TADGE DRYJA: OK. So today, Discreet Log Contracts which is very linked to Lightning Networks. So if you didn't get the last two lectures, this might not make that much sense. So I hope-- but I think everyone here is like up to date on these things.

So today, I'm going to talk about Discreet Log Contracts, which is a paper I wrote last summer, but it's evolved out of Lightning Network. And I'll try to sort of explain how it-- hopefully, you'll see the connections and how you can get from one to the other. We'll talk about oracles, talk about anticipated signatures, which has some fun math things, and talk about building the Discreet Log Contracts themselves and how that'll work.

OK, so conditional payments, I guess you could call this smart contracts. Smart contracts is a pretty vague term, and it's used a lot in this sort of Bitcoin, blockchain, Ethereum world. And usually, it's pretty-- like, a lot of times, they use it as a buzzword and it's like, oh, I'll put my land title in a smart contract and then I can buy a house.

And a lot of times, it's used in ways that make it seem as though you don't need a government anymore or a judicial system anymore to enforce these contracts. And in the case of something like Discreet Log Contracts, that's kind of true. Because what Bitcoin did and what made people pretty impressed was that, hey, you don't need a government to enforce the scarcity of the currency that's being created. So that's kind of cool.

But it doesn't mean that you can now use blockchain technology to not need a government to enforce, say, rights to property. If you say, hey, this is my land and I've got a smart contract that says my land is-- you know, this is my land on the blockchain, and then an army comes and says, well, we're taking your farm, and you're like, no, it's on the blockchain, and they don't really care.

So it's hard-- like, another way to think about it is the only thing the Bitcoin network can do is move Bitcoin, and the only thing the Ethereum network can do is move Ethereum, to some extent. It doesn't extend out to like, you know, autonomous drones to shoot you if you trespass

or anything like that yet.

OK, so the simplest and, I think, most straightforward and maybe the most useful type of smart contract is a conditional payment where it's basically, I'm going to pay you based on some external data. So I'm going to use the example of Alice and Bob betting on tomorrow's weather. If it rains, Alice gets a coin. If it's sunny, Bob gets a coin. And we need some kind of way to get this data, right. OP_WEATHER is not in Bitcoin.

And I think, yeah, bet is a word that-- it's weird. Like, when I'll meet with fancy rich people, like who are in companies that come to the Media Lab-- no, I still use the word bet. Like, to me, everything's a bet, right?

Insurance is a bet. Like, if I buy car insurance, I'm betting GEICO or whatever. Bet I'm going to crash my car. And they're like, I bet you won't. And then if I crash my car, they're like, aw, shoot, and then they give you the money.

And if I get health insurance, I'm like, I bet I'm going to get cancer. And then they're like, bet you won't. And then if I get cancer, I win and I get a whole bunch of money.

Right. Well, so they're offset by the fact that you don't actually want to get cancer and you don't want to crash your car. And so the fact that you're betting that you will do it means that, in the case that something bad happens, you get a bunch of money to defer this cost. But fundamentally, from the insurance company's perspective and your perspective in this insurance contract, it's a bet about whether you're going to do this or not.

And like almost all the financial contracts, you can look at as bets, like derivatives and futures and all those kinds of things. OK, so to keep it simple for the example, there's a very limited set of outcomes. Right? It's either rainy or sunny tomorrow, and we don't know that it's going to-- what the weather is going to be tomorrow yet, and we're going to bet the coin.

OK. So yeah, we need oracles. And I would argue that the Lightning Network script is essentially a smart contract, right, the this key now or this other key later.

In Lightning, however, we don't have external state. Right? There's no need to query a third party. There's no need to query the outside world. And all the data that is used in the channel is generated by the participants of the channel themselves. Right? They're making up random keys, throwing them-- you know, sending them to each other just so that they can negotiate

over the balances in that channel.

That said, they're probably exchanging something outside of the scope of the system. Right? They're probably trading, you know, I'll give you a little bit of Bitcoin for some cookies, or something like that, and there's some delivery of goods or services that's not in Bitcoin. But for the state itself, it's all internal.

If we want external state, we need some way to get that external state into our system. Usually, this is called an oracle. OK, so the simplest oracle would be two of three multisig, and there are places that do this.

And it's not-- I don't want to say it's like a stupid idea. It's actually quite powerful. And Bitcoin enables it in ways that you couldn't necessarily do before, but there's problems with it.

OK, so if you just say two of two multisig, where Alice and Bob both put coins into a two-- you know, into a channel-like structure where they both need to sign, and then they can say, OK, well, at the end of the end of tomorrow, whether it's sunny or rainy, we distribute the coins.

The problem is, it gets stuck if they disagree. Right? Also, rich players are at an advantage. They can say, well, I think I'm right. I'm fine not signing to get these coins out. It works great with friends, sort of gentlemen's agreement, ladies' agreement, or shake hands. OK, yeah, you're right. It was sunny. But Bitcoin is the currency of enemies, so you want a third party to decide in the case of conflict.

There's a lot of, actually, interesting sort of game theory things here where people have tried-- I've seen reports-- like, I've been to talks where people are like, "Yeah, you don't need an oracle. You just agree on it without any outside influence."

And there's some fun things, some fun attacks, where you can basically say, OK, Alice, it rains, you win, but I'm not giving you the money. And I can't get my money back either, but here's what I'll do. I'll sign a bunch of time locked transactions, and if you want a little bit of your money, you can take it right now. If you want more of it, you just have to wait weeks, months, years.

And then Bob can say, OK, it's up to you. Right? Like, how much of your money do you want back? You can wait. And so you're playing off the time value of money, kind of thing.

Anyway, so I have seen many things where it's like, you don't need an oracle. Just have these

two parties agree. And like game theory, hand wave. But I have never been convinced by them. Like ultimately, if we're in a bet and we don't trust each other, we need some arbitrator to decide the outcome of the bet.

OK, so you have a third party, right. You say, hey, we're going to bet on this weather, and if there's a dispute, we nominate Jeff, over here, and he's going to be the one that decides whether it actually rained or was sunny in order to determine who gets the money.

OK, so you have three keys, right? You've got Alice and Bob who are in the bet and you've got-- oh, well, that wasn't how I thought-- Olivia, who is the oracle. And if Alice and Bob are chill, they both sign and they don't even contact Olivia. Right, they're like, yep, it was rainy. OK, you get the coins. Don't even involve Olivia. She doesn't have to deal with anything.

But if they fight and they're saying, you know, this is totally not raining. This is like, right now, this is not rain, right? This doesn't count. They're like, there's nothing. It's basically sunny. So if they fight about this, they can ask Olivia to sign. Either of them can contact Olivia and say, hey, my counterparty is either unresponsive or uncooperative. Can you sign? And then sign off on the fact that it is, in fact, raining. I get the coins.

The problem is, let's say it's sunny, and then Alice tells Olivia, hey, it's out. Hey, it's Alice. Say it's raining, and I'll give you 0.8 points. Right. I'm not supposed to get anything, but give me 0.2, you get 0.8, win-win for both of us. So the oracle can be influenced and bribed.

Right, so the problem here? You've got interaction between the oracle and the participants in the bet. Two of three multi sig oracles, they see every contract. Right?

The contract itself is essentially not a smart contract, really, anymore because Olivia has to know like the legal-ish looking terms of the contract, right? Olivia, the oracle, has to say, OK, in what case do I side with Alice? In what case do I side with Bob?

It could be programmatic, but it could also just be a sheet of paper with a bunch of text on it, and Olivia's an actual person and determines based on that. Right, so this is just sort of a custodian arbitrator kind of thing. It's not that cool, like high-tech cryptography kind of thing, really.

Yeah, and they also decide individually, so they can equivocate. So if everyone's betting on the weather, they don't have to sign that it's sunny universally. They can sign to sunny on this side and rainy on that side. There's nothing you can do about that.

So I think what led me to think about these, it'd be better if the oracle couldn't equivocate, right? They can't deter-- you know, they can't say two different things. And it's even better if they never see the contracts, but how do we do that?

OK, so any questions about the general setup of what we're trying to accomplish here? Bets? And then you can have multiple-- you know, lots of different types of bets. But anyway, this is the essential problem. How do we do this?

And like in Ethereum, most of the time, everything's on chain, right. The data sources are published on the Ethereum blockchain. So you sort of have similar problems.

There's ideas of, OK, let's have a bunch of oracles, and then you sort of take the average of all oracles. There's different ways, but this thing is, OK, make the oracles never see the contracts. But how? OK, and then how is going to be kind of complicated.

So remember these from last time? Right. These sign with this key, and you have to wait. Sign with these two keys, and you get it immediately. This kind of same exact script is like, so that will be used.

OK, and then a think I talked about this on the third class about signatures, but we're going to go more in-depth in Schnorr signatures here. OK, so in Bitcoin, we've got this ECDSA elliptic curve. In Bitcoin, you use ECDSA, Elliptic Curve Digital Signature Algorithm, which is different than what I'm going to explain.

But this aspect is the same, right? The public keys are private keys times a generator point. So we can do this without actually using the Bitcoin signature algorithm. We can use our own external signature algorithm, and then use the keys generated by that in Bitcoin itself.

OK, so a and b lowercase, scalar. Big A, big B, uppercase, they're points on the curve. So what operations can we do? I think this is review, but this was like months ago. Regular numbers, you can do whatever you want. They're regular numbers. It's easy.

Add, subtract, multiply, divide, it's literally in the software just like BigInt.add, and the BigInt addition and mult-- you know, the BigInt operations, if you actually look underneath at the code, they're real simple. You're just adding numbers. So you can do whatever you want with these.

Points. Points are a group. They do not have two operations defined. They only have one operation defined, and we call that addition. It's sort of-- like, you could call it something else. Sometimes, they use a different symbol than plus, but addition's sort of what we call it.

And it works the way you'd think addition would work, right? You add these two things. If you add A and B, and then subtract B, you get back to A, things like that.

OK, so you can add and subtract, but you can't multiply or divide. Right? These are not defined.

OK, and you can mix these things. So you cannot add a point and a scalar. That's also undefined. So if you have like a point here, and you want to add 5, it doesn't make sense. However, you can multiply a point by a scalar. And you can divide a point-- yeah. You can divide that.

And the way you do that is you just, in the actual elliptic curve, you-- to double a point, you take the tangent. So instead of drawing a line between two points, you say, well, I'm drawing a line between this single point, which basically means the tangent, as you sort of like-- things like limit as the spacing between those two points goes to zero.

And then if you can double a point, if you can add A plus A, then you can multiply by any arbitrary number by saying, OK, I'll take 2A to be a tangent. I'll take 4A to be the tangent of that. I'll take it A to A to be the tangent of that. And then you can add up, like you can do binary decomposition-- and computers are real good at that-- and say, OK, I'm going to-- I need 5A, so I'm going to take A plus 4A. Right.

Now-- oop, yeah. So you can divide-- divide's a little annoying, but you can do that.

OK, so you've got all these operations we can do. You can do whatever you want with the scalars. You can add points, subtract points, multiply these points by scalars.

With this alone you can do-- and then, also, well, you have to throw hash functions in there for the stuff we're going to do, but you guys know all about hash functions. So with this and hash functions, you can do like ridiculous amounts of stuff. It's really kind of impressive.

Also, you can-- so yeah, part of the signature system is you pick some random point G. Sometimes, in other systems, you have G and H, two other points. But the idea is everyone just agrees, we've got some point G and everyone agrees on it.

OK, a nice property. You can add keys that you can-- you know, the addition sort of works-- addition in the scalar world works in the point world as well. Right? So if you have a times G plus b times g, and that's a point, that's equal to a plus b times g, also going to be a point. So that is what we're going to use.

And you can use this to share keys. Right, I think I explained this last time. If you know little a, but you don't know little b you can't sign. But if you learn both, you can sign for this, you know, this summed pubkey. So this is a way to like share keys.

OK. Yeah, [INAUDIBLE]. Yeah, like OK. Well, that's sort of out of order.

Anyway, Schnorr signatures, which are going to go into Bitcoin probably not even this year, maybe next year, and there's a lot of cool things you can do with them. I think I'm talking about that next week in the term. Like, that's aggregation, right?

But the basic idea of a Schnorr signature, you've got a public key that you've already told everyone. Right, so before anything happens, you pick a random private key little a, multiply it by G, and then compute big A and tell that to the world either by send-- having that as an output address in your Bitcoin script or publishing it on a website. You know, somehow, you propagate, hey, this is my public key. This is me.

OK, and then when you sign, you come up with another private key, pretty much the exact same as little a. It's going to be called little k. And then you compute R, which is k times the generator point. So this R is pretty much the same idea as-- it's another public key, but this will only be used once. And then to sign, you compute s, which is k, that thing you just made up, minus the hash of the message the sign concatenated with R times your private key.

OK, and then the signature is R and s. And then for someone to verify this, they know m, right, the message you're trying to sign. They know your public key, big A. They know R and s, your signature. And they try to-- what they do is they just multiply both sides of this equation by G. Right? So you can you can do that, same as regular math, multiply both sides by the same thing.

So they can't verify this because they don't know little a, they don't know k. But if you multiply it by G, well then you get s times G equals k times G minus this hash times a times G. Well, that is just your public key, right? This is R. And this, you also know because they gave you s, so you know G. That is, you know s times G. So that's how you verify.

And it's-- like, there's all sorts of proofs about why you can't forge these and stuff like that, but it sort of makes sense. Right? If you're trying to forge a signature, and you say, hey, I'm going to solve for s. Right, well, I can't solve for s directly because I don't know little a, I don't know k.

The problem is, I want to-- OK, I want to solve for one of these things. I've basically got an equation where I've got R minus the hash of R, and I can't tear those apart. Right, like I need-- how do I-- OK, solve for R, but there's like the hash of R and R on the same side of the equation. I could try to move this so that, OK, R equals the hash of R? Like wait, how am I going to do that?

Right, so anytime you try to solve, you're going to have an R and a hash of R in the same equation. And, like, that seems impossible, right? OK, so this is Schnorr signatures. It works great. It's got nice lots of nice properties.

Actually, all of this could work in ECDSA. It's just that ECDSA is like really ugly and complicated.

But anyway, so what Discreet Log Contracts uses is a fixed-R signature. It's the exact same equation, but you sort of move where R is considered. So before, you'd say, the public key is A, and now, my signature is R and s. Now, we're just going to say, OK, make the public key A and R, two points, and make the signature just s.

So same equation, everything's the same. It's just that you generate k at the same time you're generating A, and publish both A and R as your public key. Right, no problem.

OK, the problem is you can only sign once. That's why you don't usually do this.

There's no size difference. Right, it's like, OK, well, your public key gets twice as big and your signature gets half as big. No net gain. But you can only sign once, and why? OK, I'm actually-- it looks kind of scary, but it's actually not too bad.

So the idea is I sign. I have an A and an R, and those don't change. Right, so I have a little a and a k, and those don't change. But I'm signing two different messages. So the idea is I have signature 1, s1, which is k minus the hash of message 1 concatenated to R times my private key. Signature 2 is k minus the hash of message 2 and R times my private key. So I publish both of these signatures.

And then someone says, oh, I'm going to subtract these two signatures from each other. Right, so s1 minus s2, well, that would be this top thing minus this thing, which is k minus the hash of message 1 times a, minus k plus the hash of message 2 times a. The k's cancel out. That's the important part here. So k's are gone.

Cool, now I've got this s1 minus s2, which I can compute. Right, s is just a scalar, s1 minus s2 takes like a millionth of a second on my computer to do. Now, I've got this s-- you know, this thing I know, s1 minus s2 equals the hash of message 2, R times little a, minus the hash of message 1, R, little a.

I don't know a little a, right, so I can't figure out this. But I know what's on this side of the equation. Right? I know this sort of subtracted s. What can I do here?

Well, I can factor out a, right, because a is multiplied on both sides of this. So now I can factor-- you know, put a little parentheses. OK, this hash is message 2, R minus the hash of message 1, R times the private key here. And now, I can divide.

And well, this is the whole equation, right? I divide out. I divide by little a here. The thing I know divided by a thing I don't know equals a thing I know. Right, so I can move things around and say, OK, well then a is this thing I know, right, the difference between the two signatures, divided by the difference between these two hashes. And those two hashes, also, I know. Right? I know what the messages are. I know what the R value is. It works.

So the big thing here is if you have the same k value and the same a value, you can find their private key. So normally, this is OK because k is different. Right? So you'd have k sub 1, k sub 2 here. And then if you build this equation, you've got these k's which do not cancel out because you don't know what they are and you don't get anywhere with this.

OK, but so if you do use the same k value, you will lose your private key. Fun fact, I first learned about this in 2011 from the PlayStation 3 code-signing thing. And yeah, that was when I-- I think, because I started to get into Bitcoin and, also, it was like, oh, PlayStation 3 got hacked. Cool.

So Sony had a k-- like, they probably read up on how to do this. And they said, OK, k needs to be random, so they just made a random number and hardcoded it as the k that they used. I don't think they realized that it has to be random and different each time. So it was just a single random k.

Yeah, there's other cases of this being a problem. In bitcoin-blockchain.info, I believe, a couple of years ago, had something where k was random, but it was like only four bytes of randomness or something, and so you'd end up getting some occasional k's that were the same. And then you could compute people's private keys.

There's a couple other things like that where if-- you know, so actually in practice, now, you don't make k randomly in Bitcoin Core, in most of the Bitcoin libraries. You make k basically the hash of your private key and the message so that it changes for each message. And it's also got like secret information so no one can figure it out. That that's called RFC 6979, and it is safer because you don't keep needing new randomness each time. Oh, yes?

AUDIENCE: Is this presented for history's sake, or are people still making this mistake?

TADGE DRYJA: I mean, will people? Like, people have made it in very recent history. I wouldn't be surprised if people continue to make this mistake. You know, a couple, two, three years ago, Blockchain.info had this where the k's were the same. It's not immediately-- you know, you have to sort of look for it. But yeah, people still make all sorts of mistakes like this.

There's-- elliptic curves are actually less, I think, of a minefield then like RSA. RSA has all sorts of like, I had no idea that would break the security of the system, but it does. Like, you know, in an RSA, you've got p and q, two big prime numbers you multiply to get your modulus n. But if p minus 1 or n minus 1 is smooth, which means has many small factors, then you can break it and find their private key.

Like, I didn't know that until last year in a class. I was like, really? Oh, wow. So there's all sorts of bad things that can happen.

In elliptic curves, it's kind of nice in that your random numbers can really just be random numbers. So it's not too-- it's not as bad. But you do have to watch out for some of these things.

OK, so this new system, right, if your pubkey is A and R and a message m, right, you cannot compute s. So I know they're a public key. Right, I know R. I know they're probably-- like, I know this whole thing. I can't compute s, right, because s is in terms of k and little a. I can compute s times G, though, because s times G is in terms of R and big A, which I know.

Right, so you can't compute little s, but you can compute s times G. So s times G is

computable for any given message. So if someone publishes this new type of pubkey, A and R together, I can say, OK, well, what will it look like if you sign the message, sunny? Right, if you just take the word sunny, stick it in here, concatenate it with the R value you already told me, hash that, multiply by your pubkey, and then subtract that from your R value, I'll get a point on the curve and I can compute that. And I can compute that for any m I care to throw into this equation.

So this is kind of neat, right? I can compute a point on a curve for any message that you have not yet signed. And then the signature itself would be s. This sort of is looking like a key pair, right? So this is an unknown scalar, right. I don't know what s is, but I can compute s times G. This is a public key, private key, right? This is the exact same setup.

So now we can use this for a third-party oracle to sign messages. And then when they sign the message, they're revealing a private key to a public key you've already computed.

So when I was making this, I definitely didn't start with Schnorr signatures and do this. I had my own crazy scheme, which had like all these different equations and, like, R was multiplied by this other hash and there was all these things. And then I was talking to people and eventually was like, wait, you don't need all this. This is just a Schnorr signature where I already know the R. So OK, this is much nicer. I don't have to write this whole crazy proof and stuff.

OK, so we can use this. And we can use Olivia's signature, s, as a private key in our transactions. And we use s times G as a public key. And then we can also do the fun stuff where we mix with Alice and Bob's public keys. Right?

So I can say that Alice's pubkey plus s times G is the public key for this contract. And Alice's private key plus s is going to be the private key for this contract. So I can say-- what we can do is we can say, OK, I'm going to compute an address where it's your pubkey plus the signature of this oracle signing the word sunny times g, and that's the pubkey for Alice's sunny pubkey. Right?

Alice can only spend this if the oracle signs the word sunny and she has her private key. If the oracle signs the word rainy, Alice doesn't know how to-- you know, doesn't learn the private key.

Any questions about that part? Make sense? OK. So this is kind of cool.

So what you can do, so quick recap Lightning, you'd use the mechanism where you're adding these keys to revoke old states, right. You say, OK, if I give you this key, you can now spend it immediately. If I give you this key, you can now spend it immediately. So we enforce that with the most recent one is the OK one because if I broadcast on these old ones, you can immediately get it.

In Discreet Log, I switch it around a bit. I say, well, you know, it looks like a channel. We both put money into here. And then we build three different transactions all at the same time, before we even do anything. Before we actually fund this, we build these descendant transactions.

And the idea is that the public keys in here-- this is Bob's public key, but it's also got the sum, the sum of Bob's public key plus the oracle's signature of the word sunny. So if Bob broadcasts it, and the only way he can immediately take his funds is if the oracle assigned the word sunny, and he knows that s value.

If he broadcasts it and then waits-- right, if he broadcasts it, but it's not actually sunny, then Alice can take the money after a little while, right, after a day. You'd use the same timeout procedure where, OK if I broadcast this and I know the key, I immediately sweep it. If I broadcast it and I don't know the key, my counterparty can take everything after a day. So yeah, all states are created at the start. The validity is determined by the non-interactive oracle signature.

So the oracle signs cloudy, which is, yeah. And now, either party can broadcast the cloudy transaction and then take their money immediately. If anyone tries to broadcast, so if-- for example, if Alice tries to broadcast rainy at this point, the oracle has signed the word cloudy, they're not going to sign the word rainy. If they do sign the word rainy, they reveal their private key. So if Alice broadcasts this, her money's stuck. She doesn't actually have this key. And then Bob just waits and then takes all of Alice's money.

So you know, from the network's perspective, you can broadcast any of these transactions, but you don't want to. Because you don't know the key, it's not your funds.

OK, so yeah.

AUDIENCE: So the wait piece, how does that-- so if in the case where it's cloudy, but Alice signs the rainy key?

TADGE DRYJA: Alice broadcasts the rainy transaction? Yeah.

AUDIENCE: Alice broadcasts [INAUDIBLE] transaction, but doesn't have the proper signature of the oracle.

TADGE DRYJA: Yeah? Do you just wait because you know? Then Bob just waits because Alice's money-- yeah Alice, it's sending to Alice's public key plus the oracle's signature publicly. And Alice can't sign with that. Right, she knows her private key, but she doesn't know the oracle rainy private key. So she can't sign with this to take the money. So the money's just stuck here, but it's got that timeout period, right? It's--

AUDIENCE: OK, so before, there's no timeout?

TADGE DRYJA: Yeah, so the-- wait, did I have it here? Yeah, it's the same script, right. In Lightning, it's PubR or PubT and time, right? There's a revocable one and a timeout one. Yeah, so in Lightning, it's the opposite of Lightning. Right, in Lightning, correct is to use the timeout. Right, so in Lightning-- whoa. What just happened? OK.

In Lightning, I broadcast this. I have to wait, right? It's Alice's key that then takes a day, and then I can sweep it. That's the correct thing that happens. If I broadcast this, Bob can immediately take it because I've already given him the private key.

It's the opposite in Discreet Log Contracts, where if I broadcast the wrong thing, it's just stuck forever. I'm never going to be able to get it, and then my counterparty can get it after a day.

If I broadcast the right thing, my counterparty can still get it after a day. Right? So if Alice broadcasts this, Alice has to immediately spend this because, this Alice 5 Bitcoins is also Bob's after a day. So she needs to say, OK, I'm going to broadcast this, and then immediately sweep the funds to my own regular address.

So what's nice is that's like a better timing model. Anyway, so in cases of fraud, the recoverable key can be used, half the key revealed. So that's in Lightning, right, and Lightning broadcasts the most recent. There is no PubR, right? No one can sign this because you haven't revealed it. And then the timing happens and sends.

So you broadcast your most recent state. You wait a little while, and you don't actually have any rush. You know you've never given PubR away, so you can just leave it, and then time elapses. So maybe the timeout is one day. You don't have to spend it after that one day. You

can leave it for a week or a month, and then later spend it because you're really sure your counterparty is never going to get this private key. So that's the way Lightning works.

In Discreet Log Contracts, however, the timeout one is the incorrect one, when you publish the wrong transaction. Right, PubR is the oracle's signature plus your own private key. And PubT is just the other person gets it after a while. So you should be able-- you know, if you publish the right transaction, you can immediately spend it. If you publish the wrong transaction, you'll never be able to spend it and your counterparty will, tomorrow or next week.

So it's kind of a nicer model, I think, because, in Lightning, you have to be watching your channels. In Discreet Log Contract, nothing bad can happen if you go offline. So that's really nice, right?

In Lightning, you have to watch for fraud. Old states could be broadcast and you've got to grab it. In DLC, sweep it as soon as you want. It's easier. You have the software to do both at the same time. There's no surprises.

So for example, in Lightning, this is the current state, right? Alice has nine coins. Bob can, at any time, broadcast this transaction, and Alice must respond and say, nope, that's wrong, I'm taking all these nine coins. And your software does that automatically, sure, but you have to be online and you have to be watching the blockchain. So surprises can happen.

And you know there's software I'm working on called Watchtowers to anonymously outsource watching this to other people and things like that. But it's a downside, right? Fraud could occur if you're offline.

In Discreet Log Contracts, fraud cannot occur. Eh, I don't want to say that. There's all sorts of things the oracle can do that's bad.

But the idea if I go offline-- right, I'm Alice, I enter into this contract about the weather next week, but then I forget about it and I just don't even sign into my computer for a month, there's no risk that the wrong thing will happen, you know, assuming the oracle is signing correctly.

Bob can't do anything bad, right? If this is now endorsed by the oracle as being the correct transaction, if Bob broadcasts this or this, he can never take those coins. Right? It's just indefinitely stuck here. Alice gets her coins with no restrictions, and Bob's coins are just stuck there forever. Right, he'll never learn this the key to sign with this.

And so Alice can come back in a week, come back in a month, and say, oh, cool. Bob broadcast the wrong thing, and now I can take these coins at my leisure. If Bob broadcasts the right thing and then immediately takes his coins, Alice's coins are still there. Everything's fine.

So this is, I think, a nicer timing model in that you don't have to be online quickly. Also, it's nicer in that in Lightning, if you're being honest and you broadcast the correct transaction, you have to wait. That's annoying, right? The timeout could be a day or a week or something.

And maybe your counterparty-- you know, if your counterparty's online and co-operative, you don't have to use this transaction. You can just say, hey look, I want to close out. I've got nine. You've got one. Let's just build a transaction sending nine to me and one to you, with no weird scripts or anything. And then Bob can say, OK, sure, closing the channel. Here we go. And we just close it.

But if Bob's offline, which happens, Alice says, shoot, I have to close this channel. I want my nine coins. I broadcast this, closing the channel, and then I have to wait a few days. And that's annoying, right? That's not great.

Whereas, in Discreet Log Contracts, if you're doing on it, let's say, I'm Alice. Bob seems to be offline. I can't contact him. I'd say, OK, Bob, you're not there. Fine, well, it was cloudy. I broadcast this. I immediately take my five coins. I don't have to wait at all, and the software does both of those at the same time. So this is a nicer timing model than Lighting. Yeah, no surprises and no waiting.

The only time you have to wait is if the other party-- you know, Alice would have to wait and if Bob broadcasts-- let's say, Bob broadcasts this. Alice gets one coin immediately and nine coins after a few days. So Alice is like, well, I have to wait a little, but, hey, I get all the money, even though I was only supposed to get half. Great.

There are some attacks where, for example, let's say this was like zero or like 0.001, like basically nothing, and the actual thing that happened was it was raining. And Bob's like, shoot, I got nothing. right? Fine, I'm just going to sign this. You know-- sorry-- I'm just going to broadcast this, where I get all the money. I don't actually get all the money, right? Alice is going to take this after a day. But basically, I lost everything anyways, so fine.

Well, so that is an attack where if there's almost no money on one side, Bob can be a jerk and say well I'm going to broadcast the invalid transaction just to prevent my counterparty from

getting her money immediately. She gets it all, but maybe all is pretty close to what she would have gotten anyway, so Bob has no real loss and Alice has no real gain there. And then Alice has to waste some time waiting. So that's an attack.

The easiest way to sort of mitigate that is to have this kind of construction where it's like, well, Bob still got one. Even when Bob loses, he still gets one coin back. Right, there's extra collateral in the contract. And then he doesn't want to broadcast the wrong thing to be a jerk. Yes?

AUDIENCE: So if someone broadcasts a wrong-- the wrong transaction, the correct-- if he broadcasts the correct transaction, it doesn't matter?

TADGE DRYJA: If-- well, from the network's perspective, right, so if I'm a miner looking at the Bitcoin network, I have no idea what these things mean. Right, it just says, you know, key one coin. Other key, nine coins. You know, I just see two different transactions spending the same output.

So if you see Bob broadcast this transaction, you actually do not want to broadcast this, probably, because you'd rather have all the money instead of half, even if it takes you some time. But if you have-- you're saying, OK, I'm Alice. I broadcast this, and then Bob broadcasts this right after, you're like, oh shoot, that's a much better transaction. I would, you know, but you don't know which is going to get in because the miners have no idea what's going on.

AUDIENCE: It means [INAUDIBLE] going to be that you get your money?

TADGE DRYJA: What, sorry?

AUDIENCE: It would be in the other case where you used the attack you just talked about, where people are broadcasting, you get the money from it?

TADGE DRYJA: Oh, yeah. Yeah, so if you see that Bob is being-- the actual outcome was it was rainy. Bob sees this, and has very little money at stake, so Bob says, fine. You know what? Screw it. I'm going to broadcast this just so Alice gets her 0.001 right away, but has to wait a long time for the 9.99.

Then Alice could say, uh uh uh. I'm broadcasting this, and then immediately try to spend it with a high fee. You could do that.

AUDIENCE: If I were--

TADGE DRYJA: Yes?

AUDIENCE: --to be Alice, and it's cloudy, then I would want to just wait, in case Bob made a mistake?

TADGE DRYJA: You could. Yeah, you could hope that you're-- yeah. So there's no rush, right? The oracle signs it was cloudy. You've both got those private keys. You can just wait. You know, there's no need to immediately broadcast it. And once you broadcast this, you do need to immediately spend the part that's going to you because that also has the clause that it can go to the other guy in a week, or a day.

So but, yeah, and you can just leave it here. You could just leave it. Don't broadcast anything. And you're like, well, maybe he'll screw up and broadcast the wrong thing and then I'll get all the money. It's probably not going to happen, but--

AUDIENCE: The oracle never-- so the oracle just goes quiet?

TADGE DRYJA: Ah, yeah. So if the oracle-- there's all sorts of things the oracle can do that's bad. So if the oracle never signs, none of these will ever be valid, right? So you should-- what the contract should have is a timeout where if-- you know, so if we're betting on the price-- so we're betting on the weather tomorrow, we can say, OK, well, by Monday, if the oracle hasn't signed anything, we've got a time-locked transaction that just gives us both back five coins. Or, you know, it's a wash trade.

But, yeah, that can happen.

AUDIENCE: Set it to destroy the oracle.

TADGE DRYJA: Yeah, so hopefully the oracle doesn't do that. OK. But that's like, it's not the end of the world. It's like, OK, the oracle's not there, so we don't know how to execute this trade-- you know, this bet, so we just get our money back. It didn't work, and we wasted time.

Oh, OK, so some scalability fun stuff. You can do Discreet Log Contracts within a channel. So let's say you've got a Lightning channel, and if you want to cooperate-- if both parties cooperate, no transactions get broadcast to the blockchain at all. This is kind of cool.

OK so you've got these nested contracts. So this is a Lightning channel, right? And it's Alice has 10 coins, Bob has 20 coins, and they've been making lots of different transactions. This is, let's say, the 30th, 35th state of the transaction. There's all sorts of transactions before this.

but they've all been revoked by sharing private keys.

And then you say, OK, well Alice has 10 coins, Bob has 20 coins. Let's build a contract. So we both decrement our balances by five, and then we create a third output. So this is a-- it's sort of like having an HTLC where you've got-- you know, as this-- sorry, these arrows don't really line up the right way.

But you've got one transaction with three outputs. Alice gets five coins. Bob gets 15 coins. And a new two of two multisig gets 10 coins. From that, you build two different-- I should make colors or something. Yeah, these are two different transactions. This is all one transaction, sorry.

So from there, you say, OK, we're going to build a Discreet Log Contract that's an output in the channel. And we don't-- you know, if we want to, we can broadcast the whole thing. Any of us at any time can say, OK, I'm broadcasting this current state of the channel, which is Alice five, Bob 15, Alice and Bob 10. And then I have these transactions as well, and I can broadcast either of those.

So if your counterparty becomes unresponsive, you close the channel and close the contract. But if the counterparty is cooperative, then the oracle signs, OK, it was rainy. This now becomes valid. Both parties agree, yeah, Alice won nine coins.

Bob can be a jerk and just be unresponsive, and then Alice is like, all right, fine. Close the channel. Spend-- you know, broadcast this transaction, broadcast this transaction, take the nine coins. Alice has to do three transactions in a row to do that. But if there's nine coins, you know, that's totally doable, right? You know, this to close, this to finalize the contract, and then this the sweep the money to herself before Bob tries to get it.

But if Bob's cooperative and says, OK, yeah, you won this bet and, also, we keep doing these contracts, we keep betting with each other, this is the only valid one. The other one I'm not going to broadcast. I'll never get the money. And then we say, well, Alice got nine. Alice got plus nine. Bob got plus one. We just make a new state of the channel.

Right, so this was the old state, five, 15, 10. We went up a level, and now the new state is Alice gets 14, Bob gets 16, and then we revoke the old state.

So the contract was built. You know, all the different terms of the contract were built. One of them became known to be valid, and then they basically deleted the whole thing and they

never touched the blockchain.

Right, they saw this was all descendent transactions within a Lightning channel. And now, we've got our new balances.

So we can keep doing this. We can have like five of these at the same time and then keep sequentially making all these contracts. What's nice about this is no one ever sees anything. Right, Alice and Bob know that they were betting on the weather. Nobody else can see that they were making a bet at all, right? There's no transaction that goes onto the blockchain.

In the worst-case scenario, a transaction does go onto the blockchain. And if you see this get broadcast, the thing is, it's still not clear that it was a contract.

It could be a channel within a channel, and there's reasons to do that. You could say, OK, this is our cold channel that we keep all the keys offline and this is our hot channel that we keep transacting very quickly. And we can do some things like that, so there's reasons to do this.

But the scripts themselves, if you broadcast the transaction, look exactly the same as Lightning. It's just key A and time or key B. So this is kind of cool, I think.

OK, other-- oh yeah, I have [INAUDIBLE] Other cool things you can do, you can split the R value into an exponent and a mantissa. So it's hard to explain. OK, so let's say I did the example with like sunny and rainy, right, but, in practice, I think people are going to use this for futures contracts, where there's a bunch of different prices.

So you could say, OK, the price we'll put the price on this axis. Price is $1, $2, $3, $4, $5, for example. And then Alice gets money here. Bob gets money here. And it can be like that, or something where at $1 Alice gets all the money; at $2, Alice gets most of it; at $3, Bob gets most of it-- or half and half, things like that.

What might happen is that there's like knock-in and knock-out, where if the price is $6 or $7 or something really high, well, all the money goes to Bob. That's just how it is.

And like, you're not even sure the order of magnitude, so it might actually go $1 and then $10. Like, there might-- how do I do this? There might be like orders of magnitude where, OK, the current price of something we're betting on is $20. Right, so we actually only care about the range from $10 to $30. We only care about-- and then, like $20, it's Alice and Bob get 50-50.

We might not care about what happens if it's $0.01 or what happens if it goes to $5,000. We'll don't care. Or we care, you know, but the thing is if it goes to $30 or $5,000, the same outcome happens. So what we can do is we can say, OK, instead of just using one R value where the oracle signs the price and we have to compute messages for all the different prices and compute s times G for all the different prices, they can say, OK, well, we've got R exponent and R mantissa. And mantissa is a weird word for the thing that's behind the decimal point.

So then I can-- so exponent, you can you say, OK, well, it's either 10 to the one or 10 to the two or 10 to the three or 10 to the four, and I'm going to have an R value for that, and I'm going to produce an s times G for that. And then the mantissa could be like one, two, three, four, da da da to nine, something simple like that.

So in some cases-- so, for example, in the case of 10 to the two, 10 to the three, 10 to the four-- we don't care about the mantissa. Right? If the exponent is 10 to the two, three, or four, then Bob gets all the money, right away. And if the exponent is 10 to the zero, then Alice gets all the money. We don't have to involve the mantissa at all.

But if the exponent is 10 to the one, then we actually care, and we can just add the point. So we say, OK, sG equals R 10 to the one minus the hash times A. And then this is sG exponent. And then, sG mantissa is R3 minus the hash times A. And then we can just add these two points, so we say sG exponent plus sG mantissa equals the contract sG contract, and that's what we put in our contract for the keys.

That way, we need the oracle to sign that the exponent is 10 to the one, and the number, you know, the base is three. OK, it was $30. So that way, it's a little extra data for the oracle to sign, but pretty small, right, an extra 32 bytes. And then we can potentially save on a lot of these extra transactions out in areas that all end up being the same result to us.

Any questions about that? Does that sort of makes sense? It's an optimization. Right? You don't have to do it, but it can lead to a lot less data. Yeah?

AUDIENCE: So for the mantissa, the 1, 2, 3, 4?

TADGE DRYJA: Yeah?

AUDIENCE: Does that come from a specific decimal place?

TADGE DRYJA: Yeah. Well, it's-- the idea is the actual number is going to be R mantissa times Rx, you know, or 10 to the Rx, right? So the actual-- so if the number was 20, you say, OK, 10 to the one. R exponent's 10 to the one. R mantissa is two. If the number was 5,000, you say, OK, 10 to the three, five. And then you just release both of these.

And they're independent, so that the two users can use them independently. And the most likely case is we don't care about the mantissa, we only care about the exponent. So we're only-- some of our bets only deal with the magnitude of the price.

You could also do bets where you're only dealing with the mantissa, but that seems kind of silly, where it's like, I don't care if it's $2 or $20 or $200. But anyway, if the first digit of the price starts with two, you know, is two, I win. You could do that, but probably not as useful.

But yeah, and then in the extreme case, the extreme version of this would be to have bind-- you know, decompose the number being signed into binary, and then find every bit. And then there's only two signatures possible for every bit, and then the users can sort of combine these things in any way they want. More signature data, but it gives the users a lot of flexibility.

OK. multi-oracle, another thing you can do. So the problem is, how trustworthy is this are these oracles? The oracles can cheat, right. They can just easily sign that it's rainy when it's sunny. That's publicly knowable, and if they have a reputation, that hopefully hurts them. But you are trusting the oracle to sign the right thing.

So maybe they want to use two oracles. So that's totally doable. Right, you have the oracle, the a oracle and the b oracle, and you just add the two points that you come up with. And then when they both release their s values, you add those two s points, and you'll get the private key, totally easy.

For n of m, there's no size increase. Right? You just-- you can pick 10 oracles, add up all those things. No one can tell. And your transactions are still really small.

For n of m, it-- or usually, it's called m of n, huh? Well, anyway, there's a size blow up because you're going to say, OK, well, I want two of three, and I have to make new transactions for all these different combinations. And so you end up having a lot of transactions. For small things, like sunny or rainy, it's totally doable. For things like where you're already hitting up against limits where, oh, we have to sign thousands of transactions for all the possible different prices, then this can get very costly.

Another downside to this is if they signed different things, you get stuck. Right? So if we say, OK, we're going to use Thomson Reuters oracle and Bloomberg oracle and we're going to use their price of oil, and then one of them signs, oh, it's $50.03 and the other signs that it's $50. 05, shoot. We can't-- you know, we just used the same value for all these sG points to build all the combined points. Now, we don't have a match, so we cannot build the point-- the private key that we're looking for. So this contract either reverts.

So that's another reason why you might want to decompose it this way. You could have a construction where you say, OK, we're using Bloomberg and Thomson Reuters, but for Thomson Reuters we're only using the exponent and for Bloomberg we're using both.

So the idea is if there's a case where the exponent is different between Bloomberg and Thomson Reuters, our contract fails. But if the case is their exponent is the same, then we just go with the actual precise price from Bloomberg. So in case Thomson Reuters gets hacked and they say, oh, the price of oil is $50 million a barrel and that could cause contracts to go the wrong way, then that's sort of a safety catch on there.

So there's all sorts of things you can do like that. I don't know how this will actually work out. Will there be lots of different oracles? My guess is no. You kind of want them to be reputable. And you can't really make money doing it. And it's really scalable and cheap to be an oracle because, like, all you're doing is signing things. So it feels like the kind of thing that once you have a-- you know, once you get some momentum, you're just going to be this monopoly and everyone's going to use the same oracle. But who knows?

Yeah, oracle problems, so yeah, oracles can lie. They can also-- they can they can equivocate. Right, they can sign two different things, but then they reveal their private key. So you might want them to post coins on the blockchain with their A, you know, that's got their regular public key A, so that if they ever do sign two of the same message, then they lose their money and everyone can try to grab it.

Yeah, what are the other things? Oh, I was going to talk about novation. We have time. Yeah, so another thing that people like to do with these types of contracts is enter into contracts and then sort of sell their position in the contract.

So for example, if we were in a contract about the weather, and I look outside, I'm like, oh, it's definitely going to be rainy, so the oracle is going to sign about the weather as of noon. I'm

like, it's 11 o'clock. It's going to be rainy. I want to sort of lock in my profit. And it's not exactly sure yet, so there may still be some value to the sunny position, but it's looking pretty grim for the sunny position.

I might just contact my counterparty and say, hey, let's close early. Right? I don't need the entire profit, but give me 90% of it because it's, you know, pretty rainy. And they might agree to do that, but a better way to do it would be to say, I'm going to let someone else take my position.

So for example, if you have-- so if you've got like Alice and Bob are in a contract. And then there's like Alice gets nine, Bob gets one for sun. And then Alice gets one, Bob gets nine for rain. And then they say, OK, let's just do it early. Bob might not be cooperative. Alice wants to say, let's just get out of it right now, at some, at one of-- you know, a little bit less than this. Bob says, no, I want to keep holding it through.

What Alice wants to do is say, look, I'm going to replace myself with Carol. Alice finds some other person, Carol, to now have a Carol-Bob transaction where it's the same exact contract. So it'd be Carol nine, Bob one and Carol one, Bob nine for sun and rain.

So what's nice about this is, from Bob's perspective, nothing changes except the counterparty's public key, which Bob doesn't care about. Right? So Alice can say, hey, Bob let's switch this transaction. You have the same exact payout based on the same exact signatures from the oracle, so it's no change to you. It's just that my public key is going to change because it's actually going to be someone else.

And then Bob-- that way, Bob's software can automatically do it. This is still interactive in that Bob's keys have to sign. Right, Bob needs to make a new signature spending from here to go into this new two of two output, so that's new. So their computer has to be on and they have to sign, but you can make the computer software do it automatically because there's no user interaction needed in that it's like, hey, your position didn't change.

And then probably put in something where Alice pays Bob a little extra, right. Alice gives Bob a dollar or two to incentivize him to leave his computer on to collect fee. You know, essentially, it's like, oh, if I leave my computer on and my counterparty wants to novate and switch with someone else, I might make a buck. And my position doesn't change, so cool. So that is doable.

I do not look forward to programming that because that now has three different computers talking to each other and they all have to do messages in the right sequence and stuff, and that's not the most fun part of programming. But yeah, so anyway, so that's the idea of this.

What are some use cases? I can think of some, but there's all sorts of other ideas. I think it can be pretty useful. You could do currency futures. So I think one of the biggest would be dollar futures. So right now, in CME, I think, or CBOT or one of those Chicago kind of places, or both of them, you can do Bitcoin futures which are dollar-settled Bitcoin futures. Right, so you get a bunch of dollars based on the price of Bitcoin.

Yeah, so you can say, I want 10 Bitcoins for delivery on next week. And then you don't get the 10 Bitcoins, but you get however many dollars those 10 Bitcoins are worth based on some exchange price that they agree on. You know, we're going to use the average of these different exchanges and stuff.

With DLC, you could do sort of the inverse of that, where I have Bitcoins I entered into a contract where I have dollars being delivered-- or, sorry, I have Bitcoins, you know, the dollar value being delivered. So the idea is I have $10,000 worth of Bitcoin being delivered to me on Friday, so if the price of Bitcoin goes up, I get fewer of them. If the price of Bitcoins goes down I get more of them such that the amount of Bitcoin I receive on Friday has a value of $10,000.

And that's a pretty straightforward Discreet Log Contract ability, right? You just need to know the price of a-- it actually ends up being a lot easier to think of Bitcoin as the main currency and then dollars as the asset being traded. So you sort of want to know the price of a dollar in Bitcoins, or how many Satoshi is a dollar's worth, and then you build your contract from there. And so if the price of a dollar goes up, you get more of these things.

So one side will say, OK, I've got $10,000 worth of Bitcoin coming on Friday. The other side basically says, I get whatever's left in this contract. And that other side is essentially like a more volatile Bitcoin. So if the price of bitcoin goes up, great. Also, you get more of them. And if the price of Bitcoin goes down, you're really screwed because your Bitcoin's worth less and you lose most of them.

But I feel like people will want to do that. Like there will be people who are like, cool, double-volatile Bitcoin. You know, where do I sign up? And there's also going to be people who are like, I think Bitcoin is cool, but I want to anonymously and trustless-- trustless, except for the oracle-- you know, not so trustfully short it. Right, I want to say, OK, I'm not-- I sort of don't

have a Bitcoin position anymore.

However, it's not the same in that you don't know who your counterparty is. It's fully collateralized at the outset, so if the price of-- you know, if both parties put in 10-- you know, both parties put in 10 coins, 20 Bitcoins is all that's ever coming out of this contract. Right? You can't say, hey, I want to do like a margin call. You need to put like 40 more coins in this contract because the price went down. Your counterparty just will say, no, I'm not. So there's sort of bounded loss and gain for both parties at the outset.

Also, because you have no idea who your counterparty is, right, you can do this totally anonymously. It's-- I don't want to say it's trustless, because you are trusting the oracle to report the correct price, but your counterparty, you don't trust at all. Right? You have no idea who they are. You assume everything they're telling you is a lie, you know, like they're trying to hack you, stuff like that. So I think currency futures would be a big one, where people can then buy and sell.

Stocks might be a big one. There's a lot of issues there with insider trading because if you can, now, anonymously trade stock futures, I don't know, that's probably something people might want to do who shouldn't be doing it. But it's hard to enforce because these oracles have no idea that anyone is actually making contracts.

And if you look on the blockchain, you never see the contracts. Or you might sort of be like, well, is this a Lightning Network channel that closed or is this a Discreet Log Contract that just executed? It just looks like keys.

You can't-- oh, that's an important part. Given, even if you're the oracle and you know what you signed, the fact that you're adding these two points means that unless you knew the original Alice key, you wouldn't be able to tell, oh, that's Alice's key plus my key. Right, because you don't know what Alice's key was. It never existed independently. Or, oh, Alice and Bob knew it, but you didn't as the oracle. And so no one will-- the oracles won't know what contracts are being executed, even after the fact.

You can, however, prove that you used-- you know, if you want to. If you're Alice, you can say, hey, this was my contract. I can show you all the terms. And you can prove it by signing a message with the Alice private key and then also showing that it's, you know, this is my public key and I added it to the oracle's thing.

Yeah commodities, sports, there's all sorts of fun things you can do with it. It's pretty general, conditional payments based on any number or element from a predetermined set. Cool, so any questions about this? Yes?

AUDIENCE: So, if the public policymakers wanted to see this, they absolutely couldn't? It is [INAUDIBLE] encrypted?

TADGE DRYJA: You could talk to the oracles because they'll probably be publicly-known entities. And so you could say, hey, oracles, don't sign-- don't sign messages. I mean, yeah, but the actual counterparties A and B, if it's like OTC, if it's just Alice and Bob sort of talking to each other, hey, let's build this contract. We both trust Bloomberg. Their prices are correct. It's-- you can't see that this happened at all.

AUDIENCE: Does Bloomberg have to be complicit [INAUDIBLE]?

TADGE DRYJA: Bloomberg has to-- so some oracle, someone, somewhere, has to be saying, look, I'm going to sign the price of oil. That's it.

AUDIENCE: Even if they disagree to, right?

TADGE DRYJA: Oh, yeah. Yeah. So well, so yeah, most of these transactions never happen. If Alice and Bob see that the oracle signed, they don't have to actually broadcast it. But they do need the oracle's public key, first, to build the contract.

So as you could try to regulate public oracles, but that seems like--

AUDIENCE: Assume Bloomberg's bringing in, assume that--

TADGE DRYJA: Yeah, yeah. But, so the question is--

AUDIENCE: --they have somebody sitting in the Cayman Islands who's got a software program that grabs the Bloomberg fee?

TADGE DRYJA: Yeah. The issue there is the signatures are sort of usable by anyone. You could say, OK, well, Bloomberg charges money to provide these signatures and these public keys. But if those get out, everyone knows, oh, these are Bloomberg signatures. You know, these are-- this is the Bloomberg data. We can now execute trades using this data.

And we do have-- so this eliminates the trust between counterparties, but doesn't eliminate the

oracle trust. But the thing is, the oracle trust is sort of the easy problem. Right? There's not too many cases of-- you know, if you have some OTC contracts and you say, oh, we're using the price according to Bloomberg or the price according to the *Wall Street Journal* or whatever, generally that works.

So yeah, so what you can do is, I think the place that people will try to regulate is the matchmaking area where, if it's OTC, and Alice just calls up Bob and Alice is in the Cayman Islands and Bob is in British Virgin Islands, or whatever-- are those the same thing? I don't know. They say, hey, I want to short oil. Oh, I want to go long oil.

OK, and we just build our contract between our two computers, and Bloomberg has no idea we're doing this. No one else in the world has any idea we're doing this, and we have this smart contract. But in most cases, people will not know who they're wanting to trade with, right? They want some kind of exchange where we can meet and say, hey, I want to short Bitcoin. You know, I want dollars. And someone says, oh, cool I want double-volatile Bitcoin. Let's meet, and let's make this a contract. Those exchanges will probably be more regulated, I guess. You could sort of say, hey, we need you to KYC all your exchange participants.

But even then, like if you wanted, so similar to Altcoin's stuff now, you could say, I'm operating an exchange out of wherever. And you can't operate an oracle out of wherever, really, because people won't trust it. But you could say, I'm operating an exchange out of Hong Kong. We don't KYC anyone. Maybe we'll get shut down in a year. But anyway, you can use the Bloomberg data feed and enter into these contracts.

So yeah, it makes enforcement hard. Yes?

AUDIENCE: It strikes me that the hole here, from the public policy side, is that even oracles are semi-regulated, and there's been tons of fraud in oracles for hundreds of years.

TADGE DRYJA: Really?

AUDIENCE: Oh, and massive.

TADGE DRYJA: But they seem so useful.

AUDIENCE: You trust too much.

TADGE DRYJA: So people--

AUDIENCE: And you're a Bitcoin Core developer.

TADGE DRYJA: Yeah, I just figured that the financial system figured that out, so it's whatever. No.

AUDIENCE: Yeah, sure. Wicked loads of fraud lots of years. I used to run the Commodity Futures Trading Commission, so we did gold, currency, interest rates.

TADGE DRYJA: So there, was it that the spot market itself was being manipulated? Or the spot market says the price price is $50, the oracle says it's $80?

AUDIENCE: we would take what the oracle was publishing only because you're long or short a position.

TADGE DRYJA: But then, wouldn't you just sort of sue people and say, hey, look this oracle was wrong, like obviously the spot price was this? It's not as--

AUDIENCE: We could [INAUDIBLE]. It's often hard to say what is this spot price.

TADGE DRYJA: Right.

AUDIENCE: And sometimes, you come in at different places. Besides, it seems like the holes, the holes here, is not so much around the oracles, which there could be big holes like, somebody could just become an aggregator of the oracles because the CMA takes four prices, but somebody sitting in the Cayman Islands could say we're going to just take the same four prices right [INAUDIBLE]. And being crypto, settle. If it's fiat, settle. Just travel more ways to get into the on-ramps and off-ramps to the banking system.

But and that's why you're seeing a lot of exchanges having a hard time getting, basically, deposit accounts in banks because the banking regulators around the globe are trying to regulate on-ramps and off-ramps, but it's harder for crypto and crypto exchange.

TADGE DRYJA: Yeah, crypto crypto, start a computer. Yeah, you've got it. You don't have to ask. Yeah.

AUDIENCE: And you could have it, and you know, so over-the-counter Cayman Islands aggregator oracles, and somebody could do single-stock futures that are banned in this country, so--

TADGE DRYJA: Really?

AUDIENCE: Single-stock futures.

TADGE DRYJA: Well, options aren't, right?

AUDIENCE: Options aren't.

TADGE DRYJA: So, oh, I didn't say, you want to make options contracts with this, you just don't give the data to one side. It's basically the same construction, but like, now, Alice has the option to broadcast the transaction and Bob doesn't. But so you can make options with this-- or futures, I mean.

AUDIENCE: But you're not trying to subvert public policy norms. This could be a gray technology, or do you think that is useful for subverting public policy norms?

[LAUGHTER]

TADGE DRYJA: I think it has both use cases. One of the big ones will be sort of similar to Bitcoin, where it's like, hey, I can do these things that maybe I'm not-- you know, people don't want me to do.

And in some cases, the don't want me to do is just like scale in terms of like, OK, I'm some guy in Thailand, and I want to go long Apple. And probably, you can legally do that. But if you've got like a couple hundred bucks and you're in some random country and you want to go long, some US equity, that's not really feasible. Right? Maybe you can open accounts. Maybe you can try to open a foreign account. Like it's-- there's a lot of friction.

And so with this kind of thing, it's like, hey, we've got a Bitcoin. I trust this price oracle. OK, I'll do it with like a couple hundred bucks worth of bitcoin.

That said, there's also cases where it's like, OK, I work at Foxconn in Shenzhen and I think the Apple iPhone 11 is not going be any good, so I'm going to short Apple using this kind of technology. So there will be good and bad things.

You could also-- it could also be useful in cases between actual regular legal Wall-Streety kind of people, maybe not-- where-- then again, it's good where you've got like not a lot of trust between your counterparties, but you do trust some kind of oracle feed. And the other thing is-- like, I have talked to this sort of Wall-Streety people, and they don't really like it because they're like, wait, there's no leverage. Right? You have to put all the money you're going to get in at the beginning. And they always want to have it like super-leveraged.

You can crank up the slope. So you can say, OK, we both put 10 Bitcoins in, but a small difference in price, right, so like a 5% movement in a price will like make all the money go to one side or the other, so we hit the knock-in and knock-out really soon. So you can do that, which is kind of like leverage. But fundamentally, it's only the Bitcoins that go into the contract

come out.

So yeah, I don't know how this will be regulated. I'm working-- there's a company in Japan who's interested in building this technology. There's like-- it's a brokerage that's making this stuff. I'm working on it with some other people. We'll see. I don't know how it will be. Maybe people won't like it. It's-- yeah.

AUDIENCE: I mean, it seems to me like that's one of the big things that Bitcoin's missing. So if the financial system is for distributing risk and [INAUDIBLE] the risk, right now, I don't really do that with Bitcoin. It's kind of just a transactions channel.

TADGE DRYJA: Yeah.

AUDIENCE: So this--

TADGE DRYJA: I think it'll be cool.

AUDIENCE: --be logically, kind of the next step.

TADGE DRYJA: Yeah, oh I think this'll be cool. I think people would use it. I just, you know, it's a lot of work to program and I don't, yeah, have enough time. But there's other people starting to get-- I don't know, it's weird. It's sort of like Lightning was like three years ago, and some people were like, oh, cool. And then eventually, people were like, oh, this is a really big deal, and let's all work on it.

And it might be the same with this, where like no one really cares and then, eventually, people will be like, oh, hey we should build this and use it and start companies and things like that. So hopefully, people will get into it.

I think it's a pretty nice construction for these kinds of things. It competes-- there's things like Augur, which is-- I don't think is the best idea. It, like the Augur, the idea is the oracle is sort of everyone, and like everyone votes on what these outcomes for different things were. But yeah, I don't know. I don't know how far along they are in terms of their software. But that was also like two or three years ago.

And this could be applied more broadly, like where anytime you have Ethereum-type smart contracts or something where you want some external oracle who then can sign things. And then you can use it in a way where the oracle's signature doesn't show up after the fact. So

anyway, any questions? Sounds good?