when I get the opcode I often get the top of the stack or 2 toppings on the stack are 3 toppings on the stack and use it for executing the instruction. But in this case, they unnecessarily pop another extra value. And therefore, when you write your script, you have to add that repeat that data value that that gets unnecessarily popped, which is an annoyance, but it can be done.

(Refer Slide Time: 10:22)



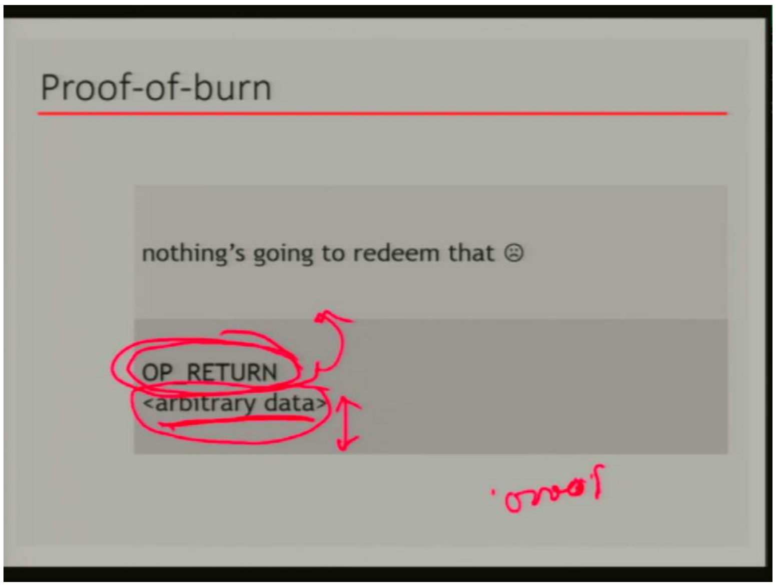
Bitcoin scripts in practice (as of 2014)

- Most nodes whitelist known scripts
- 99.9% are simple signature checks
- ~0.01% are MULTISIG
- ~0.01% are Pay-to-Script-Hash
- Remainder are errors, proof-of-burn

So in practice, most nodes whitelist known scripts with 160 or so opcodes, you can write many, many scripts, some of them actually was probably meaningless. But you can write a lot of scripts. Some of the scripts are maybe malicious doing something that you would not want to do and you did not know that it is malicious. So most nodes what they do is that they whitelist a number of scripts.

And if the script if any script defaults from their whitelisted set of scripts, then they actually did not even validate the transaction and declared the transaction to be invalid 99.9% scripts are actually simple signature check the one that we saw. And we executed on the slides. And about 1 in 100 is MULTISIG script. And there is another thing called pay to script hash that we will discuss shortly. That is another part another type of script that we see about 100. And then remainders or errors and something called proof of burn, so we will also see what that means.

(Refer Slide Time: 11:31)



Proof-of-burn

nothing's going to redeem that ☹

OP_RETURN
<arbitrary data>

So proof of burn is an instruction. So what is a burn? Burn means that you are burning money. So if you take some bitcoin as inputs, and then you say that, but you did not specify who gets it, but the script executes, then that bitcoin becomes irrelevant, irredeemable which means that you cannot redeem it ever again. Because once you have used it as input, you cannot use it as input to another currency.

On the other hand in this transaction, you did not create an output. So, you cannot refer to that output to do the next transaction. So that that bitcoin goes, but you know nowhere so, nobody can use it. So, we say that we have burned that that amount of bitcoin. So, why would I need to do that? The reason is that the way the proof of burn happens is that you there is an opcode called op return which returns without error.

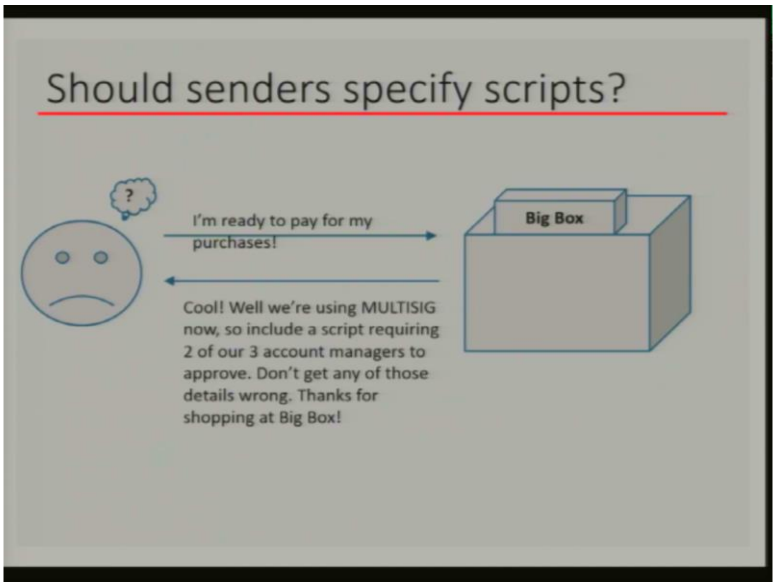
Now, we said that if when I execute the script on the stack based machine, then if I have no errors, then we assume that the transaction is valid. So, in this case, the input goes away, but the output is not gone is not going to any anybody. So, therefore, the money is gone. But what happens is that operator is such that anything written after the operator is ignored, because operated basically returns the script from execution.

So, you can arbitrary data after the operator and then the transaction will be valid and then this transaction will be eventually go into a block and will become permanent in the blockchain. So, if you want to keep some data forever into the blockchain then you use a proof of burn you pay a very small amount of bitcoin maybe 0.00001 amount of bitcoin burn it and then basically h the data forever in one of the blocks.

So, many times you will hear that people are nowadays giving like your course certificate on the on bitcoin blockchain, this is how this is done. This data about your certificate will be actually etched into a transaction using a proof of burn transaction and then it will be in some block and so, if somebody wants to in future wants to check if you have the certificate, they can actually go into the bitcoin blockchain and retrieve that information.

And since this is supposed to be non temporal and supposed to be permanent, then this is a proof that you really got the certificate and you did not forge the certificate. So that is how this is done.

(Refer Slide Time: 14:17)



Should senders specify scripts?

Cool! Well we're using MULTISIG now, so include a script requiring 2 of our 3 account managers to approve. Don't get any of those details wrong. Thanks for shopping at Big Box!

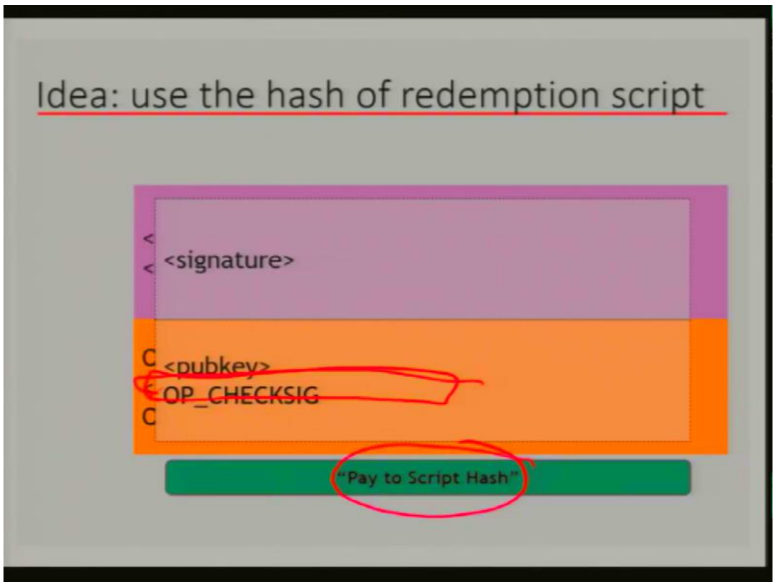
So, there is another issue that we saw that the script that we saw as the script pub key was pretty long and not very long, but it was long and it has multiple instructions and data values in the places and so on. Now, suppose you are working with the merchant and you buy stuff from him. And then one day he says that look, we have changed ownership and we now have multiple owners.

Now we actually need to sign 3 of us need to sign before so you have to send the money to a multi SIG address. Now you get confused because you are the sender. And you are used to just regular writing scripts, which has basically looks like what we just saw is the check, SIG, type of thing. When it is multi SIG, you did not know. So you will be confused. And if you make a mistake in writing the script, then that is the money cannot be redeemed by the merchant.

So what the merchant can do is that he can give you a script. But he the way he does it is that he writes a script and puts us puts a hash pointer to that script. So when you send the money, you did not have to write a long script, all you have to do is to actually send the money to that hash. So this is a technique that was introduced later in blockchain. It was not there in the beginning. And this is called a pay to script hash type of payments.

So instead of paying it to a regular address, person's address, you are sending it to a hash value, which looks like a regular address. So you from the sender's point of view, there is no change. But from the perspective of the other side, you actually have a much more complex script that you are hiding from the sender.

(Refer Slide Time: 16:17)



Idea: use the hash of redemption script

<signature>

<pubkey>
OP_CHECKSIG

"Pay to Script Hash"

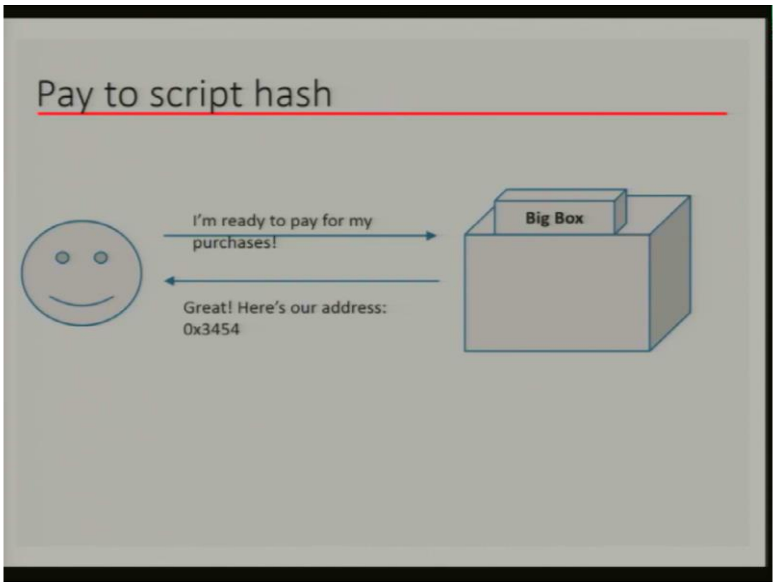
So what you do is that you put the hash of the redemption script, and then you check the redemption script. And if the hash matches, So, this part checks whether the redemption script to which you are sending is indeed the redemption script that you meant, this is all you have to write, like you did in case of a regular address. So, it could have been a regular address and you would have written the same thing.

You would have written the OP hash and you would have written the hash of the redemption script and then equal and then so then you do the pop key, then signature check and all that stuff, which was you would have done anyway. So, but what happens is that after you have done this, the actual script is replaced and that that script needs to be executed before the actual, the entire script finishes with, success.

So that is how, this works. And this is called a pay to script hash. So the basic idea is that you actually have to the receiver has a complex business situation where he needs the payment to be made to go through a complex script. But you did not want the burden of writing a complex script. So he makes it easier for you by writing a script and tell you the hash of the script.

And then you basically use the hash of the script as if you are sending to a regular address, which also is a hash of once, you know, size 160. So that is all there, to this pay to script hash.

(Refer Slide Time: 17:59)



Pay to script hash

I'm ready to pay for my purchases!
Great! Here's our address: 0x3454
Big Box

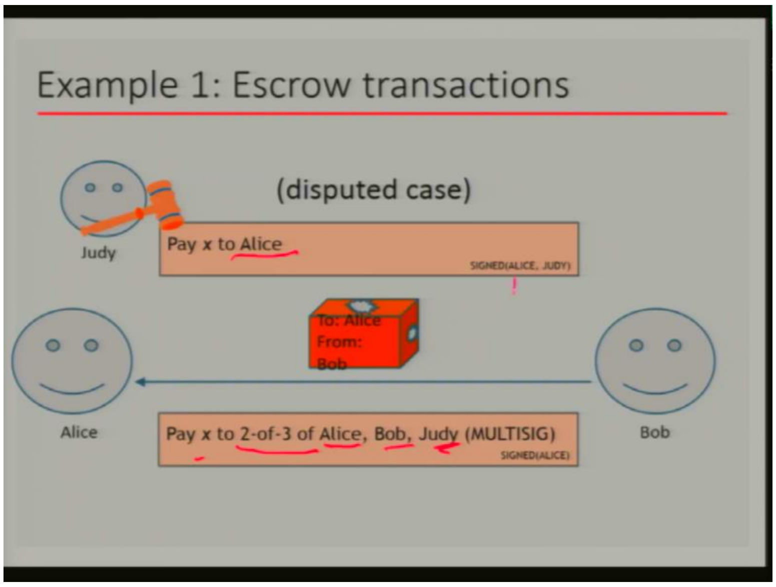
So that is how it works when you are ready to pay the purchase, these guys give you and basically a hash value as an address, you use that.

(Refer Slide Time: 18:07)

Applications of Bitcoin scripts

And that is what the pay to script hash means now, what we will see is that even though the language of bitcoin scripting, language is very simple, even with that, we can do a lot of smart things. And what we do with those is called Art sort of the precursor of smart contracts. So we can use scripts to do multiple things, where we will show you a couple of things.

(Refer Slide Time: 18:39)


Example 1: Escrow transactions

So the first thing we want to show you is the escrow transaction. And then we will also see how to do micro payments, and then we will show you how to do a green payment. So these are the three examples will show but there are many others that you can do with this scripting language. But as we said that since this is not a turing complete language, we cannot do a very complex thing with this.

And therefore, we have a, we have to have a, you know, turing complete language if we want to do very, general things with the help of blockchain. And so let us see how what is an escrow transaction. So let us say Alice wants to buy something online from Bob. And Alice does not want to pay until Bob ships and shows the proof of shipment. But the bob does not want to ship until Alice pays.

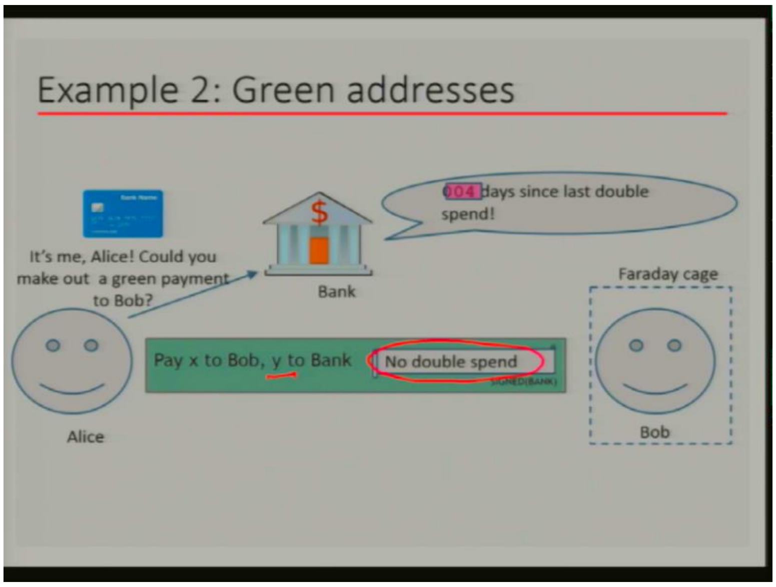
So normally, when we buy something from Amazon, what we do is that we paid by credit card, and then they actually ship the item. And then we can complain to the credit card company if the item is not good. And if we trust the merchant, then actually we can even complain to the merchant himself in like, like a muslin, and maybe we can get a refund or something. But in this case, what escrow transactions do is that it allows you to in well, we are trusted third party to keep them money on escrow.

And then if both sides trust that third party, then the Bob the merchant would know that them even though he did not get the money, it is put in escrow and the scorekeeper is trustworthy. So in case there is a problem, this crew keeper will pay the money to Bob. So what happens is that Alice writes a transaction that says that, that sends it to an escrow account, and it is it is a multi SIG instruction, which says that it requires 2 out of 3.

And if Alice, Bob and Judy, who is the adjudicator or who is the with the escrow person, so now Bob knows that you have put your money in escrow so he ships the item to Alice. Now, in a normal case, suppose Alice gets the item and is happy and they are both honest. So then the, what the Alice says is that use this transaction of the escrow transaction as input and create a new transaction that is given to Bob. And it is assigned by both Alice and Bob.

So that is that is the normal case. Now suppose what Alice gets is broken or has some problem, or it never reaches, then there is a dispute. And then Judy has to decide who is at fault. If, Judy decides that Bob is at fault, then she will sign this, and then we will get Alice will get the money.

(Refer Slide Time: 21:50)


Example 2: Green addresses
004 days since last double spend!
It's me, Alice! Could you make out a green payment to Bob?
Bank
Faraday cage
Pay x to Bob, y to Bank
No double spend
SIGNED(BANK)
Alice
Bob

If Alice is at fault, then Bob will get money. So this is what how the escrow transaction works. So you are basically putting the money in the in escrow until both sides are satisfied, it remains in escrow. If there is a no dispute, then normally, 2 sides sign, it is a 2 out of 3 signature. If there is a dispute, then Judy has to decide and Judy will sign with whichever party should get the money. Now, there is another thing called Green addresses.

So in green addresses, what happens is that say a similar problem Alice wants to pay Bob, but and Alice makes the payment transaction but Bob cannot wait for 6 verifications. Remember, like we said that there has to be 6 blocks built on top of the block that that contains this transaction in the blockchain in order to be you know, probabilistically mature. So and also Bob does not want that Alice then goes around and does a double spending.

So all this stuff or Bob may be completely offline and he has no way to check what is happening. So what Alice can do he can go to third party. In this case, we call it a bank or it could be a crypto currency exchange, some entity, that what they do is that and they have a relation with Alice, the trust Alice or Alice might have a lot of money in the bank or a lot of crypto currency in the exchange or whatever.

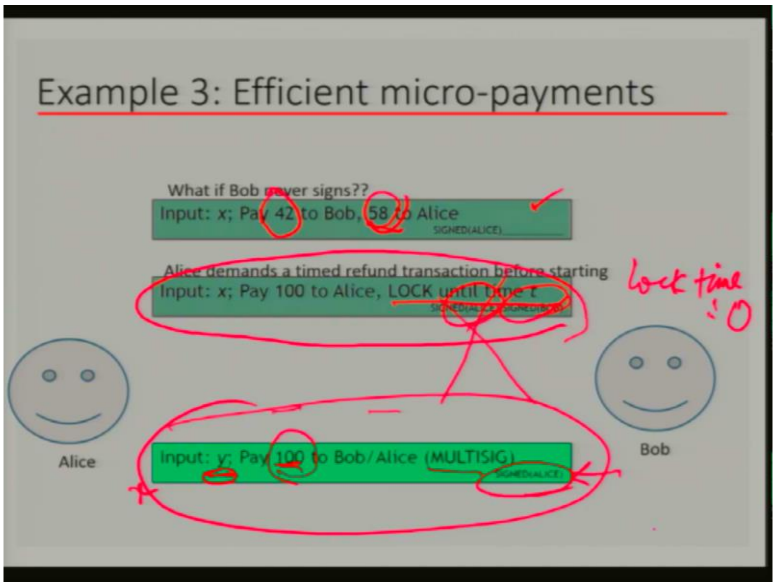
So Alice says, could you make out a green payment to Bob, and then the bank says, I will make a green payment to Bob. So what it does is puts a transaction that paid to Bob, and why to bank the white to bank parties for maybe for fees, that it charges Alice or whatever, and then Alice will then make a transaction with the bank to settle this. And then it gives a guarantee of no double spent, because Bob actually does not know Alice well, so Alice, he cannot trust.

And therefore, even if Alice paid X to Bob, and it comes on as a transaction on the network, Bob does not know that Alice will not now use the same input to another transaction and give it to her to give the money to herself. So that will be a double spin. Whereas bank is a more trusted entity so it probably would not do double spend because it worries about its reputation. Also, it may have some kind of a certificate that says that you know, it has not made a double spend transaction in so many years or whatever.

So Bob cross the bank and Bob then ships the item. Now, what happened in the double spend case is that is that the exchanges when they started this green address idea, people were excited because let us say you are ordering food, so the merchant cannot wait for 1 hour for 6 verifications to deliver your food. So green addresses idea was very welcome. So that the with the bitcoin transactions also you can buy things which can be delivered fast.

On the other hand, we merchant is also guaranteed to be paid eventually, even though he may not know the customer. But turns out that Mount Gox we talked about in a very beginning, Mount Gox was one of the exchanges that used to provide these green addresses in green payments. And there were other few other, but they all actually have run out of business. So green addresses nowadays is not very common because it did not really work out for other reasons, mostly outside the technology reason. But so green addresses in a good idea, but it did not really work that well.

(Refer Slide Time: 25:42)



Example 3: Efficient micro-payments

What if Bob never signs??

Input: x; Pay 42 to Bob, 58 to Alice SIGNED(ALICE)________

Alice demands a timed refund transaction before starting

Input: x; Pay 100 to Alice, LOCK until time *t* SIGNED(ALICE) SIGNED(BOB)

Input: y; Pay 100 to Bob/Alice (MULTISIG) SIGNED(ALICE)

lock time
: 0
Alice
Bob

The third thing that we want to talk about is efficient micro payments. So what is a micro payment say Bob is a mobile service provider and it charges Alice by the minute. So when Alice is making a call, then one possibility is that Alice is, whatever client she is running as the minutes take, she pays Bob, you know, payment for 1 minute and then and make another payment for a second minute and another payment for third minute.

But every time she does that, first of all, she will put too many transactions on the network, because this scenario could be very common if it is let us say airtel and analysis an airtel user and therefore this is not very efficient. Second thing is that the Alice might incur fees per transaction. So she does not want to do so many transactions. So, she has to pay more fees, it may be like you know, Part A minute payment may be less than the payment for the, for putting transactions into blocks.

So therefore, the idea of micro payments came into existence. So what we do is that Alice initially pays an amount y amount, let us say 100 are the maximum that Alice thinks that she would need to make this phone call from her whatever input transaction that she is trying to redeem. And then it is a multi SIG, and it is only signed by Alice. And this is not signed by Bob. So therefore, this is not going to be redeemed yet.

And then per minute, a new transaction is created, which is signed by Alice. Bob does not sign them. So now we are doing a number of double spins. So this is your X transaction. And this is kind of like you made this payment. And then from this transaction, you are redeeming this transaction. Although this transaction has not completed yet, you are redeeming this transaction with 1 to Bob and 99 to Alice.

And then next minute you pay 2 to Bob 92 Alice and so you are doing a lot of double spins. But none of them are being redeemed, they are actually not even put into the blockchain. Because all of this, this cannot go through because they are referring to the same transaction and doing various different things. And they are all double spends. But at the end when Alice says I am done, and Bob says, I will publish the transaction on the blockchain.

So what it does is that he signs the last one of them, and then publishes it. So what happens now is that the last amount that is due to Bob, so let us say it was 42 minutes. So then Bob gets 42, and 58 goes back to Alice. So all the other previous transactions are not put on the blockchain. And therefore, and even if Bob tries to put them on the blockchain, it would not work because it will be a double spend. All these transactions are referring to this transaction.

That is why it is a double spent as we explained before, so now there is a possibility that Bob never signs this transaction. Now, what that means is that Alice will lose this 58 also, she Paid 100 before and then now, she has this 58 this one was actually put into the blockchain, but it was not you know signed by Bob, but now Alice has a possibility of losing the 58. So, therefore, there is a thing called timed refund transaction.

So, in a time refund transaction, Alice demand that when she does this one, Alice demands that the bob does another one in which Bob signs and says that pay 100 to Alice. And this is remember the lock time. The lock time is what we use here and therefore, the lock time will be set to some t appropriate t so that if Bob refuses to sign this one, then Alice can already sign this so Alice can sign this and get all 100 back.

So, that is where the lock time thing that we saw before so usually we put lock time 0 for immediate use of the transaction, but here, we put a time t, and therefore, Alice can get back all the money after time t. We will come back to this in the next session. So for now, we have seen 3 different use of bitcoin scripts to do more complex transaction than just paying from one person to the other.