Next Steps with Blockchain Technology

Table of Contents

Notices

Next Steps with Blockchain Technology

Carnegie Mellon University
Software Engineering Institute

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

CARNEGIE MELLON UNIVERSITY
CERT
SOFTWARE ENGINEERING INSTITUTE

Next Steps with
Blockchain Technology

Eliezer Kanal & **Gabriel Somlo**

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

**001 Presenter: Hello from the campus of Carnegie Mellon University in Pittsburgh, Pennsylvania. We welcome you to Virtual SEI. Our presentation today is Next Steps with Blockchain Technology. My name is Shane McGraw, your audience moderator for today's presentation. And I'd like to thank you for attending.

We want to make today as interactive as possible. So, we will address questions throughout the presentation and again at the end of the presentation. You can submit those questions to our event staff at any time by using the Q&A or chat tabs on the page interface. Also, we ask that you fill out our survey upon leaving today's event as your feedback is greatly appreciated. And the link to the survey will be in the chat area soon.
Now, I'd like to introduce our speakers for today. Elli Kanal is the technical manager with the CERT division at the Software Engineering Institute, who focuses on applying machine learning techniques to the cybersecurity domain. Welcome, Elli.

Presenter: Thank you.

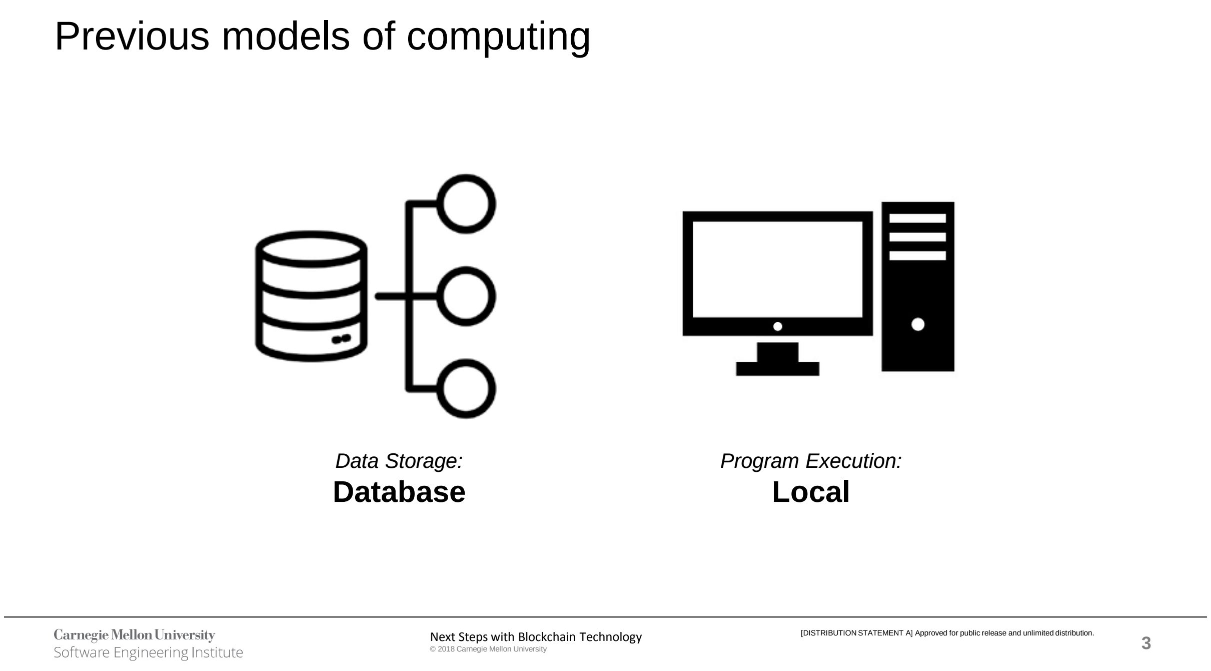
Presenter: Next, we have Gabe Somlo. And Gabe is a cybersecurity researcher within the CERT division at the SEI. Gabe, welcome.

Presenter: Thank you.

Presenter: And now, I'm going to turn it over to Elli. Elli, all yours.

Presenter: Thank you very much, and thank you very much for tuning in today as we have a conversation between Gabe and myself and with you guys about the stuff that's going on with blockchain. So, the way that I want to kick this off is, we have--

Previous models of computing



Previous models of computing

Data Storage:
Database

Program Execution:
Local

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

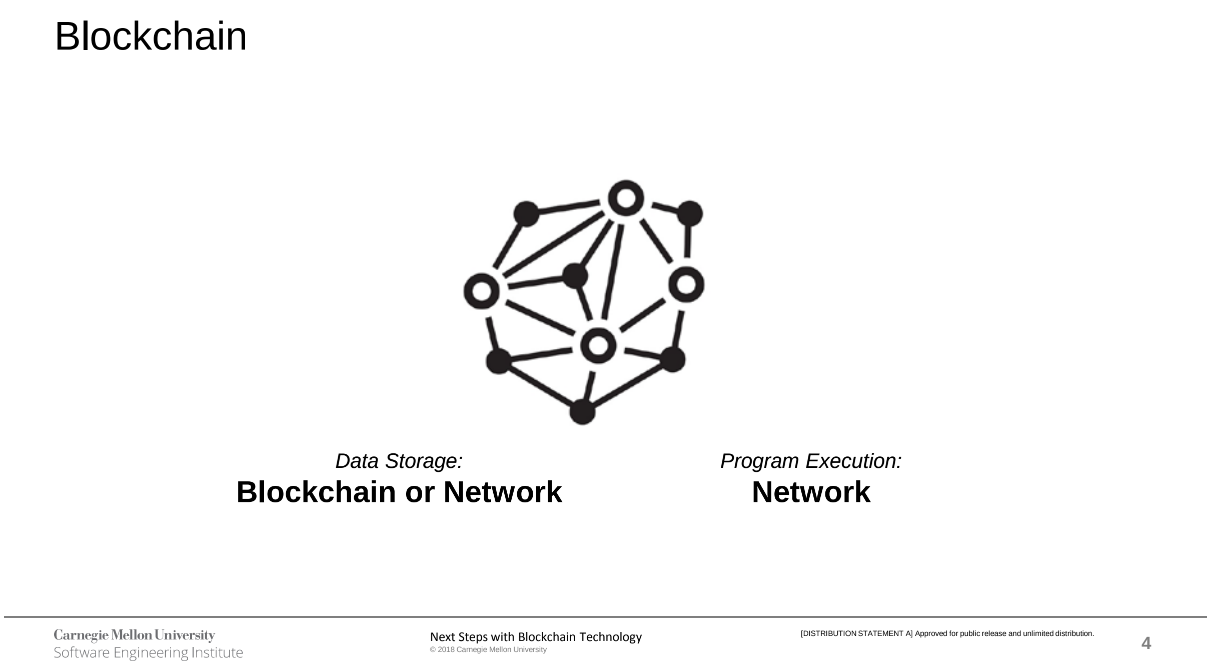
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

3

**003 A couple different things we want to talk about regarding blockchain, different interesting things that we're doing here at the SEI, some things we're seeing our customers asking us to help implement. So, we're going to talk just about some of those use cases. As well, I also want to give-- I want to start off with just a brief overview and intro just to remind everyone what is this technology we're talking about. So, I'll give a quick lead in to that, and then we'll jump into the use cases. So, to start out, those of you, if you've seen our last talk, you may remember I tried to give an introduction as to what is blockchain by starting by saying, "What did we have before blockchain?"

So, traditionally, we had really two completely separate ways of doing computing. We have a concept of a database. If you want to store any information, you put it in a database. You just sort of stick it there. If you want it later, you go to the database and pull it out and get it. On the other hand, we have a computer to do program execution. You know, I want to run some computer program, a browser or word editor. I go to a computer and I'm able to execute that program on my computer.

Blockchain



Blockchain

Data Storage:
Blockchain or Network

Program Execution:
Network

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

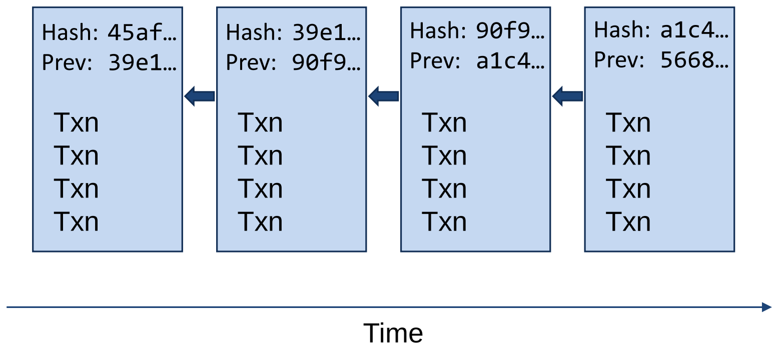
4

**004 So, blockchain technology allows us to really change that up a lot. And it puts both the data storage and the execution of the program all

on my local machine or all on the
machine that's running the
blockchain. But the more interesting
thing is it's not just my local machine.
It's actually all these computers.
They join together. They're all storing
the same data. And they're all
running the same program at the
same time. Some of you may be
familiar with distributed computing or
parallel processing. This is not that.
The concept that we have here is all
the computers running the exact
same instruction at the exact same
time and storing the exact same data
in many different places.

The question comes up why you
might want that. And it comes out--
we're going to kind of gloss over this
a little bit here. It gives you a lot of
benefits in terms of the ability to do
distributed computing, have a lot of
things-- have resilient computing--
excuse me, not distributed, have
resilient computing, have a program
execute across a whole lot of places
at once. People are more difficult to
interrupt it, interfere, take down your
system. The data is very difficult to
disrupt for a variety of reasons. And
for that-- for those reasons and
others, people have really kind of
come towards blockchain as--

a1c4… 5668…



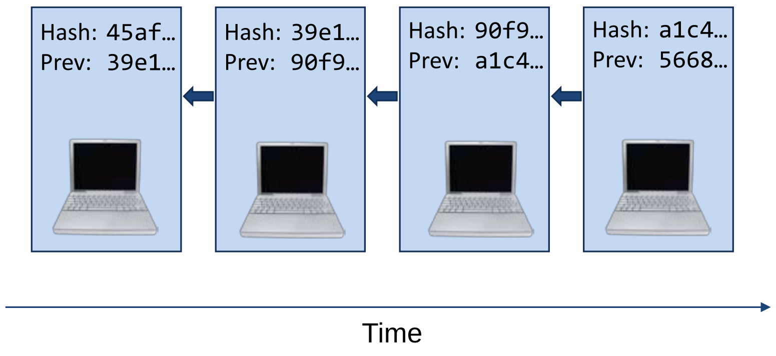
**005 A new technology that can help us make-- bring our computing into the next generation.

Presenter: Would you say it's computation by majority rules basically?

Presenter: Yeah, to a certain extent. Instead of having just my computer be the only one that's executing it where someone can come in and do something malicious on my machine, in this context, all the computers have to agree on the output of the program. So, the way a blockchain works, there is actually blocks, and they are actually in a chain.

So, to that extent, I have a little visualization here. Blockchain works by transactions. Many of you are familiar with bitcoin, which we're going to get to in a minute. You don't have to work with bitcoin. Any system of blockchain is going to have these transaction concepts. So, let's use the example of voting, which I'll talk about in just a second. When someone's doing voting, I send in a single vote. That is a transaction. Many people send in many votes, those are all transactions. The blocks help solidify when those happened in time. And what you see here, time is going from I believe that's left to right depending on how your screen looks. I have a set of transactions. And then the next block points to those previous transactions. Through this way of pointing, it turns out it's very difficult to forge any historical data. In a traditional database, if I want to update some data in the database, I can just go in there and change something. With a blockchain, it's almost impossible to do that. I can definitely submit a transaction later that shows I changed my mind. But that's a new transaction updating something I did historically. I'm not changing the original record.

a1c4… 5668…



Carnegie Mellon University
Software Engineering Institute

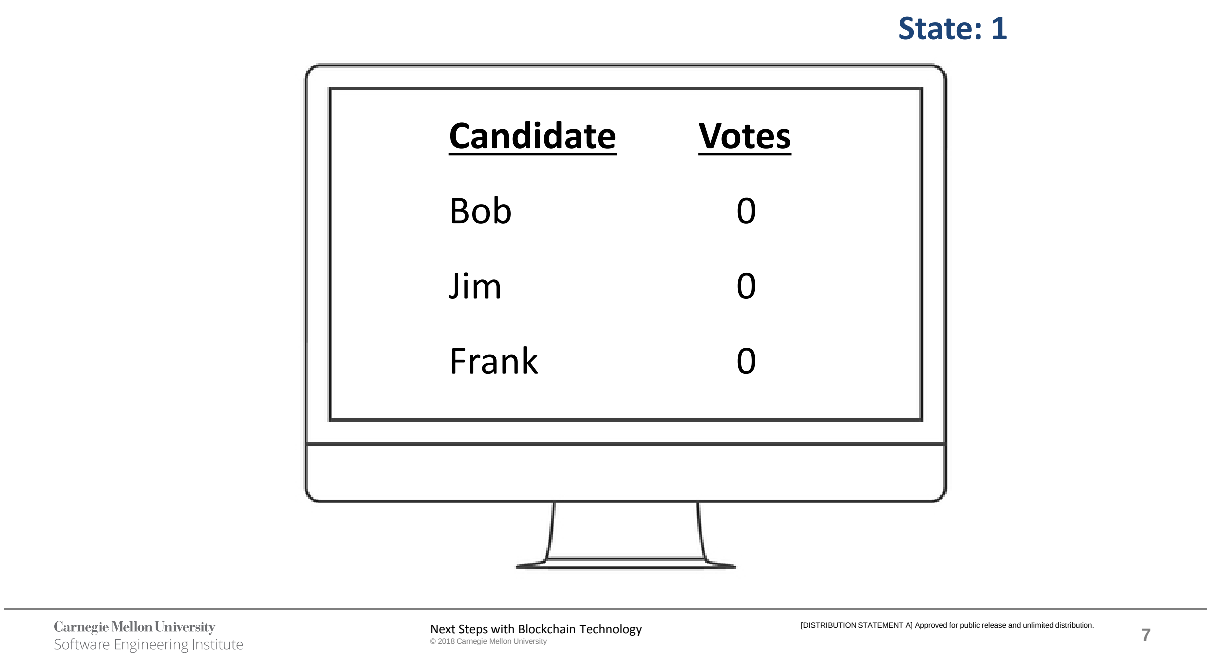
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

6

**006 So, as I was saying before, you have these transactions. Well, sometimes you'd rather not think of it as a transaction but think of it as the entire state of a computer. So, I have a computer program, which is doing something. Let's say it's recording, you know, let's do the voting application, which I'll show in a second. Each block records the state of that voting application at a moment in time. And I can fast forward it and rewind it and look to see what was the exact state.
Let's go through this voting example in just a bit more detail.

State: 1



State: 1

Candidate	Votes
Bob	0
Jim	0
Frank	0

Carnegie Mellon University
Software Engineering Institute

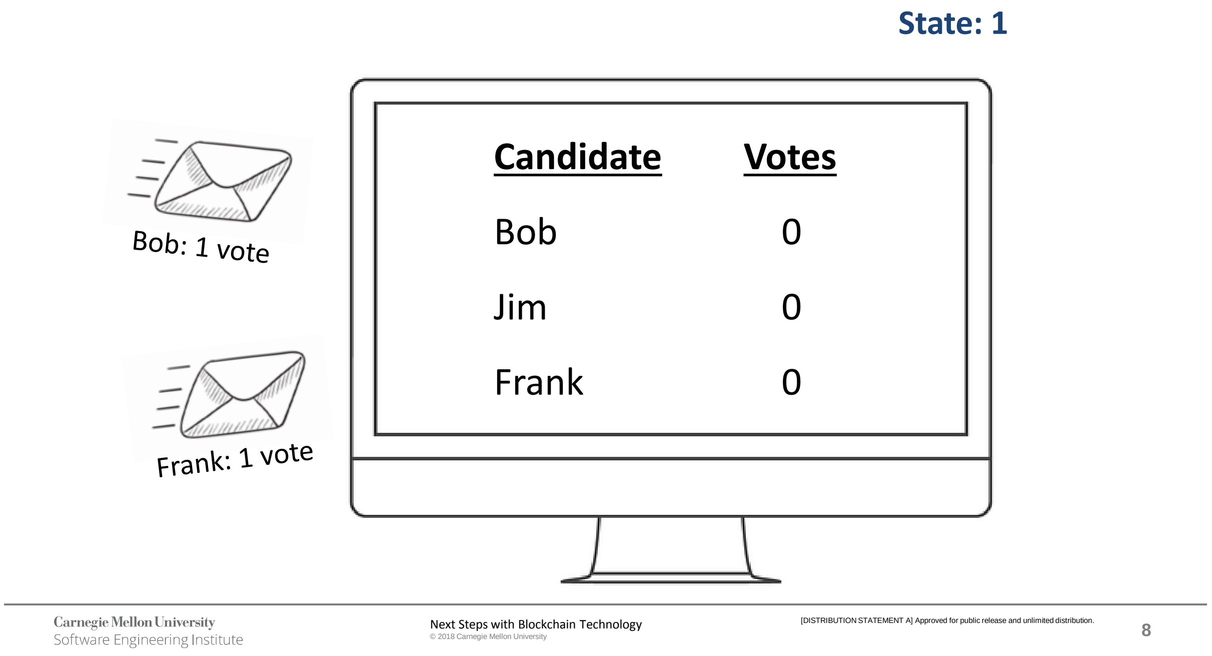
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

7

**007 Let's say I have some vote going on. These are three people who I have at work, and we're trying to vote who wears the ugliest ties, right? So, here I have three candidates, Bob, Jim, and Frank. Those guys really wear nasty looking ties every day. And at the beginning of the vote, no one has any votes yet. We haven't actually recorded anything.

State: 1



State: 1

Candidate	Votes
Bob	0
Jim	0
Frank	0

Bob: 1 vote
Frank: 1 vote

Carnegie Mellon University
Software Engineering Institute

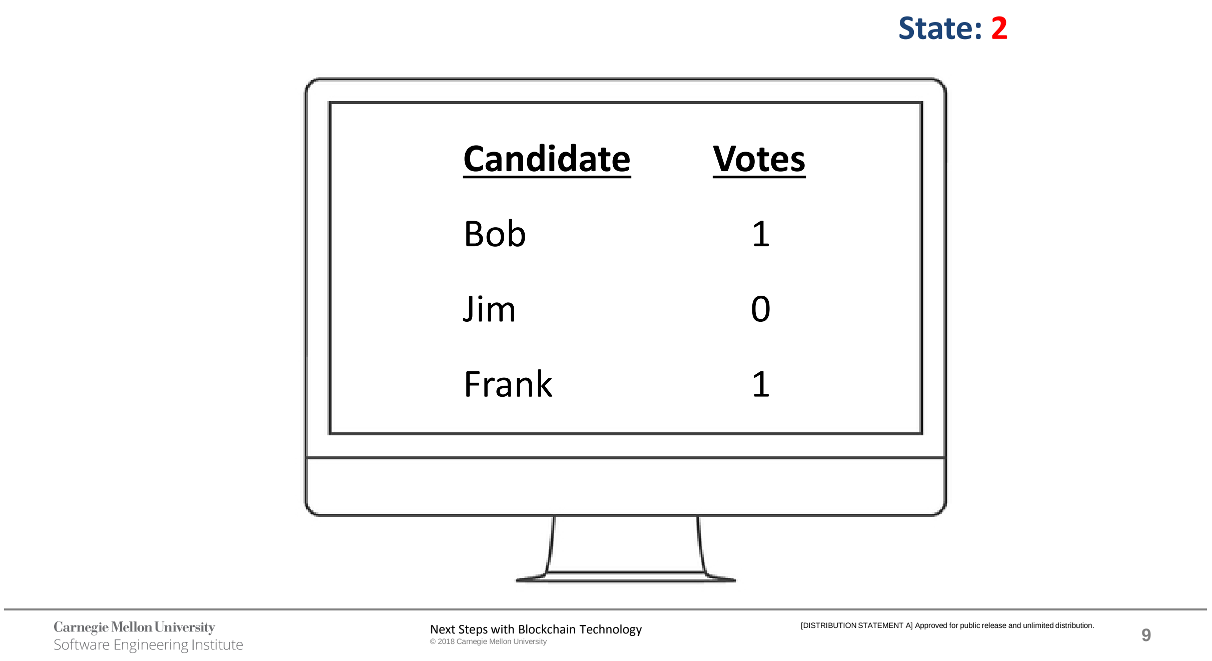
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

8

**008 That's our first state. So, let's move forward. All of a sudden, I get a vote for Bob, and I get a vote for Frank. And you can see those are coming in as messages. That's how I'm representing transactions. They arrive, and now the system's going to update. So, in state one, I only have no votes for anyone. I got these two messages.

State: 2



State: 2

Candidate	Votes
Bob	1
Jim	0
Frank	1

Carnegie Mellon University
Software Engineering Institute

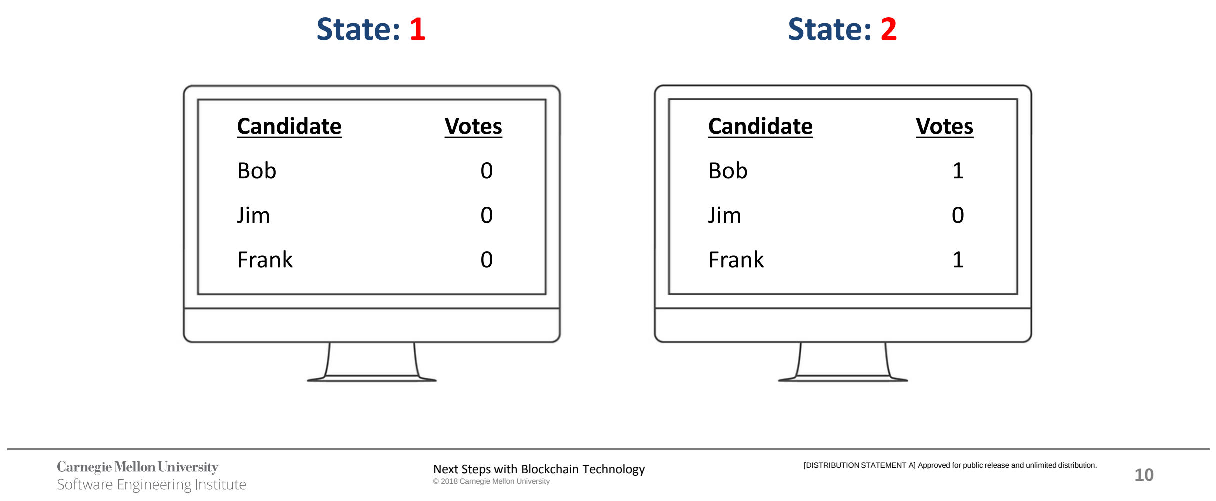
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

9

**009 So, let's move to state two. Now, I update the system, and I have one vote for Bob, one vote for Frank. You can see that this is now progressed from state zero to state one.

State: 1



State: 1

Candidate	Votes
Bob	0
Jim	0
Frank	0

State: 2

Candidate	Votes
Bob	1
Jim	0
Frank	1

Carnegie Mellon University
Software Engineering Institute

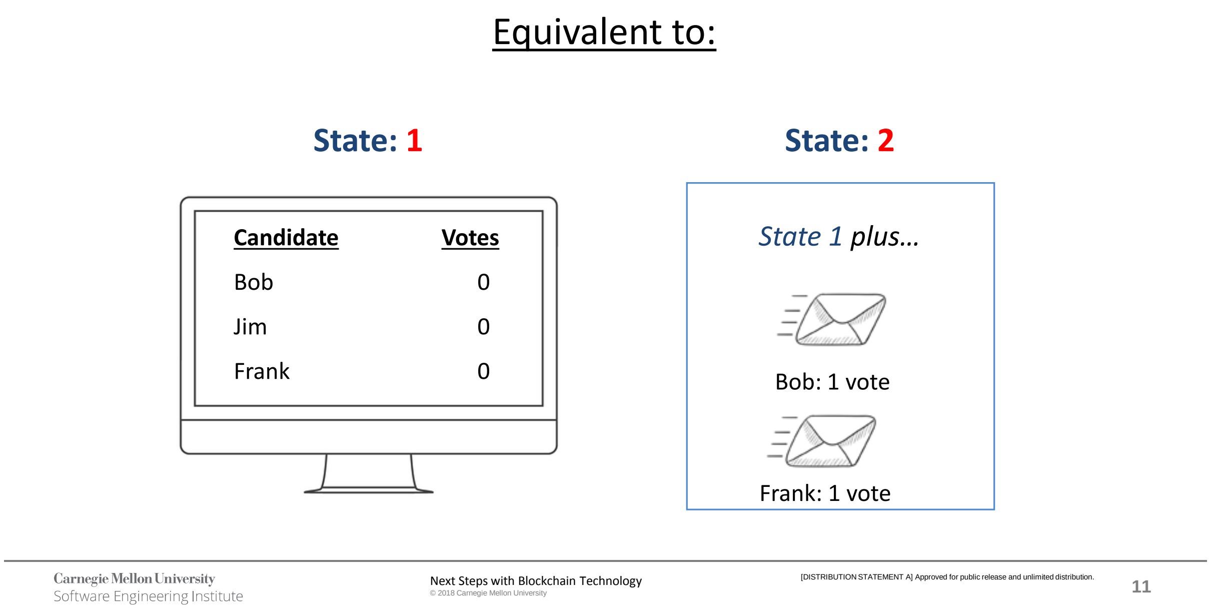
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

10

**010 When I look at this over time, I can represent this as a system of states. Even more interestingly, I don't actually have to save that state.

Equivalent to:



Equivalent to:

State: 1

Candidate	Votes
Bob	0
Jim	0
Frank	0

State: 2

State 1 plus…

Bob: 1 vote

Frank: 1 vote

Carnegie Mellon University
Software Engineering Institute

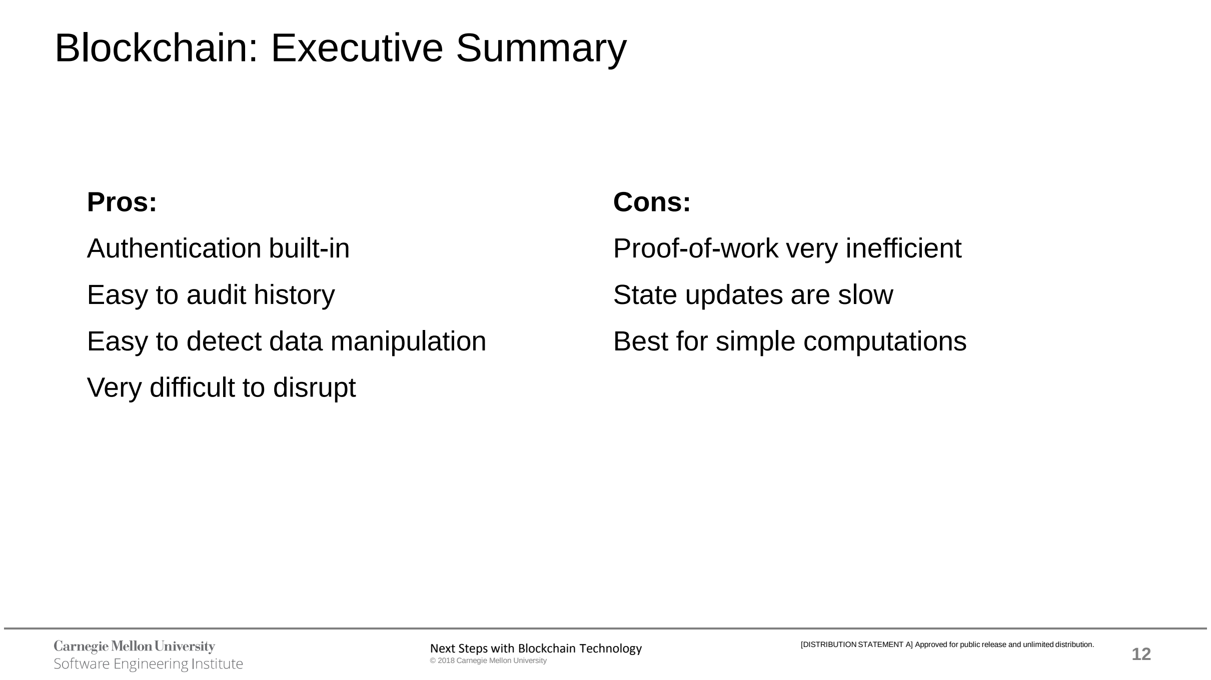
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

11

**011 All I have to change, and all I have to save, is the change requests themselves because over time, I'm going to get a whole lot of these transactions. I'm going to get a whole lot of these votes. And all I need to do is say at this block which votes were included, at this block which votes were included. And over time, I'm watching the state of my system progress through.

Blockchain: Executive Summary



Blockchain: Executive Summary

Pros:	Cons:
Authentication built-in	Proof-of-work very inefficient
Easy to audit history	State updates are slow
Easy to detect data manipulation	Best for simple computations
Very difficult to disrupt	

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

12

**012 Presenter: Speaking of voting, I just want to throw in a quick real two cents. We would need to make sure that we are-- in a real live voting scenario for elections, political process, that kind of thing, we would have to make sure to protect the confidentiality of the voter and the ability to recount, which seems at first glance to be at odds with each other. But there's really some deep dark math that actually makes those things work together. Look up linkable ring signatures if you want some extra painful math to wade through to see how that's done.

Presenter: Right, and for what it's worth, there are already blockchain technologies out there. People say blockchain. Well it turns out, that's kind of like saying computer. And yes, you can just buy a generic computer. But when you go out to buy one, you're going to buy a specific type of computer. So, you mentioned linkable ring signatures. Well so, you have the bitcoin, which is a type-- bitcoin which is a type of a blockchain, one implementation of a blockchain. Some of you may have heard of Hyperledger Fabric. You may have also heard of Monero or Zcash. There's a whole lot of these different blockchain implementations out there, some of which incorporate the technology that you're referring to over there.

Cool so, to use that as the quick reminder, quick refresher of what is a blockchain, so basically, we can-- there's a slide showing up here. There is basically four pros I would say when you want to give the executive summary why would anyone want to use a blockchain. First of all, it has authentication built in. We didn't talk about this in this particular example. But whenever someone sends a message, they have to sign it, not really with their name, with the digital equivalent of their signature. But they do sign it. And that means I know, if I can have some way of tracking who owns each digital signature, I know who gave that vote.

Secondly, it's very easy to audit history. In many cases, particularly many business applications, that's a requirement. I need to have a way to easily audit what happened in the past. Many times, when you have an application that you are trying to deploy for the enterprise, you have to buy that and add that on after the fact. In this context, blockchain so to speak comes with that. That is fundamental to the blockchain technology that you can easily see what happened.

It's also easy to detect attempts at data manipulation. If I want to change what's going on in the data, it's really hard to do. And in this context, I'm able to see that someone did it. Lastly, it's exceedingly hard to take down a blockchain application, namely because it's running on so many computers at once. And if you kill one machine, you haven't even touched the actual application. It's running just as well on all the other applications.

There's a few downsides which you might not want to use a blockchain application. One of the first ones is currently, it's actually quite inefficient as of yet. There is technologies that are being researched that will help make it more efficient. But you may have heard of this concept of mining, which I'll get into in just a second. Mining is very expensive, at least as it's being carried out currently on most of the blockchains that are out there in the general public. So, that's a thing that people want to consider. Do I really need to take on the additional cost of mining?

Another one is that the state updates are slow. When your computer is trying to-- when you hit command S on your computer to save a document, that can happen in fractions of a second, very small fractions of a second in fact. When you want to update the state of a blockchain application though, it typically takes multiple seconds to minutes. And sometimes, especially if you're working with the bitcoin blockchain, they'll recommend you wait even a whole hour just to make sure it got into the chain. For technical reasons, you don't want to have it that it's not present in the later versions of the chain. That's not something which many applications can work with. There are some technical ways to move around that. But still that's a feature of blockchain that isn't likely to go away.

The last one is, as of now, blockchain is best for simple applications, simple computations. If you're going to be doing something complicated like video editing, you're going to have to run every video editing step on every single machine tied to the blockchain, which turns out to be expensive. So, you would tend to favor having blockchain applications work mostly with applications that are simple. Think of things such as financial transactions, or other things in that context.

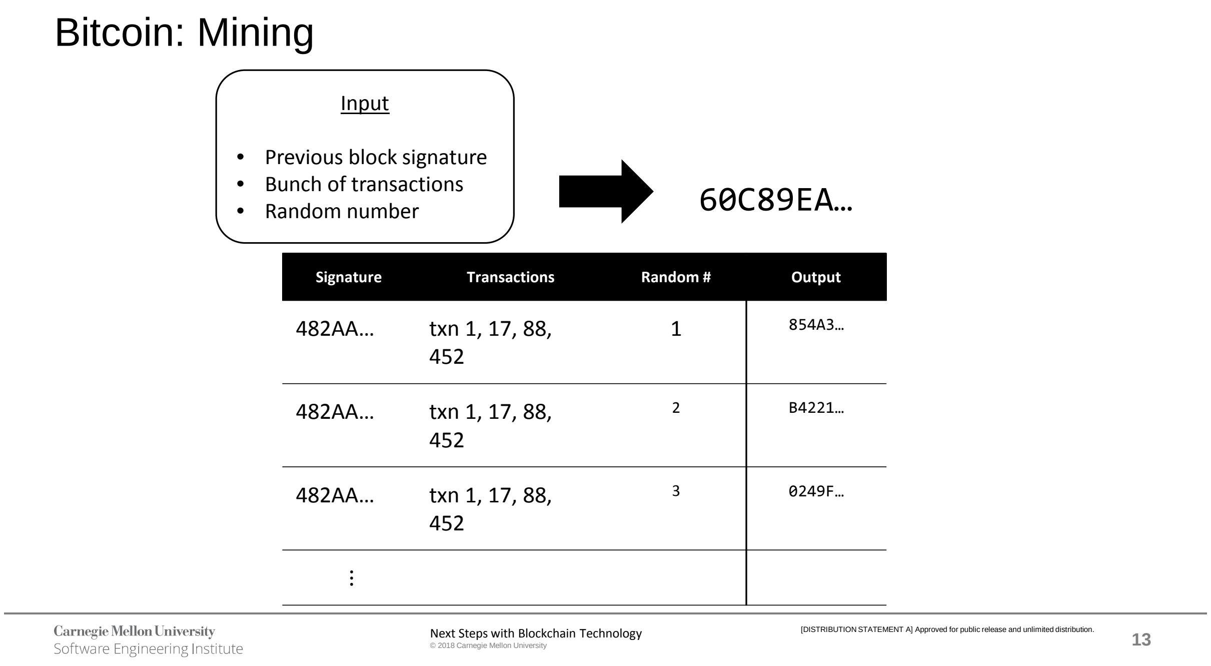
Presenter: Simple and highly important.

Presenter: Right, where you really want to store--

Presenter: Votes.

Presenter: To state. Votes are a fine example. Financial, there's a lot of talk also about medical applications as well, supply chain, a bunch of areas people are looking at.

Bitcoin: Mining



Bitcoin: Mining

Input

- Previous block signature
- Bunch of transactions
- Random number

60C89EA…

Signature	Transactions	Random #	Output
482AA…	txn 1, 17, 88, 452	1	854A3…
482AA…	txn 1, 17, 88, 452	2	B4221…
482AA…	txn 1, 17, 88, 452	3	0249F…
⋮			

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

13

**013 All right, cool. So, that's the primer. Before I turn this over to Gabe to give a talk about some of the cool stuff that they're doing, I want to give just a brief refresh regarding the-- how bitcoin works. Only the briefest aspect, but this is going to be pretty relevant to the stuff that Gabe is going to talk about. So, to that extent, just a quick recap of what is mining. So, when you want to mine, what we're essentially trying to say is I have a whole bunch of history. And I want to make sure that this new block gets stored onto the blockchain in a way that every single person agrees. We're not going to talk now about why that is I have to go through the mining process.

But what is mining? Well, mining is I take a couple inputs. I have the previous block signature. I have a bunch of transactions that I want to include in this block. And I have to find this random number. And what is the random number? Well, I won't know it until I find it. It's essentially a random number that will make the hash-- if that means something, great. If not, just think of a signature of the new block looking a very specific way. And I'm going to continuously search. And you may say, "What's the best way to search for a random number?" You literally choose random numbers. And those numbers can be as small as zero, as high as I think two trillion or something like that.

Presenter: Or you just start iterating and--

Presenter: Yeah, start somewhere and just go.

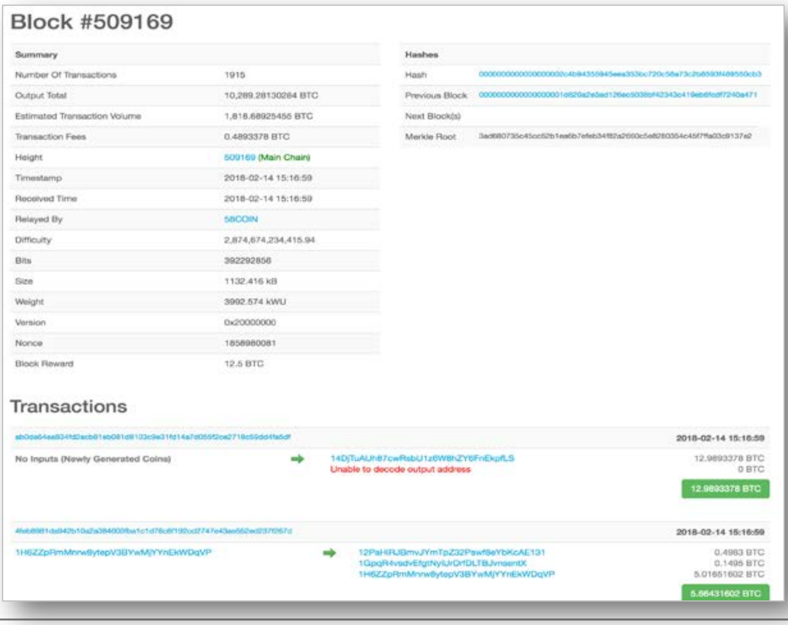
Presenter: At some point, you'll find one that matches.

Presenter: Exactly, start one-- start there and plus one, plus one, plus one.

Presenter: And there's no shortcut.

Presenter: And there's no shortcut. This is statistically proven, mathematically proven to have that. And you're going to find that. Once you find that random number so to speak, you yell it out to all the other participants in the blockchain. If you're doing bitcoin, you'd yell it out to every other bitcoin miner. They'll check, and they'll say, "Yes, that number is actually correct." And then everyone starts mining the next block.

Block #509169



Block #509169

Summary	
Number Of Transactions	1915
Output Total	10,289.28130284 BTC
Estimated Transaction Volume	1,818.68925456 BTC
Transaction Fees	0.4893378 BTC
Height	509169 (Main Chain)
Timestamp	2018-02-14 15:16:59
Received Time	2018-02-14 15:16:59
Relayed By	58COIN
Difficulty	2,874,674,234,415.94
Bits	392292856
Size	1132.416 kB
Weight	3992.574 kWU
Version	0x20000000
Nonce	1858980081
Block Reward	12.5 BTC

Hashes	
Hash	0000000000000000002c4b84355945eea353bc720c58a73c2b6593f489550cb3
Previous Block	00000000000000000001d820a2e0ad126ec5038bf42343c419ebbfcdf7240a471
Next Block(s)	
Merkle Root	3ad680735c45cc82b1ea6b7efeb34f82a2660c5e8280354c45f7ffa03c9137e2

Transactions

ab0da64ea834fd2acb81eb081d8103c9e31fd14a7d055f2ce2718c59dd4fa5df — 2018-02-14 15:16:59

No Inputs (Newly Generated Coins) → 14DjTuAUh87cwRsbU1z6W8hZY6FnEkpfLS 12.9893378 BTC
Unable to decode output address 0 BTC
12.9893378 BTC

4feb8981da942b10a2a384600fba1c1d76c8f192cd2747e43ae552ed237f267d — 2018-02-14 15:16:59

1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP → 12PaHRJBmvJYmTpZ32Pawf8eYbKcAE131 0.4983 BTC
1GpqR4vadvEfgtNyiUrDrfDLTBJvnsentX 0.1495 BTC
1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP 5.01651602 BTC
5.66431602 BTC

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

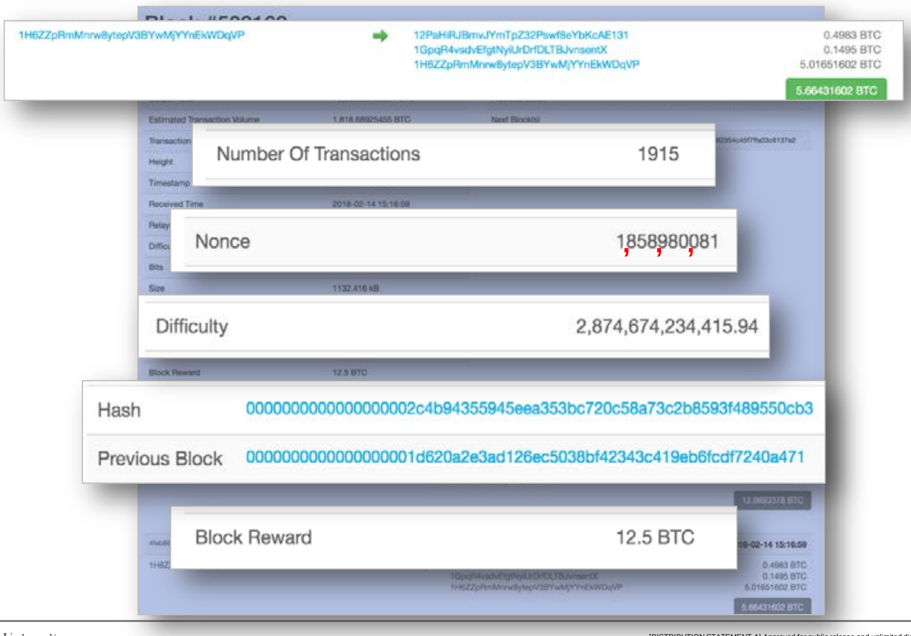
14

**014 Presenter: Gradually because you just pre-empted their attempt to win that race basically.

Presenter: Exactly, exactly. So, in bitcoin, you get something from it. We'll talk about that here. This is an example of what a block looks like showing up here. And you see there's a whole lot of metadata associated with this.

I'm going to call out just a couple pieces.

Transaction



1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP → 12PaHiRJBmvJYmTpZ32Pswf8eYbKcAE131 0.4983 BTC
1GpqR4vsdvEfgtNyiUrDrfDLTBJvnsentX 0.1495 BTC
1H6ZZpRmMnrw8ytepV3BYwMjYYnEkWDqVP 5.01651602 BTC
5.66431602 BTC

Number Of Transactions	1915
Nonce	1,858,980,081
Difficulty	2,874,674,234,415.94
Hash	0000000000000000002c4b94355945eea353bc720c58a73c2b8593f489550cb3
Previous Block	0000000000000000001d620a2e3ad126ec5038bf42343c419eb6fcdf7240a471
Block Reward	12.5 BTC

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

15

**015 And you'll see some of these also in what Gabe's about to discuss. So, first thing you see is the actual transactions. I've only highlighted one at the very, very top.

You can see the string of numbers represents a person, an individual, or a wallet. And he sent money to the strings of numbers on the sides. In this case, this one guy sent three transactions or three bits of money, two to someone else, and one back to himself. That's just one transaction in this block.

If you look at the next little block, it says number of transactions. There almost two thousand transactions similar to this one on this exact block in the blockchain. This is a block from the bitcoin blockchain. I said you're searching for a random number. Well, that's the Nonce. And if you look at that Nonce, you can see that this Nonce is somewhere around two billion. That's a pretty big number. And remember, we're really just randomly searching for a number. That's the number that they happen to find.

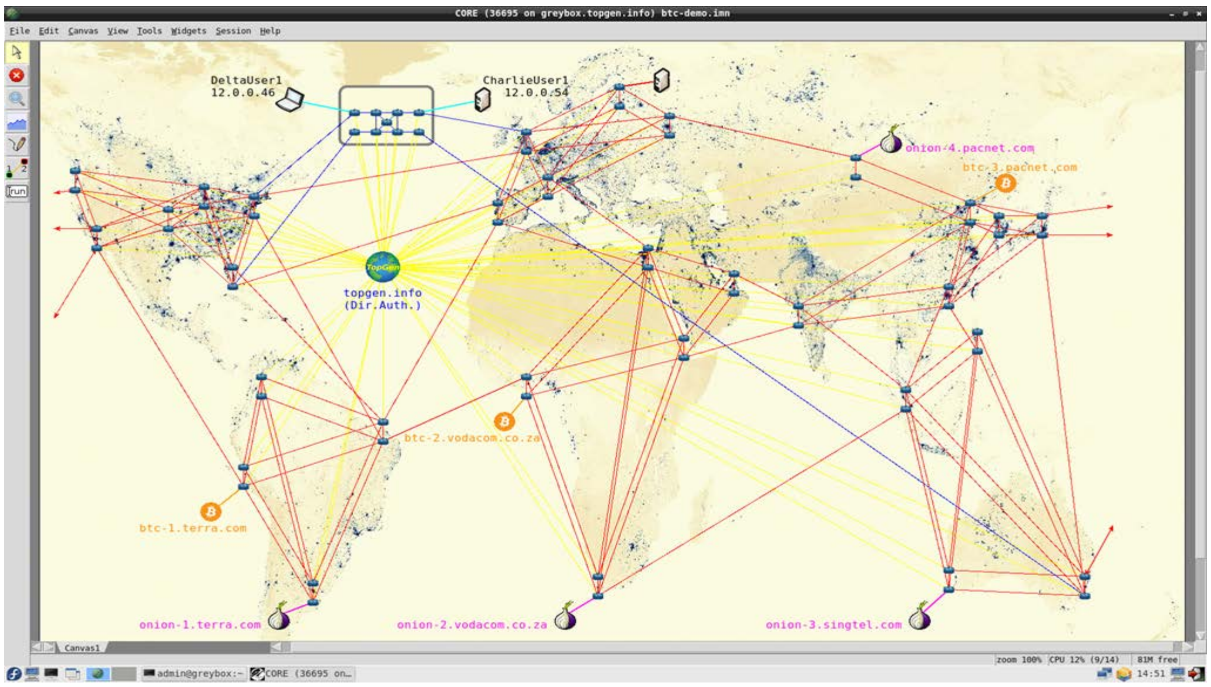
Let's skip to the very bottom. The last thing that it says down there is block reward. It's worth pointing out why do people do bitcoin mining. Well because when you mine a block, you get a reward. No matter-- let's say there are no transactions that need to be processed that day. You go out to do your mining, no one's there. Well so, when you mine that block, you get at least one transaction. That transaction is the creation of new money. And that goes to the person who found that random number.

Presenter: So, in a way, think back to 2009 when the bitcoin network actually came online for the very first time. There were no bitcoins in that network. They had to be introduced somehow. So, basically, blocks were mined, and then the miners had money. And then they started paying for things with that money. And then more people had money, even those who weren't mining. And so, that's how basically money was created and became worth more as more people were using it.

Presenter: Right, and as they actually got this money, they were able to now have some currency to spend. And this is the process that in-with fiat money, some treasury prints it. In blockchain and bitcoin case, every time a new block is created, money is created along with it. And it is handed to the person who actually found that random number. So, with that, that's the primer. And now, Gabe, why don't you give a little bit of a discussion about what it is you're doing here?

Presenter: So, my group within CERT is the Cyber Workforce Development Directorate. And our mission is to develop training materials for U.S. DoD and law enforcement personnel to help with the training for offensive, defensive, and forensic cybersecurity skills. Generally, we do this on a large simulated environment consisting of virtual machines and containers.

Image



**016 In the image, you will see we have developed basically an Internet simulator on top of a network simulator originally developed in collaboration by Boeing and the Naval Research Lab. So, what we have here are containers representing Internet backbone routers, BGP routers. And the virtual links that wired them together are done so that the delay of the packets kind of reflects the slowdown of however long it takes a photon or electric signals to make it across and sort of represent the geographic representative sort of way in the real world.

And so, in addition to BGP backbone routers and websites, we can simulate run containers that actually run the real software for various other sandbox networks. We have essentially a canned Tor network. And more relevant to today's topic, we have an actual bitcoin network where-- a very small but fully functional bitcoin network where three nodes discover each other and begin to competitively mine against each other. And they run the real bitcoin software from GitHub. The only modification we had to make to the software was adding back the capability to mine blocks with the computer's CPU. Sort of a tangent, in the real software, that functionality has been removed because due to the fact that in production, real bitcoin miners use dedicated, accelerated hardware, integrated circuits, to computer checksums that calculate those nonces.

Presenter: Find random numbers.

Presenter: No one has successfully mined a block using their real CPU that's not an accelerated piece of hardware in at least five years now if not longer. So, the upstream developers have decided because this code isn't actually getting exercise in production, they would remove it. So, the only thing we had to do to make it work in a sandbox was to just add that functionality back so that these containers could use the CPU to mine blocks against each other.

Presenter: So, to that extent, this is basically the Internet, like the whole Internet in a box virtualized.

Presenter: We call it gray box because typically there's a gray cloud shaped icon in network diagrams who are presenting the Internet. So, that's-- we refer to that as gray space. So, it's a box that represents gray space, hence a gray box.

Presenter: And on here, you have each of those little dots on the current showing slide represents, to a certain extent some of the miners, these machines that are actually running.

Presenter: So, the blue dots represent Internet backbone routers. And I have the yellow bitcoin logo icons that actually are the miners. And they're basically connected in three different places in the world. And they start up, and bitcoin software has a list of seeds that if the software has never been connected to the Internet before, it will start to reach out to a preprogrammed list of IP addresses that are typically well-known bitcoin miners. And so, these guys occupy those IP addresses representing those locations or those computers in the real world. So, eventually, like within a few minutes, they find each other and start mining against each other and generating blocks from scratch.

Presenter: And so, they're both generating blocks, but I think they're also actually making transactions. They're acting as real people would.

Presenter: Right, they do. And so, when the network starts, it's really quiet, and nobody's actually issuing any transactions. So, in the absence of user generated transactions, transferring money, and while in the absence of any money to begin with, they will just start mining empty blocks with just coin-based transactions that reward whoever wins the race. One of the three miners each time will win the rights to print in the beginning fifty bitcoins. And then later on, after a certain number of blocks has been reached, that number of bitcoins gets halved just like in the real world to twenty-five and eventually twelve and a half. The only difference from the real world is that blocks get mined a lot faster, every one minute on average instead of every ten like in the real world, just because we do want to be able to see some progress while we're playing around with the sandbox.

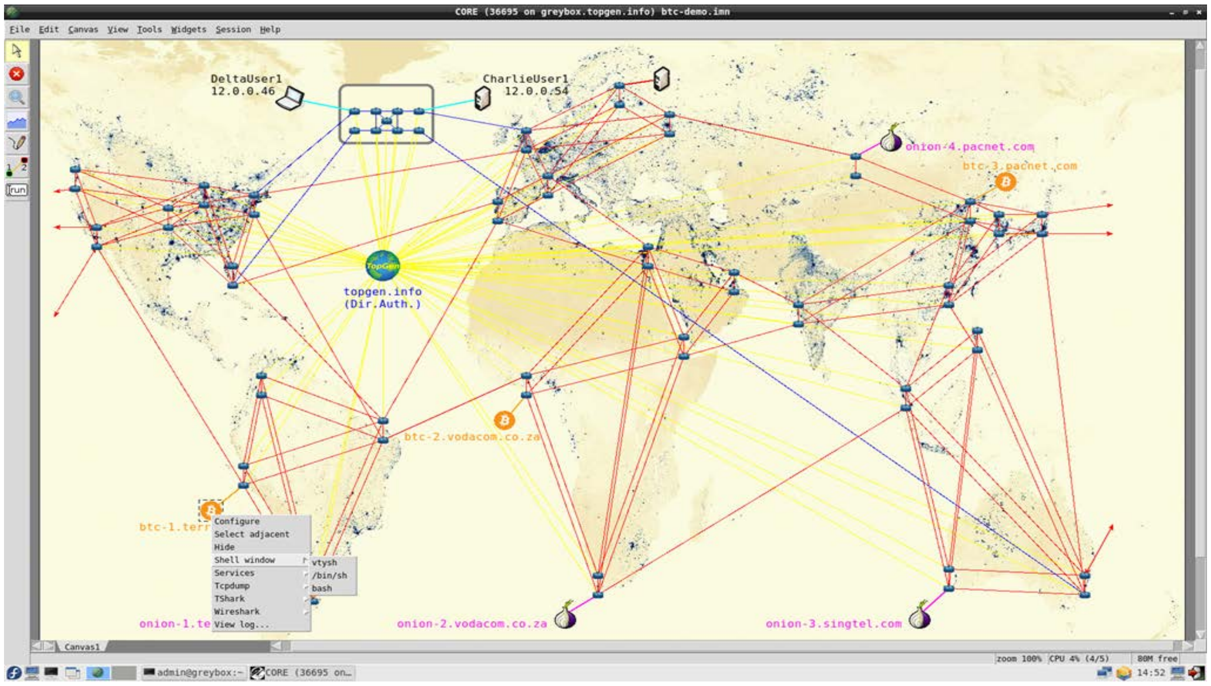
Presenter: Cool, right. So, let's take a step back for second. So, we have this cool technology where we're simulating the entire Internet. We have some bitcoin miners. We have some folks generating transactions. So, why are we making this whole environment?

Presenter: So, a lot of times, law enforcement and the other personnel, cybersecurity professionals, have to do forensic analysis, find trace transactions through the network. And so, we'd like to be able to show them how the software works and what to look for and give them the opportunity to practice just looking, analyzing the blockchain, finding transactions, seeing how they link to other transactions, tracing money that was

spent during interesting events like
ransomware payments or drug deals
that kind of thing.

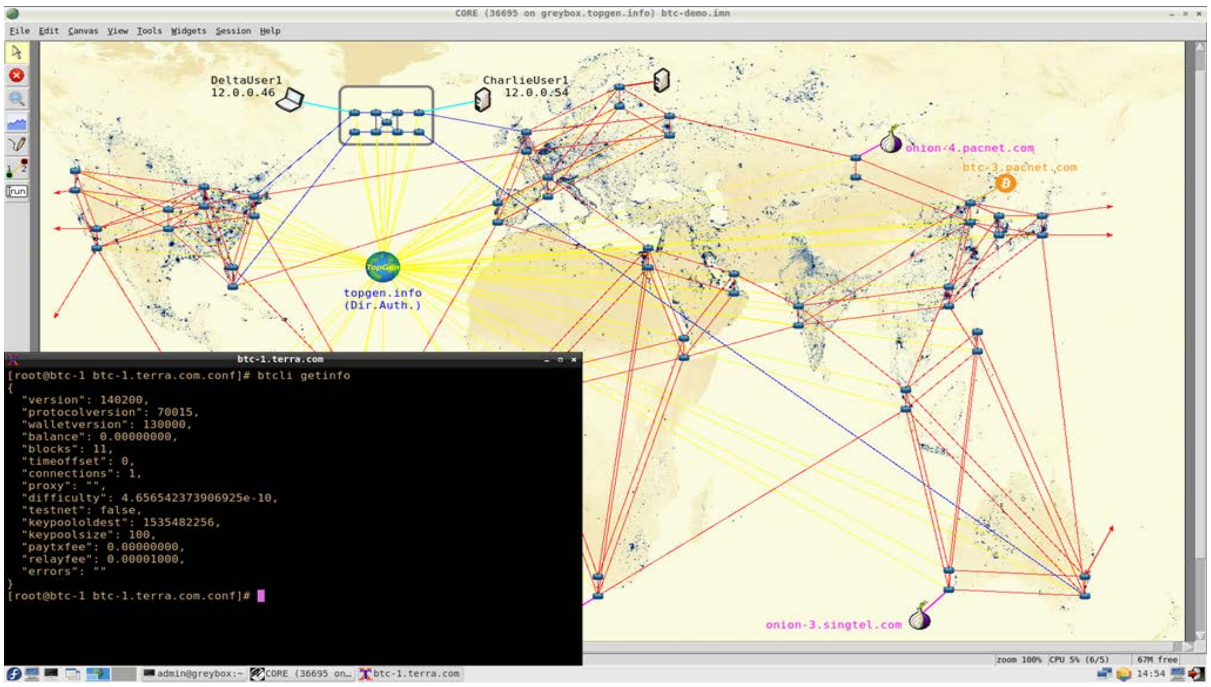
Presenter: So, and how would you
use something like this?

Image



**017 Presenter: So, once you
have the simulation up and running
and the bitcoin miners start talking to
each other--

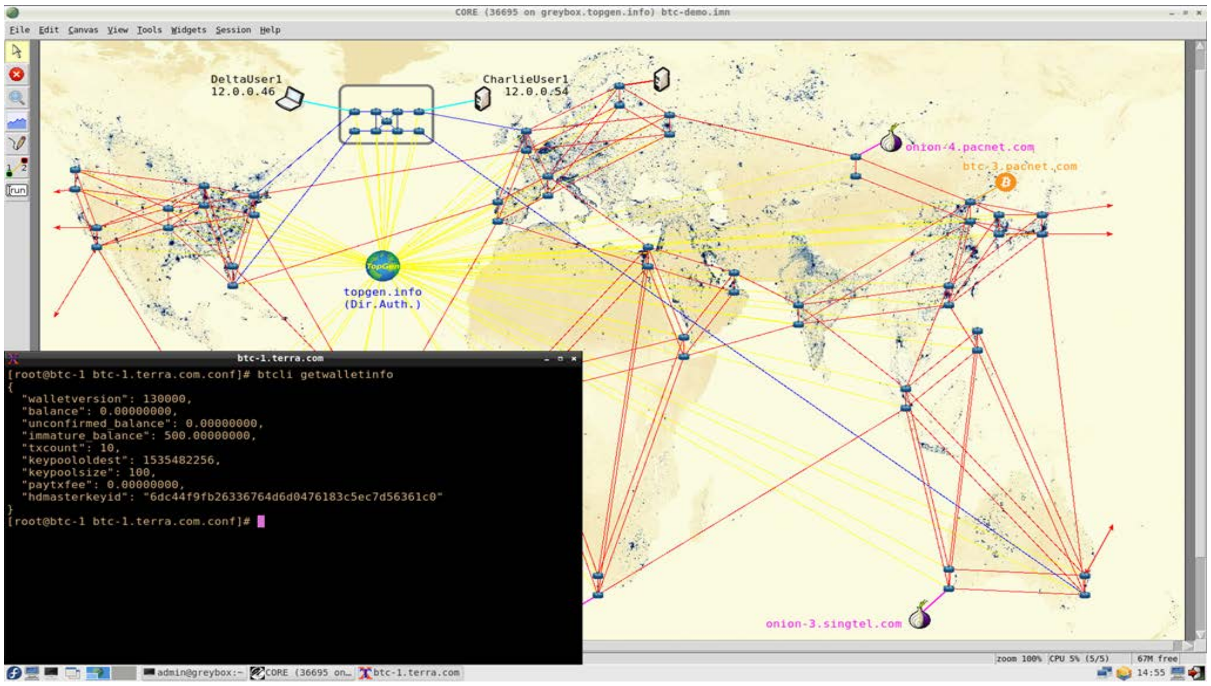
Image



```
[root@btc-1 btc-1.terra.com.conf]# btcli getinfo
{
  "version": 140200,
  "protocolversion": 70015,
  "walletversion": 130000,
  "balance": 0.00000000,
  "blocks": 11,
  "timeoffset": 0,
  "connections": 1,
  "proxy": "",
  "difficulty": 4.656542373906925e-10,
  "testnet": false,
  "keypoololdest": 1535482256,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "relayfee": 0.00001000,
  "errors": ""
}
[root@btc-1 btc-1.terra.com.conf]#
```

**018 We could open a terminal on one of the-- or either one of the nodes and poke around, look for information, see how many connections we have. This is stuff you can do if you download the bitcoin software and join the real bitcoin network. And then you can look at the real state of the bitcoin network in production. But in the simulation, you can see that maybe we've just mined eleven blocks since the simulation was started. We only have one connection out of the two possible peers that this node might be able to find eventually.

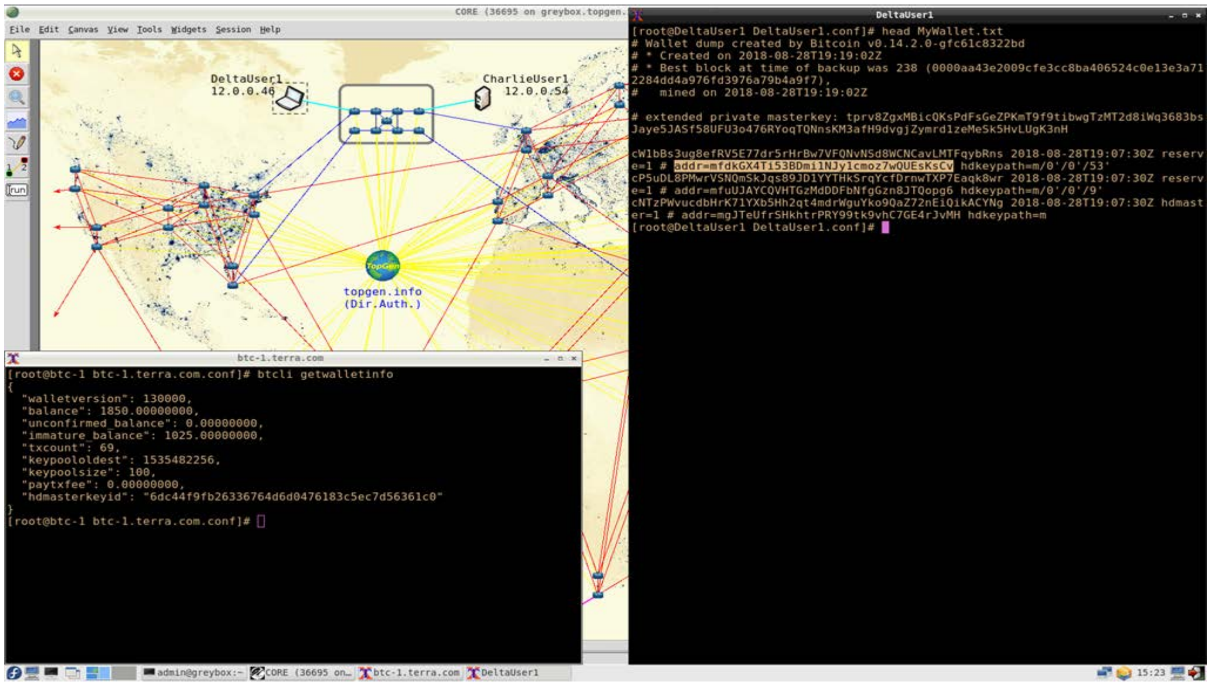
Image



```
[root@btc-1 btc-1.terra.com.conf]# btcli getwalletinfo
{
  "walletversion": 130000,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 500.00000000,
  "txcount": 10,
  "keypoololdest": 1535482256,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "6dc44f9fb26336764d6d0476183c5ec7d56361c0"
}
[root@btc-1 btc-1.terra.com.conf]#
```

**019 We can see that we don't really have a lot of money yet. We've only mined a very small number of blocks. And the money is not even reliably available to spend because generally in the network you would expect to have a lot more confirmations, like more blocks mined on top of yours before the reward that your block gives you can actually be relied upon to be accepted by the rest of the network.

Image

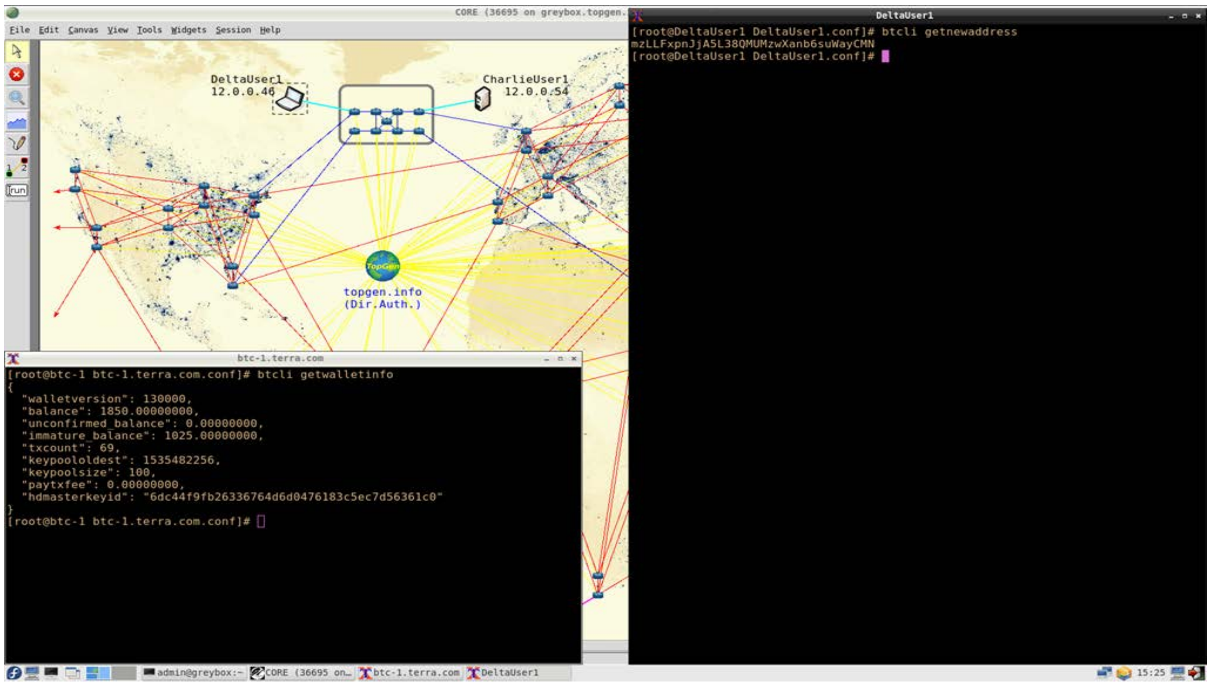


```
[root@DeltaUser1 DeltaUser1.conf]# head MyWallet.txt
# Wallet dump created by Bitcoin v0.14.2.0-gfc61c8322bd
# * Created on 2018-08-28T19:19:02Z
# * Best block at time of backup was 238 (0000aa43e2009cfe3cc8ba406524c0e13e3a71
2284dd4a976fd3976a79b4a9f7),
#   mined on 2018-08-28T19:19:02Z

# extended private masterkey: tprv8ZgxMBicQKsPdFsGeZPKmT9f9tibwgTzMT2d8iWq3683bs
Jaye5JASf58UFU3o476RYoqTQNnsKM3afH9dvgjZymrd1zeMeSk5HvLUgK3nH

cW1bBs3ug8efRV5E77dr5rHrBw7VFQNvNSd8WCNCavLMTFqybRns 2018-08-28T19:07:30Z reserv
e=1 # addr=mfdkGX4Ti53BDmi1NJy1cmoz7wQUEsKsCv hdkeypath=m/0'/0'/53'
cP5uDL8PMwrVSNQmSkJqs89JD1YYTHkSrqYcfDrnwTXP7Eaqk8wr 2018-08-28T19:07:30Z reserv
e=1 # addr=mfuUJAYCQVHTGzMdDDFbNfgGzn8JTQopg6 hdkeypath=m/0'/0'/9'
cNTzPWvucdbHrK71YXb5Hh2qt4mdrWguYko9QaZ72nEiQikACYNg 2018-08-28T19:07:30Z hdmast
er=1 # addr=mgJTeUfrSHkhtrPRY99tk9vhC7GE4rJvMH hdkeypath=m
[root@DeltaUser1 DeltaUser1.conf]#
```

```
[root@btc-1 btc-1.terra.com.conf]# btcli getwalletinfo
{
  "walletversion": 130000,
  "balance": 1850.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 1025.00000000,
  "txcount": 69,
  "keypoololdest": 1535482256,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "6dc44f9fb26336764d6d0476183c5ec7d56361c0"
}
[root@btc-1 btc-1.terra.com.conf]#
```

**021 And we can even start the software without mining, just this is a user node. We start the user node on another computer or another container in the simulation. And we could analyze the wallet, see what the wallet looks like. It's basically not just one private/public pair of crypto keys, but a hundred of them essentially. The default is a hundred keys so that there's some measure of pseudo anonymity. It's not real anonymity in the bitcoin network. It's just that each user has sort of a hundred key pairs and kind of randomly picks one whenever they want to do business with anybody else. So, if I wanted--

Image

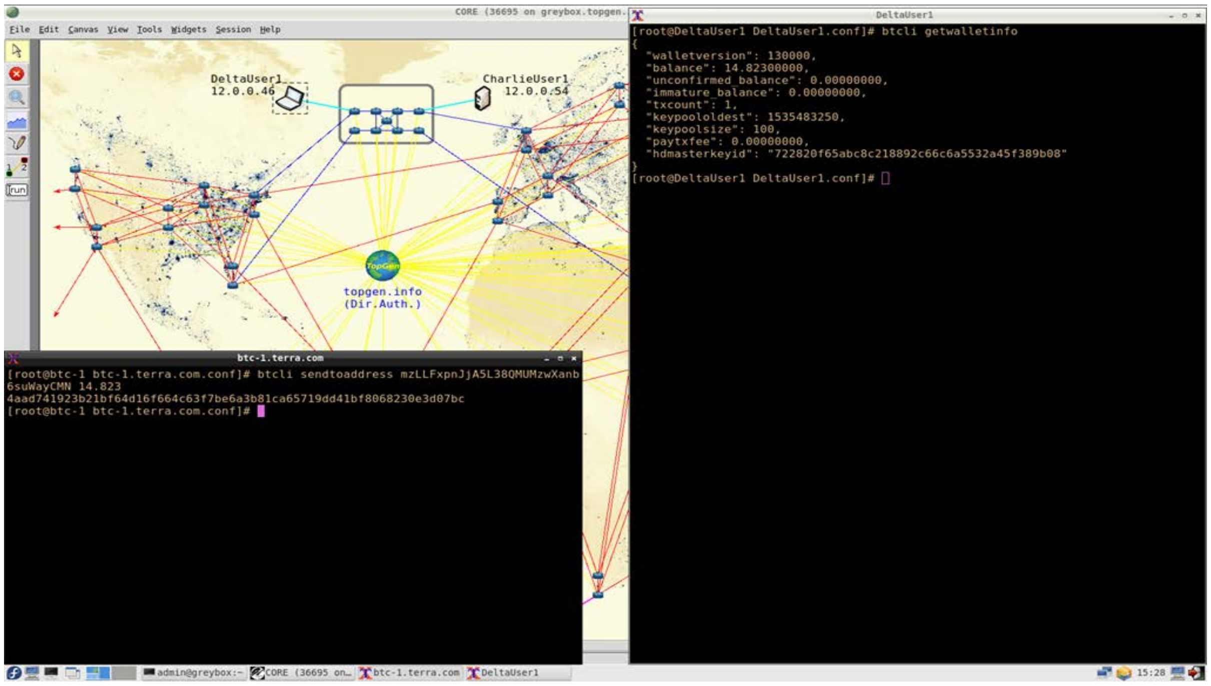


```
[root@btc-1 btc-1.terra.com.conf]# btcli getwalletinfo
{
  "walletversion": 130000,
  "balance": 1850.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 1025.00000000,
  "txcount": 69,
  "keypoololdest": 1535482256,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "6dc44f9fb26336764d6d0476183c5ec7d56361c0"
}
[root@btc-1 btc-1.terra.com.conf]#
```

```
[root@DeltaUser1 DeltaUser1.conf]# btcli getnewaddress
mzLLFxpnJjA5L38QMUMzwXanb6suWayCMN
[root@DeltaUser1 DeltaUser1.conf]#
```

**022 To receive some money I would pick a random public key that's called an address and hand it over to whoever I'm expecting to pay me.

Image



```
[root@DeltaUser1 DeltaUser1.conf]# btcli getwalletinfo
{
  "walletversion": 130000,
  "balance": 14.82300000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 1,
  "keypoololdest": 1535483250,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "722820f65abc8c218892c66c6a5532a45f389b08"
}
[root@DeltaUser1 DeltaUser1.conf]#
```

```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

**023 And they would issue a transaction where they say pay twelve or fourteen point eight whatever bitcoin to that address. And once that happened, after the block containing that transaction was mined, then my wallet would automatically show that now I have some money. I've been paid fourteen point eight something bitcoins.

Presenter: So, on that because I've seen a couple interesting things. First of all, when you started with the looking at the individual mining block, and you looked at one of the blocks, I saw there was one of the pieces of words were on there, one of the strings, said difficulty, which I recognized back when I was giving the intro earlier. That's one of-- it's the same-- just as I showed one from the actual bitcoin blockchain that difficulty is one of the parameters, difficulty is a parameter here because these blocks are identical to the ones I'm looking at. It's just that those ones are real.

Presenter: Right.

Presenter: And this one here is pretend so to speak.

Presenter: And in particular, since you mentioned difficulty, the goal of the bitcoin network in particular, but any mining based blockchain, is to have blocks generated at more or less regular predictable intervals that are wide enough apart to give the rest of the network a chance to double check the transactions and ignore the would-be liars and cheaters, right. So, you don't want too many blocks coming at you too fast because you want to be able to double check each one as you receive it. So, the difficulty increases when computing power in the collective network increases and the rate of block arrivals gets down to too fast. So, then the difficulty would go up so that miners would all have to work harder and the sort of average interval between block arrivals would go back toward the original goal, which again in bitcoin is ten minutes. So, again if things get too easy, for instance, a bunch of miners drop out because they lost interest, the difficulty would go down so that again it wouldn't take too long between subsequent blocks arriving.

Presenter: And to go on one of the other things you mentioned, there are-- each user has multiple wallets. So, the fact that I'm trying to get money on the blockchain, I don't have to just say send money to this address. I can use multiple addresses.

Presenter: You have one wallet, but the wallet itself has multiple sub-wallets, basically key pairs. And so, a public key or an address is where someone sends money to, but you would be having a hundred different addresses--

Presenter: In response to that.

Presenter: That you could-- you could randomly pick one for anyone who you would be doing business with or who you would be receiving money from so that people can't, or would have a harder time figuring out, that they're paying the same person.

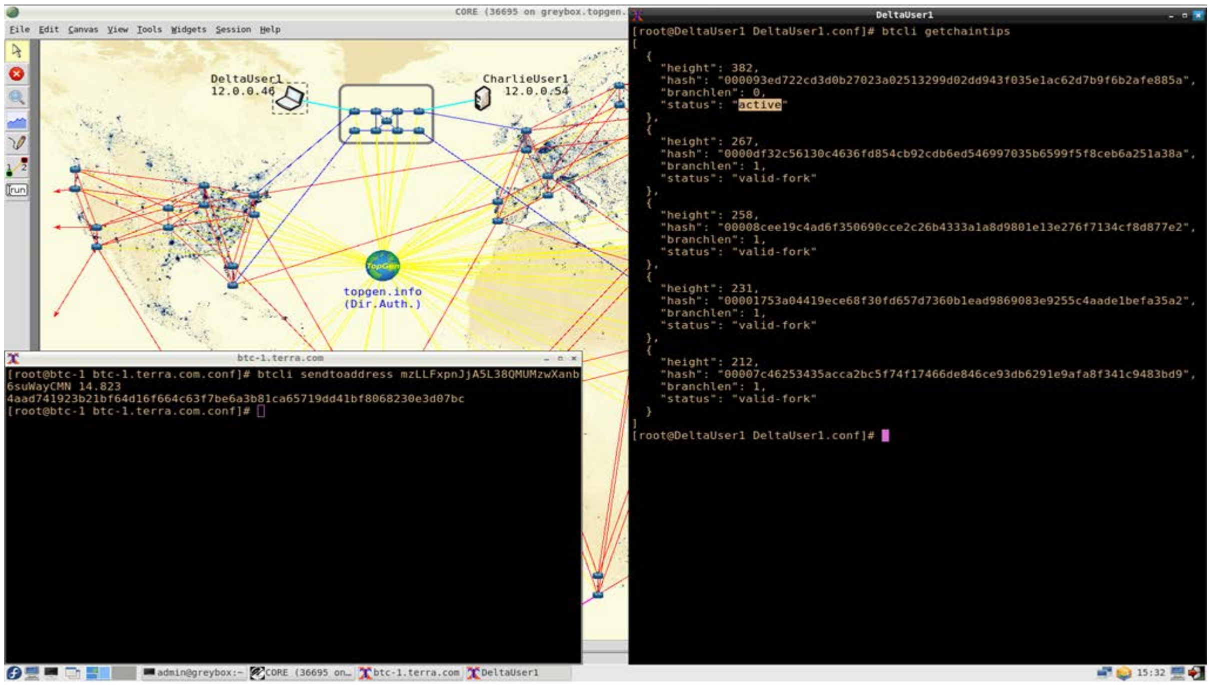
Presenter: And it's interesting because one thing which I don't think we've included too much in this conversation, if you guys want to hear about it, please put it in the comments, but there's a privacy aspect to blockchain with bitcoin, which you mentioned earlier, especially with the current bitcoin network, which is by far the most popular payment method right now based on blockchain. There's really no privacy. And there's been a lot of research demonstrating that even if I have these different hundred wallets that I can advertise or one wallet with a hundred different addresses, it's still pretty easy to figure out who people are.

Presenter: The difficulty of tracing you through the bitcoin network is just quantitatively made harder by the fact that you have a hundred keys. It's not qualitatively harder. It's basically still open to graph analysis. And we could probably figure out if you issue enough transactions that well, all those several different keys you've been using still kind of belong to the same person depending on having accumulated enough data about your financial activities on the blockchain.

Presenter: So, to get from there-- so, I guess once you have this network, and you're trying to use it as a teaching tool, how do you go about doing that?

Presenter: So, once I have accumulated the large number of pretty much empty blocks due to the lack of any transactions, and then I've generated a single transaction paying this one address, I've made enough noise--

Image

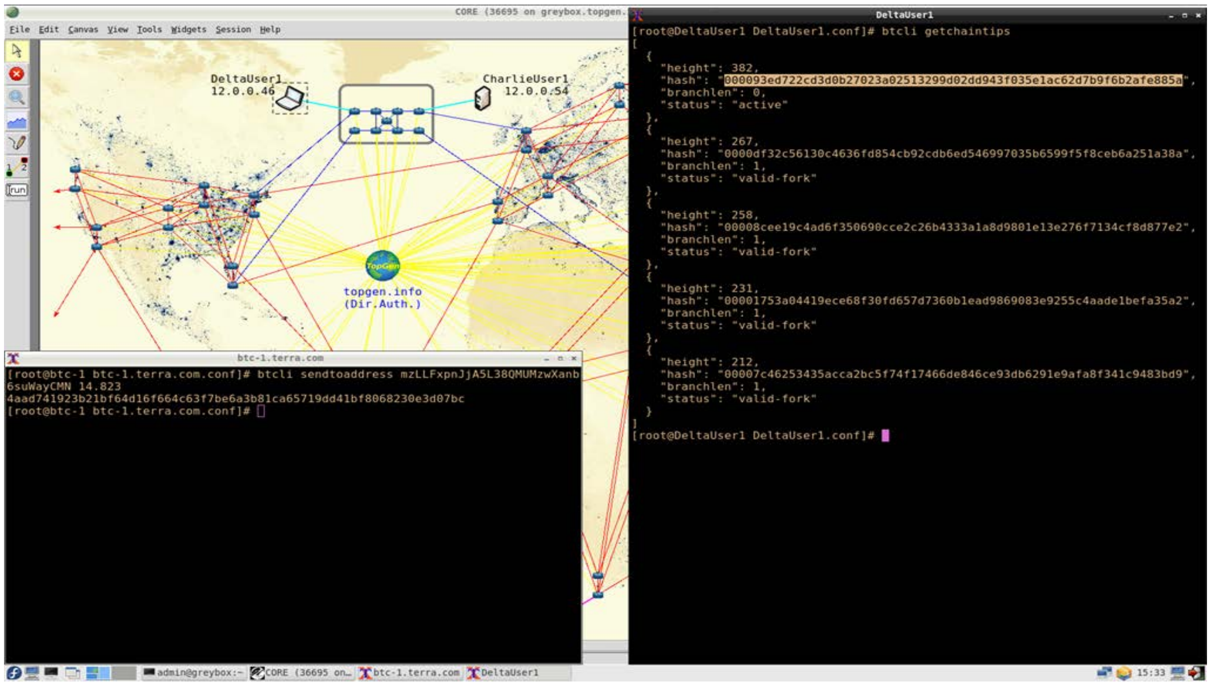


```
btc-1.terra.com
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

```
DeltaUser1
[root@DeltaUser1 DeltaUser1.conf]# btcli getchaintips
[
  {
    "height": 382,
    "hash": "000093ed722cd3d0b27023a02513299d02dd943f035e1ac62d7b9f6b2afe885a",
    "branchlen": 0,
    "status": "active"
  },
  {
    "height": 267,
    "hash": "0000df32c56130c4636fd854cb92cdb6ed546997035b6599f5f8ceb6a251a38a",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 258,
    "hash": "00008cee19c4ad6f350690cce2c26b4333a1a8d9801e13e276f7134cf8d877e2",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 231,
    "hash": "00001753a04419ece68f30fd657d7360b1ead9869083e9255c4aade1befa35a2",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 212,
    "hash": "00007c46253435acca2bc5f74f17466de846ce93db6291e9afa8f341c9483bd9",
    "branchlen": 1,
    "status": "valid-fork"
  }
]
[root@DeltaUser1 DeltaUser1.conf]# 
```

**024 In this network to be able to
be able to now just try to find that
transaction and illustrate how it's
done. So, one of the commands
would be, "Get chain tips," which
shows you the current active block
that's the latest in the blockchain,
which is what everybody's competing
to extend, build on top of.
And that block is identified by--

Image



btc-1.terra.com

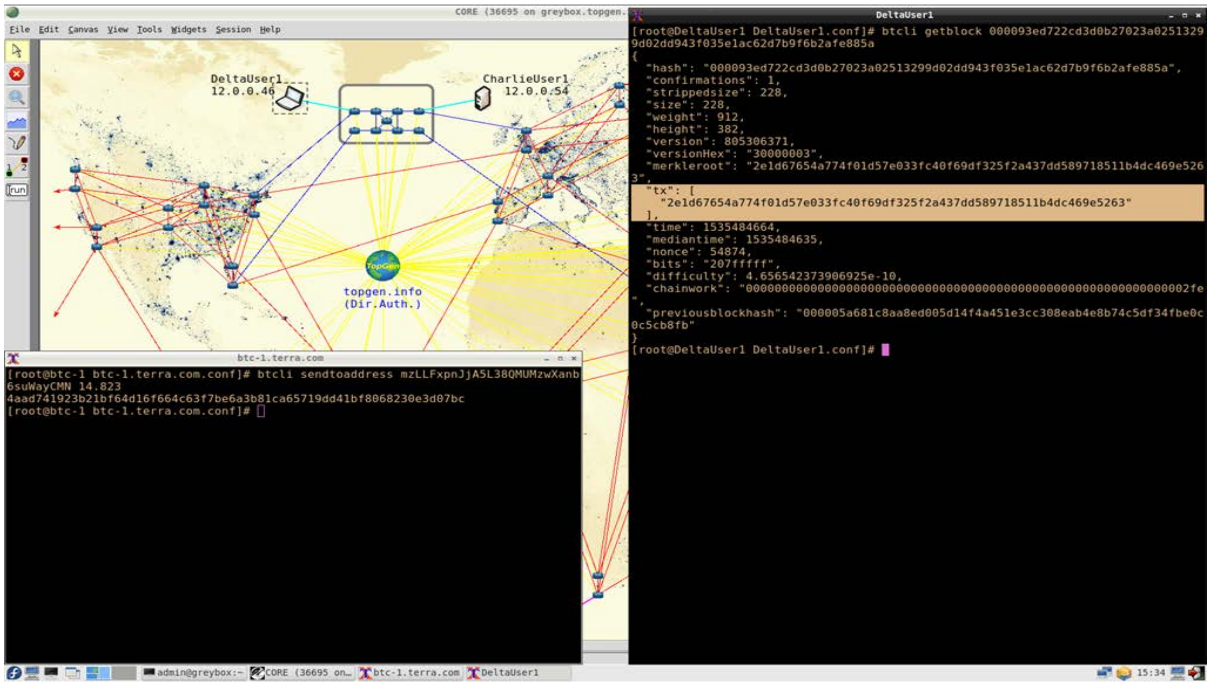
```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

DeltaUser1

```
[root@DeltaUser1 DeltaUser1.conf]# btcli getchaintips
[
  {
    "height": 382,
    "hash": "000093ed722cd3d0b27023a02513299d02dd943f035e1ac62d7b9f6b2afe885a",
    "branchlen": 0,
    "status": "active"
  },
  {
    "height": 267,
    "hash": "0000df32c56130c4636fd854cb92cdb6ed546997035b6599f5f8ceb6a251a38a",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 258,
    "hash": "00008cee19c4ad6f350690cce2c26b4333a1a8d9801e13e276f7134cf8d877e2",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 231,
    "hash": "00001753a04419ece68f30fd657d7360b1ead9869083e9255c4aade1befa35a2",
    "branchlen": 1,
    "status": "valid-fork"
  },
  {
    "height": 212,
    "hash": "00007c46253435acca2bc5f74f17466de846ce93db6291e9afa8f341c9483bd9",
    "branchlen": 1,
    "status": "valid-fork"
  }
]
[root@DeltaUser1 DeltaUser1.conf]# 
```

**025 A block ID or a block hash. And
we can then--

Image



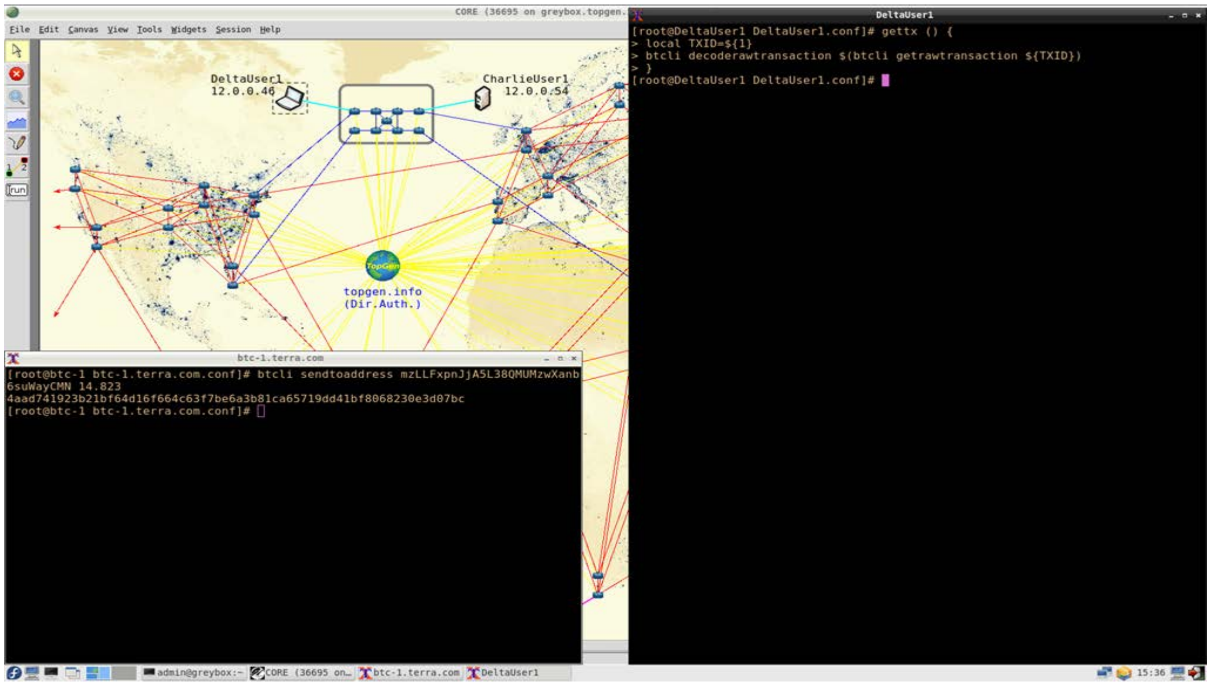
```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

```
[root@DeltaUser1 DeltaUser1.conf]# btcli getblock 000093ed722cd3d0b27023a0251329
9d02dd943f035e1ac62d7b9f6b2afe885a
{
  "hash": "000093ed722cd3d0b27023a02513299d02dd943f035e1ac62d7b9f6b2afe885a",
  "confirmations": 1,
  "strippedsize": 228,
  "size": 228,
  "weight": 912,
  "height": 382,
  "version": 805306371,
  "versionHex": "30000003",
  "merkleroot": "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e526
3",
  "tx": [
    "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e5263"
  ],
  "time": 1535484664,
  "mediantime": 1535484635,
  "nonce": 54874,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000002fe
",
  "previousblockhash": "000005a681c8aa8ed005d14f4a451e3cc308eab4e8b74c5df34fbe0c
0c5cb8fb"
}
[root@DeltaUser1 DeltaUser1.conf]#
```

**026 Get block on that hash and dump
the header information, which kind of looks
a lot like sort of the ASCII version of
the picture you had of the block header.
It has a whole bunch of metadata
about the block.

It has the height of
the block in the chain and a
comprehensive list of transactions
which is highlighted in the figure
right now, in the image. As you can
see, there's only one transaction.
This is the coin-based transaction.
This is a transaction that the miner
issued to pay themselves for being
the victor of this particular race with
the other miners and being the one
who found the nonce first and then
being the one who gets to extend the
block chain at this level, at this
height.

Image

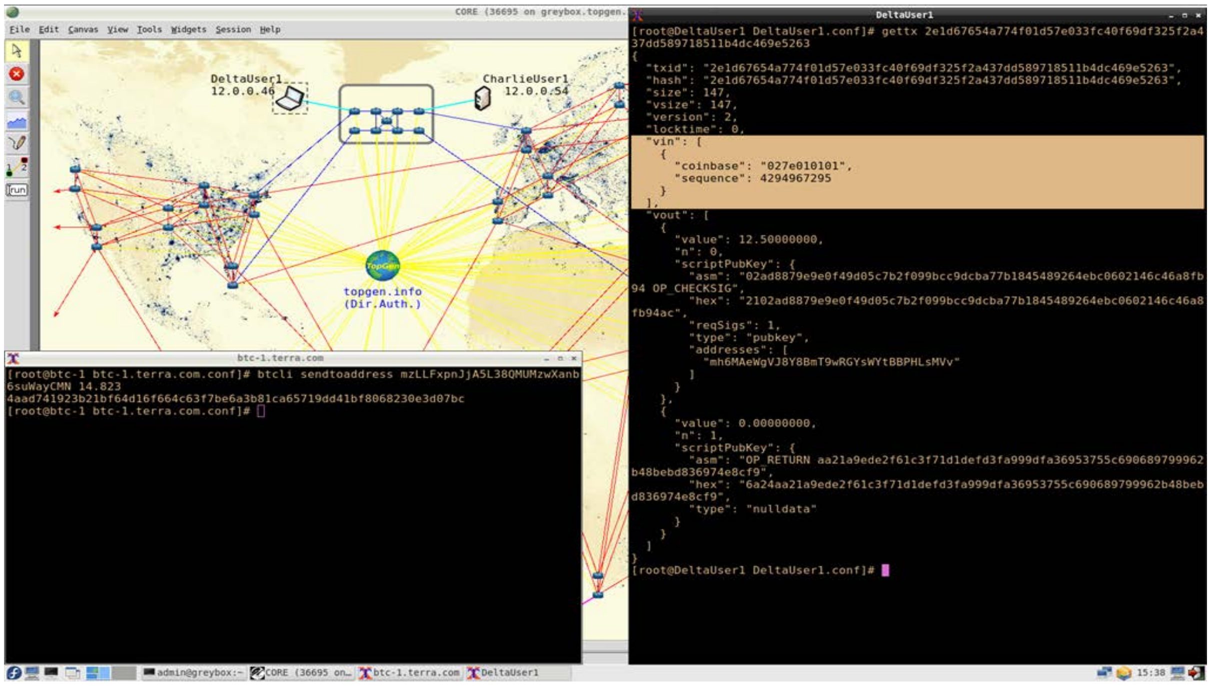


```
[root@DeltaUser1 DeltaUser1.conf]# gettx () {
> local TXID=${1}
> btcli decoderawtransaction $(btcli getrawtransaction ${TXID})
> }
[root@DeltaUser1 DeltaUser1.conf]#
```

```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

**027 And so, if we want to see what the transaction looks like, there's actually two commands. One is to get the hexadecimal encoded transaction data out of the blockchain. And the other one is to actually parse that. So, we combine them in a shell function.

Image

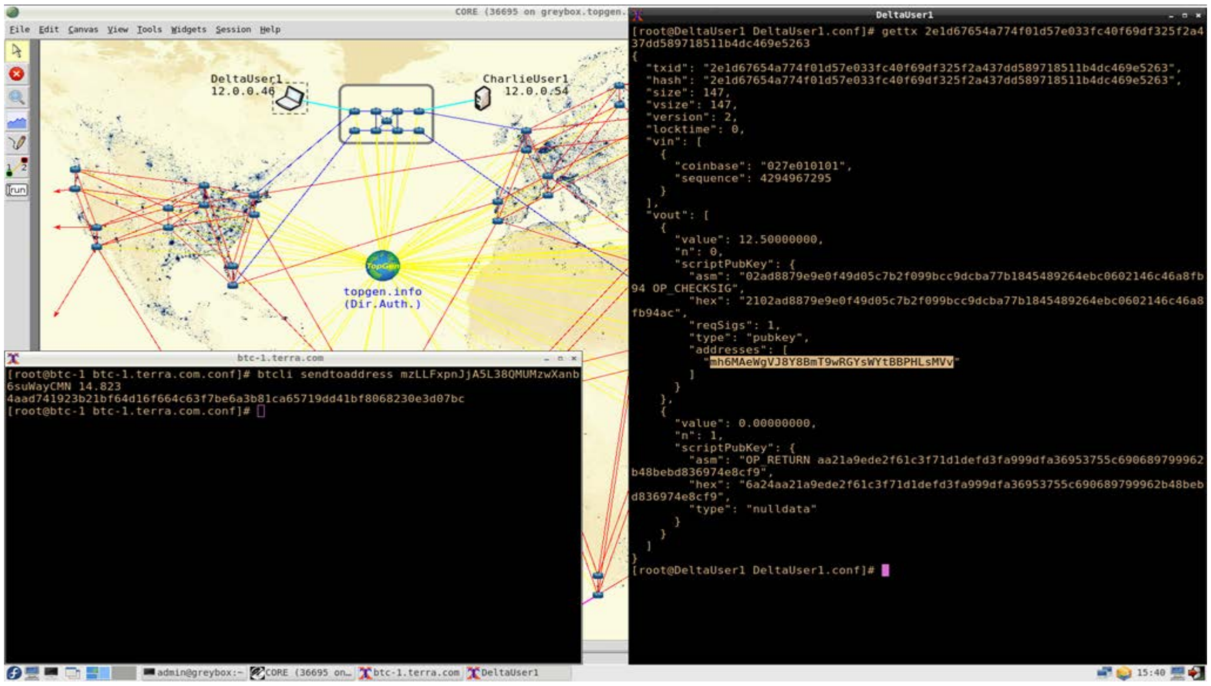


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

```
[root@DeltaUser1 DeltaUser1.conf]# gettx 2e1d67654a774f01d57e033fc40f69df325f2a4
37dd589718511b4dc469e5263
{
  "txid": "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e5263",
  "hash": "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e5263",
  "size": 147,
  "vsize": 147,
  "version": 2,
  "locktime": 0,
  "vin": [
    {
      "coinbase": "027e010101",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 12.50000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "02ad8879e9e0f49d05c7b2f099bcc9dcba77b1845489264ebc0602146c46a8fb
94 OP_CHECKSIG",
        "hex": "2102ad8879e9e0f49d05c7b2f099bcc9dcba77b1845489264ebc0602146c46a8
fb94ac",
        "reqSigs": 1,
        "type": "pubkey",
        "addresses": [
          "mh6MAeWgVJ8Y8BmT9wRGYsWYtBBPHLsMVv"
        ]
      }
    },
    {
      "value": 0.00000000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_RETURN aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962
b48bebd836974e8cf9",
        "hex": "6a24aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962b48beb
d836974e8cf9",
        "type": "nulldata"
      }
    }
  ]
}
[root@DeltaUser1 DeltaUser1.conf]#
```

**028 And then if we run it at get tx on the coin-based transaction from the latest block that we've identified, we can see that, as in the input section vn, there is no prior transaction. We're not cashing in some other transaction's output and forwarding that money to different outputs. We basically printed that money ourselves.

Image

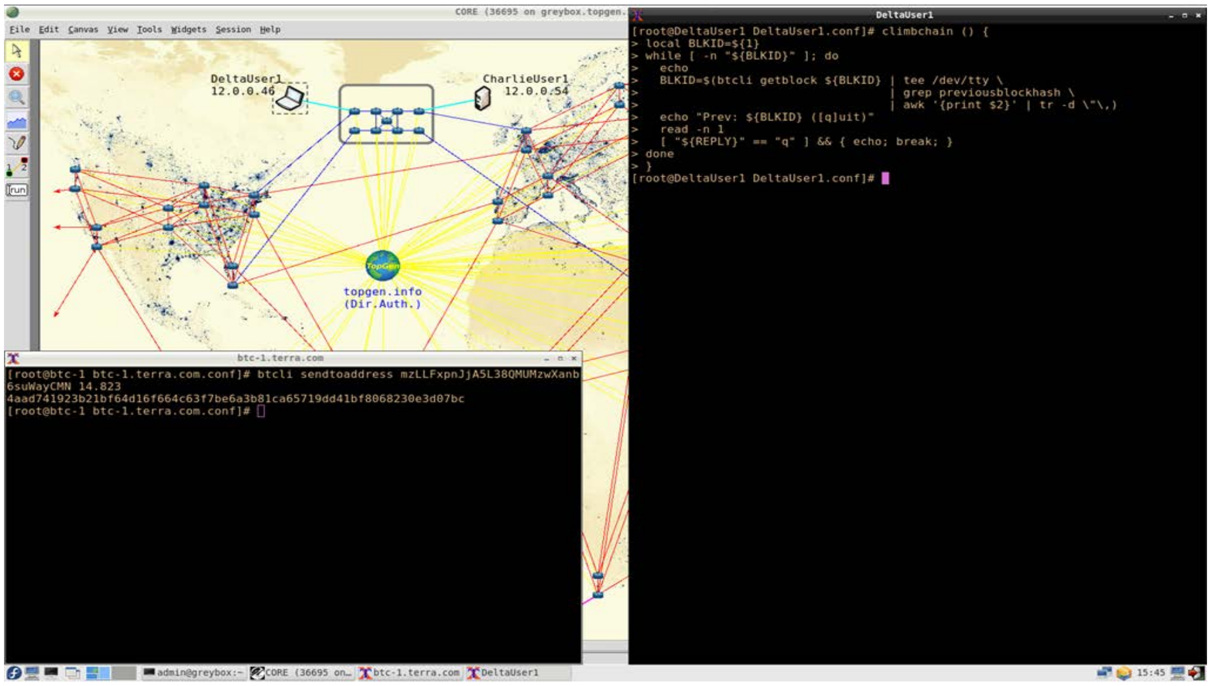


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

```
[root@DeltaUser1 DeltaUser1.conf]# gettx 2e1d67654a774f01d57e033fc40f69df325f2a4
37dd589718511b4dc469e5263
{
  "txid": "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e5263",
  "hash": "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e5263",
  "size": 147,
  "vsize": 147,
  "version": 2,
  "locktime": 0,
  "vin": [
    {
      "coinbase": "027e010101",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 12.50000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "02ad8879e9e0f49d05c7b2f099bcc9dcba77b1845489264ebc0602146c46a8fb
94 OP_CHECKSIG",
        "hex": "2102ad8879e9e0f49d05c7b2f099bcc9dcba77b1845489264ebc0602146c46a8
fb94ac",
        "reqSigs": 1,
        "type": "pubkey",
        "addresses": [
          "mh6MAeWgVJ8Y8BmT9wRGYsWYtBBPHLsMVv"
        ]
      }
    },
    {
      "value": 0.00000000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_RETURN aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962
b48bebd836974e8cf9",
        "hex": "6a24aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962b48beb
d836974e8cf9",
        "type": "nulldata"
      }
    }
  ]
}
[root@DeltaUser1 DeltaUser1.conf]# 
```

**029 And that is the-- one of the addresses of the miner who mined this block. So, basically, they get to pay themselves twelve and a half bitcoin at this point, which means we're on the second halving. So, it started at fifty for a bunch of blocks then went to twenty-five for a bunch more blocks. And after a set number of blocks has been reached, we went down to twelve and a half. And that's going to keep going down every--

Image



```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

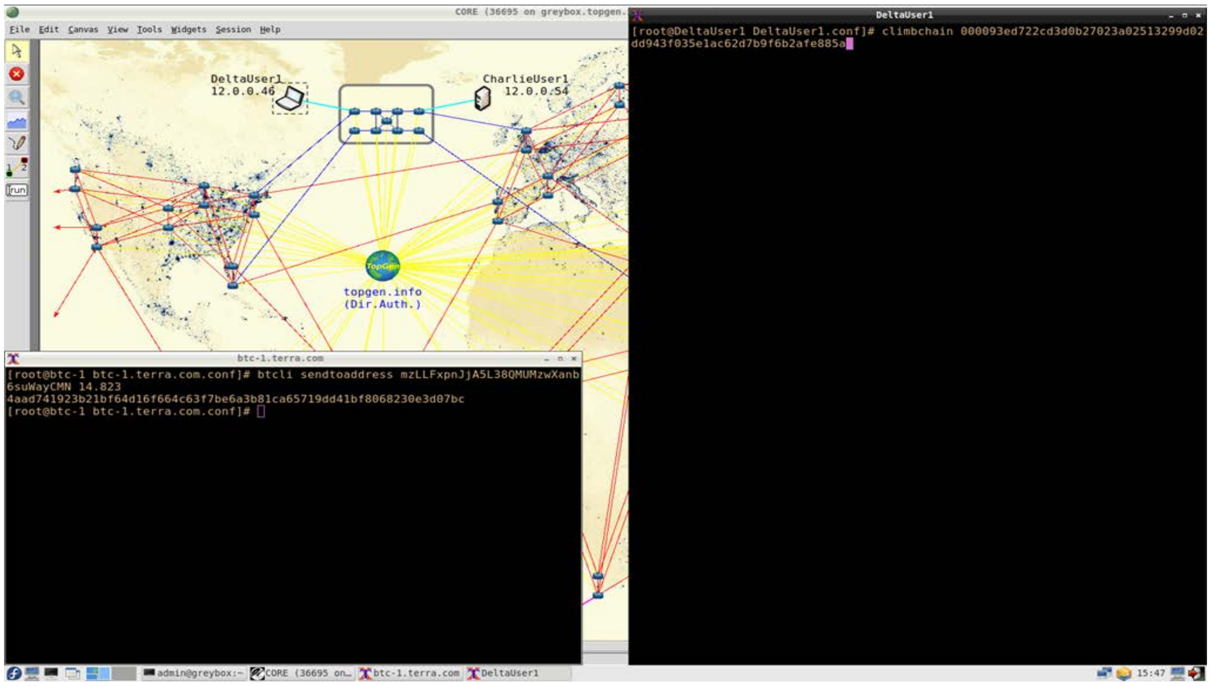
```
[root@DeltaUser1 DeltaUser1.conf]# climbchain () {
> local BLKID=${1}
> while [ -n "${BLKID}" ]; do
>   echo
>   BLKID=$(btcli getblock ${BLKID} | tee /dev/tty \
>                                   | grep previousblockhash \
>                                   | awk '{print $2}' | tr -d \"\,)
>   echo "Prev: ${BLKID} ([q]uit)"
>   read -n 1
>   [ "${REPLY}" == "q" ] && { echo; break; }
> done
> }
[root@DeltaUser1 DeltaUser1.conf]#
```

**030 Large number of blocks in the production that we're-- I think it's at least a thousand or ten thousand. I'm not exactly sure what the number is.

Presenter: So, when you run this, and you want to use this as a teaching tool, as you were talking about before, there's a forensics aspect to it, how would you go about actually using this to show people here's-- here's how you can find suspicious or malicious activity?

Presenter: So, notice on the left-hand side when we issued a payment to the non-mining node from one of the miners, there was a transaction hash or transaction ID printed out. If I had to pay off somebody with a ransomware-- for a ransomware attack to have my files unlocked, I would write down that hash and publish it on a blacklist of look, here's a list of things that happened where people got ripped off. And this was a dishonest transaction. It's tainted. If anyone tries to spend money that originated here, that's fruit of the poisoned tree. It's sort of a bad thing. If you want to be socially conscientious, you would refuse to do business with people who actually gained their money through this process or ask them uncomfortable questions if you're a law enforcement agency and sort of they happen to walk into your jurisdiction. So, we can start at the tip of the blockchain and walk down it, and look for-- and here, it's really easy because we have no transactions until we have one. So, we can walk down--

Image

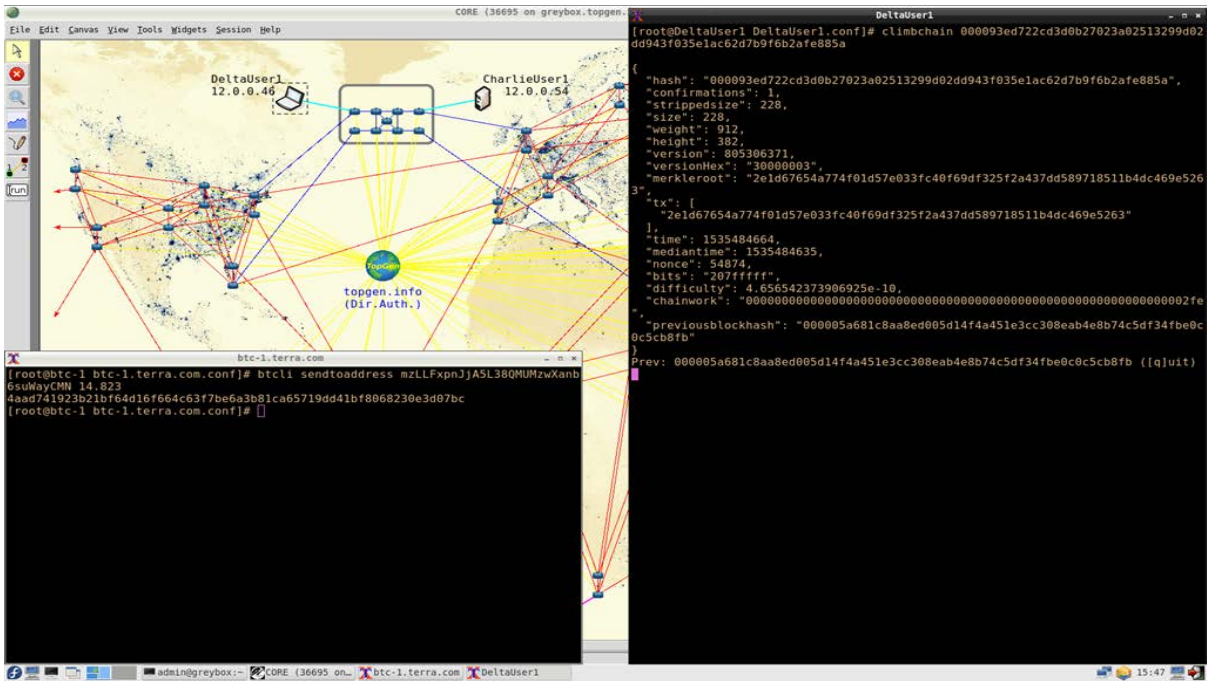


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

```
[root@DeltaUser1 DeltaUser1.conf]# climbchain 000093ed722cd3d0b27023a02513299d02
dd943f035e1ac62d7b9f6b2afe885a
```

**031 The blockchain--

Image

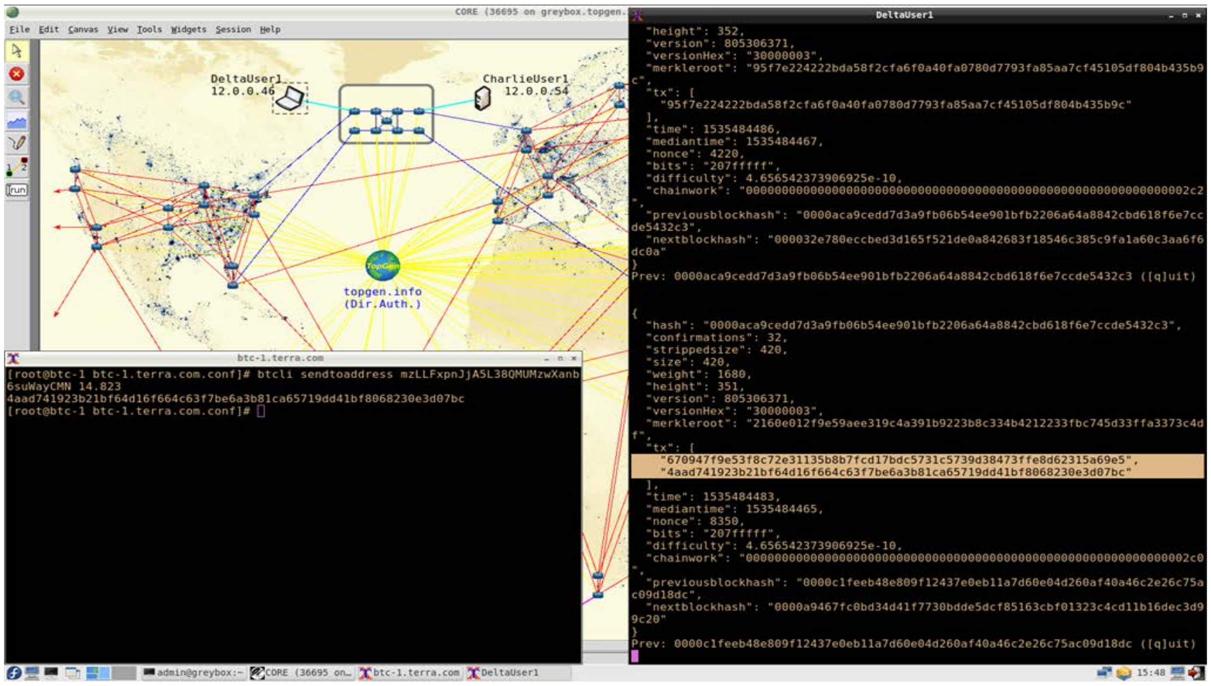


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

```
[root@DeltaUser1 DeltaUser1.conf]# climbchain 000093ed722cd3d0b27023a02513299d02
dd943f035e1ac62d7b9f6b2afe885a

{
  "hash": "000093ed722cd3d0b27023a02513299d02dd943f035e1ac62d7b9f6b2afe885a",
  "confirmations": 1,
  "strippedsize": 228,
  "size": 228,
  "weight": 912,
  "height": 382,
  "version": 805306371,
  "versionHex": "30000003",
  "merkleroot": "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e526
3",
  "tx": [
    "2e1d67654a774f01d57e033fc40f69df325f2a437dd589718511b4dc469e5263"
  ],
  "time": 1535484664,
  "mediantime": 1535484635,
  "nonce": 54874,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000002fe
",
  "previousblockhash": "000005a681c8aa8ed005d14f4a451e3cc308eab4e8b74c5df34fbe0c
0c5cb8fb"
}
Prev: 000005a681c8aa8ed005d14f4a451e3cc308eab4e8b74c5df34fbe0c0c5cb8fb ([q]uit)
```

**032 And look at each block's list of transactions. And there's usually just one--

Image

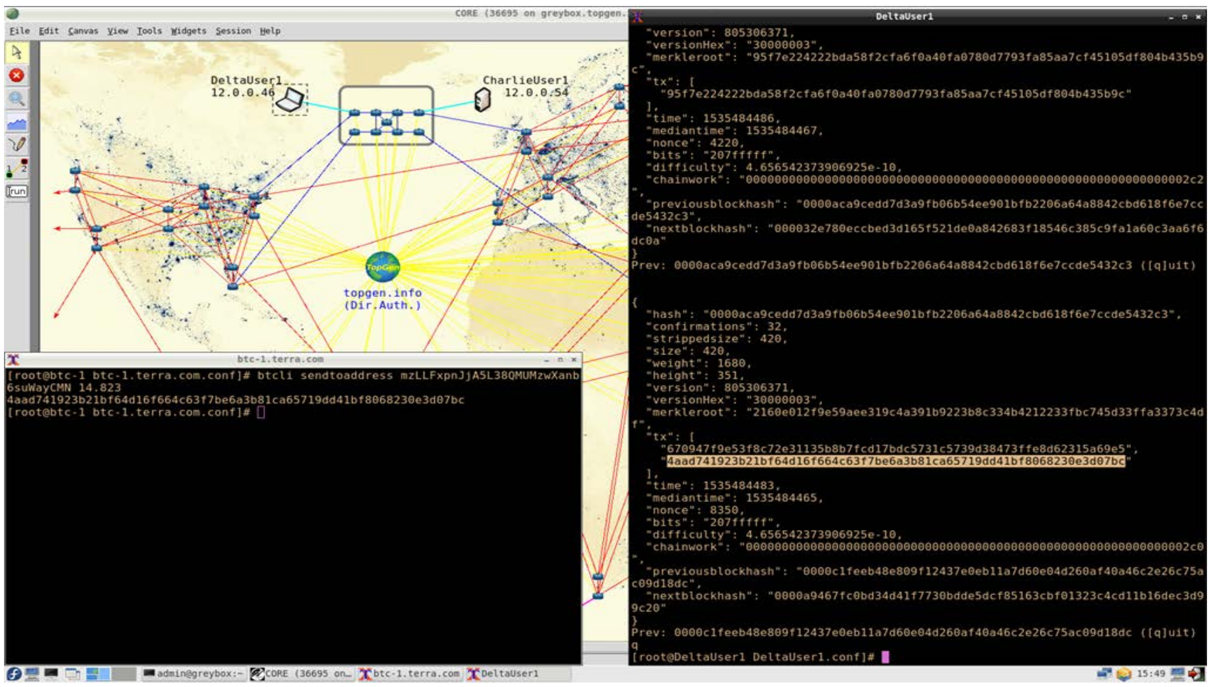


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

```
  "height": 352,
  "version": 805306371,
  "versionHex": "30000003",
  "merkleroot": "95f7e224222bda58f2cfa6f0a40fa0780d7793fa85aa7cf45105df804b435b9
c",
  "tx": [
    "95f7e224222bda58f2cfa6f0a40fa0780d7793fa85aa7cf45105df804b435b9c"
  ],
  "time": 1535484486,
  "mediantime": 1535484467,
  "nonce": 4220,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000002c2
",
  "previousblockhash": "0000aca9cedd7d3a9fb06b54ee901bfb2206a64a8842cbd618f6e7cc
de5432c3",
  "nextblockhash": "000032e780eccbed3d165f521de0a842683f18546c385c9fa1a60c3aa6f6
dc0a"
}
Prev: 0000aca9cedd7d3a9fb06b54ee901bfb2206a64a8842cbd618f6e7ccde5432c3 ([q]uit)


{
  "hash": "0000aca9cedd7d3a9fb06b54ee901bfb2206a64a8842cbd618f6e7ccde5432c3",
  "confirmations": 32,
  "strippedsize": 420,
  "size": 420,
  "weight": 1680,
  "height": 351,
  "version": 805306371,
  "versionHex": "30000003",
  "merkleroot": "2160e012f9e59aee319c4a391b9223b8c334b4212233fbc745d33ffa3373c4d
f",
  "tx": [
    "670947f9e53f8c72e31135b8b7fcd17bdc5731c5739d38473ffe8d62315a69e5",
    "4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc"
  ],
  "time": 1535484483,
  "mediantime": 1535484465,
  "nonce": 8350,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000002c0
",
  "previousblockhash": "0000c1feeb48e809f12437e0eb11a7d60e04d260af40a46c2e26c75a
c09d18dc",
  "nextblockhash": "0000a9467fc0bd34d41f7730bdde5dcf85163cbf01323c4cd11b16dec3d9
9c20"
}
Prev: 0000c1feeb48e809f12437e0eb11a7d60e04d260af40a46c2e26c75ac09d18dc ([q]uit)
```

**033 Because the network was really quiet until we find one that actually has two transactions. And there's the coin-based. And then there's the same transaction ID that we generated on the left.

Image

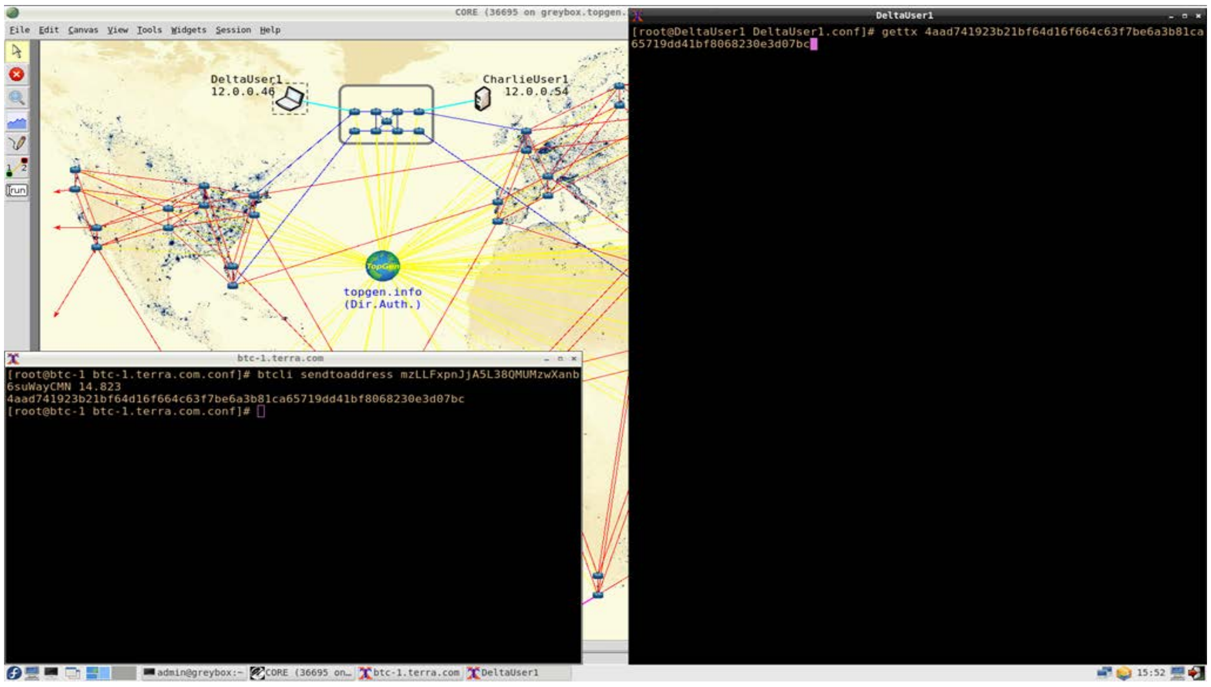


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

```
  "version": 805306371,
  "versionHex": "30000003",
  "merkleroot": "95f7e224222bda58f2cfa6f0a40fa0780d7793fa85aa7cf45105df804b435b9
c",
  "tx": [
    "95f7e224222bda58f2cfa6f0a40fa0780d7793fa85aa7cf45105df804b435b9c"
  ],
  "time": 1535484486,
  "mediantime": 1535484467,
  "nonce": 4220,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000002c2
",
  "previousblockhash": "0000aca9cedd7d3a9fb06b54ee901bfb2206a64a8842cbd618f6e7cc
de5432c3",
  "nextblockhash": "000032e780eccbed3d165f521de0a842683f18546c385c9fa1a60c3aa6f6
dc0a"
}
Prev: 0000aca9cedd7d3a9fb06b54ee901bfb2206a64a8842cbd618f6e7ccde5432c3 ([q]uit)


{
  "hash": "0000aca9cedd7d3a9fb06b54ee901bfb2206a64a8842cbd618f6e7ccde5432c3",
  "confirmations": 32,
  "strippedsize": 420,
  "size": 420,
  "weight": 1680,
  "height": 351,
  "version": 805306371,
  "versionHex": "30000003",
  "merkleroot": "2160e012f9e59aee319c4a391b9223b8c334b4212233fbc745d33ffa3373c4d
f",
  "tx": [
    "670947f9e53f8c72e31135b8b7fcd17bdc5731c5739d38473ffe8d62315a69e5",
    "4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc"
  ],
  "time": 1535484483,
  "mediantime": 1535484465,
  "nonce": 8350,
  "bits": "207fffff",
  "difficulty": 4.656542373906925e-10,
  "chainwork": "00000000000000000000000000000000000000000000000000000000000002c0
",
  "previousblockhash": "0000c1feeb48e809f12437e0eb11a7d60e04d260af40a46c2e26c75a
c09d18dc",
  "nextblockhash": "0000a9467fc0bd34d41f7730bdde5dcf85163cbf01323c4cd11b16dec3d9
9c20"
}
Prev: 0000c1feeb48e809f12437e0eb11a7d60e04d260af40a46c2e26c75ac09d18dc ([q]uit)
q
[root@DeltaUser1 DeltaUser1.conf]#
```

**034 So, we can now inspect that--

Image

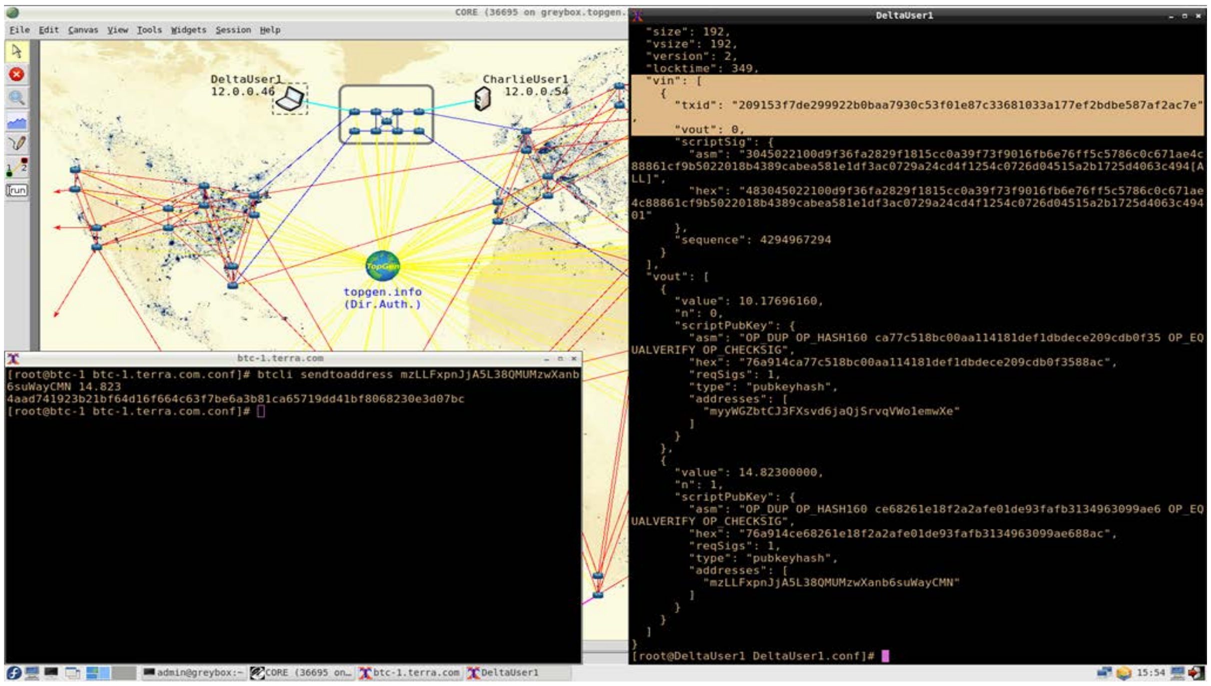


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

```
[root@DeltaUser1 DeltaUser1.conf]# gettx 4aad741923b21bf64d16f664c63f7be6a3b81ca
65719dd41bf8068230e3d07bc
```

**035 Transaction with our get tx command--

Image

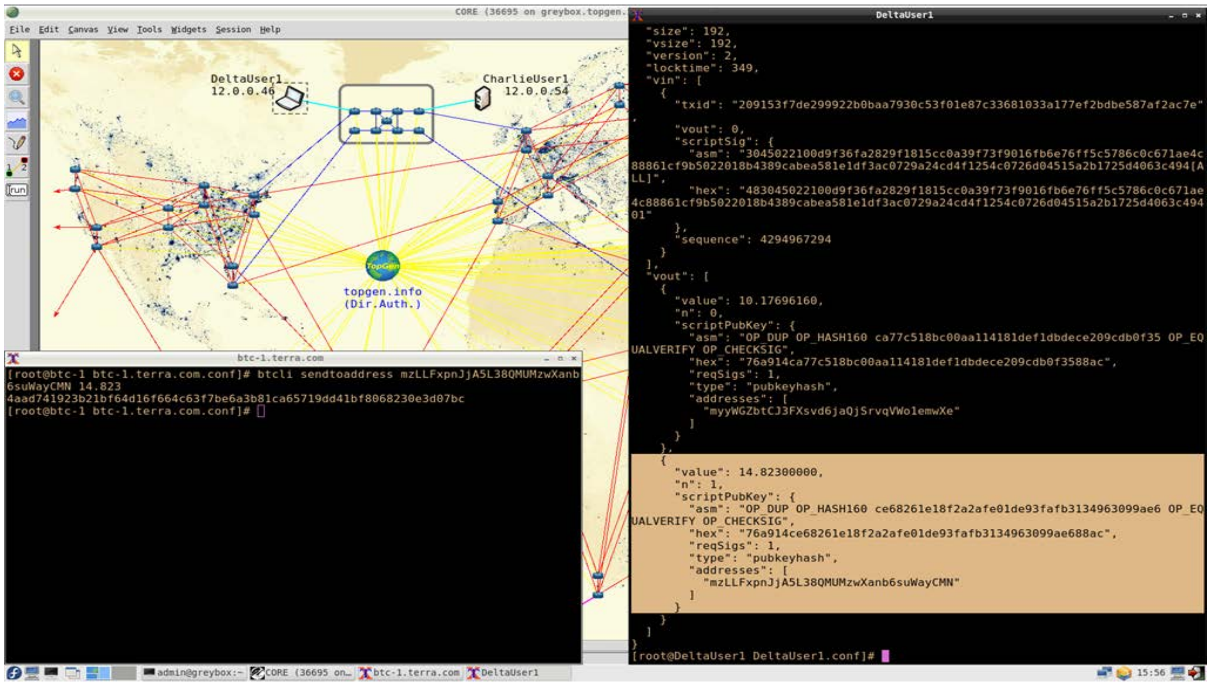


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]#
```

```
  "size": 192,
  "vsize": 192,
  "version": 2,
  "locktime": 349,
  "vin": [
    {
      "txid": "209153f7de299922b0baa7930c53f01e87c33681033a177ef2bdbe587af2ac7e"
,
      "vout": 0,
      "scriptSig": {
        "asm": "3045022100d9f36fa2829f1815cc0a39f73f9016fb6e76ff5c5786c0c671ae4c
88861cf9b5022018b4389cabea581e1df3ac0729a24cd4f1254c0726d04515a2b1725d4063c494[A
LL]",
        "hex": "483045022100d9f36fa2829f1815cc0a39f73f9016fb6e76ff5c5786c0c671ae
4c88861cf9b5022018b4389cabea581e1df3ac0729a24cd4f1254c0726d04515a2b1725d4063c494
01"
      },
      "sequence": 4294967294
    }
  ],
  "vout": [
    {
      "value": 10.17696160,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ca77c518bc00aa114181def1dbdece209cdb0f35 OP_EQ
UALVERIFY OP_CHECKSIG",
        "hex": "76a914ca77c518bc00aa114181def1dbdece209cdb0f3588ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "myyWGZbtCJ3FXsvd6jaQjSrvqVWo1emwXe"
        ]
      }
    },
    {
      "value": 14.82300000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ce68261e18f2a2afe01de93fafb3134963099ae6 OP_EQ
UALVERIFY OP_CHECKSIG",
        "hex": "76a914ce68261e18f2a2afe01de93fafb3134963099ae688ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "mzLLFxpnJjA5L38QMUMzwXanb6suWayCMN"
        ]
      }
    }
  ]
}
[root@DeltaUser1 DeltaUser1.conf]#
```

**036 That we generated earlier. And here we can see this transaction actually has another one whose outputs it spends. So, it refers to the input transaction. It says, "Hey, I'm taking my money from there.

Image



btc-1.terra.com

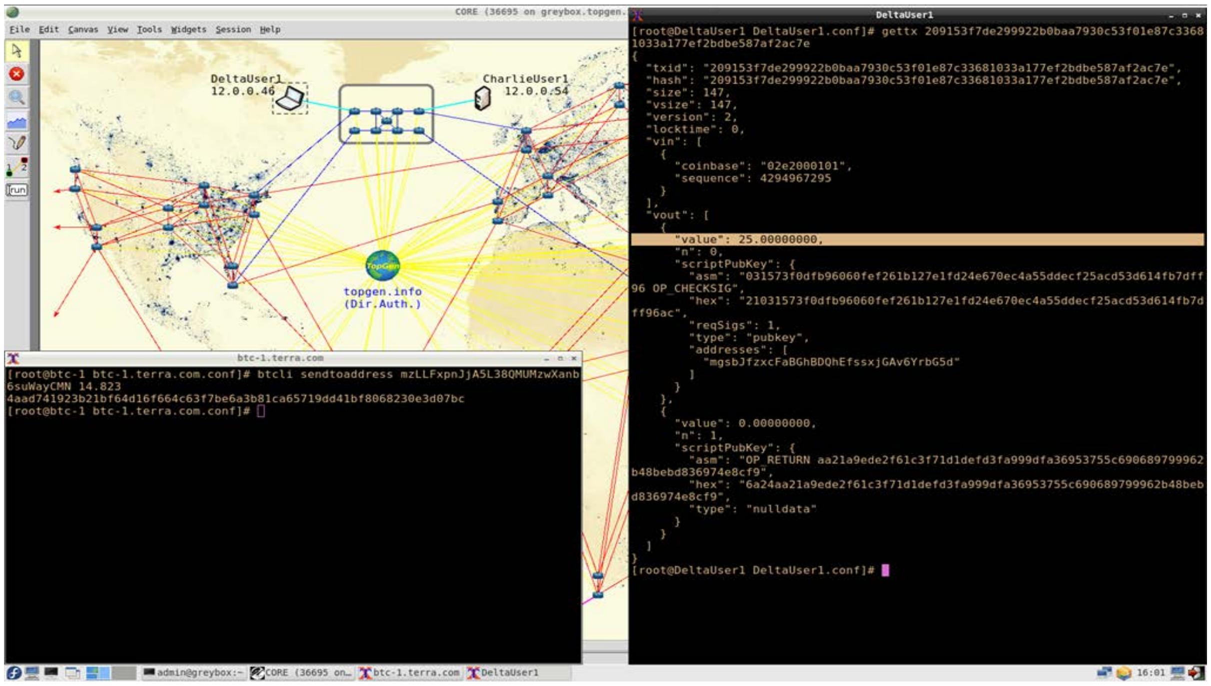
```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

DeltaUser1

```
  "size": 192,
  "vsize": 192,
  "version": 2,
  "locktime": 349,
  "vin": [
    {
      "txid": "209153f7de299922b0baa7930c53f01e87c33681033a177ef2bdbe587af2ac7e"
,
      "vout": 0,
      "scriptSig": {
        "asm": "3045022100d9f36fa2829f1815cc0a39f73f9016fb6e76ff5c5786c0c671ae4c
88861cf9b5022018b4389cabea581e1df3ac0729a24cd4f1254c0726d04515a2b1725d4063c494[A
LL]",
        "hex": "483045022100d9f36fa2829f1815cc0a39f73f9016fb6e76ff5c5786c0c671ae
4c88861cf9b5022018b4389cabea581e1df3ac0729a24cd4f1254c0726d04515a2b1725d4063c494
01"
      },
      "sequence": 4294967294
    }
  ],
  "vout": [
    {
      "value": 10.17696160,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ca77c518bc00aa114181def1dbdece209cdb0f35 OP_EQ
UALVERIFY OP_CHECKSIG",
        "hex": "76a914ca77c518bc00aa114181def1dbdece209cdb0f3588ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "myyWGZbtCJ3FXsvd6jaQjSrvqVWo1emwXe"
        ]
      }
    },
    {
      "value": 14.82300000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ce68261e18f2a2afe01de93fafb3134963099ae6 OP_EQ
UALVERIFY OP_CHECKSIG",
        "hex": "76a914ce68261e18f2a2afe01de93fafb3134963099ae688ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "mzLLFxpnJjA5L38QMUMzwXanb6suWayCMN"
        ]
      }
    }
  ]
}
[root@DeltaUser1 DeltaUser1.conf]# 
```

**037 And I am spending it here." Now, there's another output here. So, the sum of the two outputs would add up to about twenty-five-ish bitcoin. And that is because our input transaction probably was a coin-based. It was definitely a coin-based because there was no other activity in the blockchain. So, the miner who paid me took some of his or their money from the only kinds of transactions they had earlier, which was like a twenty-five dollar coin-based and then split it up basically giving me fourteen point two whatever and paying themselves the difference so that they could spend it later as part of other transactions.

Image

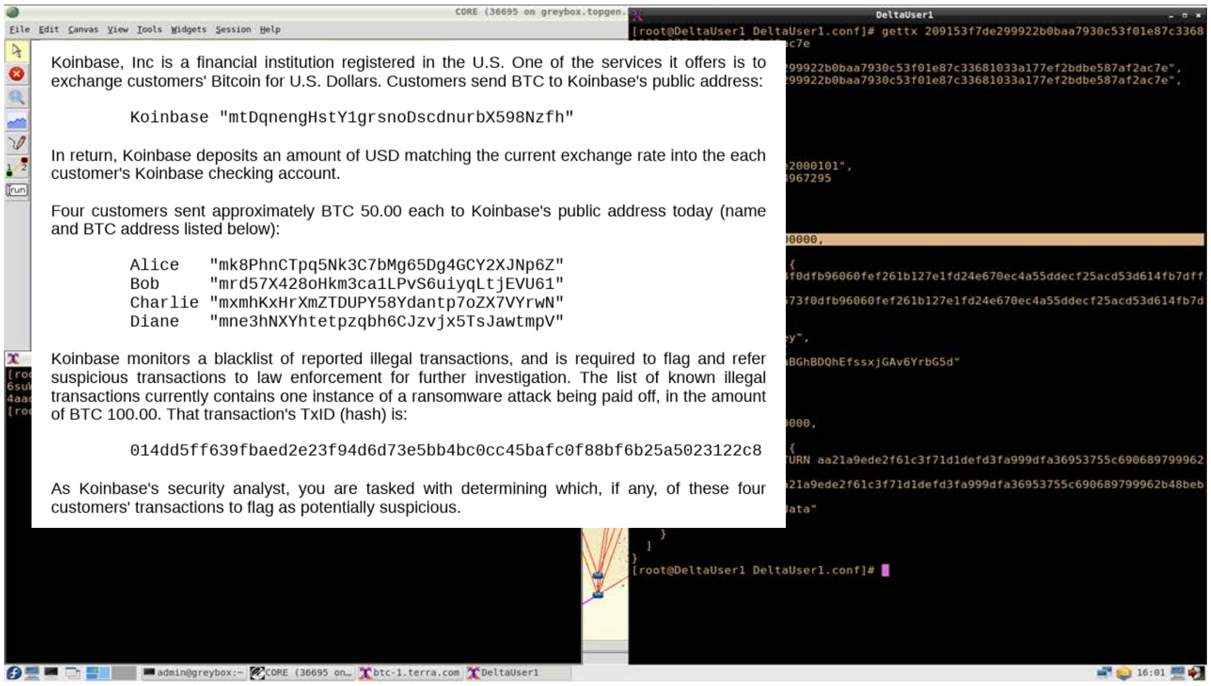


```
[root@btc-1 btc-1.terra.com.conf]# btcli sendtoaddress mzLLFxpnJjA5L38QMUMzwXanb
6suWayCMN 14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d07bc
[root@btc-1 btc-1.terra.com.conf]# 
```

```
[root@DeltaUser1 DeltaUser1.conf]# gettx 209153f7de299922b0baa7930c53f01e87c3368
1033a177ef2bdbe587af2ac7e
{
  "txid": "209153f7de299922b0baa7930c53f01e87c33681033a177ef2bdbe587af2ac7e",
  "hash": "209153f7de299922b0baa7930c53f01e87c33681033a177ef2bdbe587af2ac7e",
  "size": 147,
  "vsize": 147,
  "version": 2,
  "locktime": 0,
  "vin": [
    {
      "coinbase": "02e2000101",
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 25.00000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "031573f0dfb96060fef261b127e1fd24e670ec4a55ddecf25acd53d614fb7dff
96 OP_CHECKSIG",
        "hex": "21031573f0dfb96060fef261b127e1fd24e670ec4a55ddecf25acd53d614fb7d
ff96ac",
        "reqSigs": 1,
        "type": "pubkey",
        "addresses": [
          "mgsbJfzxcFaBGhBDQhEfssxjGAv6YrbG5d"
        ]
      }
    },
    {
      "value": 0.00000000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_RETURN aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962
b48bebd836974e8cf9",
        "hex": "6a24aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962b48beb
d836974e8cf9",
        "type": "nulldata"
      }
    }
  ]
}
[root@DeltaUser1 DeltaUser1.conf]# 
```

**038 The coin-based that is the input of the earlier transaction that I showed you is indeed a coin-based. And it is a transaction where the miner got paid twenty-five bitcoins.

Image



Koinbase, Inc is a financial institution registered in the U.S. One of the services it offers is to exchange customers' Bitcoin for U.S. Dollars. Customers send BTC to Koinbase's public address:

```
Koinbase "mtDqnengHstY1grsnoDscdnurbX598Nzfh"
```

In return, Koinbase deposits an amount of USD matching the current exchange rate into the each customer's Koinbase checking account.

Four customers sent approximately BTC 50.00 each to Koinbase's public address today (name and BTC address listed below):

```
Alice   "mk8PhnCTpq5Nk3C7bMg65Dg4GCY2XJNp6Z"
Bob     "mrd57X428oHkm3ca1LPvS6uiyqLtjEVU61"
Charlie "mxmhKxHrXmZTDUPY58Ydantp7oZX7VYrwN"
Diane   "mne3hNXYhtetpzqbh6CJzvjx5TsJawtmpV"
```

Koinbase monitors a blacklist of reported illegal transactions, and is required to flag and refer suspicious transactions to law enforcement for further investigation. The list of known illegal transactions currently contains one instance of a ransomware attack being paid off, in the amount of BTC 100.00. That transaction's TxID (hash) is:

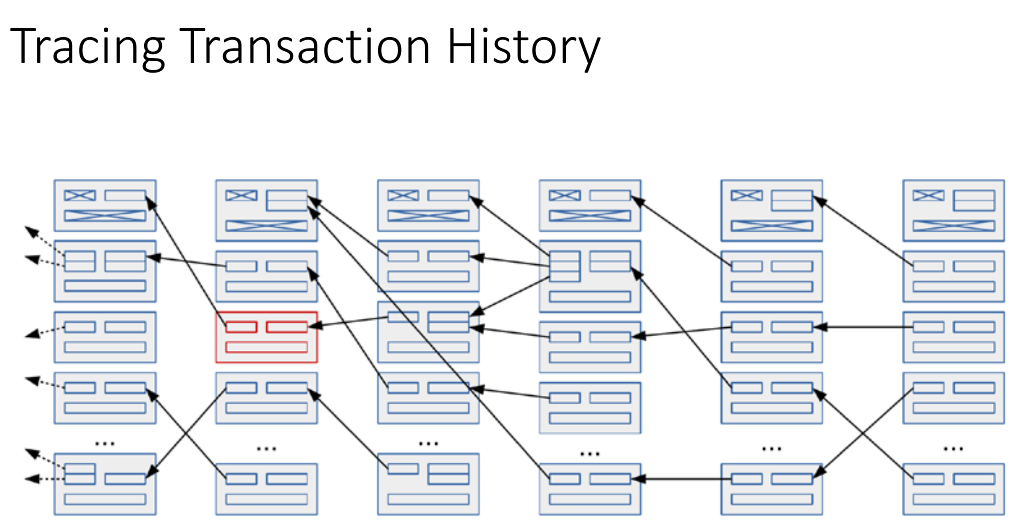
```
014dd5ff639fbaed2e23f94d6d73e5bb4bc0cc45bafc0f88bf6b25a5023122c8
```

As Koinbase's security analyst, you are tasked with determining which, if any, of these four customers' transactions to flag as potentially suspicious.

```
[root@DeltaUser1 DeltaUser1.conf]# gettx 209153f7de299922b0baa7930c53f01e87c3368
c7e
299922b0baa7930c53f01e87c33681033a177ef2bdbe587af2ac7e",
299922b0baa7930c53f01e87c33681033a177ef2bdbe587af2ac7e",
2000101",
967295
00000,
{
f0dfb96060fef261b127e1fd24e670ec4a55ddecf25acd53d614fb7dff
573f0dfb96060fef261b127e1fd24e670ec4a55ddecf25acd53d614fb7d
ey",
BGhBDQhEfssxjGAv6YrbG5d"
0000,
{
TURN aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962
21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962b48beb
data"
      }
    ]
}
[root@DeltaUser1 DeltaUser1.conf]#
```

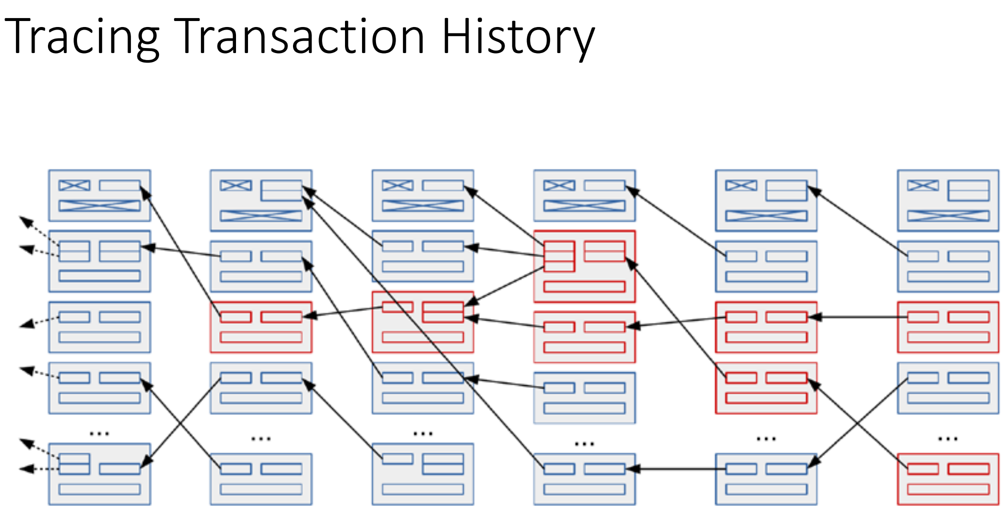
**039 And so, with this type of experience, we can now actually ask trainees to solve puzzles of the form illustrated here. Some number of customers walk into a shop that has-- well, virtually walk into a shop that is under U.S. government jurisdiction, and they have the know your customer's laws. And they have to be able to tell when they're issuing real U.S. dollars in exchange for whatever the user pays, bitcoin in this case. They have to be able to a flag forensic analyst, or this bank or financial institution would have to be able to refer transactions for further investigation to law enforcement if they're suspicious of the IDs. There was a bad transaction in the past where somebody had to pay off a ransomware attack, or it was a drug deal, whatever it was. And the question is, "Who, out of those four users, have that bad transaction in their ancestry?"

Tracing Transaction History



**040 And the idea is if you flag a transaction somewhere earlier in the blockchain, then any of its descendants--

Tracing Transaction History


Tracing Transaction History

**041 Are tainted by it. And so, the idea is, at the very end, or at the current level, if somebody is subject to the jurisdiction of the investigator, they can be asked questions. Where did you get your money? How did you arrive to be in possession of these bitcoins because we know originally, or back further back in the blockchain--

Presenter: Something was wrong.

Presenter: You're spending money that was ill-gotten gains in a way of speaking?

Presenter: And to a certain extent, I mean this is exactly-- you have this in the smaller scale of a virtual network. But this is exactly what happens when you're doing large-scale investigations with blockchain fraud.

Presenter: All of this information is one hundred percent public except the real blockchain is-- was a few months ago a hundred and seventy or a hundred and ninety gigabytes in size compressed. So, I believe it must have grown somewhat since then.

Presenter: But even to that point, if you make a transaction on the blockchain, and someone else receives it and then spends the money again, even though we have this hundred and ninety gigabyte database or probably now who knows how big it is, it's still this transaction literally points directly to where I need to look next.

Presenter: Right.

Presenter: And that one points where I need to go after that. And when you do forensics, all this information is public.

Presenter: It's an exercise in six degrees of Kevin Bacon essentially. We're trying to figure out how many intermediate steps were between the bad event that we wrote down as a transaction number and the money you're trying to spend currently that's under scrutiny.

Presenter: And one of the interesting cases which came out of that when we-- when there was all the recent attacks with the crypto-- what was it called? The ransomware.

Presenter: Right.

Presenter: The word we're trying to find earlier. When there were all these attacks with the ransomware, so if someone says send bitcoin to this address, this is on the blockchain, which means I can literally look who sent it from which address to which address. When did that happen? What did that second address do with it? If they tried to bring it to a government exchange, now I know exactly where to go. And I can walk to that exchange and say, "Hey, who's your customer that interacted with this wallet? That person is working with stolen goods."

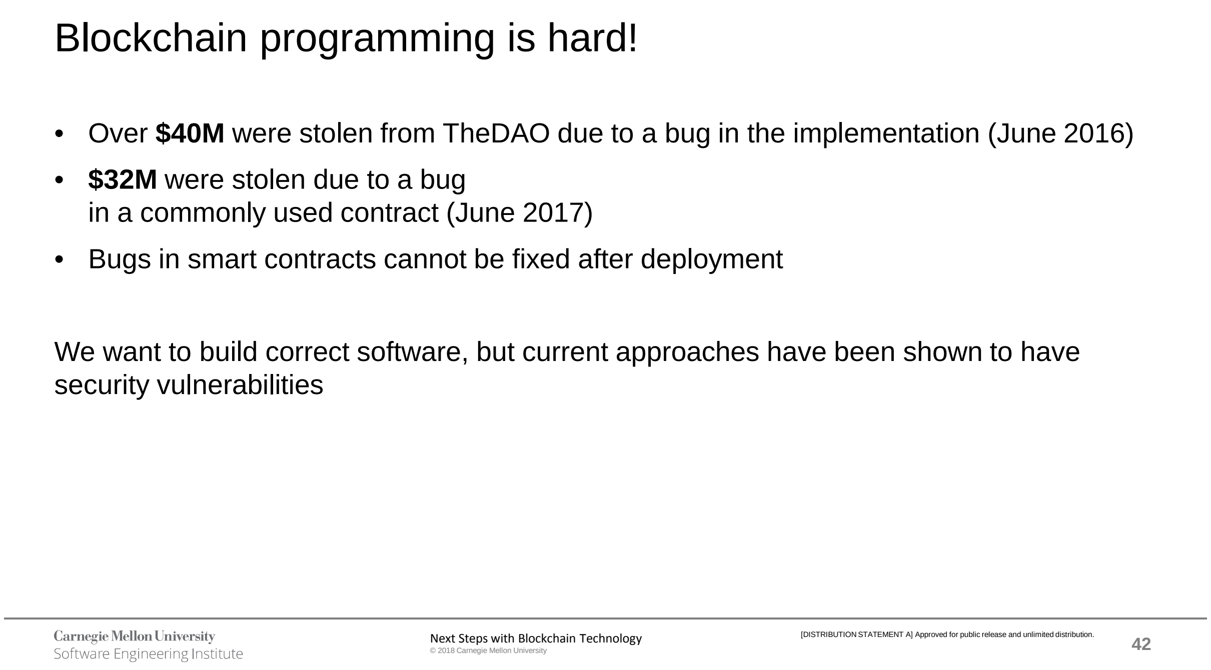
Presenter: So, your bitcoins are anonymous. And your identity is anonymous as long as you completely stay on the blockchain and never try to exchange your bitcoin for--

Presenter: Fiat currency.

Presenter: Fiat currency under the juris-- in a jurisdiction that actually cares to see and sort of pay attention to-- or maintains a blacklist.

Presenter: Right, and so far, I'm not familiar with any grocery chain, major grocery chain, that accepts bitcoin in exchange for bread and milk. So, the day will come.

Blockchain programming is hard!



Blockchain programming is hard!

- Over **$40M** were stolen from TheDAO due to a bug in the implementation (June 2016)
- **$32M** were stolen due to a bug in a commonly used contract (June 2017)
- Bugs in smart contracts cannot be fixed after deployment

We want to build correct software, but current approaches have been shown to have security vulnerabilities

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

42

**042 Presenter: Eventually.

Presenter: So, that's cool. So, that's a fascinating way that you can use technologies that we're developing here. And these are actually things that are developed all around to investigate fraud that happens on the blockchain. And honestly, we're talking about bitcoin. The same concept exists on larger blockchains. So, with respect to Ethereum, there have been a couple of notable instances of money theft taking place with Ether instead of bitcoin. And the same exact techniques can apply because you're just-- you're tracing what sent-- who sent money to who, who sent money to who. And you can watch that path progress.

Presenter: Now, one other point that I would like to emphasize is that generally speaking, crypto is hard for most people to deal with. And then you have situations where you store your wallet, private keys included, on a website that manages it on your behalf. And it's not the crypto that gets broken when your money get-- or your bitcoins or your Ethers get stolen. It is, generally speaking, an attack against the website that has your private key. And then the thieves get access to your private key. When they do that, they could impersonate you.

Presenter: And do anything with it.

Presenter: But that's not the cryptographic mathematics fault. That is basically just opsec.

Presenter: Right, exactly. And--

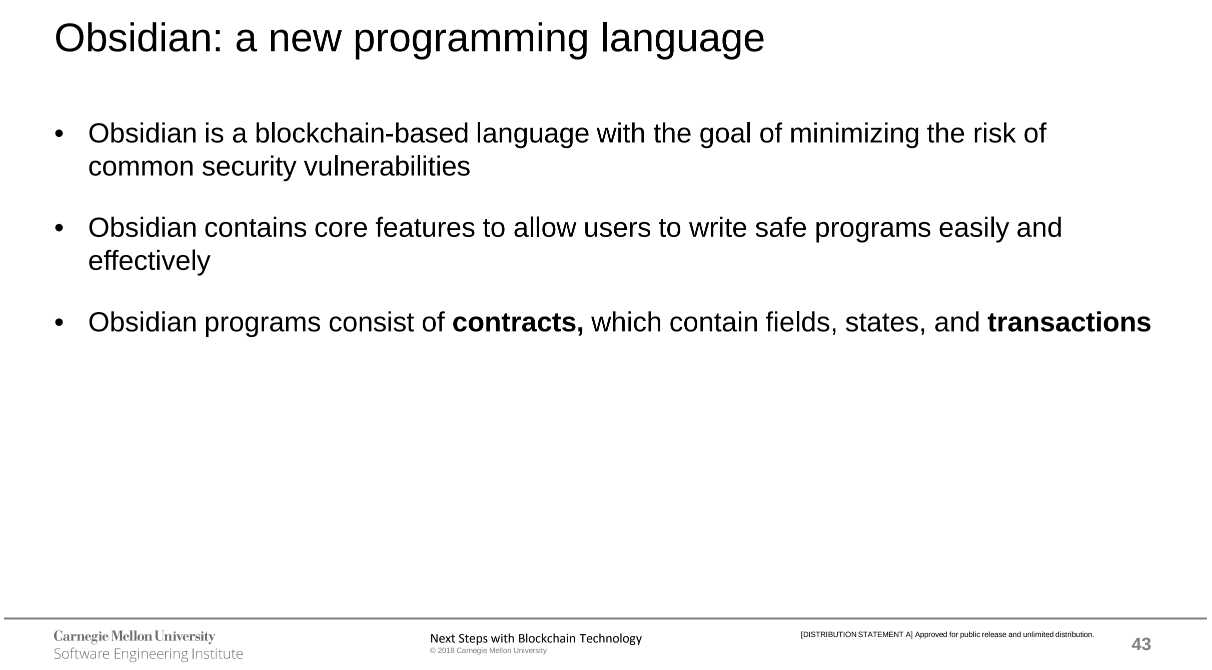
Presenter: And I say just opsec, but that's really a big, big deal that we need to focus on.

Presenter: Well, so to that point, and that's a great segue, so I want to focus on a different application, some other cool research that's going on. And this is actually an area of larger research. We're just playing a small part of it here at SEI and the broader CMU. So, as you saw on that slide, blockchain programming is hard. And we're talking about thefts, and particularly with respect to Ethereum. There have been some-- blockchain allows you to send money. Ethereum allows you to create programs. Most of the interest in the blockchain technology-- I keep on saying the blockchain technology. Most of the interest in blockchain technology comes from the ability to create computer programs of custom nature, whatever you want to do, and then implement that on a blockchain. So, to that extent--

Presenter: Colloquially, we refer to as smart contracts.

Presenter: Right. To that extent, people are looking at these abilities to make smart contracts and trying to implement all sorts of applications including one called the DAO. And they're doing it wrong. And it turns out it's really hard to create a good smart contract for a variety of reasons. But one of the main ones is it requires a different kind of programming than has been used in the past. People are used to writing programs that run on a single computer and programs that don't interact with other users in the same way that blockchain applications do. Block chain applications, because of these nuances, you need to consider a whole lot of edge cases that you wouldn't need to think about with traditional programs. So, to that extent--

Obsidian: a new programming language



Obsidian: a new programming language

- Obsidian is a blockchain-based language with the goal of minimizing the risk of common security vulnerabilities
- Obsidian contains core features to allow users to write safe programs easily and effectively
- Obsidian programs consist of **contracts,** which contain fields, states, and **transactions**

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

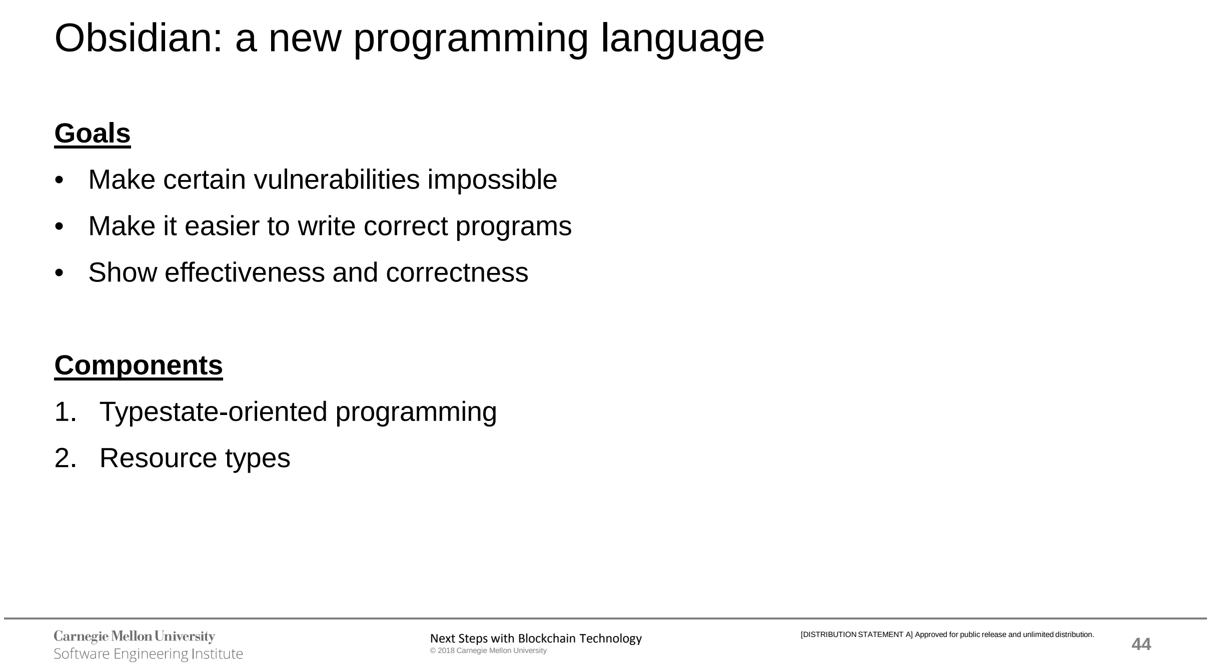
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

43

**043 One of the areas that we've been involved in is trying to develop a language for creating blockchain applications that's more secure by design. What we're looking at is a way of minimizing the risk associated with creating it, making it both harder to create bugs and easier to create bug-free software. So, if you're-- have a developer, and he's going to try to make some application, I want it that he's not able to make a bug. And in the case that he does make a bug, I want him to find it really quickly.

So, in this context, what we've done is make this language called Obsidian. Obsidian is a program that has contracts, which are essentially the same as we were talking about before. They're the applications. But the contracts themselves include these things called state.

Obsidian: a new programming language



Obsidian: a new programming language

Goals

- Make certain vulnerabilities impossible
- Make it easier to write correct programs
- Show effectiveness and correctness

Components

1. Typestate-oriented programming
2. Resource types

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

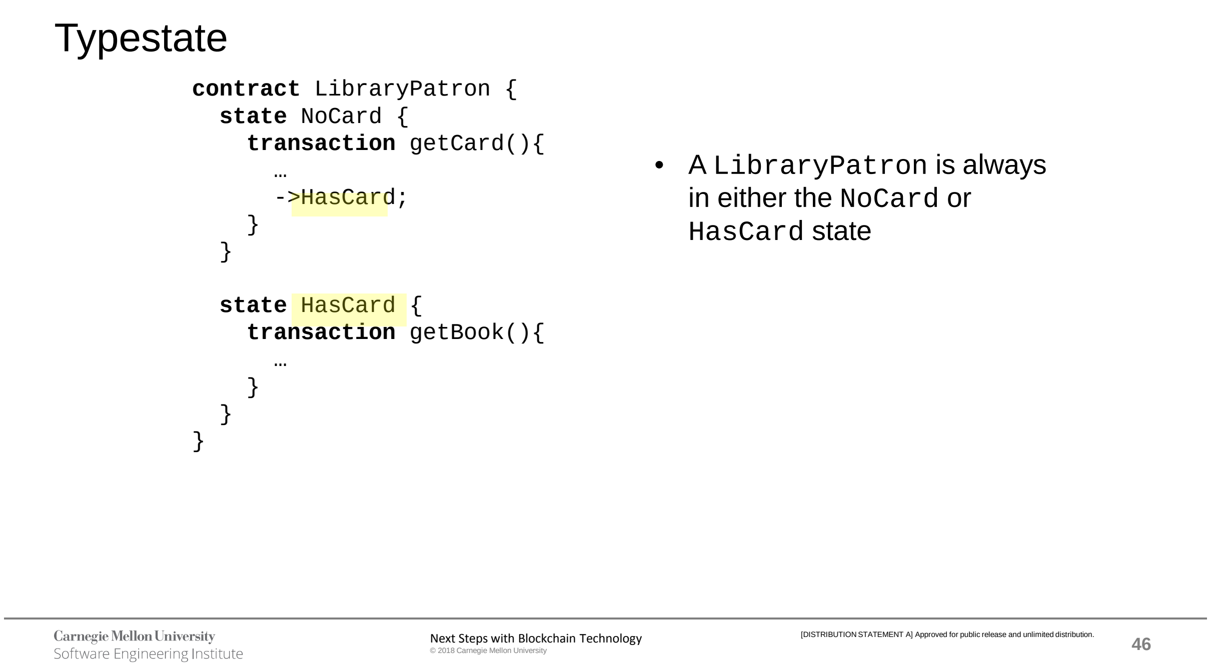
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

44

**044 And to talk a little bit about that, what we have is two aspects that make Obsidian somewhat unique in this. First of all, we have this thing called typestate oriented programming. With typestate oriented programming, and as you'll see in a second through an example, any application that we write, it is very clear at any given moment what my current state is. We were talking before about voting. And we'll get to that as an example. But you can imagine there are certain states in voting. I can either have not voted. Or I can have voted. And there may be other states you want to talk about. Regular applications, I don't tend to have to keep that in mind as such. With respect to the blockchain, if I make a mistake, given that I can't change history, it can be really costly. The second thing is something called a resource type. Very frequently in blockchain applications, I need to deal with money. When I'm dealing with money, I have to be very, very careful that I don't accidentally lose it. And I'm not talking necessarily about changing a three to a four. I'm talking about when you program, one of the things you have to use a lot is something called the variable. Those of you who are software engineers are very familiar with this. Variables are essentially my way of saying here is something. A common programming error is to use a variable incorrectly. I can put too much stuff in the variable. I can have too little. I can forget that I used it. I can forget to get rid of it.

Well, in blockchain programming, it's very important to keep track of certain kinds of variables, particularly those representing money. So, to that extent, we added that as a feature to the Obsidian language.

Typestate



Typestate

```
contract LibraryPatron {
  state NoCard {
    transaction getCard(){
      …
      ->HasCard;
    }
  }

  state HasCard {
    transaction getBook(){
      …
    }
  }
}
```

- A `LibraryPatron` is always in either the `NoCard` or `HasCard` state

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

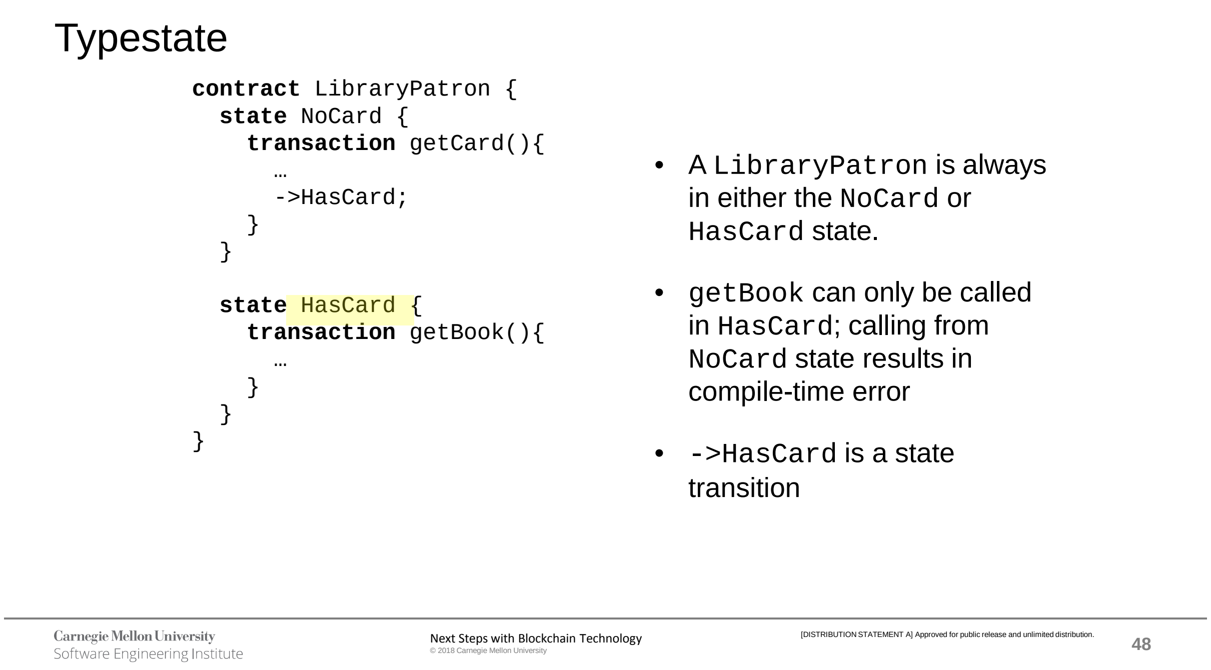
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

46

**046 Let's go through an example here. This example has to do with a person called a library patron. There is a couple states that I want to allow this person to exist in. Either-- and this is of course a simplified example. You can imagine in your head how you would develop this out. Either this is someone who does not yet have card. Or this is someone who has a library card. If it's someone who does not have a library card, then probably what they want to do is get a library card. But you can imagine that if it's someone who has a library card already, they can't get a second one. So, to that extent, I want to make it such that only certain types of behaviors are allowed in each state. This is not-- this type of way of thinking of programming is not a feature of most programming languages because it's not as important for them to get this part exactly correct. But in blockchain, it's very important.

So, what you see here is I have a person who is get card. I can then call the has card state once they get that card. And now, they're in the has card state. When they're in the has card state, they can't call the get card transaction. Why? Because they already have a card. It's not available to them. That allows us to have a lot of security to make sure that a programmer doesn't accidentally call a function they're not supposed to. To give this a motivating example, one of the larger bugs that was present in I'm going to say the DAO-- and I might have this incorrect. So, if I get this wrong, forgive me. One of the larger bugs that was present in one of the Ethereum hacks was the ability to accidentally call-- the programmer of the smart contract allowed users to accidentally call the wrong function at the wrong time. And that allowed them to syphon off a lot of money that they weren't supposed to have access to simply because the state was wrong. So, this allows that to be much more explicit.

Typestate



Typestate

```
contract LibraryPatron {
  state NoCard {
    transaction getCard(){
      …
      ->HasCard;
    }
  }

  state HasCard {
    transaction getBook(){
      …
    }
  }
}
```

- A `LibraryPatron` is always in either the `NoCard` or `HasCard` state.
- `getBook` can only be called in `HasCard`; calling from `NoCard` state results in compile-time error
- `->HasCard` is a state transition

Carnegie Mellon University
Software Engineering Institute

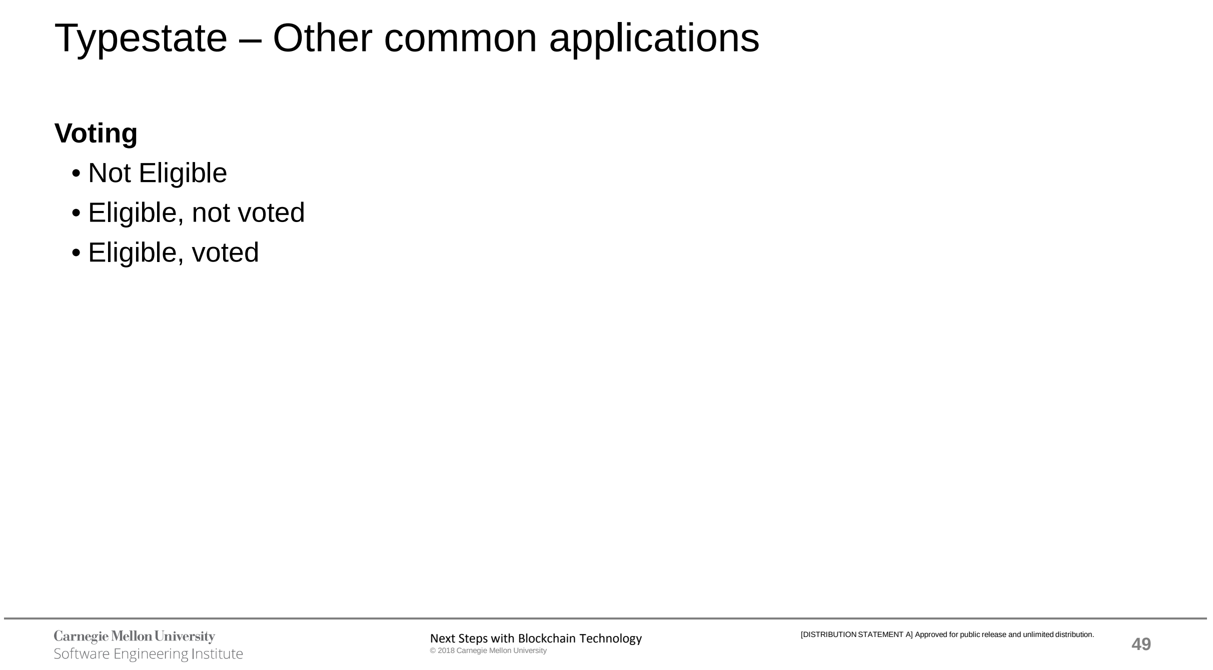
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

48

**048 We want to make sure that they users who are trying to interact with this don't have that type of problem.

Typestate – Other common applications



Typestate – Other common applications

Voting

- Not Eligible
- Eligible, not voted
- Eligible, voted

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

49

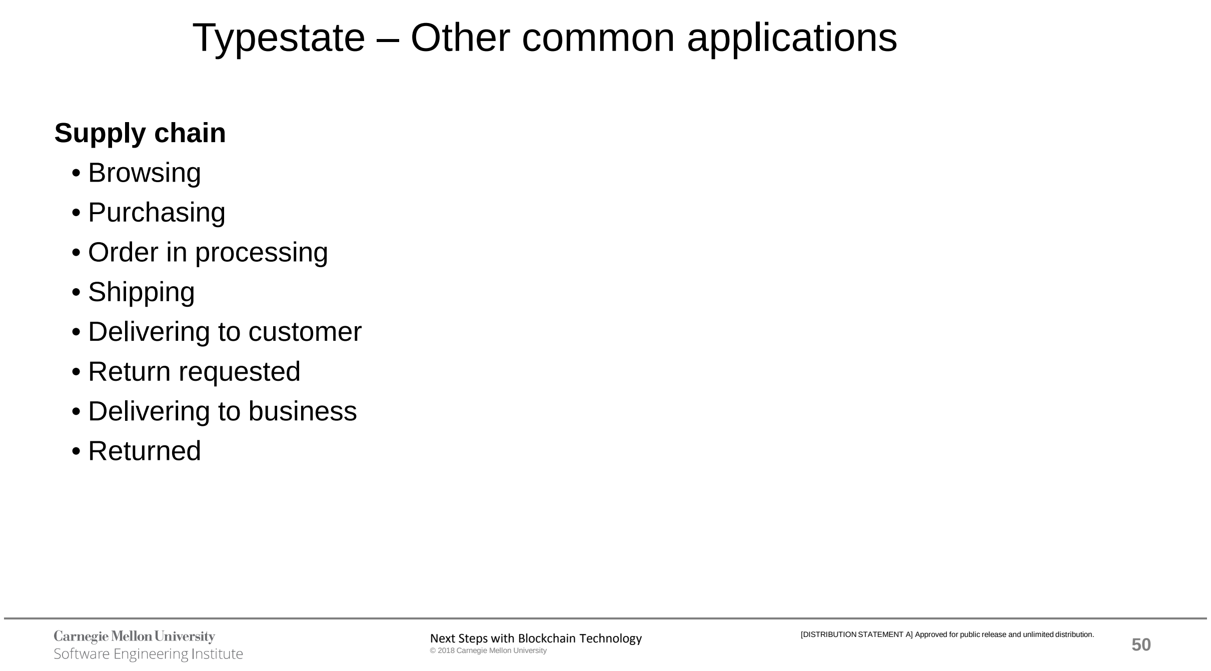
**049 Presenter: Would you say Obsidian is a language dedicated at expressing state machines in a way then?

Presenter: Very much so. Very much so. And we were talking before about--

Presenter: State-transition diagrams.

Presenter: Exactly, actually the case we were showing before was voting. Right? That was the one we used at the very beginning of the talk. So, you can imagine there's a couple different states possible with respect to voting. A person can be not eligible to vote for whatever reason. Maybe we're doing the ugly tie vote as we were talking about before, but this person doesn't work here. So, he can't vote for the ugly tie. We can have a person who is eligible, but hasn't yet voted. So, that person, one of the things they can do is vote. We can have a person being eligible and have voted. So, one thing they may be able to do is change their vote. But they can't submit their vote again because that would add to the count in a way that we don't want to allow it.

Typestate – Other common applications



Typestate – Other common applications

Supply chain

- Browsing
- Purchasing
- Order in processing
- Shipping
- Delivering to customer
- Return requested
- Delivering to business
- Returned

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

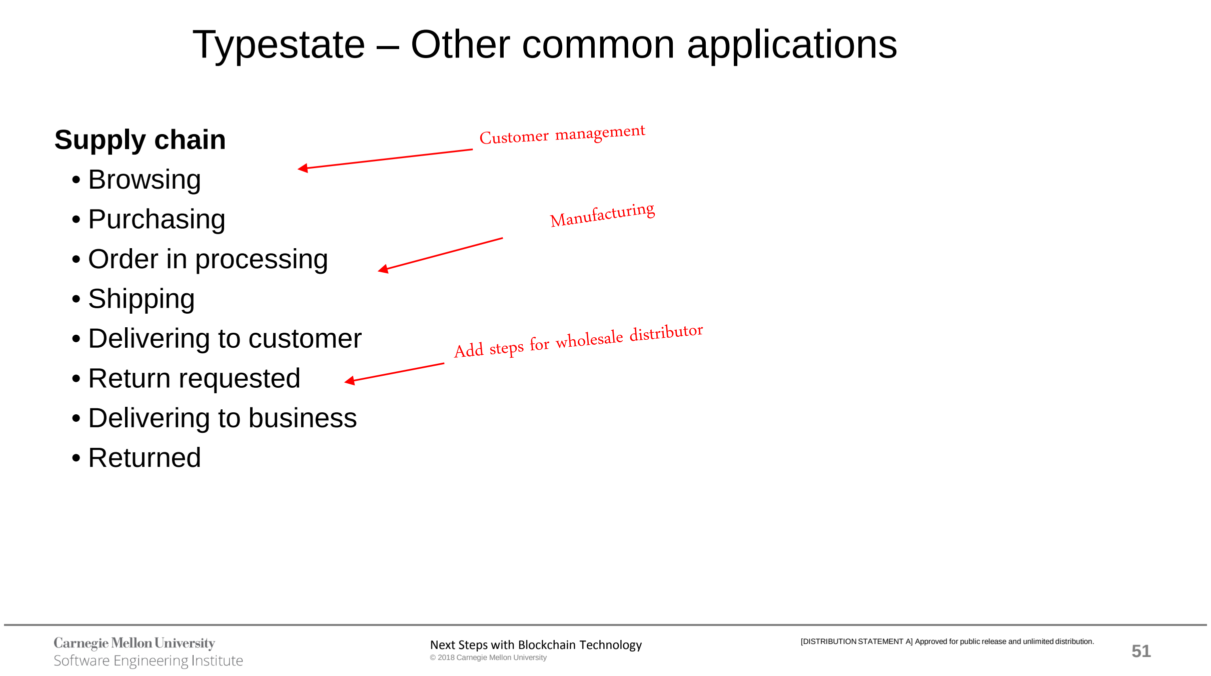
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

50

**050 Another one which is actually seeing a lot of attraction in the industry is supply chain. And I've put on here, on this slide, eight possible states ranging from when a person is just browsing and trying to find something to purchase all the way down to when someone has delivered whatever it is that they're trying to make to the customer as well as if they want to return it.

But one of the interesting things is because this is-- we're giving you a programming language. We're trying to make it that this is a way to express applications that you're going to be writing. You can develop this however you want.

Typestate – Other common applications



Typestate – Other common applications

Supply chain

- Browsing
- Purchasing
- Order in processing
- Shipping
- Delivering to customer
- Return requested
- Delivering to business
- Returned

Customer management

Manufacturing

Add steps for wholesale distributor

Carnegie Mellon University
Software Engineering Institute

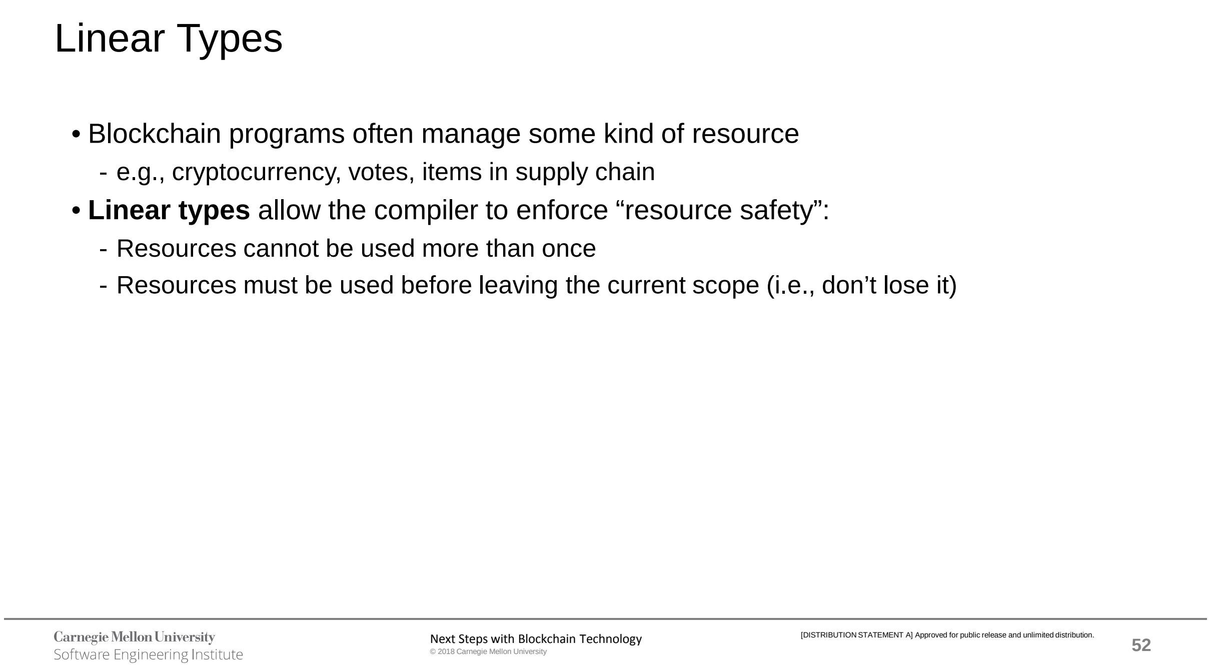
Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

51

**051 So, I've added here three additional states that person might want to include based on the needs of their own company. You can see I have customer management. Maybe before I even talk about browsing, I need to state whether or not this is a customer who I already know. So, I'll have to add something is historical customer, is not historical customer. I may want to add in a whole series of steps regarding manufacturing, whether or not I have to build the thing. I may want to add in something for distribution of it. It's interesting because this allows us to give a much more expressive way to talk about all the things I can do but be very careful that I'm only allowed to do the correct things in the individual states.

Linear Types



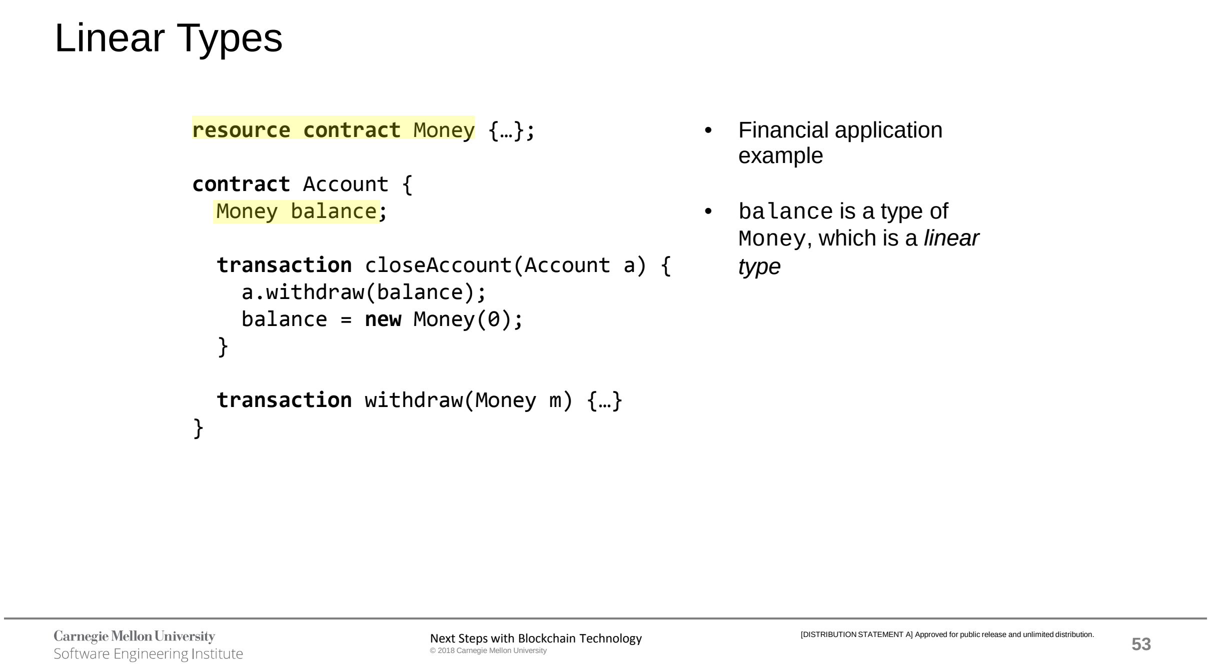
Linear Types

- Blockchain programs often manage some kind of resource
  - e.g., cryptocurrency, votes, items in supply chain
- **Linear types** allow the compiler to enforce “resource safety”:
  - Resources cannot be used more than once
  - Resources must be used before leaving the current scope (i.e., don’t lose it)

Carnegie Mellon University Software Engineering Institute | Next Steps with Blockchain Technology © 2018 Carnegie Mellon University | [DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution. | 52

**052 So, let's put aside the typestate security. And let's talk for a minute about linear types as we were talking about earlier. This allows us to have this resource safety. And this one's a little more difficult to think about because typically, when people think about keeping money safe, they're thinking about accounting, which is making sure that all the ones and zeroes add up. In this case, what I'm trying to make sure is that I didn't lose a variable. So, let me give-

Linear Types



Linear Types

```
resource contract Money {…};

contract Account {
  Money balance;

  transaction closeAccount(Account a) {
    a.withdraw(balance);
    balance = new Money(0);
  }

  transaction withdraw(Money m) {…}
}
```

- Financial application example
- `balance` is a type of `Money`, which is a *linear type*

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

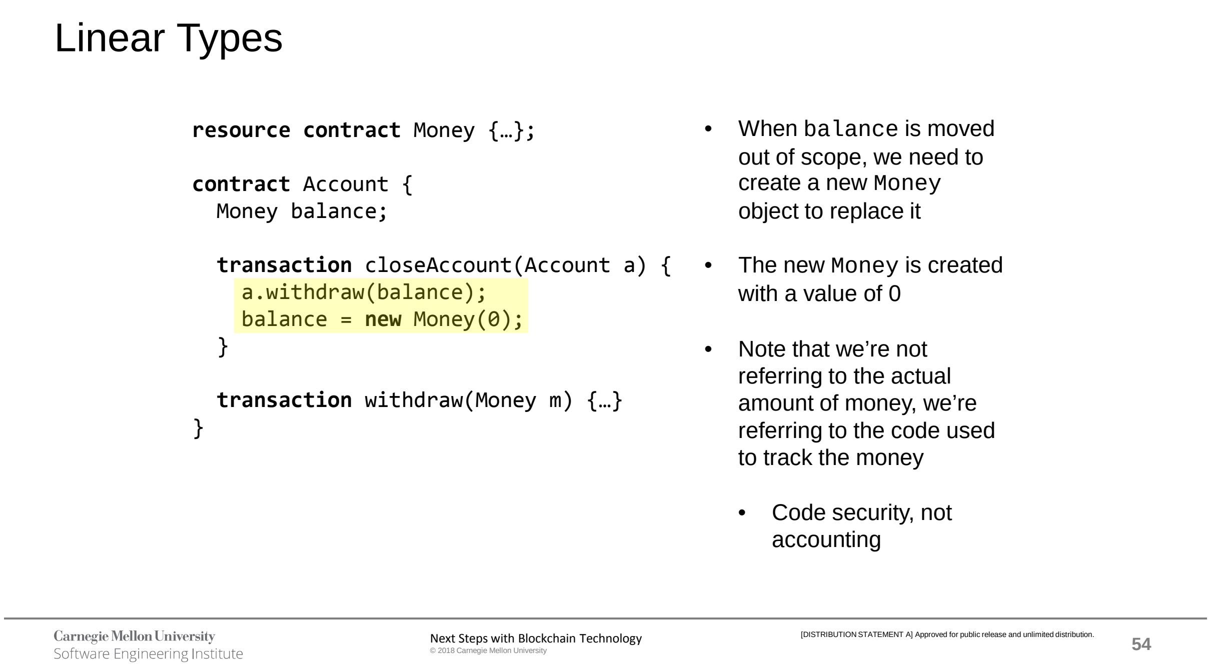
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

53

**053 What I think might be one kind of example. Here's a contract. And I want to-- you can think about this representing maybe a bank, but any sort of financial transaction. So, I'm going to define at the very top, resource contract, money. So, in this case, I have a variable. I'm going to call it money. And I want-- or actually, I have a variable type. And I want to say whenever I use this thing called money, whenever I make a new thing called money, track it really carefully. Don't let me lose it. And don't let me use it more than I'm allowed to.

So, we start this. We have this thing called balance, which is a type of money, right? So, now I'm tracking this thing called balance. It is now one of the monies that I'm tracking.

Linear Types



Linear Types

```
resource contract Money {…};

contract Account {
  Money balance;

  transaction closeAccount(Account a) {
    a.withdraw(balance);
    balance = new Money(0);
  }

  transaction withdraw(Money m) {…}
}
```

- When `balance` is moved out of scope, we need to create a new `Money` object to replace it
- The new `Money` is created with a value of 0
- Note that we're not referring to the actual amount of money, we're referring to the code used to track the money
  - Code security, not accounting

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

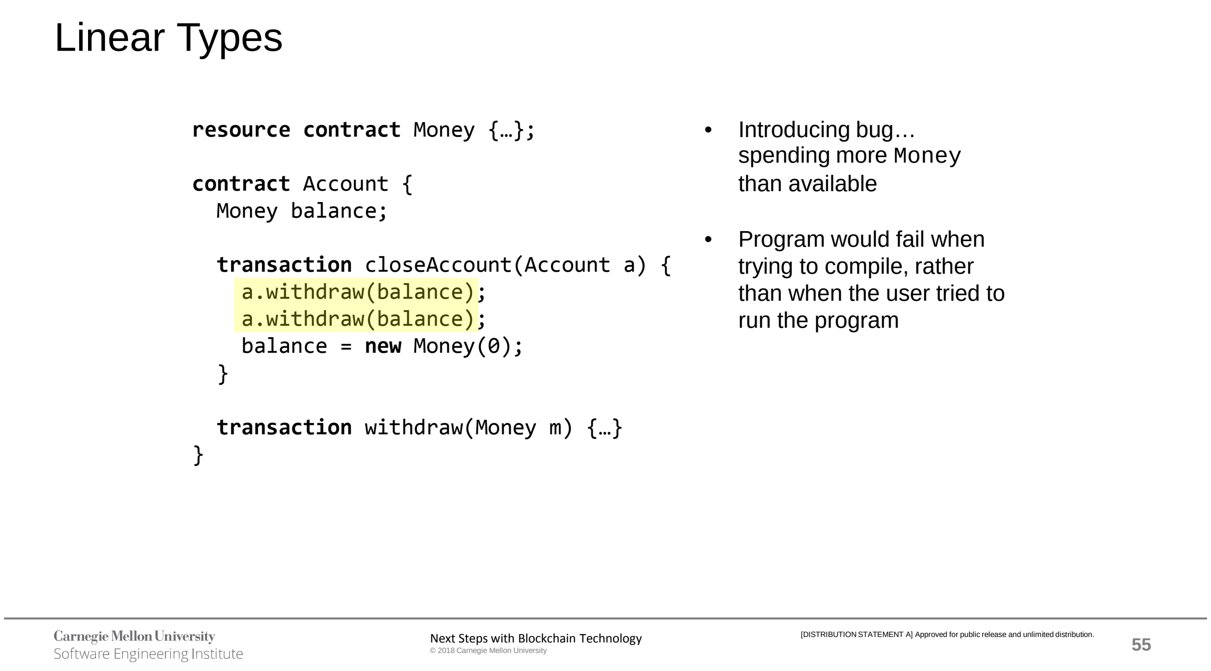
54

**054 And you can see, I go down here. So, when I want to close my account, I'm going to withdraw my balance. Now, I sent that balance now elsewhere. So, I have to make sure if that-- if the balance is gone now, I sent it out, I have to make sure I have something left at the end. I can't get rid of money. So, I have to make sure that there is a new money thing. And it's at zero by the time I'm done. So, I sent balance out of scope. And now, I have to create a new balance, a new money object, with a value of zero.

Presenter: And presumably, somebody else's balance has increased by the very same amount somewhere else.

Presenter: Exactly, so I have that function down there, withdraw. And that will accept that money. And they're going to now track it to make sure that they don't lose it. And it's important that they do that. So, one thing which I can try to protect myself from in this context is--

Linear Types



Linear Types

```
resource contract Money {…};

contract Account {
  Money balance;

  transaction closeAccount(Account a) {
    a.withdraw(balance);
    a.withdraw(balance);
    balance = new Money(0);
  }

  transaction withdraw(Money m) {…}
}
```

- Introducing bug… spending more `Money` than available
- Program would fail when trying to compile, rather than when the user tried to run the program

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

55

**055 Let's say I accidentally, for whatever reason, I have this here as only a three-line function. There's only three lines in this piece of code. You can easily imagine a bank would have hundreds of lines. And keeping track of what's where gets very difficult. You can imagine that for some reason I withdraw twice. I might have some checks and other things that I'm concerned about. In this context, the program would fail.

Now, for those of you not familiar with how programs fail, there's a lot of ways a computer program can fail. The one that you guys are all familiar with is probably the blue screen of death as you call it where all of a sudden, the whole computer stops working, or you're running a computer, and the program crashes. Those types of bugs are referred to as bugs that were caught at runtime. The program is running. There's a bug. Boom, everything dies. That's the one we want to avoid. That's the bad guy because if you've caught it, after the program is in your hands, that's terrible. Now, the developer has-- you have to go tell the developer, "There's a bug in your program." And they have to go fix it. And they have to somehow get it to you. And you have to run it and hope it doesn't break anything else.

Another kind of bug, which we can find, is one that we call compile time. So, the programmer writes all of his computer code in this text editor. It's kind of like Word but a little more fancy, intended--

Presenter: Like Notepad.

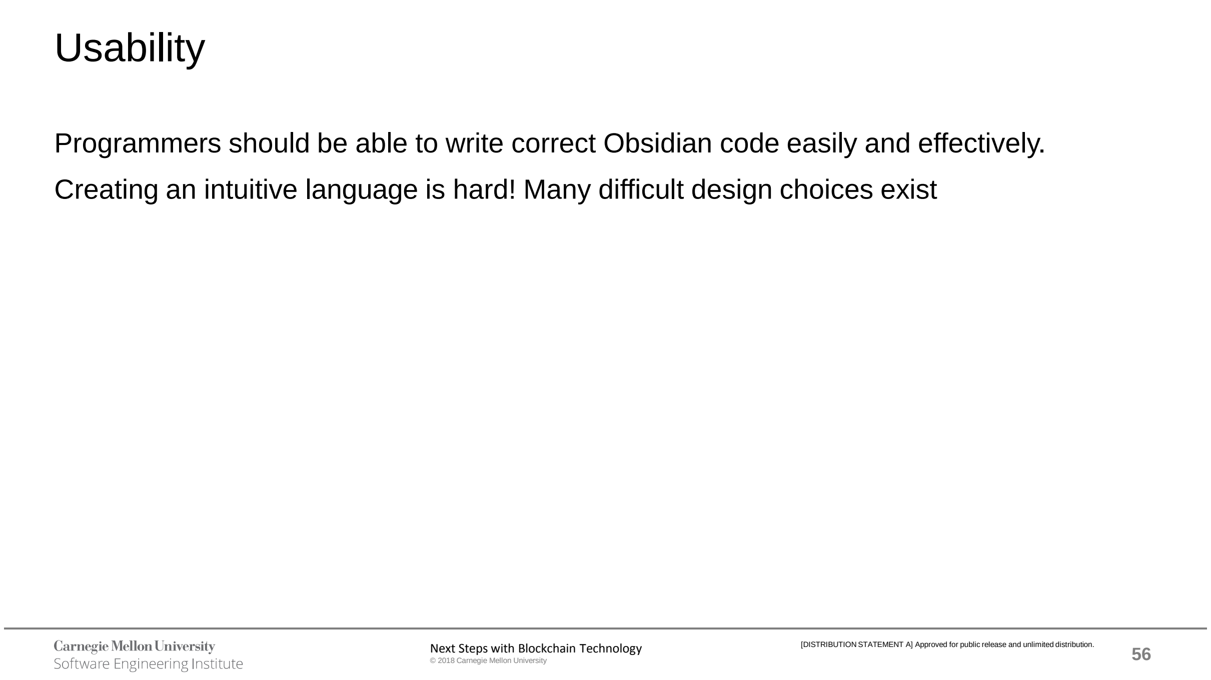
Presenter: Like Notepad, intended to be specifically for writing code. And then he builds it into a program. So, when the computer program is building that code, it does a whole lot of checks. And what we've done with Obsidian is add the checks to make sure that the money is being treated correctly at that build time, at that compiled time we call it. By doing that, we're hoping to make it that no one is going to be able to lose money accidentally. And it makes it much-- we can have a lot more confidence in

our treating of these money
variables. And again, I was talking
before about motivating factors.
Some of the other bugs that have
been seen are definitely-- were
definitely caused by people not
tracking how the money is being
spent.

Presenter: And for those of you in
the audience who like state-transition
diagrams, basically, withdraw is not
available from the state where
withdraw just took you into.
Presenter: Right, exactly. Unless
you create a separate-- if I have
whatever the state is over there--
Presenter: If it's zero, for instance--
Presenter: Yeah, I might be able to
use it in that context.

Right, it's only available
where I am right now.

Usability



Usability

Programmers should be able to write correct Obsidian code easily and effectively.

Creating an intuitive language is hard! Many difficult design choices exist

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

56

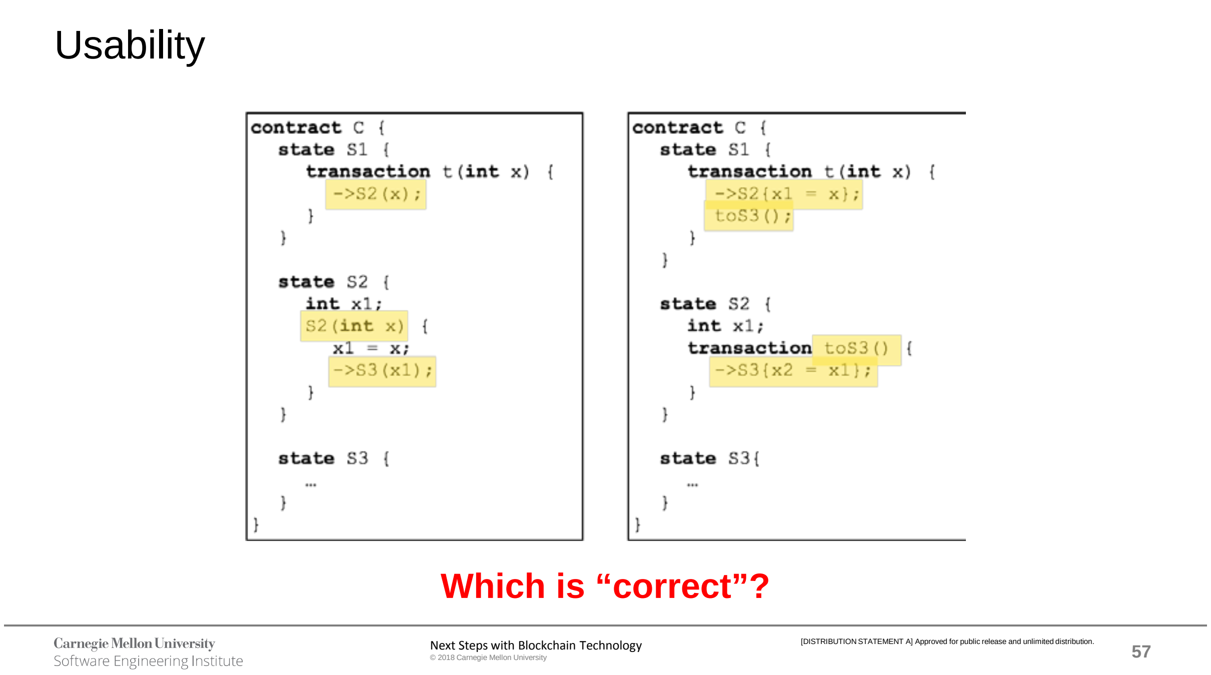
**056 So, an interesting aspect when you're trying to develop languages is how do I make a language that people actually understand how to interact with. We are used to using the English language. And the English language is wonderfully expressive.

But sometimes, it's hard to say certain things. And that's why it's interesting for those of you who are linguists. You compare words in the English language to words in a different language. It never matches up perfectly.

So, when you're writing a computer program, when you're developing a language that you want to use to write computer programs, the same exact concept exists. I want to have a language that is expressive enough to allow me to do what I want to do, but not too complicated that I can't actually use it on a daily basis. I don't want it to get in my way.

So, how do you go about figuring out what-- how should I structure this programming language? How do you users actually interact with the language I'm designing?

Usability



Usability

```
contract C {
  state S1 {
    transaction t(int x) {
      ->S2(x);
    }
  }

  state S2 {
    int x1;
    S2(int x) {
      x1 = x;
      ->S3(x1);
    }
  }

  state S3 {
    …
  }
}
```

```
contract C {
  state S1 {
    transaction t(int x) {
      ->S2{x1 = x};
      toS3();
    }
  }

  state S2 {
    int x1;
    transaction toS3() {
      ->S3{x2 = x1};
    }
  }

  state S3{
    …
  }
}
```

Which is "correct"?

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

57

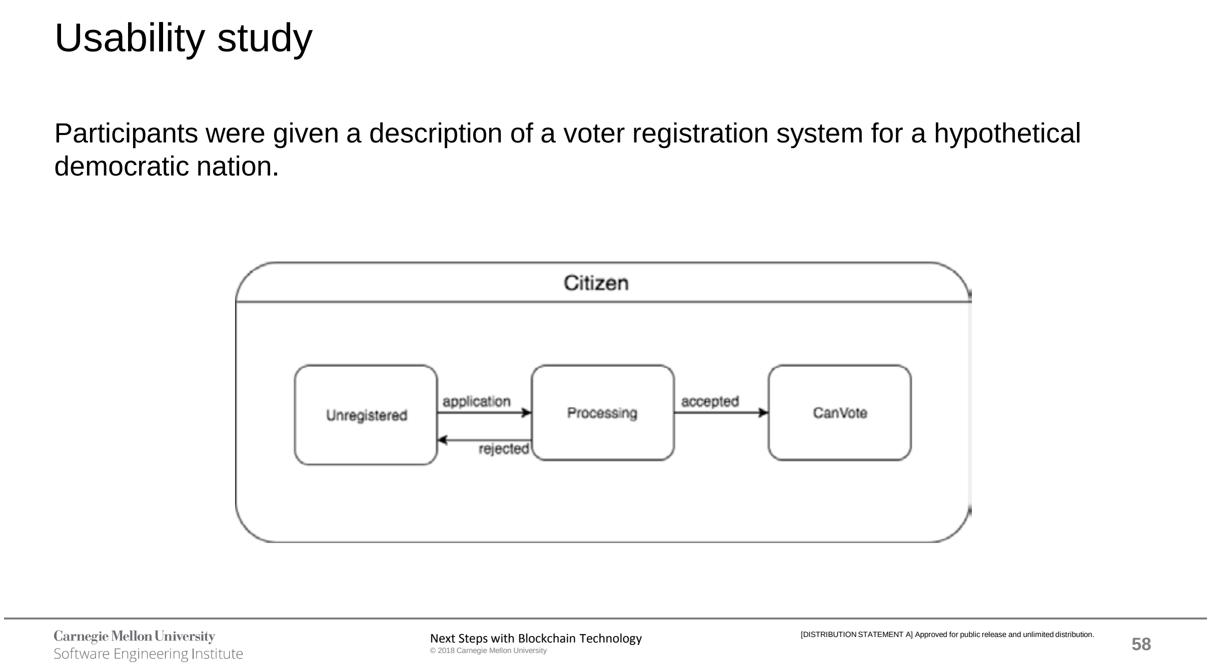
**057 So, one of the interesting things we've seen, and you can see here, we want to have some way to demonstrate that I'm moving from state one to state two. You can imagine before, I don't have a library card. I get a card. And I now have a card. How do I make the computer programmer indicate that in the concept of the program? And remember, I want to have the computer programmer do this in a way intuitively so that he can look at this code and understand what's going on. Some of you may think well, have the guy just read the documentation. That's sort of like saying everyone should look at a dictionary every single time they want to speak. You want to have it be intuitive. And you want to have them understand what it is they're working on and what the language is trying to say. That'll help an awful lot in uptake, adoption, and making sure they use it correctly.

Presenter: Doing the right thing should be easier than doing the wrong thing, preferably.

Presenter: Very, very much so. So, to that extent, we have this up here. You can see these two different demonstrations.

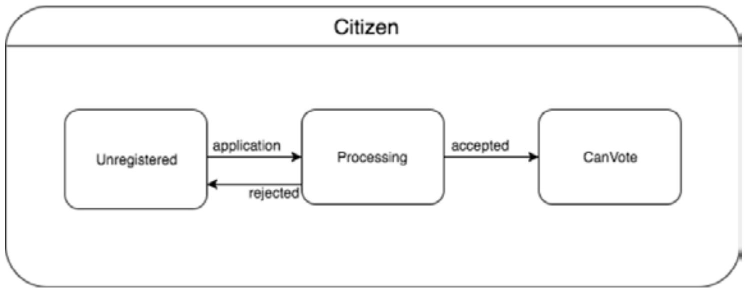
These are both
proofs of concept regarding how we can actually structure the language. And the question they put on here is which is correct. And the answer is really neither is correct. Right? So, how do you test this? How do you figure out which why is right? One way I have these arrows. One way I have to have this to where I demonstrate how this works.

Usability study



Usability study

Participants were given a description of a voter registration system for a hypothetical democratic nation.



Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

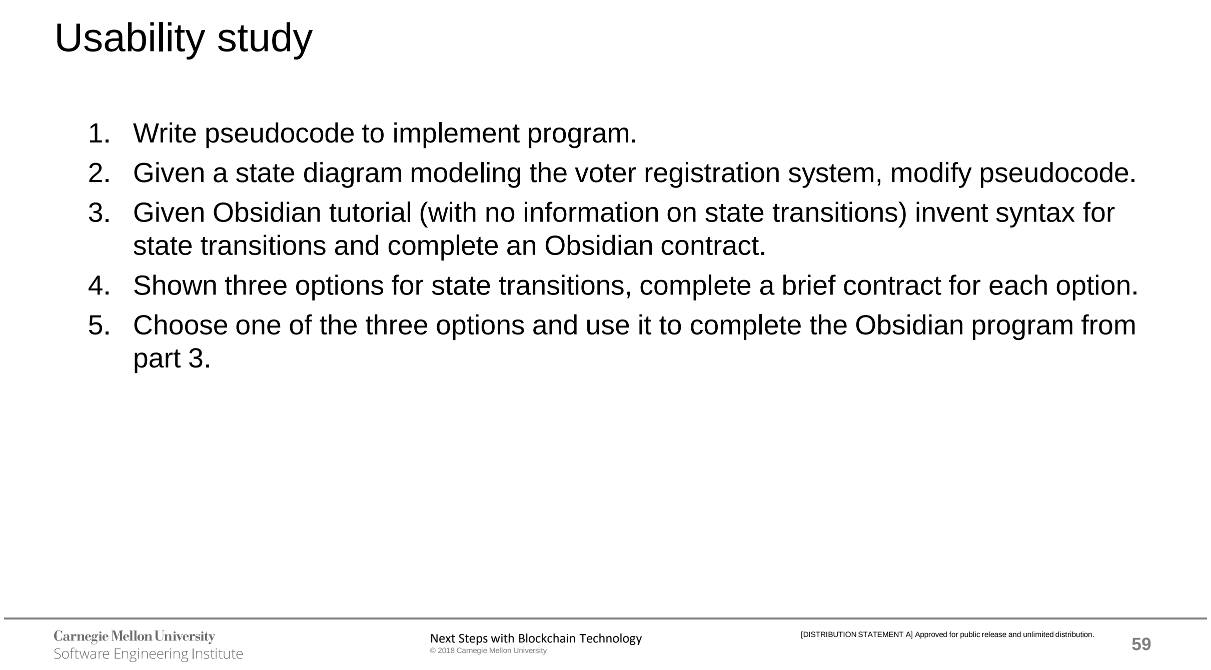
[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

58

**058 So, the answer is we actually run user studies the same way anyone else does. And in order for us to-- we gave them this whole complicated example. In this context, it was we're trying to-- the toy example we gave was create an onboarding immigration process for new citizens. Someone goes from being unregistered. They put in an application. Then it goes into this processing state.

And then it moves out of processing, either to accepted, where now they're allowed to vote because they are a member of the society, or they're rejected. And they have to reapply again to see if they can get in that next time.

Usability study



Usability study

1. Write pseudocode to implement program.
2. Given a state diagram modeling the voter registration system, modify pseudocode.
3. Given Obsidian tutorial (with no information on state transitions) invent syntax for state transitions and complete an Obsidian contract.
4. Shown three options for state transitions, complete a brief contract for each option.
5. Choose one of the three options and use it to complete the Obsidian program from part 3.

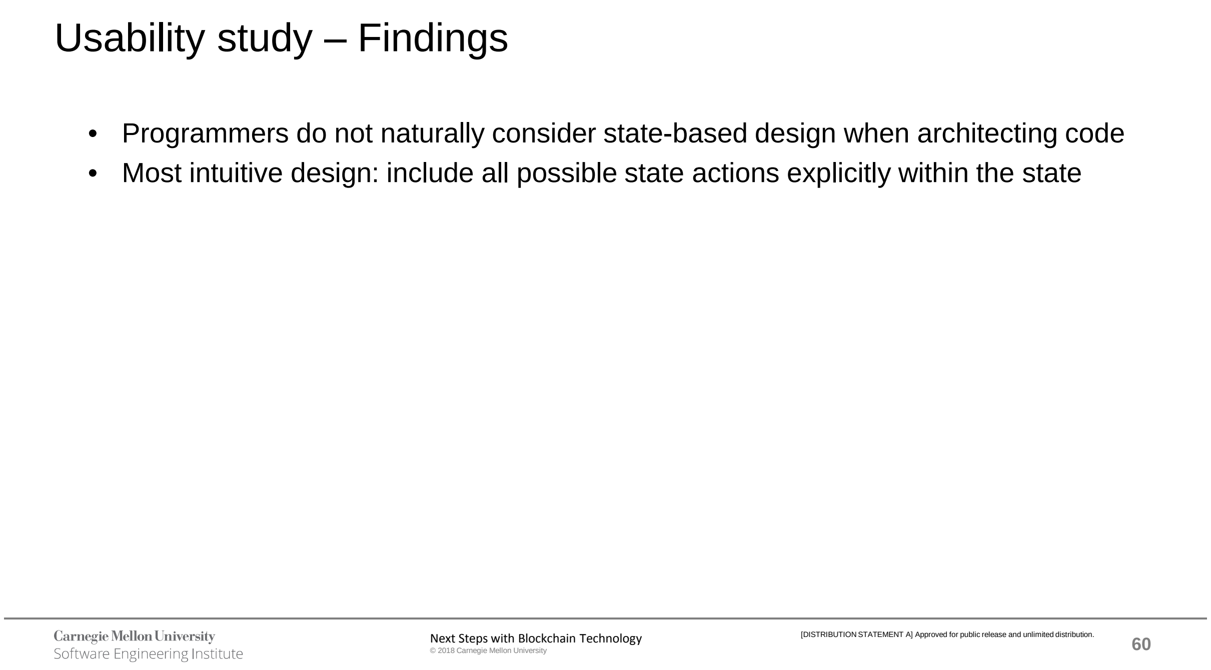
Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology

59

**059 We gave a number of people in a whole bunch of different studies, we brought in a whole bunch of different people and said, "What would be the easiest way for you to do this? Which way does it make most sense to you? If you would change it, how would you change it?" We went through.

Usability study – Findings



Usability study – Findings

- Programmers do not naturally consider state-based design when architecting code
- Most intuitive design: include all possible state actions explicitly within the state

Carnegie Mellon University
Software Engineering Institute

Next Steps with Blockchain Technology
© 2018 Carnegie Mellon University

[DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

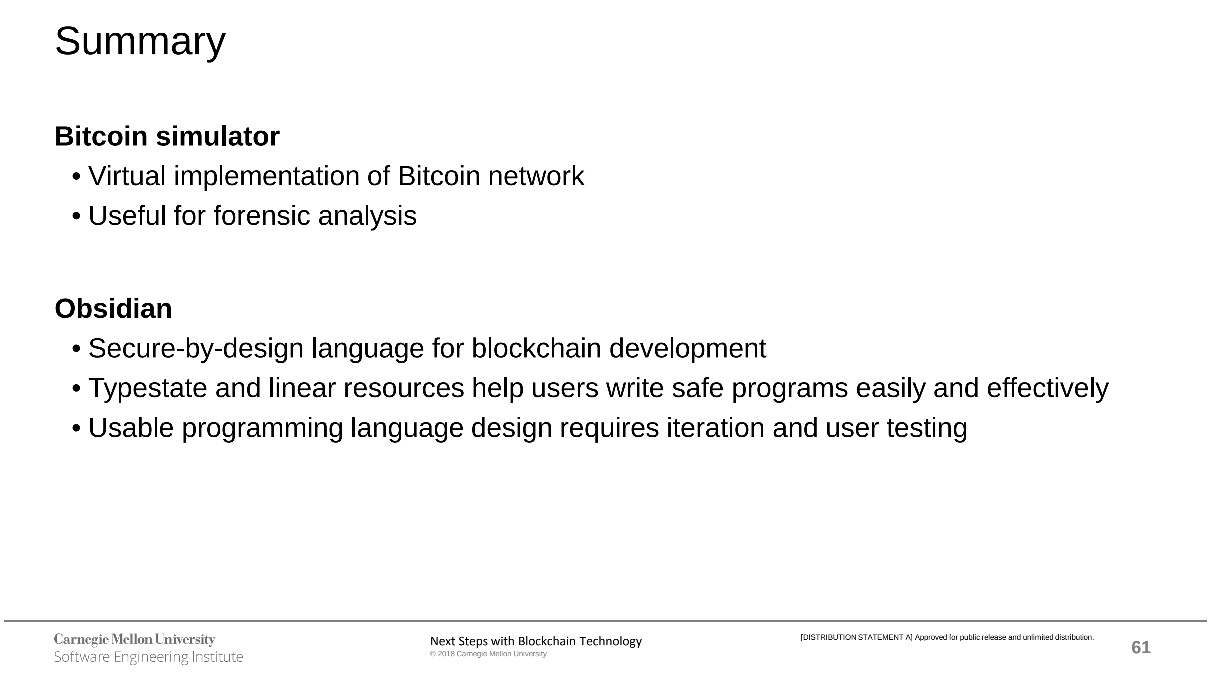
60

**060 It gave them different options.
And the interesting thing which we found is
that this type of design, this type of
development, actually is very not
intuitive for most people. When
people are writing code, because
they've been trained on this
traditional way of writing applications
that are designed for computers
rather than designed for this
blockchain style network of
computing, they're not thinking about
these state transitions. So, again,
we're trying to make Obsidian a
language that will help people
understand what it is they're
supposed to be thinking about. And
it's interesting that we have to really
train them exactly how they should
be thinking because it's not even
something which is intuitive to them
in the first place. So, by developing
the language, we're learning some of
the reasons why we're having so

many bugs in these blockchain applications.

Presenter: So, we're mostly the Software Engineering Institute. So, we're talking to software people who have been trained in procedural imperative languages more than anything else. There's also functional languages out there which are sort of more suitable to expressing state-transition diagrams. And we really should--

Summary



Summary

Bitcoin simulator

- Virtual implementation of Bitcoin network
- Useful for forensic analysis

Obsidian

- Secure-by-design language for blockchain development
- Typestate and linear resources help users write safe programs easily and effectively
- Usable programming language design requires iteration and user testing

Carnegie Mellon University Software Engineering Institute | Next Steps with Blockchain Technology © 2018 Carnegie Mellon University | [DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution. | 61

**061 Be talking more to hardware engineers. State-transition diagrams is what they live and breathe.

Presenter: Very much so. And in fact, not even engineers like who do hardware, but the database-- the folks who work with databases have been thinking about this a lot because when someone's writing to a database, the last think you want is two people to write to the same part of the database at the same time. So, there's been an awful lot of work in databases to really carefully structure and manage how it is that I can lock the table for this minute, do a write, unlock it, allow the next on to go in, create that all as a package so I can roll a whole bunch of stuff back--

Presenter: Those are known as atomic transactions.

Presenter: Exactly, exactly.

Presenter: Nobody is interfering while your transaction is being processed.

Presenter: Very cool. Before I go to the summary slide, I actually want to take-- there's one question here, and thanks to the person who asked it because it's good to have this come up at least once in every blockchain talk. Someone was asking whether quantum computing helps or hurts and whether or not that can crack hashes in seconds. It's an interesting question. It comes up very frequently. Do you want to take a stab at that?

Presenter: So, to the extent that I'm familiar with the theoretical promise of quantum computing is that certain types of encryption and certain types of cryptography would be sort of quantitatively easier to break. But in the large scheme of things, there is currently mathematics to implement the same sort of crypto that would be much harder to break even with a quantum computer. So, you go to larger key sizes. You go to better algorithms, elliptic curves instead of RSA, for instance, for public/private key cryptography. And so, it's likely that the current implementations would be easily broken by quantum, but there's ways to recover from that actually develop crypto to actually cope with it. That's sort of my two cents.

Presenter: Yeah, and I would say even more than that. So, whether or not cryptography-- crypto-computing would break the blockchain, I would say if we get to the point where quantum computing is able to actually do any of these sorts of applications, we have much larger problems because the same cryptographic concepts that underlie the blockchain also underlie the entirety of modern Internet commerce and behavior. So, the day someone breaks that--

Presenter: We have an upgrade on our hands.

Presenter: Yeah, we have a lot of bigger problems than just someone broke your blockchain, for which, by the way, you probably haven't even implemented a real application yet. But definitely, that's going to-- right now, quantum computing is limited to a very small number of applications. And it doesn't seem like it's going to be tackling large problems like this--

Presenter: Twenty years away essentially, just like twenty years ago.

Presenter: Right, exactly. Exactly, it's one of those things that's perpetually five years in the future. So, let's head up to the summary here. So, we want to go over the two things we've really reviewed with you. We gave you the general overview of what blockchain is. Then we went into this bitcoin simulator, which kind of is a tool for forensic analysts to learn if you're going to doing these type off-- if you're going to be doing this type of investigative work into bitcoin, but really any kind of blockchain application, here's a tool for you to look at it and understand exactly how do you find bad actors on a blockchain network.

We also talked a bit later, I went through the case of Obsidian, where we were trying to demonstrate there's been a lot of issues with applications that have been built already. And here's some technologies that we're looking at rolling out and we're building up right now that are going to let it be easier to develop things well, make them secure, and hopefully develop good applications in the future.

Presenter: Guys, great presentation. We're just about two o'clock. If we can get one quick question here right before end. Someone asking, "With all the excitement about blockchain that intends to decentralize the existing system, how can blockchain survive in the face of formidable challenge by the current centralized ecosystem?"

Presenter: So, I'm not entirely sure what that would be referring to. I mean there's a-- when you say centralized ecosystem, you can be referring to quite a lot of things. So, blockchain applications, when you want to roll them out, typically, you're going to either roll out a brand new application on the blockchain, which literally means you're writing as much lines of code as you have currently, probably that much plus more, rewriting the entire application. You know, if you want to migrate a bank, for example, that's millions of lines of code, putting it onto a blockchain.

Some of the vendors who are creating blockchain platforms are also creating tools though that allow you to slowly migrate parts of your application onto the blockchain. It turns out that you need to kind of wonder what is the purpose of bringing only small parts. If you're going to bring small parts on, why am I doing that? Do I need to have just this small aspect of my application on a chain? But for some types of applications, that's very relevant. And that's a much faster way to get the benefits of blockchain without having to do all the recoding that comes in. And then in some cases, that's a total waste of money. So, you really wouldn't want to do that. So, it's important to understand what you're trying to get for your money there.

Presenter: Great. Ellie, Gabe, thank you, wonderful talk today. We appreciate everybody's time. An archive will be available by tomorrow morning. We'll send out an email with the location for that file. And lastly, you'll get an invite to our next upcoming even, which will be September 27th. And our talk will be secure coding. Thanks, everyone. Have a wonderful afternoon.