**TADGE DRYJA:** OK. So today, I'll talk about payment channels and the lightning network. And I am sorry if I gloss over things. I've explained this, like, a lot in the last two years because that's, sort of, what I work on. So I might skip over something, and you're like, wait, how did you get there? So let me know.

So I'll have uni-directional payment channels, and then lightning channels. And then, multi-hop. OK, so the idea of payment channels. Why do we need to do all this complicated, weird stuff with payment channels? Well, the problem is these networks are really poor in scalability. Right?

Every transaction on the blockchain, you know, that whole idea doesn't scale. People say it's O of n squared. Depends on what you define as n and then things like that. So if you double the number of nodes, it actually doesn't. You can say, well, if you double the number of nodes, each node is going to be making some transactions. Right?

And then, you're like, OK, yeah, it's n squared if you count the total number of computations throughout the entire system. But at the same time, it's like, well, you've doubled the number of nodes. So really, it's O of n, right? For any one computer, your amount of computation you have to do scales with how many other computers there are. So it's a little questionable.

But basically, it doesn't scale well. Right? It doesn't scale to what you want. And so I put a little history at the first response of anyone ever referring to Bitcoin. So there was a 2008-- like October, Halloween 2008, or maybe after midnight, so November 1st-- and Satoshi wrote, I've been working on a new electronic cash system that's fully peer-to-peer, with no trusted third party.

Then, he has a little blurb. The first reply on this mailing list was some guy-- James A. Donald-- saying, we very, very much need such a system, but the way I understand your proposal, it does not seem to scale to the required size. So that was the first thing anyone's ever said about Bitcoin. And we're still talking about it almost 10 years later.

I wonder who James Donald is. Maybe he's got a ton of Bitcoin now and he mined it. So it was written about in late 2008. The actual software was released in 2009. There was a couple emails. No one really looked into it much, at the time.

OK, so you can start with a one way payment channel. I think Satoshi, sort of, mentioned these ideas. Never really implemented anything. And the way he talked about it didn't actually work. Some people in 2012 I remember looking at this. I think [INAUDIBLE] and Mike Hearn gave a talk about how you could make payment channels. That didn't quite work either, but the idea made sense.

So the idea is you have some kind of shared balance between two parties. And you can move coins in one direction without making new transactions on the blockchain. So if you look at a transaction-- you probably have seen this kind of thing through homework-- it's got a lock time. And in this case, the lock time is 0. That means this transaction is valid at any point.

Above height 0, it's valid. And every block is above height 0. You can also find transactions, let's see, which has a block time of this. So this looks like a Unix time where you set a time. This is like a Unix time. So Unix time, if you don't know, it's seconds since January 1, 1970. That's, sort of, like the first second.

And I don't know. That's probably a week ago or something. So you can have that field in a transaction. The idea is the transaction is only valid after that time or height. It's a little weird. Below-- what is it-- 500 million, it counts as a height. Above 500 million, it counts as a Unix time, which ends up working because the Unix times below 500 million are in the 80s. So you know, Bitcoin didn't exist then. There's no real need to have time locks them. Sorry?

**AUDIENCE:** What about when you get that far in block height?

**TADGE DRYJA:** About what? Sorry.

**AUDIENCE:** Don't you eventually get that far in block height, though?

**TADGE DRYJA:** 500 million? Yeah, couple thousand years. So yeah. No, there's way more problems with Bitcoin before that. The time field in the header is four bytes. So if it was signed, it'd be 2038 that it flips over and goes negative. But it's unsigned, so you got until 2106. There's actually a hard fork to fix that.

So you know, there's all sorts of weird, like, wait, what do we do in the hundreds of thousands of years? We have to make these little tweaks, which, hopefully, wouldn't be super controversial because it's like, well, it'll stop working unless we change this little thing. OK. So yeah, you can have these lock times. And you can use those. So what you can do is make a channel.

And a channel is really just a multisig output. Multisig, a lot of times, is used for-- I don't want to say two factor authentication-- but you could have two of two multisig where your phone needs to sign, and your desktop needs to sign. And then, if someone steals your phone or you know hacks your phone and gets the key out, well, they have to also get your computer.

So this is useful. But in these cases, it's, sort of, adversarial multisig where it's not just you and you. It's not like your phone and your computer. OK, it's Alice and Bob. They're completely separate people. They're not necessarily friends. You know, it could be a customer merchant kind of thing. And they create a transaction together. So they fund it.

And Alice is funding this channel. And so she takes her own coins, her own txid:index, her own output, signs it, and sends to an Alice and Bob multisig and sends 10 coins. So in order to spend this, they both have to cooperate. But before doing this, Bob gives her a refund transaction.

So the idea is, if Alice just broadcasts this and says, OK, I'm taking my coins and I'm sending it to Alice and Bob multisig, Bob can just disappear. And then, Alice is stuck. And the money's stuck there forever because Bob can never sign. Or Bob can say, oh, Alice, looks like you sent 10 coins. How about this. I'll give you nine of them. You just give me one of them. You're not buying anything. This is just I'm holding your coins hostage.

Bob could even make a even better threat and say, I will sign a bunch of time lock transactions. If you want your money today, you only get one coin. If you want your money in a week, you'll get two coins. And you know, sign these things with lock times. Hand all the transactions over to Alice and say, well, you can pick. How long do you want to wait to get your money back?

And then, Alice's only response is, oh shoot, well, if I wait years, maybe I'll get most of my coins back. But if I want my money now-- you know, there's all sorts of horrible hostage situations, even if they both have money at stake. One person could have less time value of money. So they're like, look, I don't need this anytime soon. So I'm willing to wait.

So this is dangerous. So what they do beforehand is Alice says, hey Bob, give me a refund transaction. So Bob signs a refund transaction with a lock time of one week from now, March 28. And the input is the fund transaction ID. Even though the fund transaction comes later, thanks to segwit, you can say I know what the Tx ID's gonna be. I'm gonna reference a spin for that.

And Bob's signature is going to be on there. Bob signs it. Hands it over to Alice. Alice's signature is not there yet. So Alice can sign it and store it on her drive, or she can sign later. But until March 28th, this transaction is not happening. If you try to broadcast it, everyone will reject it. In the output from this refund transaction, it just sends to Alice's address, all 10 coins. Yes?

**AUDIENCE:** How do they know the transaction in the future, the transaction ID in the future?

**TADGE DRYJA:** Oh, how do they know the funds Tx ID? Alice can calculate it and tell Bob. Alice can say I'm building a fund transaction. Alice knows her inputs. She doesn't actually have to sign it all because the Tx ID won't contain the signature and what their key is. So Alice can compute the Tx ID and tell Bob, hey, sign this. You know, this is what the transaction is going to look like. Sign it.

And Bob, at this point, has no risk. Bob, I guess, needs to make sure that he's not inadvertently signing his own money. So if Alice says, hey, here's the Tx ID, Bob should make sure, OK, this is not my Tx ID. Right? But it actually doesn't matter since he, sort of, commits to Alice's pub key here. So even if you've swapped the Tx ID, when you sign, you say this is two of two multisig.

So if this was Bob's single key, it wouldn't it wouldn't even work. Yeah, so Alice has to give some information to Bob. Bob signs, hands the refund over to Alice, then Alice is able to create this fund transaction. And now Alice's risk is minimized because she says, OK, I'll send this to this channel and, worst case, I have to wait a week. Right?

If Bob disappears as soon as I broadcast this, well, then Bob's gone. I can't do anything. We can't sign. But then, next week, I'll be able to use this transaction and get all my money back. Yes?

**AUDIENCE:** You said that if this refund transaction is broadcasted, all the notes will be discarded.

**TADGE DRYJA:** Yes.

**AUDIENCE:** But then--

**TADGE DRYJA:** This week.

**AUDIENCE:** --it has to be sent before Alice sends out the other transaction, right? Because what makes sure that Alice will have this transaction up there in the systems before the other transaction, before they're bought?

**TADGE DRYJA:** Which transaction? So there's the fund and the refund. Right?

**AUDIENCE:** Right. So the refund must be sent before the fund.

**TADGE DRYJA:** No, it needs to be created before the fund. Right? So it's not going to be on the network. But Bob creates this first, signs it, and gives it to Alice. That way Alice knows, well, I have this that will be valid in a week. And so that means it's safe to sign and broadcast this. Yeah.

**AUDIENCE:** And the refund just requires Alice's initial signature.

**TADGE DRYJA:** Yeah.

**AUDIENCE:** They're [INAUDIBLE]. OK.

**TADGE DRYJA:** Yeah. So the refund transaction, it's got Bob's signature. And that's what hand you hand over. And then, Alice can sign later. She can sign it immediately on reception and then store it. Or she can sign it on March 28th. But the idea is she's got Bob's signature endorsing this. And it's not valid yet, but she knows minimize risk. Right? Worst case, wait a week.

OK, so then what can you do with this? You can make, sort of, similar to refund transactions, but they give Bob more money. So you make two outputs. You say, OK, I'm spending the fund transaction ID. I put Alice's signature. And I say Alice gets nine coins. Bob gets one coin. And Alice hands that over to Bob.

So you make a transaction like that, right. It spends the fund transaction output. Alice and Bob's key are both needed. And Alice signs this and says, OK, I get nine. You get one. Hands it over to Bob. And Bob says, basically, I just got the money. I can broadcast this if I want. Right?

Bob's got Alice's signature. Bob can add his signature. Broadcast this transaction, and receive one Bitcoin. OK, but see, Bob doesn't sign his side in broadcast. Bob just waits. Bob's like, cool, I've got this coin. I'm not even going to sign and broadcast this. I know it's as good as having received the actual coin because there's nothing else Alice can do on her own. She can't take these 10 coins.

She can, eventually, get the refund in a week. But for now, Bob's safe. And then, maybe a day later Alice says, OK, I'm giving you another coin. What I'm doing is I'm signing a new transaction. It spends the same output. This is still there. These have not been broadcast.

But in this one, Alice gets eight coins. Bob gets two coins. Alice signs it. Hands that over to Bob. Bob's like, great, I got another coin. I will also just wait. Bob can actually delete this because, well, this is better. Right? From Bob's perspective, I'd rather have two than one. And any of them are valid.

According to the network, the network itself, no one's seen these things. They're not broadcast. Only this exists as an output. And you can keep doing this. You can keep doing this as many times as you want. Well, within limits, right? What's the obvious limit here?

**AUDIENCE:** 10 BTC when Alice bought it.

**TADGE DRYJA:** Yeah. Sure. Once Bob gets all 10, there's no reason to keep it open anymore. Also, you can't reverse. Right? So Alice is always sending money. If Bob says, hey, I'm going to give you some money back, and Bob signs something and say, hey, now I have two coins and you have eight coins.

Alice is like, yeah, but you have this. Right? You have this Alice 7 Bob 3 transaction. I know you can broadcast that whenever you want. So there's no credible way for Bob to say like, oh, I'm giving you the money back in this channel. Right? Yes?

**AUDIENCE:** But what if the next transaction Alice signs-- so she gave 1, 2, and 3. And the next one, she gives Bob 2.5 and keeps 7.5. Is that right?

**TADGE DRYJA:** Yeah, Bob would just ignore it. Bob would be like, yeah, but I already have three. Why would I want 2 1/2? And it's more like, in that case, if Alice says, oh, I have two, Alice is trying to take some money back. Essentially, Bob is paying Alice, in that case. But it's not credible.

Alice is like, yeah, I can sign this again. Or I have all 10 again. But you know, Bob's idea is like, I'm just going to sign the one where I get the most money. So you know, that's a limitation. It's one direction. Alice can keep paying Bob. Bob can't pay back. But that's still pretty useful, in some cases. If you have some kind of recurrent payment, you're streaming videos or something. You open a channel, you can keep paying.

One thing that's nice is these can be really fast. So this, you have to actually wait till it's in the blockchain. But these transactions, there's no blockchain involved. You just sign it, and hand it over to the other party. And as soon as they've received it, they're like, yep, I got the money. So this can be very low latency. So this is pretty cool. There are limitations.

So, yeah. Bob keeps getting these half signed transactions with more and more money going to him. The old ones are useless. Why would I keep this old transaction where I got one coin? Just delete it. So Bob only actually has to store one thing. Bob also has to be very careful, though. He's got to sign and broadcast one of these before next week.

If he doesn't, Alice could just broadcast the refund transaction. And you know, he thinks he's getting three coins, but Alice is like, no, I'm just taking all 10 back. And Bob was like, shoot, I just got ripped off. So from the Bitcoin network's point of view, no one's seen any of this. All they see is the fund transaction. Nothing happens for days.

And then, they see one of these payment channel closing transactions. And they might see multiple. Right? Bob could broadcast all three. It would be dumb. Bob only wants this one. But you're, basically, using double spends to create these channels. yes?

**AUDIENCE:** So the refund transaction is pretty much insurance, right, for Alice?

**TADGE DRYJA:** Yeah. If it happens, that means either Bob went down or Bob never-- you should never use it in real life. It's insurance so that you're not worried that Bob's just going to run off and get your funds stuck.

**AUDIENCE:** So in the case that, both, Allison and Bob are honest actors, I guess, in this situation, if a week does go by, there's no reason for Alice to actually use that refund transaction, right? There's no guarantee that she's going to use it in a week.

**TADGE DRYJA:** Right. But you can't keep using the cha-- like, at that point, there's so much trust involved. If you keep trying to use the channel after the weeks gone by, Bob now-- like, it's the same reason you can't make it bidirectional. Right? If Bob says, hey, here's a transaction where you get more of the money, Alice is like, yeah, but that's not credible because I know you can broadcast and take more.

And similarly, if Alice says, let's just keep the channel open, Bob's like, no, that's not credible. I know you can just take all the money at any time. So you handing me these new signatures, like, it's, sort of, meaningless. There's so much trust involved that like, why bother with this whole payment channel thing, right? So this is nice in that you don't have to trust the counter parties. The worst they can do is disappear.

I mean, yeah, they can hurt themselves. Right? Bob can broadcast this after this exists. And Alice is like, OK, that was dumb. Thanks for the two coins back. Or Bob can just disappear. And Alice is like, hey, I'm going to do the refund. You're supposed to get these three coins, but you're gone. And the refund is my only option, so I'm broadcasting it.

So you don't have to worry about who is your counterparty. And you don't have any debt. There's no custody or anything like that. So that's really nice. You can add trust, but we already have that with Coinbase and Gemini-- you know, the exchanges.

OK, so this is pretty useful. Time limits. Limits on back and forth. Refund transactions have to be built before the fund transaction, and so malleability hurts that. Before segwit, this was very risky, slash you couldn't really do it. The refund transaction has to spend the fund transaction. And without segwit, you don't know what that transaction ID is going to be before it gets confirmed.

And so you can try to anticipate, oh, well it might get malleated to these five different things. Let's do it. But it's risky. So with segit, it's nice. You don't have to worry about it. OK, so any questions about the simple, one direction payments? Makes sense?

OK. So how can you make it even better? What if you want to make it bidirectional and last forever? That would be much more useful. Right? But it's a tricky problem. The refund transaction is there, and the clock's ticking.

Also, how do you delete or revoke these old transaction? if you've signed over, OK, you've got eight coins. I've got two coins. That's there forever. And the blockchain doesn't know that it's not valid anymore. Right? The whole point of the blockchain is to say, OK, this is gone.

Actually, between the unidirectional and lightening, I think-- how did the order go? Like, Christian Decker wrote a thing about decrementing time locks. And I don't think he published. I think he published after we wrote lightning. But the idea was these would have lock times. It's like, you could broadcast this on the 27th. All three of these could be broadcast on the 27th.

And then, the idea is Bob wants to send money back. And you put a lock time on the 26th. And the idea is like, oh, now these all are 27th. But now there's a Bob 2 Alice 8 over here that's the broadcastable on the 26th. And so the idea is, well, Alice can get the money back. Bob can go back to paying Alice because it's sooner, and Alice can raise.

So that helps a little in that you can switch back and forth a few times. But each time you do it, the time window gets sooner. So that's, kind of, a cool idea. But I think the lightening one is nicer because it makes it-- OK, the goal is back and forth between these two parties. And it lasts forever. And you can do as many transactions as you want.

OK. So how do you do that? So there's two timing up quotes that have been added to Bitcoin. And CHECKSEQUENCEVERIFY is pretty much for lightning, in my opinion. That's the real use case. There's also CHECKLOCKTIMEVERIFY. So the idea of SEQUENCEVERIFY is it's a relative lock time. So it doesn't specify, OK, this transaction is valid on March 28th. It says, this transaction is valid when the input it's spending is a week old.

And by a week old, it's in a week's worth of blocks. In the actual software, we don't usually use time. We always use block height. It's simpler to think about, in a lot of ways. But you can say, OK, well I send to an output script. And the output script specifies that you can only spend these coins after the transaction creating them has been 100 blocks deep.

So that's, kind of, an interesting, useful thing. Yeah, and confirmations. If not, you can't spend it. OP_CHECKLOCKTIMEVERIFY. It's a lot absolute lock time opcode. So you require that the transaction be confirmed at high end or above. So what we said in the refund transaction, the refund transaction, the transaction field has a lock time.

Where is it? Oh, well, in this case, yeah. But the transaction itself has a locktime. But that's difference from the opcode. What the opcode does, it says, OK, this output has a locktime because the output script ensures that the transaction itself has a locktime.

So basically, you say OP_CHECKLOCKTIMEVERIFY 500,000. And then, it checks that the transactions locktime field is exactly equal to 500,000. So it's a way to enforce a transaction wide field inside of an output. It's a little weird. But both of these become opcodes that you can use in scripts.

And now, you can specify, like, make an address that has these weird timing properties. OK. Any questions about these two?

**AUDIENCE:** Can you use both of them?

**TADGE DRYJA:** Yep. You can mix them and match them. CHECKSEQUENCEVERIFY is specific to an input. So there's a sequence field in each input that was completely useless until like--

**AUDIENCE:** This or--

**TADGE DRYJA:** What?

**AUDIENCE:** Like, this or that.

**TADGE DRYJA:** Well, OK, so if you look, there's a sequence. And it was always just FFFFF. You know, that's like 2 to the 32 or whatever. 2 to 32 minus 1, I think. And it didn't do anything. Satoshi put it in because he thought, OK, this will let you indicate finality where you can increment the sequence number. So you say, OK, this is sequence 1. And then, I can replace it with sequence 2. And it didn't work because you can't have consensus on those things.

So this was a field that was, sort of, unused in the inputs. And what they did is they said, OK, we'll make that a sequence number where if you now make this, say-- how did it work-- you make the sequence number 100, for example. And then, in your transaction, it will say, OK, this transaction output that you're spending, is it 100 blocks old? And if so, we're good. If not, this input spending fails.

So it's another check in addition to the signature. And then, you can also ensure that, with CHECKSEQUENCEVERIFY, you can put in the opcodes. You can put in the script. OK, this must have a sequence of 100. And then, check that sequence is equal or greater to the depth of the transaction. And yeah, you can mix them and match them in the same script.

It's a little ugly to use because it was introduced as a soft work. And they rename and op and op. A no op opcode got renamed to this. This is, like, no op 3. 6 And so you have to push the number you're checking on the stack, OP_CHECKSEQUENCEVERIFY. And then, you have to drop the number off the stack because this opcode doesn't actually consume anything off the top of the stack so that it looks the same as a no up to nodes that don't know about it.

Anyway. So you've got these two ways to specify output timing. OK. Questions there? Good.

So then, what you can do is you can revoke based on timing. So the idea of the script-- and this is, sort of, in C like notation, not the actual opcodes for Bitcoin because the actual opcodes are, kind of, confusing-- the idea is, OK, you can spend with 2 of 2 multisig, essentially, right? keyA and keyB, or keyC, and wait 100 blocks. And by wait 100 blocks, it means that whatever you're spending has to be confirmed 100 blocks ago or more.

So A and B together, they can spend any time they want. Together? Oops. C can spend, but must wait. And A and B can grab their coins first. So this is a revokable transaction. So the idea is you've got some-- and we call it a commitment transaction in the lightning paper. You've got some input. The funding transaction output. You're spending that. You need 2 of 2 multisig.

So like, Bob's signs and sends it over to Alice. And then, you've got that script where it's, OK, it's Alice's key, and you wait 100 blocks. Or Alice's key and you make a new key for Bob. Like, the Bob revoke key. And this has 2 coins. This has 8 coins.

Oh. Oops. This should be our-- hold on. I should change that. That's, kind of, going to be really confusing. Wait. I put the r in the wrong place. Let me redo this. That's a typo, but it's super confusing. Yeah, that is opposite. Wait. AliceR and Bob. Alice and Bob are-- wait. So I put the same thing on both. Right?

Alice and Bob are-- this one is AliceR and Bob. Yeah, and when you're actually coding this stuff, it's pretty easy to screw up. There's a lot of like, wait, who is Alice and who is Bob? So in any of the code, I don't use Alice and Bob. I use mine and theirs. But a lot of times, it's not clear because it's held by Alice. So this is the transaction that Bob creates, Bob's signs, transfers to Alice, and then it sits on Alice's hard drive. Yes?

**AUDIENCE:** AliceR?

**TADGE DRYJA:** OK, so you make these separate keys. Right? It's a key that Alice creates. Alice creates another pub key pair, and tells the pub key to Bob. And this is her revocable key.

**AUDIENCE:** The revocation.

**TADGE DRYJA:** Yeah, where Alice can then-- basically, what Alice does is Alice gives the private key to Bob in order to revoke this transaction. So they're, sort of, mirror images. In both cases, both parties agree, look, Alice has got 2 coins. Bob's got 8 coins. Right?

But the transaction that Bob creates and signs-- yes, the transaction that Bob creates and signs sends Bob 8 coins in the clear. This is just a regular old pay to pub key hash where Bob gets 8 coins. The coins that are Alice's though, Alice has to wait 100 blocks. If she broadcasts this commitment transaction, she can't spend her money for a day. Like, whatever 100 block is.

Bob can get the money, but only if he knows Alice's R key, Alice's revocable key. So Alice has this Bob doesn't and in the Bob case it's similar you know the outputs are the same but the ideas Alice creates this. Alice signs it. Hands it over to Bob. And in this case, the transaction is, well, Alice gets 2 coins in the clear. This is immediately spendable.

No fancy scripts or anything. Bob gets the money, but he has to wait a day. Or Alice can spend the money if she knows Bob's revokable key is BobR key. So either party, at any time, can broadcast these transactions. The thing is, the money they're supposed to get-- so in Bob's case, Bob can have Bob's signature broadcast, but now he has to wait.

No big deal. Wait 100 blocks. OK. In Alice's case, OK, she can close it and put her signature on here, broadcast, wait a day, spend the money. But what's nice is you can revoke it. Right? So Alice can tell Bob I'm not going to do this. I'm not going to use this transaction. You know, I deleted it. And Bob's like, yeah, sure you deleted it.

The way I'll prove to you that I deleted this is I will tell you the private key. I'll tell you AliceR. So now Bob knows because Bob created this transaction. Right? Bob signed it. Bob knows what it looks like. And Bob knows, well, yeah, this is the script.

Since Bob knows AliceR and Bob knows his own key, Bob knows, look, this money is mine. This money's mine too because I can do this immediately. So if you broadcast this transaction, I don't have to even touch this because this was my money. And this output, I can spend immediately while you have to wait 100 blocks.

And you know, my software is going to automatically do that. So I know AliceR's private key. I know Bob's private key. I just take these 2 coins. As soon as I see this transaction, I take these 2 coins, I spend them back to Bob's address and get all 10. So that's a pretty convincing way for Alice to say, look, I'm not going to use this. I'm deleting to show, even if I do use it, here. Here's the key. Any questions about this?

**AUDIENCE:** Does Alice have to voluntarily give her key up?

**TADGE DRYJA:** Yep. Yeah, so Alice has to hand it over. Right? This is just a regular 32 byte private key that Alice created and same idea for Bob. They both create, essentially, the same transaction but with everything swapped. Bob can broadcast this. If he does, he has to wait a day.

But if he reveals BobR private key to Alice, Alice knows, OK, I get two coins in the clear. And then, these 8 coins, I can sign this. I can sign this. I can get these 8 coins immediately. Bob's got to wait a day. So that's a way to say, look, I'm not going to use these. Any other questions?

OK. So what you do with that is, yeah, either party can broadcast. They have to wait. They exchange these revocation private keys. And now, if they broadcast, the counterparty can take all the faults. So it's pretty good. So the idea is this is a lightning channel. You've got this output. You create these dates.

OK, the first date is Alice gets 1 coin. Bob gets 9 coins. Then, Bob wants to pay Alice. So Bob says, hey, Alice, I'm paying you 4 coins. Here. I'll sign and create this new output, this new transaction. Hand it to Alice. And Alice says, OK, but now we have to revoke this one.

Alice has this. It's signed. Alice can broadcast it, but Alice also wants to be sure that Bob can't broadcast this anymore. So they, sort of, trash it. They reveal their R private keys to each other. So in practice, it could be, well, only Bob has to reveal the private key, in this case because Alice is going to broadcast this, not this.

It's just so much simpler if they both reveal their R keys because then you don't have to worry about something. You know, what happens in the future when Alice has 0.5 and Alice still has this thing? So it's really simple. They both reveal. Now this transaction-- this state 1-- it's, basically, unbroadcast. You know, you're not going to use it. And they can keep doing that. And they can go back and forth. Right?

So now Alice pays Bob and says, hey. Whoops. I wanted to make it go back and forth. Oh well. Oops. Anyway, you can make this like Alice back to 2 and Bob back to 8 or whatever. Basically, arbitrary numbers. You can keep going in both directions and deleting the old one.

So that's pretty useful. There's no lock time for the channel itself. The channel can just keep going on forever. And Alice and Bob can keep sending these things. It's a little more complicated. Yeah, I didn't put the messages.

But they have to send four messages. You can optimize it to three. Right. So if you have Alice and Bob, the first thing they do is Alice sends a signature. So let's say they're at state 5. OK, so she says, here's a signature for state six. And Bob says, OK, I'll give you a signature for state 6. So now, they both build the state 6. All right? Let's say, go from 5 to 6.

And then, Alice can say, OK, I'll give you your revocation key for state 5. And then, Bob can say, OK, I'll give you a revocation key for state 5. And then, they're done. They've created the new state and revoked the old state. Alice can not put rev 5 here in the same message. Do you know why it went on?

**AUDIENCE:** You need rev key.

**TADGE DRYJA:** Well, yeah. Rev key just abbreviate to rev. So why would this be bad if Alice says, look, here's state six, and I'm revoking my claim on state 5? What goes wrong if you do that?

**AUDIENCE:** Bob just takes them all.

**TADGE DRYJA:** Not quite. Bob broadcasting state 6 is OK, in that case.

**AUDIENCE:** Bob can sign the rev key for 5.

**TADGE DRYJA:** No because state 5 is not broadcast. Right? Only Alice can sign and broadcast state 5. So the problem with this is, if you do this, Bob's fine. Alice is stuck because Alice does not have state 6 on her drive. All Alice has on her drive is state 5, and she's just revealed the key for state 5 to revoke it.

So basically, Alice has nothing, at this point. And Bob can just wait, and Alice is stuck. And that's a really good position for Bob to be in because Bob is like, hey, I can close the channel. I can close the channel at state 5 or state 6. Probably, he wants state 6. Usually, you're sending money. But I can close this channel, and you can't. So give me some money, and I'll close the channel. That's the attack.

**AUDIENCE:** So conceptually, a new state has to be created before you can revoke it.

**TADGE DRYJA:** Right. And you want it on your side. So this is creating a new state for Bob, but it's revoking the old state for Alice if and when you do this. So that's dangerous. Alice doesn't want to do that. So the simplest way is, OK, create the new state by signing 6, revoking 5, revoking 5. You can optimize this though. Can you guys see the-- it reduces the three messages?

**AUDIENCE:** He can send rev 5 back?

**TADGE DRYJA:** Yeah. He can send 6, 6 rev 5 at the same time. So you can do it down into three messages. So that's cool. It's basically, here's state 6. OK, here is state 6, and I revoke my claim on state 5. OK, revoke my claim on state 5. Then, they both update their-- once this happens, the UI is finished.

**AUDIENCE:** And you're generating keys on the fly, right? You don't need to pre-generate them?

**TADGE DRYJA:** It ends up being faster if you pre-generate, like, one key ahead where you say like, hey, here's the next public key I'm going to use so that I can-- because Alice, when she's building state 6, she needs to know Bob's revocation key in the state 6. So you pre-share one or two steps ahead so that you don't have to say, hey, can you give me the next key? OK.

Otherwise, you'd have this message where it's like request state six key. Send it. Send the signature back. So there's more data in these messages without optimizing.

**AUDIENCE:** So I assume with public keys, in that case, it's similar.

**TADGE DRYJA:** Yeah. Yeah, so you also put some public keys in here. And this is the private key. Yes?

**AUDIENCE:** Where does state 6 [INAUDIBLE]?

**TADGE DRYJA:** In here. You're sending the signature, but you can also send here's how much I'm giving you. Here's what the times and stuff are. Most of it can be computed because the transactions don't change that much. The scripts are the same. Everyone knows what this is going to look like, so you don't have to actually send too much data.

Another optimization I didn't put in is you're going to, potentially, have to retain all of these R private keys. So if you make a million states and 900 and whatever are revoked, Bob has to keep track of all those million old Alice private keys. And Alice has to keep track of all those old Bob private keys. That's, kind of, annoying. Not the end of the world, but it's like 32 bytes each. Can be a couple hundred megabytes if you use this channel a lot.

So I thought I came up with it. But at the same time, when I was coming up with it, I was like, no, I'm pretty sure this is in some paper somewhere, and I can't find it. And then, talking to people here, apparently, it's Goldwasser. Like, a professor here came up with it in, like, the '80s. I called it elkrem because spelled backwards it's merkle, right?

So you have a binary tree. And you say, OK, 0, 1, 2, 3, 4, 5. 6 Is the root, for example. And when you want to descend to the left, you take the value here, append an l, or maybe 0 is left and 1 is right or something or l or r, and you hash it. So you say, OK, I'm taking this value, sticking something on hashing to get down here. And similarly to here, you make a big tree.

The idea there is you can reveal things sequentially and say, OK, I reveal 0. Fine. I'll store it. I'll reveal 1. Store it. I reveal 2. Now, you can delete 0 and 1 because you know the parent, and you can compute 0 and 1 from knowing 2. Right? So I give you 0. Give you 1. Give me 2. You delete 0 and 1.

Going to give me 3, 4, 5. You delete 3 and 4. When I give you 6, you delete 2 and 5. So that way, you only have to store login secrets. Given 0 and 1, you can't compute 2. And given 2, 3, and 4, you can't compute 5 or 6. But once you've got them, you can compute all the old ones.

So if we do that for the secret keys, that reduces storage a lot. So that's kind of a nice optimization. It was pretty quick. And so then, your total data you're storing for these channels never gets more than a kilobyte or two. So that's pretty useful. OK. Other questions on this? This alone, actually, is really complicated.

**AUDIENCE:** You only want one active state, right?

**TADGE DRYJA:** So there is a, sort of, superposition time here where-- so let's say this is time 0. This is time 1. This is time 2. This is time 3. So at time 1, Bob's got 2 states he can broadcast. Right? Bob's like, I can do 6 or 5. Alice can only broadcast 5. But Alice knows that Bob can broadcast 6.

So Alice is also like, well, I'm not sure which it is. From Alice's perspective, if I close, I'm at state 5. If Bob closes, he's at state 6. And then, at state 2, Alice can only broadcast state 6 but Bob can broadcast-- sorry, Alice can broadcast either, but Bob can only broadcast one.

So yeah, during this process, it's not clear which state it is. And you know, if something happens, like Alice sends sig-- signature for 6-- and Bob goes offline. Then, it's like, OK, I never got a response. I'll just try to close at state 5, I guess. And then, I guess that payment never went through.

But then, Bob could say, no, I got it and broadcast it. So things can get weird where you're not really sure what happens here. So it's more a UI issue where, in the UI, you only show things after you get to the end.

Hopefully, in most cases, this whole process takes a fraction of a second because you're just sending some messages to each other. So that's the nice part. That's, sort of, the lightning-y thing. It's fast right. Yes?

**AUDIENCE:** Does this require both actors to be online at the same time?

**TADGE DRYJA:** Yeah. Yeah, right. So they're sending messages to each other. They can't really pre-compute these things because they don't know how much money is getting sent back and forth. So when you're signing state 6, you're also signing the output values.

And when Bob receives state 6 and signs his version of state 6, he also is like, you know, I don't know how much money I'm getting yet. So I need to sign off on that. So yeah, both parties need to be online to receive, which is different than normally in Bitcoin. That's the same as the uni-directional payment channels.

In the unidirectional one, where Bob's the receiver, he doesn't have to sign. But he still has to be online because the transaction just goes directly to him. It doesn't go onto the block chain. None of these are publicly known. OK.

**AUDIENCE:** One question.

**TADGE DRYJA:** Yes?

**AUDIENCE:** Why do we care about old states? Say you had a million states. Why do we care about the old ones?

**TADGE DRYJA:** Well, this was state 1 where Alice had 1 and Bob had 9. And then, you keep going, and the general trend is Bob's losing money. Right? You go back and forth but, at the end, Bob has 1. Bob needs to worry that Alice holds onto state 1. And then, months later, broadcasts it while Bob's not watching.

Sorry, wait. No. Opposite. Alice has to worry about Bob broadcasting state 1 in the future.

**AUDIENCE:** Well, everything's going to be revoked, right?

**TADGE DRYJA:** Right. So you still need to keep the secret, though. It's revoked, but the whole process of revocation is Bob gives a key to Alice. So if Bob forgets that key-- sorry. If Alice forgets this key, Bob can broadcast it. And then, Alice can't grab it.

**AUDIENCE:** It doesn't stop them from broadcasting.

**TADGE DRYJA:** Yeah, they can broadcast. So from the network's perspective, these are all equally valid. The network has no idea which came when or anything like that. The only way you enforce it is, if Bob broadcasts this, Alice is like, uh uh, I'm taking this one because I have the key. So that's also a pretty significant, different security model. It's a security reduction. Right?

Now, you have to actively defend your channel because if you go offline-- I'm Alice. I'm supposed to have 9 coins. I go on vacation. I turn off my computer, and Bob knows I've turned off my computer. Then, Bob can say, oh, Alice is out like. It's this 100 block delay. She's not going to be online.

I'm just going to broadcast this, get the 9 coins, wait 100 blocks, and then sweep them as soon as I can. And then, Alice comes back, plugs in, and it's like, oh, you were-- wait. Oh, it says I lost 8 coins. What? Because Bob defrauded, so you have to be online.

I think I'm going to get into it next time. There is a way to outsource that so that you don't have to be watching. You can have someone else watch for you. And I'll, probably, talk about that next time. But then, the last 20ish minutes-- yeah, so this is cool.

This is lightning channels. Two parties. Indefinite. You still need to create a channel to pay people. So you're going to do 1 on chain transaction to open the channel. 1 to close the channel. Potentially, 2 to close the channel, but that's pretty rare. If you need to immediately grab it, then you're time sensitive, and you're going to have two closing transactions.

If you close normally-- like, you close the current state-- you need to wait 100 blocks for your money to be active. But you don't have to spend it immediately after that. You can just leave it because you know. So in this case, let's say, this is Bob's held transaction. Bob broadcasts it.

Bob knows, OK, Alice gets her 2 coins. I get my 8 coins. I have to wait 100 blocks, but I don't actually have to spend it after 100 blocks. I can just leave it in my wallet because Bob knows he never gave Alice this key. So this clause-- this or Alice and BobR key-- this is not going to happen. So Bob's safe and just leaves it.

So he can just use that, as well. However, if it's the no you don't kind of thing, then you have to do two transactions. That's rare though. So this is pretty good, but you still have to do two transactions. So if you use this, you open a channel, you use it once, and then you close it. Well, that was dumb. You just paid twice as much in fees that you needed to.

So it's useful for some things, not for others. Could we do something like multiple party channels? So there's research about this. It gets real ugly real fast. Like, I want a single channel with three or four users or, like, a tree of channels where the first top output has 4 of 4 multisig and then branches into these 2 of 2 multisigs.

There's a lot of interesting ideas, but it gets a little ugly. But the off chain scalability gets bad. There's a lot of n squared kind of stuff. So what about a forwarding network of point to point channels? So OK, we have these two party channels, and people are connected with them. So Alice and Bob have a channel, and Bob and Carol have a channel.

So this would be cool. Alice can just say hey, Bob, I will pay you if you pay Carol. And you can do that. Alice says, look, I'm just making a new state where you have an extra coin. Now, please, give this extra coin Carol. And Bob says, yeah, OK and does. So the word there that should have immediately raised a, uh oh, that's not how this stuff works, is the word please. You don't say please in Bitcoin.

So yeah, the other option is Bob just keeps it. So Alice says, OK, here. I'll add a coin to your output in this channel. And please add a coin to Carol's output in your channel with her. And Bob says, yeah. No, I'm just going to keep the coin. Thanks. So this might not be super crazy in that, well, you got a channel with them. Maybe they're [INAUDIBLE] chain.

Also, you can you reduce the number of coins you send per update. You can say, look, I'm going to send one Satoshi at a time. And so worst case, you keep one Satoshi that you shouldn't have. Right? You could. Right? It's not as scalable, though, because let's say you want to send a lot of money to Carol. You're going to have to keep doing these little things. It's, kind of, annoying.

Also, what if you want multiple hops where you're not even sure who's in the middle of your chain? It doesn't work for that. OK. So you can have what's called HTLCs. And these are even more complicated.

OK, so the idea is it's Hash/Time Locked Contract. Or Hash/Time Lock Contract. Whatever. The script is, essentially, this. KeyA and preimageR. You need to present the preimage of a hash. Or KeyB and some absolute lock time. Op check lock time verify. And it's important that this is an absolute time, not a relative time because these should be able to expire, regardless of when they get broadcast onto the blockchain.

So the idea is, in this case-- so let's say A is Alice, B is Bob-- Alice gets the money if she knows this preimageR. This 32 bite. So what actually goes here is the hash of R. It's like a Pay-to-PubKey Hash, where you say op dup, op hash 160, the thing, op equal verify. So you need to know some preimage, and you need to sign.

Or you can be Bob, and then you can sign after a certain date. So this is, kind of, nice. It's like, well, if R is known, then it's Alice's money. If R is unknown, then it's Bob's money after a certain period of time. OK, so this was the revokable transaction where Alice got 2 coins, Bob's got 8 coins, and then you add a third output, this HTLC output where now Alice has 2 coins, Bob has 7 coins, and Alice might be getting an extra coin if she learns R.

Bob gets the coin back after height 500,000. So similarly, with this it's like, OK, this is Bob's money. But if he does something bad, Alice can grab it. This is Alice's money if she knows R. If she doesn't, Bob gets it back. So there's no-- like, really, this is Bob's money. Right? This only happens if someone's trying to rip someone off.

This, on the other hand, is well it depends. Is Alice going to learn R or not? And what is R? OK. So the multi-party adversarial payment model. So the lightning forwarding is Alice wants to pay Carol. Right? She doesn't want to just ask Bob to keep forwarding the funds because she doesn't trust Bob.

So she connects to Carol-- and by connect I mean, like, regular TCP connection-- and says, hey, Carol, make a random number R, and send me the hash. carol OK, here you go. Carol knows H, which is the hash and R, which is the pre-image. She sends over H to Alice.

Alice then says to Bob, hey, I'm going to add an HTLC to our channel. So basically, Bob, if you know the pre-image of H-- if you know R-- you can get this coin. If not, I get the coin back after 5:00. And then, Bob says, OK, cool. I know H now, but I don't know R. So I can't grab this money. I don't know this, so there's no way this is going to be my money. It's just going to go back to you after 5:00.

And Alice says, yeah, but Carol knows R, so ask Carol. Bob says, OK, I get it. I will forward this on to Carol. And then, I create a similar output. I make sure that these are the same R or the same H. I don't know R yet. But I say, hey, Carol, if you know R-- if you know the pre-image of H-- you can get this coin. If not, after 4:00 o'clock, I get the money back.

And then, Carol says, oh, cool. Well guess what? I know R because I made R up. So from

Carol's perspective, Carrol could broadcast this onto the blockchain immediately. And she will be able to grab the money because she knows R. She has to grab it soon because, by 4:00 o'clock, Bob can grab the money.

And if Carol does this-- she closes the channel, she uses R because she knows it to take this coin from this output. Carol doing that will reveal R to everyone in the world because it's on the blockchain now. And so then Bob can say, ah, I know R now. I need to get this money. So Bob can close the channel. Grab the money. Yes?

**AUDIENCE:** Those times are actually block heights, right?

**TADGE DRYJA:** In the software, yeah, we use heights. You could do it with time. The main reason for heights is, what if a block's full because height's, sort of, variable. Like, maybe a bunch of blocks come out quickly, and maybe they come out slowly. So time is a little dangerous in that you could say, OK, 4:00 o'clock.

But maybe in the next five hours, only a couple blocks come out, just by chance. And they're full, and you're not able to broadcast. You're not able to get this in, and it's risky. So the height takes into account the variable nature of it a little bit better.

**AUDIENCE:** What if there's some situation where blocks are [INAUDIBLE]?

**TADGE DRYJA:** Yeah. Well, child pays for parent-- yeah. So the question is, what if the fees get really high? The thing is, this whole process should take a few seconds. So it's not like these outputs have weird fees that are old. One issue is what if you're building these states back and forth. And then, you sit on one for a few weeks. And then, the fee rate goes way up.

And you're like shoot, if I tried to broadcast these transactions, the fee's too low and it might not get confirmed. It's not a huge risk because if you're not trying to rip people off, you don't actually have any time risk. In this case, this gets built. This gets built. These are, sort of, in real time. And so you know what the fee rate is in the network, so you can adapt to it. But yeah, that's an interesting one. Yeah?

**AUDIENCE:** Are there any problems in this system being slow? Because I think, as we were reading literature prior to blockchain, like 1980, that I read some stuff about distributed computing that if you n number of nodes, you need a minimal number of 3 [INAUDIBLE] or something in order to get consensus. But does blockchain sort of make something underneath all of the--

**TADGE DRYJA:** You don't need global consensus in that-- I did only 3, so you can't show. If you have four, you can sort of say-- let's say you have Alice, Bob, Carol, Dave. You don't actually have to know the whole thing. You just have to know the person before you and the person after you.

And you don't care what the actual endpoints are. So to some extent, Bob doesn't need to know. Maybe Alice got this HTLC forwarded from someone else before. And Bob's like, all I know is I could get money from Alice if I know R, and Carol gets it from me if she knows R. And maybe that keeps going or something. You don't need global consensus.

**AUDIENCE:** And that's because of how it's chained?

**TADGE DRYJA:** Yeah.

**AUDIENCE:** Like the message?

**TADGE DRYJA:** Yeah. You're only concerned with the channels you're participating in. If there's something else happening before it, you might want to watch in case ours shows up. But you don't have to worry about it. And then, these will all get broadcast. The failure mode is broadcast everything onto the blockchain, and let the blockchain sort it out.

And that's very high latency because it's, like, 10 minutes. But the idea is the blockchain already has that global consensus because it's really high latency and everyone can agree on it. So I think that's the basic idea. Yeah, so what you could do here is you could just close it. Right? You could broadcast this whole transaction on the blockchain and get the payment through.

That's really inefficient. You want to keep your channels open so what instead happens-- when everything's working. It could happen. right? Carol could just broadcast it. OK, that's annoying. Whatever. But no one loses money. Or Carol can go offline, at this point. And then, these things get stuck. And there's all sorts of weird things that can go wrong, but you don't lose money.

But what happens when things go right is Carol says, hey, Bob, I know R. And so really, this whole crazy HTLC thing, it's just my money. I know what R is. You're not getting this money at 4:00 o'clock because I know R. So here. Look. I'll tell you R. And when Carol tells Bob R, she's doing two things. She's proving that this HTLC is her money.

She's also giving Bob R so that he can then grab Alice's money. And so as soon as she reveals R to Bob, Bob says, yeah, I agree. This HTLC is superfluous, at this point. I know you know R, so this is just your money. And if it gets close to 4:00 o'clock, I know you're going to broadcast this channel onto the chain if I don't let you clear it out.

So let's just make a new state through these state updates where you just have the extra money. Let's clear out that HTLC. So that 1 coin goes to you. And Carol's like, OK, cool. I agree. We'll sign this, and they'll all just get rid of the HTLC output. Then, Bob goes to Alice and says, hey, Alice, guess what? I know R. So this part, I'm going to do it. I know R and Bob. Alice, after 5:00 o'clock, it's not happening.

If you stop answering me and it gets to 4:30, I'm just going to broadcast the channel on chain and grab it that way. And then, Bob reveals R to Alice. Alice says, oh yeah, you do know R. OK, we'll clear the HTLC up, and you get the money.

And then, Alice gets R. And that's, sort of, a receipt for the payment. And Alice knows, well, Carol was the one who told me H. Carol must have gotten the money. Right? So that's my confirmation of that.

**AUDIENCE:** How does creating a new channel state of the HTLC-- how does that provoke the HTLC itself?

**TADGE DRYJA:** Oh, so I didn't put it in the script. But it's revocable the same way the regular outputs are revocable. There's also this, which you don't even have to do because the idea is that the HTLCs are very small in comparison with the other outputs in the state that, if you try to broadcast an old one, you'll lose way more than the HTLCs worth.

But you can also tack on a or other clause to the HTLC script with a relative lock time. So there's actually a couple different ways to do it. The simplest is you just say, look, I've got 2 coins, 7 coins, and then a 1 coin HTLC. I'm not going to risk my 7 coins to go back and try to get this 1 coin in HTLC. But that's not how most of the software does it. You just add a new clause.

OK, so this is a multiple party adversarial. Whoops. So yeah, that would be really useful. Right? You build lots of nodes with channels connecting. And they form a big graph. And then, you can request payment routing via these HTLC outputs. And if you open a few channels, then you can pay lots of different users on the network. Yes?

**AUDIENCE:** I may have missed this, but what is Bob's incentive in relaying transactions?

**TADGE DRYJA:** OK. So yeah, Bob could charge a fee, or he could just be a nice guy. That's, basically, the--

**AUDIENCE:** But you said you never ask for [INAUDIBLE], because you know Bob's not a nice guy.

**TADGE DRYJA:** Yeah, so you can pay him. It's going to be hard to pay him much because Alice is going to say, look, we've got a channel. We're using it. If you charge me a ton to forward payments to Carol, I'm just going to close the channel and open with someone who does it cheaper. So yeah, you can make the amounts slightly less.

So Alice says, I'm sending you 1 coin. And Bob says, OK, I'm sending 0.9999 coins to Carol, and I'm keeping the difference. You can have that. And then, it's really up to Carol to decide if the fee-- because all Alice knows is, hey, I'm sending one coin.

What if Bob puts a crazy fee, like 1/2 a coin? Carol sees 0.5. And Carol's like, no, that's way too low. I don't know what happened, if Alice sent 0.5 and Bob forwarded it to me or if Alice sent 1 and Bob forwarded me 0.5. But I'm not accepting this. Right? I'm not going to reveal R to you. I'm just going to say no.

So you can do that. At this point, Carol has not yet revealed R. Carol can refuse and say, look, Bob, I'm not going to accept this HTLC. We can just clear it out right away without revealing R and just, basically, undo what we just did. Yeah.

**AUDIENCE:** So two follow-ups to that. The first is, can there be a situation where it's an efficient market place for these relaying points in the network where Alice asks for a bid, and the lowest transaction fee wins and she routes it from that?

**TADGE DRYJA:** You can. So let's say Alice is connected to three Bob's in the middle, and then asks all of them, hey, what are you going to charge me to forward a coin to Carol? Then, they can all answer. It's a little weird because they can lie, at that point because they haven't actually signed anything.

And this, Bob can say, oh, I'll do it for free. And this Bob says, oh, I'll do it for 1/2 a coin. And this Bob says, I'll do it a tenth of a coin. And then, you try to go for the free one, and Carol gets some really low amount and says, I'm not telling you R. Let's cancel this.

And then, if Alice trusts Carol, Carol could be like, Alice, Bob just tried to charge a ridiculous fee. Don't use this. You know, this Bob is bad. And then, you try the other Bob, and the other Bob does the right thing. Yeah.

**AUDIENCE:** Can't Alice just send directly to Carol?

**TADGE DRYJA:** Open a channel? So yeah, you could. You just have to open a channel.

**AUDIENCE:** Is that expensive or something?

**TADGE DRYJA:** Yeah. You have to go on chain. We, sort of, saw a preview of it in November, December, and January where it was actually, kind of, expensive to use Bitcoin. Now it's cheap again. Costs a few cents. But for a month or two, it was like, oh shoot. Like, this costs $5 to make a on chain transaction.

And so creating a direct channel from Alice to Carol, it might cost $5. Or it might cost $20. Or long term, if everyone's trying to use the blockchain and the fees go really high, it could be, kind of, expensive. And so the idea is, well, if I've got a channel already, I'd rather use that and move my money around that way without having to touch the blockchain. Yeah.

**AUDIENCE:** Wouldn't it be a graph problem trying to figure out what the cheapest way to get to Carol is? Would that be [INAUDIBLE], and would that be scalable?

**TADGE DRYJA:** So you can sort of-- so it's [INAUDIBLE] or it's like [INAUDIBLE], kind of, if you know the whole graph.

**AUDIENCE:** Yeah, but then you have to ask.

**TADGE DRYJA:** Yeah. So the one part is if you know the whole graph. And you can know the whole graph because the graph never gets bigger than the UTXO set. Every channel is a transaction output. And so, if you're running a full node and you have the whole UTXO set, the entire graph is on the order of that size. So maybe it's twice as big in that you have to have some extra metadata about what these outputs are.

And Bob and Carol can make proof. So like, oh, here is our channel, and it's in the UTXO set. And we can reveal our pub keys, the big O in here. So if you have the whole graph, you could do some routing that's in login kind of speed. But then, it might not work.

So there's actually tons of ways things can go wrong in this. That's the, sort of, fun part of dealing with it. One of the ways it can not work is you're like, yeah, I found a path. The simplest is, I found a path, and I query Bob, and Bob is just offline. I'm like, oh, OK. Well, that's not an active path. OK, what's the second shortest path? OK, this other Bob, and he says, yes.

But the fee seems really high. So OK, I found the--

So most of the algorithms, though, will get you the second and third and fourth and fifth shortest in much less time than recomputing the whole thing from scratch. So you can get a bunch of pretty good ones and try them all. Like query and the--

**AUDIENCE:** By the time you try them all, what if the first Bob is online and you try the first Bob?

**TADGE DRYJA:** Yeah. The thing is, in practice, it's all real fast. Well right now, for a small network, it's nothing. But even total number of UTXOs is on the order of, like, millions. Right? So if you've got a graph with millions of edges, that's easy for a computer. Right? They'll do it in less than a second.

And then, your main thing here is network latency for all of these things. For building new states, it's the time taken for the data to get from one side to the other. It's much larger than computing the signatures, verifying the signatures, building the hash tree thing. All that stuff is real fast compared to a network.

So the same with routing. Looking at the graph and dissecting it is going to be faster than contacting all the things. There's a bunch of other things that can go wrong. So for example, this gets built, get to here, and Carol just stops responding. So now you've got these two HTLCs. They're a third output in your channel transaction, and Carol's just not saying anything.

And then, Alice goes to Bob. He's like, Bob, what's up? Like, we're going to clear this HTLC. Usually, these only take a second. And Bob's like, uh, yeah, hold on. And so it's, kind of, awkward because now we've got this HTLC and you don't know whose money it ends up being.

The way to mitigate that, Rusty-- yeah, I think it works; Rusty's the other guy who works on this stuff-- was that he has a timeout period-- I think 30 seconds, a minute, whatever-- where Alice says, OK, Bob, I made this HTLC a minute ago. Minute's up. You need to either shut down this channel right now because you're doing something bad, or you need to prove to me that a channel has been shut down with this HTLC, this H value in it.

And then, I know where to ascribe blame. So if Bob's like, yeah, I can't contact Carol. Carol's offline. Then, Alice says, all right. Well, then, shut down the channel.

**AUDIENCE:** Isn't that pretty expensive considering Carol isn't necessarily malicious?

**TADGE DRYJA:** Yeah, so that's a downside. Right?

**AUDIENCE:** Yeah, it seems a little harsh on Carol.

**TADGE DRYJA:** Well, the thing is, if you don't do that--

**AUDIENCE:** It just seems to me like running a lightning node is not anywhere near as voluntary as running a Bitcoin node in this world where it's, like, being live really matters. Responding even faster than the timeouts really matters.

**TADGE DRYJA:** OK. OK, one thing to mitigate that. Carol responded by signing off on the HTLC, and then immediately goes offline. It's not like you accident--

**AUDIENCE:** This is all protocol. She didn't truly break the rule by doing so.

**TADGE DRYJA:** Well, part of the protocol is, if you have an HTLC and you don't remove it within 30 to 60 seconds, it's like, you know, that's a violation of the rules, and we're going to close their channels because if you're just straight up offline, and then it's like, well, OK, I'm not going to use that channel. But it's like, oh, I accepted this, and immediately said, OK, I'm not talking to you anymore.

And they're like, well, now it's stuck because the attack vector is build a ton of HTLCs. You could add multiple per channel and build them all, and they all get stuck. And one party is just not responding with their R values. And then, everyone's like, OK, if we try to close, our transactions now have, like, 100 outputs. And they're huge and have all these fees.

Also, all these HTLCs, yeah, they might be really little, and we have to wait. And then, it charges a lot of fees to recover them and stuff. So you don't want people to be able to, sort of, blow up the number of HTLCs in your channels. So the way to do it is, if one of them is stuck, you need to find someone to blame.

And yeah, it's possible. Right? But the thing is, you accurately find the person to blame. If Carol accepts the HTLC, and then, a split second later, unplugs their computer, Bob's like, what? OK, fine. I'm closing the channel. You just did something really annoying.

And Bob's like, if I don't close this channel, Alice is going to close the channel on me because Alice is going to think I'm the one acting weird. So yeah, there's all sorts of weird edge cases.

There's all sorts of weird stuff that can happen. It's definitely more complicated than just the straight up UTXOs in Bitcoin. Yes?

**AUDIENCE:** Couldn't Carol respond to Bob, and then Bob be malicious and close the channel? And then say, hey, Alice, check it out. Carol was malicious.

**TADGE DRYJA:** Yep. That's OK.

**AUDIENCE:** And then, you wouldn't get the blame, but Carol presumably--

**TADGE DRYJA:** Bob blames Carol. Carol says, no, it was actually Bob. From Alice's point of view, like, whatever. Like, something got closed. There was a cost to adding this HTLC. You know, there was a cost to having me stuck with this HTLC for five hours. Right? Something unchained happened. So the thing is, having one HTLC stuck for a few hours is no big deal.

After 5:00, Alice just goes to Bob and is like, look, well let's clear it. Right? You don't know R. If you knew R, you would've told me by now, so let's just get rid of this thing. And Bob's like, yeah, fine. And they clear it out. So the idea is it's not a big attack if you only have one of them. You want some cost to performing that.

And so if Bob closes the channel-- whether it's Carol's fault or Bob's fault-- Alice is sort of like, look, well something happened, so someone's bearing the cost for this. So they're not going to keep doing it. OK.

**AUDIENCE:** So in reality, is it maybe like five hours, or is it more like couple minutes?

**TADGE DRYJA:** You actually need hours because-- oh, other attack.

**AUDIENCE:** Yeah.

**TADGE DRYJA:** Let's say these were both at 5:00. Right? And then, Carol closes the channel right at 5:00. Right before 5:00 o'clock, Carol closes the channel, uses R, grabs it. Bob then says, oh shoot, there's R. Alice is like, nuh uh. Alice closes the channel right after 5:00 and grabs hers by signing with her key.

And then, Bob's screwed. Right? Bob's like, shoot. Wait. Carol just got the money from me using R. And then, Alice closed and got the money from me using the time out. And I didn't get anything. I lost the money on this side, and I didn't get the money on this side. So Bob just had a huge loss there.

So Bob really wants to make sure that this is later. Like, Alice can't get the money until well after Carol can get the money with the timeout function. Or sorry, well after Bob can get his money with the timeout.

**AUDIENCE:** Did Bob set that time though?

**TADGE DRYJA:** Yeah. Hold on. Bob sets this time, yes. Then, Alice sets this time.

**AUDIENCE:** So he just has to make sure they're [INAUDIBLE].

**TADGE DRYJA:** Yeah, he has to make sure his earlier. And continuing that, if Carol makes HTLC today-- if she makes it at 3:00 o'clock or whatever-- I don't remember what the defaults are in the different-- like, I do it all in testnet. And so in testnet, I do five blocks because who cares. I think in the main net one's people are trying out, they do quite a bit more because it's better to have your money stuck for a day than to lose it, I guess.

So you can make different time outs and stuff like that. OK, what else? I'm, basically, done. There's a whole bunch of other cool things you can do. And I think I'll talk about that next time. Yeah, you can do cross chain swaps. So this is a preview of after-- just real quick.

This channel and this channel do not need to be on the same blockchain. Totally fine for this to be Bitcoin and this to be Litecoin or this to be Dogecoin. It actually totally works. So that's kind of cool.

Security. So you want to monitor your channels in case someone tries to rip you off. You can outsource that in a nice way where you can give it to a third party and say, hey, I've got this channel. Watch it for me. Not only that. I've got a channel. Watch it for me. I'm not going to tell you what channel it is, how much money I have. Basically, anything. And you can still protect it for me. So that's, kind of, cool. You can anonymously do that.

Stuck HTLCs, I just talked about. Dust and fees gets really ugly. There's a lot of cool stuff you can do. So I think the schedule is I'll talk about the rest of lightning on Monday after spring break, and then go into discrete log contracts, which is another similar construction on the Wednesday after. OK. Any questions about all this stuff? I'm sure, yeah.

**AUDIENCE:** What are the time factors that miners pose to this situation?

**TADGE DRYJA:** Miners can make time outs not happen. If they're coordinated, they can say we're just going to not accept this transaction till later. So that's bad. You know, there's other weird attack factors. The one that I worry about most-- not most-- the one that I worry about is, if you make a backup of your wallet and you restore it, you lose all your money.

So you make a backup here at state two onto your USB stick, and you go home, and you drop your laptop in the pool. And you're like, shoot. Well, good thing I got my backup. Well, sorry. You make a backup here at work on your USB stick. And then, you keep spending money. And then, you drop your laptop in the pool. And then, you restore from your USB stick to here. And then, you try to broadcast this.

But you've already revealed your private keys. And so Bob's just like, what? Like, that's red. Like, you revealed it. And then, he takes all the money. So yeah, don't make backups. The reason I worry about it is it's very counterintuitive. There's nothing in the software that does this. The software I can write, like, you know, don't make backups. Always only keep this, but you can just copy the files.

And it seems like the kind of thing people might do and put warnings on the read me's and stuff. So that's, kind of, a big risk. So there's a lot of how do we make this safe. And a lot of people criticize it. Like, this is too complicated.

There's enormous communities on the internet that hate this and hate me. And go to RBTC sometimes. They really don't like the Lightning Network. On the other hand, it's like, well, you don't like it, you don't have to use it. Right? I'm just running the software for free for you guys but anyway.

But yeah, and you know, there's legitimate, like, this is kind of complicated. It's not going to work for everything. There's all sorts of limits to this too.