ALIN TOMESCU: My name is Alin. I work in Stata, in the Stata Center. I'm a PhD student there in my fifth year. And today we're going to be talking about one of our research project called Catena. And Catena is a really nice way of using bitcoin to build append-only logs. Bitcoin itself is an append-only log. And a lot of people have been using it to put data in it. And I'll describe a really efficient way of doing that and its applications.

And if there is time, we'll talk about a tax and maybe colored coins and some other stuff. We'll talk about the what, the how and the why. And that's the overview of the presentation. So let's talk about this problem called the equivocation problem. So what is this? In general, non-equivocation means saying the same thing to everybody.

So for example, if you have a malicious service and you have Alice and Bob, the service should say the same thing to Alice and Bob. So it would make a bunch of statements. Let's say s1, s2, s3 over time and Alice and Bob would see all of these statements. So this is very similar to what bitcoin provides, right? In bitcoin you see block one, block two, block three. And everybody agrees on these blocks in sequence, right? Does that make sense?

So this is non-equivocation and in some sense this is what bitcoin already offers. And in general with non-equivocation, what you might get is some of these statements might actually be false or incorrect. Non-equivocation doesn't guarantee you that this statement is a correct statement. But it just guarantees you that everybody sees the same statements. And then they can detect incorrect statements. In bitcoin you get a little bit more. You actually know that if this is a block, it's a valid block, assuming there are enough blocks on top of it, right?

So equivocation means saying the same thing to everybody. So for example, this malicious service at time four, he might show Bob a different statement than Alice. So Bob sees s4 and Alice sees s4 prime. This is what happens in bitcoin sometimes. And that's how you can double spend in bitcoin by putting the transaction here sending money to the merchant, and then putting another transaction here sending money back to you. Right, are you guys familiar with this? Yeah, OK.

All right, so why does this matter? Let me give you a silly example. Suppose we have Jimmy and we have Jimmy's mom and Jimmy's dad, right? And Jimmy wants to go outside and play. But he knows that mom and dad usually don't let him play. So what he does is he tells dad, hey dad, mom said I can go outside. Right, and then he tells mom, hey mom. Dad said I can go outside. And let's say mom and dad are in different rooms and they're watching soap operas and they're not talking to one another. So they can actually confirm that. You know, mom can confirm that dad really said that. And dad can really confirm that mom said that. But they both trust Jimmy.

So you see how equivocation can be really problematic because now mom and dad will say sure, go outside as long as the other person said that, right? But let me give you a more practical example. So let's look at something called a public-key directory. A public-key directory allows you to map user's public keys-- a user name to a public key. Right, so here I have the public key for Alice and here I have the public key for Bob. And they look up each other's keys in this directory. And then they can set up a secure channel.

How many of you guys use Whatsapp, for example? So the Whatsapp server has a public-key directory. And when I want to send you a message, I look up your phone number in that directory and I get your public key, right? If that directory equivocates, the following thing can happen. What the directory can do is it can create a new directory at time two where he puts a fake public key for Bob and he shows this to Alice, right? And at time two also, he creates another directory for Bob where he puts a fake public key for Alice, right?

So now the problem here is that when Alice checks in this directory, she looks at her own public key to make sure she's not impersonated. And Alice looks in this version and sees, OK. That is my public key. I'm good. She looks in this version, OK. This is my public key. I'm good. So now I'm ready to use this directory. And I'll look up Bob and I'll get his public key. But Alice will actually get the wrong public key. Does everybody see that?

And similarly Bob will do the same. So Bob will look in his fork of the directory, right? And he looks up his key here and his key here. And he thinks he's OK. He's not impersonated. But in fact, Alice has impersonated there. OK? And now as a result, they will obtain fake keys for each other. And this man in the middle, attacker who knows the corresponding secret keys for these public keys can basically read all of their communications. Any questions about this?

This is just one example of how equivocation can be really disastrous. So in a public-key directory, if you can equivocate, you can show fake public keys for people and impersonate them.

So in other words, it's really important that Alice and Bob both see the same directory. Because if Bob saw this directory, the same one Alice saw, then Bob would notice that this is not the public key he had. He would notice his first public key and then that there's a second one there. And then he would know he's impersonated. And he could let's say, talk to the New York Times, and say, look. This directory is impersonating me.

So in conclusion, equivocation can be pretty bad. So this idea that you say different things to different people can be pretty disastrous. And what Catena does is it prevents that. So in general, if you have this malicious service that is backed by Catena, if it wants to say different things to different people. it cannot do that. It has to show the same thing to everybody. And the way we achieve that is by building on top of bitcoin. And that's what we're going to be talking about today. So any questions about sort of the general setting of the problem and our goals here?

So let's move on then. So why does this matter? So this matters for a bunch of other reasons, not just public-key directories and secure messaging. It matters because when you want to do secure software update, equivocation is a problem. And I'll talk about that later. So for example, at some point bitcoin was concerned about malicious bitcoin binaries being published on the web and people like you and me downloading those binaries and getting our coins stolen. Right, and it turns out that that's an equivocation problem. Somebody is equivocating, right? It's equivocating about the bitcoin binary. It's showing us a fake version and maybe other people the real version.

Secure messaging, like I said before, has applications here and. Not just secure messaging but also the web. Like when you go on Facebook.com, you're looking up Facebook's public key. And if somebody lies to you about that public key, you could be going to a malicious service and you could be giving them your Facebook password. Does that make sense? And also it has applications in the sense that if you have a way of building a append-only log, you really have a way of building a blockchain right for whatever purpose you want. And we'll talk about that as well.

So the 10,000 feet view of the system is we built this bitcoin based append-only log. And the way to think about is that bitcoin is already an append-only log. It's just that it's kind of inefficient to look in that log. If you want to pick certain things from the Bitcoin blockchain, like you put your certain bits and pieces of data there, you have to kind of download the whole thing to make sure you're not missing anything. And I'll tell you why soon. So instead of putting stuff naively in the Bitcoin blockchain, we put it in a more principled way. And in a sense we get a log and another log. We get our log and the bitcoin log. And this generalizes to other cryptocurrencies. Like you could do this in light coin, for example, or even in ethereum. Though I don't think you guys yet talked about how the ethereum blockchain works.

And the cool thing about this Catena log that we're building is that the Catena log is as hard to fork as the bitcoin blockchain. If you want to fork our log, you have to fork bitcoin. However, unlike the Bitcoin blockchain, Catena is super efficient to verify. So in particular, remember, I described the log in terms of the statements in it. So if you have 10 statements, each statement will be 600 bytes to audit. So you don't have to download the whole Bitcoin blockchain to make sure you're not missing a statement. And you also have to download 80 bytes per bitcoin block.

And we have a Java implementation of this. And if you guys are curious, you can go to my GitHub page, and take a look at the code. All right, so before we start, I know you guys already know a lot about how the Bitcoin blockchain, but it's important that I reintroduce some terminology just so we're on the same page. So this is the bitcoin blockchain. We have a bunch of blocks connected by hash chain pointers. And we have Merkle trees of transactions. And you know, these arrows indicate hash pointers. It means that block n stores a hash of block n minus 1 in it, right? So in that sense. Block n has a hash pointer to block n minus 1.

All right, and like I said, each tree has a Merkle tree. Each block has a Merkle tree. And everybody agrees on this chain of blocks via proof of work consensus, which all of you already know. And importantly, in the Merkle trees, we have transactions. And transactions can do two things. Right, a transaction can mint coins, create new coins. So here I have transaction a. It created four coins by storing them in an output. Are you familiar with outputs? So you guys already discussed transaction outputs and inputs. So that's what we're going over here again real quick. So an output specifies the number of coins and the public key of the owner of those coins, as you already know. And the second purpose of transactions is that they can transfer coins, right, and pay fees in the process.

So here if you have a transaction b, this transaction b might say, hey, here's a signature from the owner of these coins here. Here's a signature and transaction b. And here's the new owner with public key b of three of those four coins. And one of those coins I'll just give it to the miners as a transaction fee. Does this make sense? How many of you are with me? All right, any questions about how transaction inputs and outputs work? So if you remember in the input here, you have a hash pointer to the output here.

All right. So and the high level idea here is that the output is just the number of coins in a public key. And the input is a hash pointer to an output plus a digital signature from that output's public key, OK? And yeah, so what happens here is that transaction b is spending transaction a's first output. Right, that's the terminology that we're going to use. And I think you guys have already used terminology like this. And in addition, what we're going to talk about today is the fact that in these bitcoin transactions you can actually embed data.

And I think you touched briefly on this concept of op return transaction outputs. Right, so this is an output that sends coins to a public key. But here I can have an output that sends coins to nobody. It just specifies some data. And in fact, I'll use that data to specify the statements that I was talking about earlier. So that the high level point here is that you can embed data in bitcoin transactions using these operations. And there's a bunch of other ways to do it, actually. Like initially what people did is they put the data as the public key. They just set the public key to the data. And in that sense, they kind of wasted bitcoins.

Right, they said hey, send these three bitcoins to this public key, which is just some random data. But nobody would know the corresponding secret key of that public key. So therefore those coins would be burned. Did you did you cover this in class already? Maybe, maybe a little bit. So but that's kind of inefficient because if you remember, the miners have to build this UTXO set. And they have to keep this output that has this bad public key in their memory forever because nobody's going to be able to spend it. So we don't want to do that. We want to build a nice system, a system that treats bitcoin nicely, the bitcoin miners. So that's why we use these return outputs.

All right, so the high level here is that Alice gives Bob three bitcoins. And the miners collect a bitcoin as a fee. And of course, you can keep doing this, right? Like Bob can give Carol these two bitcoins later by creating another transaction with an input referring to that output and with an output specifying Carol's public key. All right, and the high level idea of bitcoin is that you don't-- you cannot double spend coins. And what that means is that a transaction output can only be referred to by a single transaction input. So this thing right here in bitcoin where I have two inputs spending an output cannot happen, right? How many of you are familiar with this already? OK, good.

So actually this is the essential trick that Catena leverages. And we'll talk about it soon. And yeah, the moral of the story, the reason I told you all of this is because you know, basically if you have proof of work consensus in the bitcoin sense, you cannot do double spends. So this thing right here, that I said before, just cannot occur in the Bitcoin blockchain unless you break the assumptions, right, unless you have more mining power than you should. In that case, you either have to tx2 or tx2 prime, but not both, right?

And what Catena really realizes is that if I put statements in these transactions now, what that means is that I can only have a second statement. I cannot have two second statements. I cannot equivocate about the second statements if I just restrict my way of issuing statements in this way. I put the first statement in the transaction and the second statement I put it in a transaction that spends the first one. So does everybody agree that if I do things in this way and I want to equivocate about the second statement, I would have to double spend?

So that's the key insight behind our system. Any questions about this so far? You know it's really hard to talk if you guys talk back, it's way easier. Yeah?

AUDIENCE: A question. And maybe, I think I'm getting this right. If you're setting up the first transaction, then you're adding data on? You're burning bitcoins every time you're doing it, so at some point, you're going to run out.

ALIN TOMESCU: That's an excellent question. So let's-- we'll go over that. But the idea is that in this output, I won't burn any bitcoins. I'll actually specify my own public key here. And I'll just send the bitcoins back to myself. And in the process I'll pay a fee to issue this transaction. Does that make sense? Yeah? And we'll talk about it later. OK, so real quickly, what did previous work do regarding this? So how many of you are familiar with blockstack? 1, 2, only two people? OK how many of you are familiar with Keybase?

OK, so blockstack and Keybase actually post statements in the Bitcoin blockchain. And they're both public directories. They map user names to public keys. And for example, Keybase, what Keybase does is they take this Merkle route hash, and they put it in the transaction. And then six hours later, they take the new route hash, and they put it in another transaction, and so on. Every six hours, they post the transaction. But unfortunately, they don't actually do what Catena does. So they don't have their new transaction spend the old one. And as a result, if you're trying to make sure you see all of the statements for Keybase, you don't have a lot of recourse other than just downloading each block and looking in the block for all of the relevant transactions.

Another thing you could do is you could sort of trust the bitcoin miners-- the bitcoin full nodes to filter the blocks for you. So you could contact a bunch of bitcoin full nodes and say, look. I'm only interested in transactions that have a certain IP return prefix in the data. And they could do that for you. But unfortunately, bitcoin full nodes could also lie to you very easily. And there is no cost for them to lie to you. Everybody can be a bitcoin full node. So then it becomes a very bandwidth intensive process because you have to ask a lot of full nodes to deal with the fact that someone might lie to you.

But you either need to download full blocks to find let's say, a missing statement like this one that a bitcoin full node might hide from you, or you can trust the majority of bitcoin full nodes to not hide statements, which is not very good, right? So I don't want to trust these full nodes because all of you guys could run a full node right now in the bitcoin network. It doesn't cost you anything. And if I talk to your malicious full node, I could be screwed. So our work just says, look. Instead of issuing transactions in sort of an uncorrelated fashion, just do the following thing. Every transaction you issue should spend the previous one. So as a result, if someone wants to equivocate about the third statement, they have to double spend like I said before. Right? Yeah?

AUDIENCE: So what's the connection back to Keybase and blockstack again?

ALIN TOMESCU: Yeah, so Keybase and blockstack are public directories. And what they do is they want to prevent themselves from-- is there a whiteboard I can draw here? Yeah so, let's say, you know, if you remember the picture from the beginning, I can have a public directory that evolves over time, right? So this is v1 of the directory, and it might have the right public keys for Alice and Bob. But at v2, the directory might do this. It might do v2. It might have Alice, Bob, but then put a fake key for Alice. And at V2 prime, it might do Alice, Bob as before, and put a fake key for Bob. So in other words it keeps the directory append-only. But it just adds fake public keys for the right people in the right version.

So here this one is shown to Bob. And here this one is shown to Alice. So now Alice will use this fake key for Bob. So she'll encrypt a message with b prime for Bob. And now the attacker can easily decrypt this message because he has the secret key. So what the attacker can do then is re-encrypt it with the right public key for Bob and now he can read Alice's messages. And the whole idea is that the attacker-- you know, this b prime is the public key, is pk b prime, let's say. But the attacker has this sk b prime. He knows sk b prime because the attacker put this in there.

The attacker being really, blockstack or Keybase. And of course, they're not attackers in the sense that they want to be good guys. But they're going to be compromised eventually. So they want to prevent themselves from doing things like these. Is that sort of answer your question? Yeah. All right, so yeah, a really simple summary. If I had two slides to summarize our work, this would be it. Right, they would be these two. Look, don't do things this way. Do them this way.

All right, so let's see, let's look a little bit at the design. So remember we have these authorities that could equivocate about statements they issued like blockstack and Keybase. So what we propose is look, these authorities can run a lock server, a Catena lock server. And they start with some funds locked in some output. And what they can do first is they can issue this genesis transaction to start a new lock. So for example, Keybase would issue this genesis transaction starting the log of their public directory Merkle routes.

And this genesis transaction can be thought of as the public key of the log. Once you have this genesis transaction, you know it's transaction ID. You can verify any future statements, and you can implicitly prevent equivocation about statements. And what the lock server is going to do is it's going to take these coins and send them back to the server to answer your question. So if there was a public key, if these coins are owned by some public key, they're just sent back to same public key here and paying some fees in the process. Right, so we're not burning coins in the sense that we're just paying fees that are miners, which we have to do.

OK, so now what you can do is if you want to issue the first statements, you create a transaction. You send the coins from this output to this other output. You pay some fees in the process. You put your statement in an op return output. And as a result, if this lock server wants to equivocate, it has to again, double spend here, which it cannot do unless it has enough mining power. So Keybase and blockstack, if they were to use a system like this, they could prevent themselves from equivocating.

And this can keep going, right? So you issue another transaction. Spend the previous output. Put the new statement. Yes?

AUDIENCE: This doesn't seem like a new problem. How have authorities prevented equivocation in the past?

ALIN TOMESCU: This doesn't seem like a new problem. How did authorities do it? The problem is not new. The problem is eternal. So you are correct there. How did they do it in the past? They just used a Byzantine consensus algorithm. So in some sense this is what we're doing here as well. We're just piggybacking on top of Bitcoin's Byzantine consensus algorithm.

AUDIENCE: So you're rolling down to a newer Byzantine consensus algorithm basically.

ALIN TOMESCU: Sure, I'm not sure what rolling down means. But yeah, we're piggybacking on top of bitcoin. The idea is that look, a byzantine consensus is actually quite complex to get right. We already have a publicly verifiable business consensus algorithm. It's bitcoin. Why can't we use it to verify, let's say, a log of statements super efficiently? So up until our work, people didn't seem to do this. So Keybase didn't do this. Blockstack didn't do this. They kind of forced it to download the entire bitcoin block chain to verify let's say, three statements.

So in our case, you only have to download a few kilobytes of data to verify these three statements, assuming you have the bitcoin block headers, right, which we think is a step forward. And it's sort of like the right way to use these systems should be you know, the efficient way, not the inefficient way. Because bandwidth is expensive, right? Computation is cheap. Bandwidth is expensive.

All right, so anyway the idea is that if the lock server becomes malicious, if Keybase or blockstack gets hacked, they cannot equivocate about the third statement. They can only issue one unique third statement. And the advantages are, you know, it's hard to fork this lock. It's hard to equivocate about third statement. But it's efficient to verify. And I'll walk you through how clients verify soon.

The disadvantages are that if I want to know that this is the second statement in the log, I have to wait for six more blocks to be built on top of this statement's block, right? Just like in bitcoin, you have to wait for six blocks to make sure a transaction is confirmed, right? Why do you do that? The reason you do that is because there could be another transaction here, double spending because sometimes there are accidental forks in bitcoin and things like that. What are some other disadvantages that you guys can point out? Yeah?

AUDIENCE: It's going to be expensive.

ALIN TOMESCU: It's going to be expensive to issue these trends. So Alin-- Alin and I share the same name. So Alin is pointing out that you know, every time I issue these statements, I have to pay a fee, right? And if you remember, the fees were quite ridiculous in bitcoin. So that's a problem, right? So let's see, is that the next thing? The next thing was you have to issue-- you can only issue a statement every 10 minutes, right? So if you want to issue statements really fast. you can't do that.

All right, like Alin said you have to pay bitcoin transaction fees. And the other problem is that you don't get freshness in the sense that it's kind of easy for this lock server to hide from you the latest statement. You know, unless you have a lot of these log servers and you ask many of them, hey, what's the latest statement? And they show you back the latest statement. If there is just one log server and it's compromised, it could always pretend no, no, no. This is the latest statement.

And if you don't trust it, the best recourse you have is to download the full block and look for the statement yourself. So you don't get freshness. Those are some disadvantages. Now let's look at how clients audit this log? So I was claiming that it's very efficient to get these statements and make sure that no equivocation happened. So let's say, you have a Catena client and you're running on your phone with this client. And your goal is to get that list of statements. And there's the Catena log server over there in the back. And there's the bitcoin peer to peer network which at the moment has about 11,000 nodes.

And remember, I said the first thing you need is the genesis transaction. Does everybody sort of understand that if you get the wrong Genesis transaction, you're completely screwed, right? Because it's very easy to equivocate if you have the wrong Genesis transaction, right? I mean, you know there's the right GTX here where you have, let's say, a s1. And then you have s2 in their own transactions, right? But if there's another GTX prime here and you're using that one, you're going to get s1 prime, s2 prime, different statements. So if Alice uses GTX but Bob uses GTX prime, Alice and Bob are back to square one.

So in some sense, you might ask, OK, so then what's the point? What have you solved here? I still need to get this GTX, right? So what we claim is that this is a step forward because you only have to do this once. Once you've got this GTX, you can be sure you're never equivocated to, right? Whereas in the past, you would have to for each individual statement, you'd have to do additional checks to make sure you're not being equivocated to, like you would have to ask in a full node, let's say.

Right, so as long as you have the right GTX, you're good. And how do you get the right GTX? Well, usually you ship it with the software on your phone. And there's some problems there as well, like there is no problem solved in computer science in some sense. But you know, we're trying to make progress here. OK, so let's say you have the right GTX because it got shipped with your software. Now the next thing you want to do is get the block headers. So you have header i, but there are some new headers being posted.

Let's say the bitcoin peer to peer network sends them to you. You have these headers. You verify the proof of work, right? So this only costs you 80 bytes per header, right? Does everyone see that this is very cheap? So far, so far I have the GTX, which is let's say 235 bytes. And now I'm downloading some headers. And now I'm ready to ask the log server what's the first statement in the log, right? And what the log server will do is he's going to reply with the transaction with the statement, which is 600 bytes and the Merkle proof. So all of this is 600 bytes, actually. Right, and now what the Catena client will do is he'll check the Merkle proof against one of the headers so to see in which headers does it fit.

And then he'll also check that the input here has a valid signature from the public key in the output here. All right, I want at least one question about this. Yeah?

AUDIENCE: Sorry I came in late, but is the Catena client over there similar to the SPV?

ALIN TOMESCU: Yeah, so that exactly. It's an SPV client. Yeah, so the idea is that we want SPV clients. We don't want these mobile phone clients to download 150 gigabytes of data. We want them to download let's say 40 megabytes worth of block headers, which they can discard very quickly as they verify. And then we want them to download 600 bytes per statement, but still be sure that they saw all of the statements in sequence, and that there was no equivocation. Yeah?

AUDIENCE: So in our previous classes, we discussed, if you can have a full node and an SPV node, do these sort of vulnerabilities exist with the [INAUDIBLE] client?

ALIN TOMESCU: So which vulnerabilities that you did talk about?

AUDIENCE: I forget.

ALIN TOMESCU: Did you talk about--

AUDIENCE: There was just a box that was like less secure. And then there was another box that was something else bad. There was one about [INAUDIBLE]. The clients would lie to you about--

If you say, here's some transactions or here are some unspent outputs, then they could just tell you something different.

ALIN TOMESCU: Yes. Yeah? Sorry.

AUDIENCE: Well, they can't tell you that transactions exist or don't exist. They can just not tell you--

ALIN TOMESCU: Yeah, they can hide transactions from you, which gets back to the freshness issue that we could discussed. They could also-- this block header could be an header for an invalid block. But remember, that before you accept this tx1, you wait for enough proof of work, you wait for more block headers on top of this guy to sort of get some assurance that no, this was a valid block because a bunch of other miners built on top of it. Right?

So as long as you're willing to trust that the miners do the right thing, which they have an incentive to do the right thing, you should be good. But like you said, what's your name?

AUDIENCE: Anne.

ALIN TOMESCU: Anne? Like Anne said, there are actually bigger problems with SPV clients. And if there is time, we can talk about it. But it's actually easier to trick SPV clients to fork them. And there's something called a generalized vector 76 attack. Have any of your guys heard about this? So it's like a pre-mining attack but it's a bit easier to pull off. Actually, it's a lot easier to pull off on an SPV node than on a full node. And if there's time at the end, we can talk about it. If there isn't, you can read our paper, which is online on my website. And you can read about these pre-mining attacks that work easier for SPV nodes.

But anyway, this can keep going, right? You get block headers, 80 bytes each. You ask the log server, hey what's the next statement in the log? You get a Merkle proof in a transaction. And then you verify the Merkle proof. You put this transaction in one of these blocks and you verify that it spends the previous one. Right, and as a result, you implicitly by doing this verification, by checking that hey, this is a transaction and a valid block. This block has enough stuff built on top of it and this transaction spends this guy here, you implicitly prevent equivocation.

Right, then you don't have to download anything else. Right, you only have to download these Merkle proofs and transactions in these block headers. Whereas in previous work, you could be missing these s1's, these s2's, they could be hidden away in some other branch of the Merkle tree. And you'd have to do peer to peer bloom filtering on the full nodes. And those full nodes could lie to you.

Yeah, so the bandwidth is actually very small. So suppose we have 500k block headers-- I think bitcoin has a bit more right now-- which are 80 bytes each and we have 10,000 statements in this log, which are 600 bytes each. Then we only need now with 46 megabytes, right? What's the what's the other way of doing it? You have to download hundreds of gigabytes. All right, so let's talk about scalability a little bit. So suppose the system gets deployed widely. Let's say Whatsapp starts to use the system to witness-- to publish their public directory in bitcoin, right?

And everybody, a lot of you here have Whatsapp. And this is you guys. Let's say there are 200,000 people using Whatsapp. I think there's more like a billion. So what are they going to do? Remember that part of the verification protocol is asking for these block headers from the peer to peer network, right? And in fact, if you're SPV clients, you usually open up around eight connections to the peer to peer network. And if you're a full node in the bitcoin peer to peer network, you usually have around 117 incoming connections. That that's how much you support. You support 117 incoming connections as a full node.

So that means in total, you support about a million incoming connections. So you know, this guy supports a million. But we need about 1.6 million connections from these 200,000 clients, right? So it's a bit of a problem if you deploy Catena and it becomes wildly popular. Being a bit optimistic here, but you know, let's say that happened, right? So how can we fix this? How can we avoid this problem because in this case, what we would basically be doing is we would be accidentally DDOSing bitcoin. And we don't want to do that.

Does everybody see that there's a problem here, first of all? OK, so the idea is very simple. We just introduced something called a header relay network. And what that means is look, you don't really have to ask for these block headers from the Bitcoin peer to peer network. You could just outsource these block headers anywhere because they're publicly verifiable. Right, the block headers have proof of work on them.

So you can use volunteer nodes that sort of push block headers to whoever asks for them. You could use blockchain explorers like blockchain.info, right? You could use Facebook. You could just post block headers on Facebook. Right, like in a Facebook feed. You could use

Twitter for that. You could use GitHub gists. You know, so you could-- there's a lot of ways to implement this. The simplest way is have servers and have them send these headers to whoever asks for them.

So it's easy to scale in that sense because if you now ask these header relay network for the block headers, you know, it's much easier to scale this than to scale the bitcoin peer to peer network, which has to do a bit more than just block headers. They have to verify blocks. They have to verify signatures. Yeah?

AUDIENCE: Did you consider having the clients being peer to peer?

ALIN TOMESCU: , Yes so another way. And I think we'll talk about that in the paper-- is you can implement the header relay network as a peer to peer network on top of the clients. Yeah, so that's another way to do it. There's some subtleties there that you have to get right. But you can do it, I think. Yeah?

AUDIENCE: Would you you expect that if a company like Whatsapp decided adopt Catena, they would run their own servers to make sure that there was the capacity for it?

ALIN TOMESCU: For the header relay network?

AUDIENCE: Yes.

ALIN TOMESCU: I mean, I would be just be speculating, wishful thinking. They could. There is some problem with this header relay network as well. And we talk about it in the paper because this had a really network could withhold block headers from you. So you do have to distribute it. Like usually you don't want to just trust Whatsapp who's also doing the statements, who's also pushing the statements in the blockchain. You don't want to trust them to also give you the block headers. You actually want to fetch them from a different source that Whatsapp doesn't collude with. Yeah, Anne, you had a question?

AUDIENCE: Are there other header relay networks that are deployed already?

ALIN TOMESCU: Yeah, there was actually one on ethereum. There's a smart contract in ethereum, I think, that if you submit bitcoin block headers to it, you get something back. And then you can just query bitcoin block headers from the ethereum blockchain. Is anyone familiar with this? So I guess I didn't include it here. But another way to do it is to just publish the header is in an ethereum smart contract. Yeah. So there's crazy ways you could do this too. Yeah?

AUDIENCE: Will that one go out of gas? I don't know, my understanding of ethereum is pretty decent, but would that smart contract eventually run out of gas and not publish anymore?

ALIN TOMESCU: So to fetch from it, you don't need to pay gas. But I suspect to push in it, I actually don't know who funds that contract. So I guess you fund that when you push maybe. Maybe not because then you also want something back. Why would you push? I'm not sure. But we can look at it after. Yeah. It's a good question. So anyway, even if this header relay network is compromised, if you implement it in the right way, you've distributed on a sufficient number of parties, you can still get all of the properties that you need to, meaning freshness for the block headers. That's really the only property that you need to. The header relay network should always reply with the latest block headers.

So let's look at costs since Alin was mentioning the costs. So to open a statement, you have to issue a transaction, right? And the size of our transactions are around 235 bytes. So and the fee as of December 13 was $16.24 for transactions if you guys remember those great bitcoin times. I think it went up to $40 at some point too. So it was it was really hard for me to talk to people about this research back then. But guess what, the fees are today? So today, this morning I checked. And there were $0.78, right?

When we wrote the paper, they were like $0.12. So you know, here I am standing in front of you pitching our work. In two minutes they could be back to $100, but who knows? Yes, you had a question.

AUDIENCE: Maybe I'm wrong, but in a transaction, you can have some outputs. Can you have several statements in there?

ALIN TOMESCU: So that's a good question. So can you batch statements? And then the answer is yes. You can definitely batch statements. What we've said so far is in a transaction-- in a Catena transaction, you have this output. And you have this op return output where you put the statement, right? And you know, it spends a previous transaction. But as a matter of fact, what you can do and some of you may already notice this.

There is no reason to put just one statement in here. There is some reason-- so the only reason is that it only fits 80 bytes. So you could put easily, let's say, two or three statements in there if you hashed them with the right hash function. But a better way to do it is why didn't you put here a Merkle root hash? And then you can have as many statements as you want in the leafs of that Merkle tree, right? In fact, here you could have I don't know, billions of statements. So keep in mind, you will only be able to issue billions of statements every 10 minutes. But you can definitely have billions of statements in a single transaction if you just batch them.

So now, remember the blockchain will only store the root hash. This Merkle tree will be stored by the log server perhaps or by a different party. They don't have to be the same party. Does that make sense? Does that answer your question? Yeah. Right, that was my next point. Statements can be batched with Merkle trees. OK, so let's talk about the why since so far we've been talking abstractly about these statements. But what could these statements actually be. So let's look at a secure software update. So how do you do secure software update? An example attack on a software update scheme is that somebody compromises the bitcoin.org domain. And they change the bitcoin binary to a malicious binary.

And they wait for people to install that malicious binary. And then they steal their coins. They steal their data. They could execute arbitrary code. And an example of this was sort of this binary safety warning on the bitcoin website. At some point, they were very concerned that a state actor is going to mess with the DNS servers and redirect clients to a different server and make them download a bad bitcoin binary. So does the attack make sense? Does everybody see why this is possible?

You do need to sort of accept that the DNS service that we currently have on the internet is fundamentally flawed. It's not built for security. So the typical defense the typical defense for this is that the Bitcoin developers-- they sign the bitcoin binaries with some secret key. And they protect that secret key. And then there's a public key associated with the secret key that's posted on the bitcoin website, right? And maybe some of you'll notice that sometimes it's also very easy to change the public key on the website, you know, if you can just redirect the victim to another website.

And another problem is that not everyone checks the signature. Even if let's say you have the public key on your computer, you know what the right public key is, only if you're like an expert user and you know how to use GPG, you will check that signature, right? And the other problem that's probably I think a much, much bigger problem is that for the bitcoin devs themselves, it's very hard to detect if someone stole their secret key. Like if I'm a state actor and I break your computer and I steal your secret key, I will sign this bitcoin binary. And I'll give it to let's say, one guy. I'll give it to you. You know, and I'll just target you individually because

I'm really-- I know you have a lot of bitcoin.

Right, and then the bitcoin devs will never find out about it unless you know you kind of realize what happened. Then you take your bitcoin binary and you go with it to the bitcoin devs. And they check the signature on it and they say, oh wow. This is a valid signature and we never signed it. So somebody must have stolen our secret key. So this is really a bad-- kind of the core of the problem is that it's hard for whoever publishes software to detect that their secret key has been stolen-- to detect malicious signatures on their binaries. Does that make sense?

So the solution, of course, is you know, publish the signatures of bitcoin binaries in a Catena log So now if there's a malicious binary being published by a state actor, people won't accept that binary unless it's in the Catena log, which means people in the bitcoin devs will see the same binary. Right, so let me let me show you what I mean with a picture. So we have this Catena log for Bitcoin binaries. And let's say, the first transaction has a hash of the bitcoin 0.001 tar file, right, the bitcoin binaries. And this hash here is implicitly signed by the signature in this input because it signs the whole transaction.

So now if I put this hash in a Catena log, I get a signature on it for free. And now, if let's say, a state actor compromises the log server, gets the secret key, he can publish this second malicious binary in the log, right? But what that malicious state actor will want to do, he will want to hide this from the Bitcoin devs and show it to all of you guys. All right, so he'll want to equivocate. So as a result, he will want to create a different transaction with the right bitcoin binary there, show this to the bitcoin devs while showing this to you guys.

All right, so the bitcoin devs would think they're good. This is the binary they wanted to publish while you guys would be using this malicious binary published by the state actor. Of course this cannot happen because in Catena you cannot equivocate. Right? Does everybody see this? Right, any questions about this? There has to be a question on this. No? So this mechanism is called software transparency. It's this idea that rather than just downloading software like a crazy person from the internet and installing it, we should just be publishing these binaries in a log that everybody can see, including the software vendors that created those binaries.

So in this way, if somebody compromises a software vendor, that vendor can notice that in the log there's a new version for their software that they didn't publish. So you know, this isn't to say that it'll prevent attacks. You know, what a state actor can do anyway is they can just do this. They can post this H2 prime in here, show it to you guys including the bitcoin devs and still screw everyone over. But at least these attacks then go undetected anymore. All right, so it's a step forward in that sense.

Yes, so the idea is that you have to double spend to equivocate. And the other example that really-- the reason I wanted to start this research had to do with public key distribution. So let's say we have Alice and we have Bob. And they both have their public keys. And I'm using this letter b to denote Bob and his public key and the letter a to denote Alice and her public key. And they have their corresponding secret keys, right? And Alice and Bob, they want to chat securely, right?

So they want to set up a secure channel. And there's this directory which stores their public keys. So this guy stores Alice, pk Alice. This guy stores Bob, pk Bob. All right, and the directory gets updated over time, maybe Karl, Ellen and Dan registered. And if you have non-equivocation, if the attacker wants to impersonate Alice and Bob, he kind of has to put their public keys, the fake public keys in the same directory, which means that when Alice and Bob monitor-- they check their own public keys, they both notice they've been impersonated, right?

So again, the idea is that you can detect. Now how can this attacker still trick Alice to send an encrypted message to Bob with Bob's fake public key? Is there a way even if you have non-equivocation? So what's the attack? Even I have non-equivocation and I claim that the attacker can still get Alice to send a fake, an encrypted message to Bob that the attacker can decrypt. What should the attacker do? So pretend that we are here without the ability to equivocate.

So the attacker cannot equivocate. But I claimed that the attacker can still trick Alice into sending a message to Bob that the attacker can read. So now it's time to see if you guys paid attention. Somebody? Alin? Oh, you?

AUDIENCE: Does the attacker have to have the secret key?

ALIN TOMESCU: No, no. He does not. Yeah. He does not have to have the secret key. The attacker just creates fake public keys. Here's a hint.

AUDIENCE: If you only changed one person to choose, they don't know that it's a fake key so they could send it to a fake key for a bit?

ALIN TOMESCU: Yeah, so whose person should they attack or change the key for?

AUDIENCE: Like if they change Bob's, Alice will still think Bob's is correct so she'll send it to the fake Bob until Bob checks it.

ALIN TOMESCU: Exactly. So that's exactly right. So what's your name?

AUDIENCE: Lucas.

ALIN TOMESCU: Lucas. So what Lucas is saying is look, even without equivocation, I had this directory at T1. I had another one at T2. But at T3 and both of these had keys for Alice and Bob, right? But at T3, Lucas is saying look, just put the fake key for Bob here. And that's it. Don't put a fake key for Alice there, just for Bob. And now when Alice looks up this public key for Bob here, she sends a query to the directory hey, what's Bob's public key? She gets back b prime, which is equal to Bob pk Bob prime, right?

Alice can't tell if that's really Bob's fake public key. That's the reason she's using the directory in the first place. She wants sort of a trustworthy place to get it from. Bob can tell if Bob looks. But by the time Bob looks, it might be too late. Alice might have already encrypted a message, right? So again, what's the point of doing all of this? It's not like you're preventing attacks, right? And the point of doing all of this is that you get transparency. Bob can detect, whereas right now Bob has no hope.

In fact, so you said, a lot of you use Whatsapp. So you know in Whatsapp, if you really want to be sure, so I have a conversation here with Alin Dragos. So, Alin, do you what to to bring your phone here? Do you have Whatsapp? So if you really want to be sure that you're talking to the real Alin and not some other guy, you have to go on this encryption tab. Can you tape this?

So my phone is black and white. It's going through a depression phase. I apologize. So you have to go here and there's a code here, right? And Alin, can you do the same thing? You know what I'm talking about? I really hope I don't get some weird text message right now with the camera on the phone. OK, so now with Alin's phone, is that the same code? Can somebody tell me? I can't see it.

AUDIENCE: It's hard to see.

ALIN TOMESCU: OK, so we have 27836 and yeah. So it's the same code. Can you see it on the camera? So now because we have the same code here. With this code here, it really is a hash of my public key and Alin's public key. And if we've got the same hash of both of our public keys, then we know we're talking to one another. But we won't really know that's the case until we actually meet in person and do this exchange, right? So what this system does instead is it allows Alin to check his own public key and it allows me to check my own public key. This way if we check our own public key, we'll always know when we're impersonated even though I might send an encrypted message to the wrong Alin, Alin will eventually find out.

It's a bit confusing because we're both called Alin. So does that sort of makes sense? Yeah?

AUDIENCE: So what do you do when you realize that your key is the wrong key?

ALIN TOMESCU: Good question. So that's really the crucial question. What the hell can you do? Right, so this directory impersonated you. In fact, you can't even-- here's the problem. If you're Bob here and you see this fake public key. And you go to the New York Times and you say hey, New York Times, this Whatsapp directory started impersonating me. And the New York Times can go to the directory and say, hey directory, why did you impersonate Bob? And the directory can say, no, I did not impersonate Bob. Bob really just ask for a new public key. And this was the public key that Bob gave me.

And it's just a he said, they said, kind of a thing, right? So it's really a sort of an open research area to figure out what's the right way to whistle blow here. So for example, one project that we're trying to work on is there a way to track the directory somehow so that when he does stuff like this, you get a publicly verifiable cryptographic proof that he really misbehaved, right? No, there's no cryptographic proof. The fact that that public key is there could have come from the malicious directory or could have come from an honest Bob who just changed his public key. So yeah.

So again, a step forward but we're not there. You know, it's just that there's much more work to do here. And I think you also had a question.

AUDIENCE: Can't Alice just ask Bob if this is you?

ALIN TOMESCU: So that's a chicken and an egg, right? So we have Alice. We have Bob. And we have the attacker. Alice asks Bob, is this your public key? You know, let's say, b prime. Let me make this more readable. So attacker, right? So Alice asks, hey Bob. Is this b prime your public key? The attacker changes it. Hey Bob, is b your public key? The attacker-- Bob says yes. Attacker forwards yes to Alice, right? Remember, Bob and Alice don't have a secure channel. That's the problem we're trying to solve with this directory, right?

So the attacker can always man in the middle people. Well, people like Alice and Bob. If the attacker can man in the middle everything, then there's really no hope. And we're living in a very sad, sad world if that's the case. Yeah, it actually might be the case but we'll see. Anyway, so yeah, so I claim here that this is a step forward. But there's still much work to do. All right, so we get transparency. Bob can detect. He'll know. He won't be able to convince anybody. But if a lot of Bobs get compromised, you're still like in a place where everybody knows that something's off and will all stop using Whatsapp, for example. Right?

By the way, Whatsapp is a great tool. You should continue using it. I'm just saying it's difficult to use right like if somebody really wants to target it, they can play a lot of tricks to still trick you. So yeah, so all right, we already talked about this. And yeah, again if the director can equivocate, then you know, all bets are off because now Bob will look in this directory, he'll think he's not impersonated. Alice will look in this directory. She'll think she's not impersonated, right? So the reason we started this research is because I really want to-- my thesis is on building these directories that are efficiently auditable and have a hard time impersonating people.

So that's why we decided to look at how could you do this with Bitcoin. Yeah, so of course there's one project called KeyChat that we're working on with some high school students. And we're using the key based public key directory. And we're witnessing it in a Catena log so that stuff like that doesn't happen.

OK, so now let's talk about the blockchains. In general, people nowadays like to say I need a blockchain for x, right? I need a blockchain for supply chain management. I mean a blockchain for cats, for whatever. I've heard a lot of crazy stories. IOT, self-driving cars, blah, blah, blah. And I think the right way to think about blockchain is to never ever say that word unless you use quotes, first of all. And second of all, to understand what Byzantine state machine replication is. Right, and if you understand what Byzantine state machine replication is or a Byzantine consensus, you understand blockchain. And you understand all the hype. And then you can make some progress in solving problems.

In the sense that what is blockchain? So what we're doing here is we're doing a Byzantine consensus algorithm. We're agreeing on a log of operations. Right, by the way, that's what Catena does too by piggybacking on bitcoin. It agrees on a lot on a log of operations. Right, but the other thing that SMR or Byzantine consensus does is that it also allows you to agree on the execution of the ops in that log. So in Catena, you don't agree on the execution, you just agree on the statements. But there is no execution of those statements in the sense that you can't build another bitcoin on top of bitcoin in Catena because you can't prevent double spends of transactions that are Catena statements, right?

Like the Catena statements, you have to look in each one and tell if it's correct. So to detect a double spend in a Catena backed cryptocurrency, you would have to download all of the transactions and because you cannot execute it like the bitcoin miners do and build this UTXO set. I'm not sure this is making a lot of sense. But let's put it another way. In bitcoin, you have block one and then you have block two, right? And there's a hash pointer and there's a bunch of transactions here, right? And remember that what prevents me from double spending something here-- I can have two transactions in this block that double spend the same one here. What prevents me from doing that is exactly this execution stage, right?

Because in the execution stage, when I try to I execute this first transaction and I mark this output as spent, when I execute the second transaction, I cannot spend that output anymore, right? In Catena, you can't do anything like that with the statements. You just agree on the statements. In Catena, you would put this transaction in the log, this one and then that one and someone would have to detect that the second one is a bad one by actually downloading it. That's kind of what I'm trying to say here.

So in general, the way should you should be thinking about blockchain is through the lens of Byzantine consensus. And that'll get you ahead of the curve in this overly hyped space, right, because it's really just this. You're agreeing on a log of operations. And then you're agreeing on the execution of those operations according to some rules. The rules in bitcoin are transaction cannot-- there cannot be two inputs spending the same output more or less. There's other things too. What that gives you is it allows you to agree on a final state which in bitcoin are the valid transactions. That's the final state right. In ethereum, for example, the final state are the valid transactions, and the account balances of everything, and the smart contract state for everything. And I guess you'll learn about that later.

So you can build arbitrarily complex things with Byzantine consensus or with blockchains. And the high level bit, if you want to look at it another way, is that you have a program p, right, which could be anything, could be a cryptocurrency. And then, instead of running this program p on a single server s, what do you do is you distribute it on a bunch of servers, s1, s2, s3, s4, right? And now as a result, to mess with this program p, it's not enough to compromise one server, you have to compromise a bunch of them. right?

OK so, and some of you might also be familiar with this term permissioned blockchain. So when you distribute this program o amongst n servers where n is equal let's say, three f plus 1 and f is equal to 1 in this particular case. In a permission blockchain, this n is fixed, right? Once you've set n to 4, it has to stay 4. These servers have to know one another. They need to know each other's public keys. And only one of the servers, f is equal to 1, can fail. If more than one server fails, then all bets are off. Your program can start doing arbitrary things. In particular, if your program is bitcoin, it can start double spending. I'm moving a little bit fast, so I'll take some questions up until this point before I go on. Yes?

AUDIENCE: In a permissioned blockchain, do they use proof of work?

ALIN TOMESCU: No, you don't have to. And that's kind of what the hype is about. This stuff, there is like-- the first interesting paper on this was 1976 or something like that. So this is 40-30, 40-year-old research. We've known how to do permissioned consensus-- we used to call it Byzantine consensus for 30 or 40 years, right? So there's nothing new there. It's just that it's very useful nowadays to say blockchain then to say consensus because then you get 10 more million from your venture capitalist folks.

AUDIENCE: But if you don't have to do proof of work, do they ever do proof of work?

ALIN TOMESCU: It would be such a bad idea technically to do prefer working a permissioned consensus algorithm. It just-- completely unnecessary, plus probably insecure too. Yeah, so now the reason you do proof of work is because in a permissionless blockchain, this n is not fixed. n could go, let's say, n was 4. It could go to 8. Then it could go to 3. Then it could go to 12. In other words, people are joining and leaving as they please.

And the reason you need proof of work in bitcoin, one way you can look at it is that you're really turning a permissioned consensus algorithm into a permissionless one. And a consensus algorithm is just voting. These n folks are just voting. And you need 2f plus 1 votes to sort of move on, right? And if this n changes over time, like if the n becomes bigger, it's very easy to take over a majority of the voters. If I can just add fake voters to a permissioned consensus algorithm, I can just take over the consensus algorithm. In other words, I can take over more than f nodes.

Right, so the trick there is you have to prevent that from happening. And the only way to prevent that is to say, look if you're going to join and then make my n bigger, you better do some work. So that it's not easy for you to join because if you're a bad guy and you want to join, you know, you can do that very easily unless I require you to do some work. So that's kind of the trick in turning a permissioned consensus algorithm into a permissionless one. And in fact, the way these permissioned animals work is completely different than bitcoin. They are much more complex. Bitcoin is incredibly simple as a consensus algorithm.

If you ever read a consensus algorithm paper, you know, it's a bit insane. Also to implement, it's a bit insane. Bitcoin is very simple to implement compared to these other things, I mean, bitcoin is, of course, a complex beast as well. But you should look at let's say, practical Byzantine fault tolerant paper, PBFT, and try and implement that. So anyway, why am I telling you all of this? The reason I'm telling you all of this is because if you want to do a permissioned blockchain for whatever reason, one way to do that is to use your favorite Byzantine consensus algorithm. So that would be-- let's say pbft. This was 1999 from MIT.

So you could use that. You could have a lot of fun implementing it. Another thing you could do is you could take your program p and just give it to an ethereum smart contract. And you know, that the ethereum smart contract, if the ethereum security assumption holds, it will do the right thing. It will execute your program p correctly, right? But the other thing that you could do actually, is you could use Catena to agree on these logs, on the log of operations for your program. And then you could use another 2f plus 1 servers or replicas to do the execution stuff so that you can agree on the final state.

And this gives you a very simple Byzantine consensus algorithm. So remember Catena doesn't give you execution. It allows you to agree on the log of ops. To get the execution, you'd basically take a majority vote. If you see you have 2f plus 1 replica servers and if you see f plus 1 votes on a final state, you know that's the right state because only f of them are malicious.

So in fact, if you use Catena with 2f plus 1 replicas, I claim that you can get sort of a permissioned blockchain that sort of leverages the bitcoin blockchain to do the agreement on the log of ops. So in that sense, it's sort of a mix of a permissioned and permissionless. We haven't studied this like we don't know what properties it would have. So that's future work. And if you don't need the execution, for example, if all you're doing is you're agreeing on a public key directory-- like here, there's no execution. This directory just is supposed to stay append-only, we have some research that allows you to prove that every transition is an append-only directory.

And if you only need execution, you can just use Catena directly as I already told you guys for the software transparency application for the public key directory application. And if you want to do a permissionless blockchain then, of course, you would have to roll your own. But you have to proceed with caution there, right? It's not an easy thing to do. OK, so let's conclude now. What we did here is that we enabled these applications to efficiently leverage bitcoin's consensus, right? So clients can download transactions selectively rather than full blockchain and prevent equivocation.

Right, and you only need to get 46 megabytes instead of gigabytes from the Bitcoin blockchain. So why does this matter? These are just I think the three killer apps-- secure software update, public key directories-- by the way, the public directories also are applicable to https. So when you go on Facebook, you have to get Facebook's public key. The certificate authorities that sign the public keys are often compromised. So they're often fake search for Google, for big companies like that. And you might use it.

But if you have a public key directory, Facebook and Google can immediately notice those fake sorts. It's a step forward. And for more, of course, you can read our paper. It appeared in a IEEE security and privacy 2017. And I'll post the slide on GitHub too. So there are links there. Yeah so, again this is the high level overview of everything that's previous work and our work. The difference is very small.

And now we can also talk about other stuff. In fact, I have more stuff to talk. But before we do, I'd like to have a discussion with you guys if you have questions. So?

AUDIENCE: So could you implement Catena in the actual bitcoin node? Would that be something that they would want? It seems like it would be a good feature to add? Or is it strictly separate?

ALIN TOMESCU: Yeah, I don't think you need to. That's the whole point, right? The whole point of the research is how do we use bitcoin without getting the miners to accept a new version of bitcoin, without changing bitcoin in any way? So no, I don't think-- there's nothing to do really, we're just-- we're taking bitcoin as it is and we're piggybacking on top of it. We couldn't change it. I mean, there's a lot of things you can do in some sense. But then you get a very different system. Very different, we can talk about it more if you want. Yeah?

AUDIENCE: You were talking about how you can use this system to verify software binaries and how you want this to run in SPV modes on phones So how do you install software on the phones say, through apple and the app store. Is there a way to sign the binary that you actually get from the app store?

ALIN TOMESCU: I think your question is really about how do appstore binaries-- how do you Verify App store binaries? It's a chicken and an egg in some sense right, is that what you're saying? Yeah, you're right. Eventually, I mean in the best case, wishful thinking would be to say that look, the app store does this already for all of the binaries that they publish, allowing the developers to make sure nobody is posting malicious binaries on the app store for them. Yes?

AUDIENCE: Who do you envision running it?

ALIN TOMESCU: So I'd really like to see Keybase run Catena, it seems like a missed opportunity that they don't do this. I'm sure they have better stuff to do but it's just really easily allow Keybase-- let's say Keybase has a mobile phone app, it would allow that mobile phone app to verify the directory and get much, much, much more security. You know, no equivocation as long as nobody forks bitcoin. Since Keybase is already publishing these digests but they cannot be audited efficiently on a mobile phone. I mean they can but not securely you know, because full nodes can lie.

So there's a big problem with everything I said so far. And nobody caught it. So one problem is that what do you do when you run out of funds? Remember I said the log server starts with two bitcoins. Let's say it issues thousands of transactions, starts paying those $40 fees and it runs out of funds. What do you do? Then Yeah?

AUDIENCE: You can maybe reload the new transactions with this transaction [INAUDIBLE].

ALIN TOMESCU: What's your name?

AUDIENCE: Raul.

ALIN TOMESCU: Raul. So Raul is saying you can reload. And that's exactly right. We just have to change the transaction format slightly. We talk about this in the paper as well. But just to demonstrate real quickly. Suppose, let's take a ridiculous example which hopefully will never happen in bitcoin. But suppose that the Bitcoin fee is 1 bitcoin. So now I have one bitcoin here. And I have s1 here. And now I have zero bitcoins here. All right, so zero bitcoins in this output. And maybe s2 here.

So that would be terrible. Right, now I can't go on. Does everybody see this as a problem? Right, so what Raul is saying is look at another input here and make it take coins from some other transaction whatever, 20 bitcoins. And now you get 20 bitcoins here. Right, so you can easily refund transactions. There is a bit more subtlety there in the sense that you don't want to join logs-- let's say if you have two logs, GTX and GTX prime for different applications. Right, and they start issuing statements-- s1, s2. You don't want to be able to join-- I'm sorry, s1, s1 prime-- these two locks to a single log for certain reasons right.

But this doesn't actually allow you to join them in the sense that-- let's say you actually do this and join them. Right, so let's say this transaction here came from GTX prime, right? And there was an s1 prime here. And you did this, right? So the problem is this is no longer a valid Catena transaction for this log because a valid Casino transaction-- the first input spends the previous transactions output. But in this chain, it's the second input that spends the previous output. So I cannot join logs. And this matters for a bunch of reasons that we don't have to go into.

Yeah, so we talked about the about batching statements. I want to show you guys some previous work, so how did some previous work do this since there we seem to have a bit of time. OK, so there are some previous work called liar, liar, coins on fire. Have you guys seen this? So the idea here is that it is a really nice piece of work and lots of people in the bitcoin community already know about this. And I think it was an idea before the paper or maybe not, I'm not sure. But Tadge has a similar idea.

So for example, suppose we have this authority. Imagine this is a Catena block server and it publishes a transaction which locks to bitcoin and locks those bitcoins to that public key. And this authority sometimes will want to say two different things. It will want to say s and s prime, right? And it will sign the statements with their secret key. But the secret key that the authority uses to sign statements is also a bitcoin secret key. It's the same secret key that the authority used to lock to bitcoins, so $20,000 or something. I'm not sure if bitcoin plummeted since yesterday, can never be sure. Does the setting make sense?

So I have an authority. It issues statements just like before. And it numbers them with i, let's say. And we want to prevent this authority from equivocating. We're not actually going to prevent it. We're just going to disincentivize it in the sense that if this authority equivocates like this for the same statement i, what I claim is that anybody can then steal that authority's bitcoin because equivocating like this reveals the secret key. And the reason it reveals the secret key is because the signature shares the same i here.

So how many of you guys actually, you did cover Schnorr signatures, right? So did Tadge talk about how to do this with Schnorr?

AUDIENCE: This is the [INAUDIBLE].

ALIN TOMESCU: But did Tadge cover it?

AUDIENCE: Yes.

ALIN TOMESCU: OK, great. So yeah so let's go over that less briefly. So again, the idea is that if someone observes these two signatures on conflicting statements for the same i, there is this box where you can put the two signatures and get back the secret key. And once you have the secret key, we can spend this transaction and get that authority's bitcoins. And then there's a lot of details to get it right because you might notice that if the authority does this, before they do this, they might already be spending this transaction themselves so as to prevent you from taking it. And details on how to prevent the authority from doing that are in those paper.

Yeah, and then you know, whoever discovered this can spend those bitcoins. And the idea is that this disincentivizes equivocation by locking these funds under the secret key of the bad authority. But it does not prevent it. Right, so in Catena we actually prevent equivocation. We say, if you want to equivocate, you better fork bitcoin. Here they say, if you want to equivocate, you're going to lose $20,000. Right?

But you have to understand like this could be a good authority that locked $20,000 here. But the attackers are going to be-- they're not going to care about those $20,000. They're just going to steal the secret key, equivocate and then the authority is going to be left without money. If the authority is the attacker, then this makes sense. But if the attacker is not the authority, then this makes less sense because the authority sort of risking their bitcoins on the assumption that no attacker can compromise them, which you know, if you could do that in computer science, I wouldn't be sitting here talking to you guys.

OK, so now how do you do this? So do you remember Schnorr signatures real quickly? An easy way to do this is using Schnorr signatures. And I think-- I could be wrong, but I think this new SegWit update to bitcoin allows Schnorr signatures, right? So with SegWit and with Schnorr signature, as you can definitely do this. And the idea is that Schnorr signature if you recall, is just k plus h of m, g to the k s, where s is the secret key. Right, so this is a Schnorr signature on m, right? How many recall this?

Right, and of course, it's not just this. I mean, the signature is really this and this h of m, g to the k. So it's these two things, right. But now I want to show you that if I sign two different things, I can actually get s. But if I sign them in a certain way-- so remember what I said before is I would like this authority to sign i m and i m prime. Right, and if the authority does this, I claim that I can get the secret key out. But I have to have the same i. So we relax this in a sense that we're not going to use i here, we're just going to use this g to the k.

So if the authority uses the same g to the k to sign, then we can extract the secret key. And the way we can do that is I'll just show an example, sig1 would be k plus h of m1 g to the k. And it'll be sig 1 and I guess the associated hash E1 would be h m1, g to the k. Does everybody see this? And then sig2 would be k plus h of m 2 g to the k. So again, oh, you guys are clearly not paying attention. I forgot the secret key. E2 is hash of m2 g to the k. OK, so I'm using the same g to the k and the same k.

So now how can I extract the secret key? Does anybody see a solution to this?

AUDIENCE: It's now just system of two equations and two variables where k and s are the unknown variables.

ALIN TOMESCU: So I think what Rahul always saying is that s is just sig1 minus sig2 divided by e1 minus e2. Is that right? Yeah. See this, because if I subtract this from this, I just get h m1-- I'm not going to have space here. I get-- let's say it here-- h of m1, g to the k minus h of m2 g into the k times s. All right? And now I take this here in the denominator and I simplify and I get s. All right.

So it turns out that that's kind of the trick that this word leverage is more or less. The only sort of caveat here is that remember I said, there is a position i for the statement. But now I'm saying, there's no longer a position. We have to use this g to the k. And how do we map positions that are g to the k is another trick that you have to do but you can do it. And if you want more details, you can read the paper and I'm sure Tadge will tell you even more details about this. I think the lightning network also leverages this trick in some cases.

All right now, let's talk a little bit about some attacks if there is time. The most interesting attack would be the generalized vector 76 attack. So this is a screenshot from a paper. Can everybody see this? So the generalized vector 76 attack is very interesting. So you have you have an attacker, right? And remember, the goal of the attacker is to replace-- let's say this

TX1 with that TX2. It's going to show this TX1 to a merchant saying hey, I paid you money. And then it's going to show this TX2 to the merchant sending the money back to himself.

So the whole trick is to fork the merchant on this side and then to switch him back to that, right. It's a sort of a pre-mining attack and this attack is much easier to pull on SPV notes because what the attacker does is-- let's see-- he works on the secret chain with TX1 in it. And at some point, the main chain might take him over, might win. So the attacker kind of gives up and keeps trying. But at some point the attacker gets ahead, right in the second chain. He gets ahead of the main chain. This stuff is not posted yet.

And let's see this merchant only for some reason because they're silly, they only need one confirmation to accept the TX1. And for other silly reasons, this merchant is an SPV merchant, right? Let's draw a figure here. So this merchant is an SPV merchant that only needs one confirmation to accept the transaction. So we have the attacker. And we have let's say, the SPV merchant. Right, and now the attacker's sent you know, this was the main block there, let's say, this was bi. And the attacker forked it. He put TX1 here. And then had a confirmation on it, right? So he sends this chain to the merchant.

And the other miners haven't found anything yet. Right, so the attacker is a bit ahead. So he's pre-mining. So far, so good. Are you all with me? And remember that he really just is showing block headers. So these are not full blocks. He's just showing block headers. They're much smaller blocks, 80 bytes. And the blocks are missing. All right, so when he shows TX1, he is just showing like a Merkle path to TX1 to the merchant. You guys with me?

OK, so now he did this to the merchant. And now what the attacker is going to do, he's going to relax, sit back, and post the TX2 the mines-- give TX2 to the miners. Let's say the miners are here. Hes going to post TX2 to the miners. And he'll stop mining the attacker. He's not going to mine anymore. But he got the merchant to accept the payment. And the merchant shipped the goods. Maybe this is an online purchase. And so now the miners, they will find the next block bi plus 1. They'll put TX2 here. They'll find the next block and then the next block. And they'll take over. Their chain will be the main chain.

And eventually this merchant will hear about this main chain. And he'll just see the headers, of course, because he's an SPV merchant. So now the SPV merchant just got double spent. Does everybody see this? That I just double spent the merchant? And sort of the fundamental problem here is that the merchant only received block headers. So he cannot take these block headers and broadcast them to the miners because miners don't mine on top of block headers. Miners mine on top of full blocks.

So this merchant cannot has no protection against this if he's doing SPV-- if he's accepting SPV payments. If these were full blocks, what the merchant could have done was the following. The merchant would have would have received the full block here with the transaction in it, right. So again the attacker got ahead. But now the merchant because he saw a block, he's going to ship this block with TX1 to the miners because he's a full node. Full nodes do that. When they hear about a block, they broadcast it.

So now the miners know about this block. They're going to continue mining on top of it. And in fact, the merchant will send both blocks, right to the miners. So now miners will continue building here. So now the attacker has a bit of a tougher problem on his hands. There is still a way to trick even full nodes, even if this guy is a full node, you can sort of leverage some timing assumptions to still trick the full node. And the details are in that paper over there. But that's one way-- somebody was asking, I think you were asking Anne, right-- about SPV nodes and how they're less secure.

And this is one fundamental way in which they're less secure. If you accept payments with SPV nodes you're really playing with fire. You know, you do need a sufficiently powerful attacker who can get ahead. But yeah. OK, so with that I think that kind of concludes the lecture. Any final questions? All right, cool. Thank you guys.