MITOCW | 17. Anonymity, Coinjoin and Signature Aggregation

TADGE DRYJA: OK, so today I'll talk about CoinJoin, signature aggregation, well before that privacy, but privacy, CoinJoin, CoinShuffle, signature aggregation, and how these all connect.

It's a little bit of a leap to put these both in the same class, but not really. They're connected. But they are somewhat separate things. Today, I'll talk about privacy, CoinJoin, and various ways to do that, and then aggregate signatures, so Schnorr multi-signatures, and then aggregation, and attacks on the system.

OK, so the idea is privacy. And there's a bunch of terms we can use here, like anonymity and fungibility. And whatever the term for this, I don't need any, because I have nothing to hide.

So I don't need any privacy. I don't need any anonymity. I'm a good person. I don't break the law. I'm a boring guy. I don't do anything crazy.

I'm joking, but-- no, I'm not joking about being boring and not doing anything crazy.

That is actually pretty true. I mostly just work on this stuff.

But, yeah, that's a fairly common thing that people say. And the sort of clearest example is if you don't have anything to hide, you don't have any bitcoin, because literally, if you don't hide your private key, someone will just immediately take your bitcoin. So you do have secrets.

Your secret is your private key and your password. And if bitcoin and these types of systems really take off, potentially most of your money and a lot of your wealth could be tied up in what is a secret. So if you're not able to keep your secrets you could lose a lot of money. And larger, I sort of think that going forward-- this is very thought leader, future kind of thing.

But going forward, it seems like sort of all you have is your secrets. To the extent that you can own anything, the thing that you really have control of, you're like, OK, I got these things in my head that no one else knows. And maybe I own my car. But that's also sort of property rights, and legal systems, and maybe you can get repossessed.

Maybe I own this land. But, again, that's sort of this thing with the state. But I definitely really have the stuff in my head. And I can at least try to keep that.

So even if you think you have nothing to hide, you do have your private keys to hide. And that's sort of the obvious one. But larger, you want privacy because, in general, you don't want to reveal stuff about your coins. This is specific to bitcoin.

But my philosophy is I generally don't want to reveal stuff ever. And it's a lot of times at stores. I remember like Radio Shack, when it was still a thing when I was in high school, they would always ask you your phone number when you bought anything. And when I was little, I would just tell them my parents phone number at my house, because they asked.

But then once I was in college, I was like, wait, no I'm just going to make something up. And now I do that all the time. And it sometimes gets annoying for other people, because I would like go to a restaurant. They ask to put my name down. I'll say I'm Fred.

And then my other friend comes. And she was like, wait, you're not listed. And then she put her name in. I'm like, no, I'm already there. I'm Fred, and confusing things like that.

And I hope, and I sort of think that we're starting to see this kind of thing. Like, there's like Facebook hearings now. And people are all shocked that the company Facebook can read everyone's messages and see everything. Well, yeah, that's how it works. You're giving it all to their servers.

So I think it's starting to be, like where before companies would all be like, we just want to get all the user data possible, hopefully going forward data will be seen more as a liability and less as an asset. We're not there yet, because the companies who get sort of hacked and lose all this user data don't really get punished. So the Equifax thing, and all these different data breaches, there's not a ton of consequences yet.

Maybe there will be in the future. And maybe companies will start to not want all this data. But it's not as clean cut as, well, I didn't do anything wrong, so I don't want to hide anything.

Sort of default hide is my stance. And there's so many conflicting interests where every time you're on the websites, it's like, this website wants your location. And you're like, wait, how do I always disable that? I don't know the settings. But everyone wants this stuff.

So another reason, specifically in the case of money, is what's called fungibility. And it sounds like a weird word. It just means that every bitcoin is the same, or every dollar is the same.

So dollars are fungible, in that, physically, this $20 bill and this $20 bill are worth the same. And different denominations, so I will give you a 20 for two 10's. And anyone will do that. They're worth the same.

You don't have, well, this dollar is not worth quite as much. And sometimes that happens, even with dollar bills. Like in other countries, I think in the Philippines when the new $100 bills were issued that have the like little blue shiny thing, people didn't like them.

And people were like, no, I won't trade my old $100 bill for your new $100 bill. I don't trust this new one. Eventually, they got used to it.

But there's things, that even money can be not fungible. So the classic example is, OK, gold is fungible. Diamonds are not. Diamonds, I have little experience with diamonds they seem kind of silly, but they're all unique.

They've got different grades, and different cuts, and different sizes, and carrots, and little inclusions, and imperfections. And you've got this whole industry, where people with these little eye things look at the diamonds, and figure out, oh, this is a good one. And this one is so, so.

Whereas gold is, sort of also has essayists, and good delivery standards, and all these things for the gold bars. But it's much more of a standard. There's no judgment calls.

It's like, OK, this is gold. It's 99.95% pure. It weighs this much. This is how much it's worth. And you can chop gold up and-- sorry, divisibility is different than fungibility.

So if diamonds were all identical but not divisible, they could still maybe function as money. You would just need a lot of little diamonds. But the fact that gold is also divisible is really is a bonus. But the fungibility is really what's nice about it, and what makes it sort of used as money.

Currency's fungibility is actually-- I'm not a lawyer, but I hang out with them. It's enforced by the law itself. So there's an interesting case, Crawford v. The Royal Bank, in Scotland a couple hundred years ago, where there was a guy.

And he wrote his name on a 20 pound note. It's like Crawford. This is his money. And I think he was trying to mail it someone. He lost it.

Years later, the note shows up at a bank. And he's like, no, see that was my money.

I lost it. And he demands the money back.

I lost that money. This is my property. Give it back. It's got my name on it. I can prove it.

And then the court says no, no, no, that's not how money works. The bank has the money.

You can't just write your name on money and then have it be yours.

And this is very different from property. So if you steal a bicycle, and then you go go sell it on-- sorry, if someone steals a bicycle, sells it on Craigslist, and you buy it, and you didn't know, it was just, whatever, some guy's selling a bike on Craigslist. The police show up, and say, hey, that bicycle was stolen. We're taking it back.

And you're like, well, I didn't know. I just bought it on Craigslist. I paid a couple hundred bucks. Police are like, sorry, you're not in trouble. We're not going to charge you with a crime.

But these are stolen goods. We're taking them back, giving it to the rightful owner. And that's how the law works. You can try to complain about it, but they're taking it.

Money is different. If you're running a pizza store, someone comes in, gives you a 20, and buys a pizza, and then presumably eats it or whatever, and then a couple hours later the police come in and say, hey, that guy stole that $20 bill. And we're taking it back, and giving it to its rightful owner.

As the shopkeeper, you're like, I don't even know which $20 bill. Even though there are serial numbers, and the police could say, it was this $20 bill. We're taking it back, or returning it to the person he stole it from, legally they can't do that.

So this is from talking to lawyer people. So money's different. Money is not property, in the same sense. And the fungibility is enforced by the state.

And the divisibility and fungibility are enforced by the banks. The US dollar isn't worth anything.

The government and the banks don't have to give you anything for $1.

They used to sort of have this agreement, where, OK you give us $35. And we'll give you an ounce of gold. That agreement is longer is in place. But they do have to give you a change.

So they do have a duty where if you have a bunch of old, beatup $20 bills, the bank sort of does have to honor those and give you new $20 bills. And that's one of the things that the Bureau of Engraving and Printing, and the Fed, they all have this system. So they maintain the currency, in that sense.

So there's all these things that like you don't really think about in that how money operates. But fungibility, divisibility, those are really important. And most of it is enforced by the legal system.

Bitcoin does not have these legal protections. If want bitcoin to work, even where it's not legally recognized as being money, and our goal is to make it like money. So if they're stolen bitcoins, and you can trace them, the law enforcement can say, no, those bitcoins are stolen. We're recovering them.

So it's not treated as a currency. It's also not controlled as fungible. So in the absence of these sort of protections from the legal system, the software needs to help enforce the fungibility and divisibility of these things. So you can sort of treat the bitcoin software and ruleset as sort of a legal-- it's not a legal system.

But it's a system of rules that governs how the bitcoin transactions work. And so the fungibility has got to be in the system. Right So gold, for example, they don't have-- I don't want to say they don't have rules about gold. There's probably all sorts of old laws about gold.

But you don't really need a law enforcing the fungibility of gold, in that you can sort of melt it down, and it's untraceable. If you have this old gold coin from this empire, and this other gold coin from this empire, you melt the gold, you put it into a new coin, no one can tell. So, similarly, bitcoin needs to enforce fungibility that way.

OK, so any questions so far about these definitions or objections? I don't think bitcoin should be fungible.

AUDIENCE: So bitcoin is more like gold than diamonds. But, in a sense, right after your transaction happens, the probability that it-- say it's only one block deep. I don't think it's actually treated this way.

But someone could potentially, oh, I'll just consider that 90% of one bitcoin, because the probability or whatever--
TADGE DRYJA: They're not valued the same, because they haven't been fully confirmed.

Yeah, that's an interesting way to look at it. I'm not talking about sort of the time-based things here, because I guess in many cases you say, OK, I'm going to wait a day.

If it's got 100 confirmations, it's good. I'll consider it full value. But, yeah, there could be things where you look at the probability of a double spend, and then sort of assign value to it, increasing as time passes or something. That's sort of a unique thing to bitcoin, where the sort of finality of the transfer increases with time, which is not really the case with gold or dollars, where you're like, oh, I got it.

The fact that I've had it for one minute or 10 minutes doesn't really change that I've got it. So that's an interesting point. I'm not really addressing it here.

In this conversation, just assume you wait a day. And then all the probability of it going back is sort of negligible. But it's more the fungibility in that you can trace where it came from, and so assign values to different coins based on that.

So there's a real world example of this. I don't want to name specific names. But customer buys some coins. You go to exchange. You send them a couple hundred bucks.

You say, I want to buy some bitcoins. And the customer buys the bitcoins, and then transfers the coins to a betting shop, a betting company in the UK, and bets on a soccer game, for example. And he wins.

Great, so he bought a coin, sent it over to this UK gambling company, bet on a soccer game, or football I guess they call it, wins. Now, he has two coins. Transfers those two coins back to the US exchange.

And he wants to sell them. Before he can click sell, his account is closed. And the exchange website says, no, you violated our terms of service. We're closing your account.

Take your bitcoins. We've sent all your dollars back to your bank. Withdraw your bitcoins.

We're out.

This happens. So whether you agree with gambling being legal-- so in the UK, however, there's no law violated. And as a United States-- so this is sort of the gray area. I'm pretty sure that if I'm a US citizen, I take a plane over to London, I go to a-- I don't know the lingo, punters shop, betting shop? And then I bet on something, and I win.

Well, I'm pretty sure I have to pay capital gains on those winnings to the US, IRS. But I haven't broken a law, because I'm not in the US when I'm doing this thing. So, similarly, if casino gambling is illegal in Massachusetts, sort of, I can go to Las Vegas. I can gamble.

And I can come back to Massachusetts, and it's OK.

So this is probably a gray area. I don't know if there's actual court cases about this, where you're still sitting in Massachusetts at your computer. But you've sent your money to the United Kingdom. I don't know. I don't know of anyone who's gotten in trouble for this.

I don't know if there's been prosecution against UK betting shops for this kind of thing. But this happens. A lot of companies run betting shops like this.

But I have seen that US exchanges will shut down accounts because of this. And they say, look, you can buy bitcoins. But if you're using these to gamble, we don't want to get involved. It's just more risk for us.

And so they close the accounts. So the problem here is-- what did I say next? From the perspective of the legal system, for the user it's obviously a problem, because it's annoying.

What's the problem for bitcoin here? Why should bitcoin care about this, like as people working on bitcoin? Why do I care?

AUDIENCE: Two reasons. It's the usability of bitcoin, that bitcoin is not usable that's public and private. So that's

the different between fiat currencies, that can be like government, state. But, two, is then what happens to these two bitcoins? Do they just get burned?

They're suspended until there's some decision as to what to happen. They're not owned by the exchange.

TADGE DRYJA: They gave them back. But, yeah, the real thing is these two bitcoins are sort of worth less than two other bitcoins. Now, you've got a sort of different value for these, where these two bitcoins are hot. And I need to get rid of them, and launder them, or something.

AUDIENCE: That happens with $100 bills too.

TADGE DRYJA: Really? OK, well.

AUDIENCE: I had a suitcase full of them in Bogota.

TADGE DRYJA: You had a suitcase full of hundreds, and you just walk to the Bank of America, and say, here, I want to deposit this, probably you've got some problems.

AUDIENCE: [INAUDIBLE] 1% into the banking system, to move it from fiat to digital, digital fiat.

TADGE DRYJA: Yeah, like actual physical currency notes.

AUDIENCE: That's not money laundering.

TADGE DRYJA: So, yeah, so now you've got the sort of money laundering that now people are going to try to do using bitcoin. It would be preferable if-- so the way that they can tell about this, is that people reuse addresses. Bitcoin is not inherently as fungible as, say, gold.

Dollar bills have these serial numbers. And I don't know to what extent people track these things. But bitcoins are much more like the dollar bills with serial numbers. And it's all on the internet. It's all on the computer. So it's really easy to track things this way.

AUDIENCE: [INAUDIBLE] could just simply transfer them to a new address, and no one would know it was still him until it was spent.

TADGE DRYJA: Well, yes and no. So you say, OK, I'm at the exchange. They've got a giant pool of addresses. I transfer to the casino.

And then the casino, in many of these cases, reuses addresses extensively. So it's really easy to see that this is the casino. And then from the casino, goes to the exchange again.

Exchange flags that and says, hey, we know where this came from.

Instead, you say, OK, I go to User A. And then I go to the exchange. In some cases, that might-- if their algorithm is, hey, just look at where it came from, sure. But if you say, wait, there was one output. This transaction is one input, one output.

So we know exactly where it came from. You can trace it back that way. So it depends.

Would the coins be worth less? This is un-money like.

And since legal tender is a whole other thing. And I'm pretty sure bitcoin will never be legal tender anywhere, in terms of debts can be settled. But at least fungibility, we want.

So we want to make bitcoin more money like, how do we fix this? The first, and simplest, and possibly the biggest is, address reuse. So if the casino keeps using the same address over and over, it's really obvious.

And so things like vanity addresses, so a lot of them will use vanity addresses. A vanity address is when you continually perform computations to try to get a human readable address. The addresses are random numbers.

So for example, I ain't rich. So Greg Maxwell has this address, which is 1gMaxwellbo8. So you can see that the first part of that addresses is Jim Maxwell.

And you get that just by continually attempting millions and millions, or billions, of different keys, and seeing which turn into the address you want. So people do that. People spend a lot of resources on making cool addresses.

And then the casino can say, OK, this is my address. But that really hurts the fungibility and privacy, because now it's clear when a customer sends to that address, everyone in the world can see. Yeah?

AUDIENCE: Does the casino just do this, because it's cool? Because it's not like it's easier to type in, because--
TADGE DRYJA: You still have the stuff. It's branding, vanity address, it cool I guess.

But, yeah, Satoshi Dice does this too, right? Or did.

So the Satoshi Dice address is-- yeah, it always starts with one dice. So branding, I guess, makes it easier for-- it does make it easier for people to recognize. If they have a list of the addresses, they're like, oh, that's the casino address. I want to deposit to the casino, and just copy and paste that in there.

So it does help usability. And that's one of the issues in bitcoin. It's like, having these giant, ugly addresses that you have to get right, and not sent to the wrong, it's not a great user experience kind of thing. So I can see that

vanity addresses do improve the user experience a little bit.

But it really hurts the privacy of the system. And address reuse is a problem, because people keep using it. It looks cool. Also web explorers, so if you look at blockchain.info. This is advanced mode, enable, disable.

Even with advanced mode, it'll give you a link to the output. So no one is going to click that. Anyway, so the idea is this shows, oh, here's a transaction. It's coming from two addresses, sending to two addresses.

And it even gives you a helpful, hey, this is Greg Maxwell. And I'll give you a link to-- where does this link go? You're being redirected. OK, well, go for it. Gmax-- OK, so it's Greg's account on bitcoin talk. Interesting.

So you guys know enough about the system now, that this is not true. If you click on an address, and now it gives you all the transactions, way too many, that were involved in this.

But if you click on a transaction, it's not spending from addresses and to addresses.

It's spending from transaction outputs, and spending to addresses that can later be consumed.

So the way that web explorers-- and it'll show a balance. If I click on an address, it says, hey, here's the number of transactions involved. Here's the final balance. And you can look at the current balance, and things like that.

But that's not actually how the system works. And it hurts privacy and vulnerability, to have this idea of, oh, this is sort of my account. And it's got a balance that can increase and decrease, because then that implies that I'm going to keep using it. Whereas, really, if they're one-time use, that really makes it harder to trace things.

So another aspect that hurts this, if you want to trace things-- so if it's one input, one output, right here, it's real easy. But even if it's not one input, one output-- so, for example, imagine a transaction where the input has 10 coins in it. There's output a, which is one coin, output b, which is 8.9997 coins.

Which do you think is the change address? They're all different addresses. But OK, whoever is doing this, they're sending one coin. And the remainder, minus the fee, is this.

So it's often pretty clear, even though looking at it, all the addresses are different. OK, are any of these change addresses? Is it just a sending to b and c? Is it a sending to b, and back to a? Often, it's pretty easy to figure out.

And you're guessing. But you can get the guessing pretty good. OK, so what we're going to talk about now-- anonymity sets.

So bitcoin is not actually anonymous. There are sort of identities attached to these things, in that you have addresses. The addresses are not your name.

But you can think of them as a pseudonym. And you can create a bunch of pseudonyms.

But there are these keys. There are these publicly known addresses.

And so we want to expand an anonymity set. And so the idea of an anonymity set is, how many possible different identities could be the owner of these coins? And the idea of expanding your anonymity set-- so even if bitcoin were perfectly anonymous, in terms of the anonymity set was everyone who had Bitcoin, that's still actually not that many people.

So if you see a bitcoin, you're like, well, I don't know who owns it. But I know that the person who owns it, owns bitcoin. And, actually, there's not that many people who own bitcoin. So I've just eliminated 99% of the people, my suspects, just by doing that.

So just having more people using bitcoin makes it more anonymous, in that sense. If it's very niche kind of thing, and then the police say, OK, well whoever did this crime, they're a bitcoin user. Well, now you can find-- there's not that many bitcoin users. So you want to try to increase your anonymity set for a specific transaction.

So the traditional-- I don't want to say traditional-- the way you do this is basically money laundering.

It's a bitcoin mixer. And these mixers still exist. I'm pretty sure, right? They're still around?

They're still around. I don't know why. But a lot of times they use Tor. I mean, I know why. But it's like, there's so many better ways to do this.

So you've got coins at address A. And you say, OK, I'm going to send 10 coins to the mixer, which has address me. And then later, some different, not from that output, but somewhere else in this sort of giant mixer account, four coins get sent to address C, and six coins to address D. So you basically pool all the coins into this mixer, which uses lots of different addresses, and then split it up over time, different amounts.

And it gets really hard to figure out where the coins came from.

So your anonymity set is bigger. The problem is mixers were well. Potential anonymity set is all the other users of the mixer, if it's well designed. The problem, the mixers disappear with everyone's money, very consistently.

The mixers are certainly not publicly regulated companies. I'm pretty sure you couldn't do that. So they're just sort of these anonymous, like, hey, I'm a mixer, bitcoin cloud, or bitcoin fog, or whatever. And a lot of times they're on Tor. So you don't even know where the mixer exists. And you sort of hope for the best and send your money. Yeah?

AUDIENCE: Do they take a transaction fee?

TADGE DRYJA: Yes, of them take fees. The big fee is when they keep all your money and don't give it back. But a lot of times, they will take a small cut. But it's not actually hard.

The cost is pretty minimal. Need some kind of Tor service. And then you just have some software that just runs this, and allows deposits and withdrawals.

The conference in Puerto Rico, Financial Cryptos, there was a talk about a-- what was the word they used? Not traceable, a mixer that you could prove defrauded you. So you could have these proofs that, oh, I can prove that they stole my coins, which is I thought was kind of useless, because the whole idea is they're anonymous.

And maybe you can prove that they ripped you off. But they still ripped you off. OK, so then the better idea than a mixer is I taint rich was a blog post from Greg. That's why I've got this up.

In 2013, and it was kind of fun. Ever since I was a wee lad, I had a dream, a dream of being incorrectly assessed as impossibly rich by a braindead automated analysis. Now, with your help, I can be.

So he wanted to mix inputs from different people within the same transaction. So you could have two different people in the same transaction. And in bitcoin, this is secure, because the signature signs the whole transaction.

So you say, OK, I'm user A. I have my 10 coin input. I'll match up with user B, who's got his two coin input. And we both sign this whole transaction, which sends two coins to C, and 10 coins to D. So now, you can't trace where these coins went right. Right? Well, you sort of can.

But if you just look at the graph of transactions, it's not as obvious. So the first transaction is really fun. I remember seeing this.

So there's three inputs, 40,000 bitcoins, 0.1337 bitcoins, and 1 bitcoin from Greg, and then 40,000, and 31337, and then 0.82. For some reason, it was even more-- yeah, 40,000 bitcoins is pretty good today. It was no less impressive at the time in 2013, even though I guess it was only worth about half a million dollars at that time.

But it was like-- it was pretty cool. And then Greg was like, wow, I've never seen that much money. Yeah, so the guy who posted the 40,000 coins, he's called loaded. And nobody knows who he is. But he shows up from time to time, and has a lot of money.

So, yeah, then Greg made a transaction with Loaded. I've handled pricey assets before, the most I've ever moved on a single key press. So it's pretty cool. This was very manual.

That was on a message board, sort of goofing around. But there's no risk. You're not sending money to someone, and then getting it back. You're just saying, I'll sign off on this transaction. And then transactions are atomic.

You can't say, oh, I'm going to cut off this bottom part, and only have 10 coins going in, and two coins come up. The whole transaction is the thing that gets signed. OK, so what's the problem with this model?

Any way to get a mapping from C and D, to A and B? There's one really obvious one on this screen. It's an x, right? Yeah, well, gee, I think it's A goes to D, and B goes to C, because the amounts are completely different. And, actually, we'll talk about amounts next week.

But how about this? Well, you've got 10 coins coming in, two coins coming in here. And I've got address C, D, E, and F, 1, 7, 1, and 3. Better, maybe? No, nice try. You can still easily.

The 7 goes to the 10, the 7 to 3, the 1 and 1. I don't think there's any way it could be anything else. How about this?

Address C has two coins. Address D has two coins. Address E has eight coins. Well, now, that actually works, right? It's not clear if C is from A or B, same with D.

These two have some anonymity, in that you're not sure which user it's from. E, on the other hand, is obviously from A. But B's coins are now sort of-- the anonymity set has doubled.

You don't know whether it's C or D. B's address is now unclear.

So that's kind of cool. How do we scale this? Well, have more users, and a bigger anonymity set. So one issue with this is, as you scale to more users-- let's say you do 10 different users, where they all put in their inputs.

They all put in their set of outputs. And now, you've got this big transaction, where it's hard to tell what the mapping is. The problem, as you gain numbers of inputs, numbers of users, the users themselves know the mapping, because someone's actually doing the, I put in my thing, you put in yours.

So there's a user that knows the mapping. And they can leak that info. And that hurts the anonymity. So maybe just the transaction graph itself won't tell you, but somebody knows.

And they can reveal that at a later date. So that's not good. So there's a really cool protocol called CoinShuffle, so pre-CoinJoin messaging to shuffle the inputs and outputs.

And this allows you to have 10, 20, 30 different people doing it. And if at least two participants are honest, then the mapping cannot be determined. The way it works-- this is super quick, because I don't have time.

Everyone has their inputs that they want to put into the transaction. And they also have the output addresses that they want as their outputs. So they make a new set of public keys that they're not going to use on bitcoin at all.

They're just making public keys for encryption purposes for this game. And they also tell everyone, here's my inputs. My input is A. My input is B. So the inputs are known to the people participating.

And then the idea is, OK, I know everyone's publicly that they've given. So I encrypt my output. So I've got an input I've told everyone. I've got my output.

And I encrypt that, the address itself and the amount, with everyone's public keys sequentially.

So, for example, I use encryption on key C to encrypt the thing on encryption on key B, encrypt the encryption on key A of my output. And then I hand this to user A.

So this is like onion routing, sort of onion encryption, where you take a plaintext, encrypt it, encrypt it again, encrypt it again. So I receive these encrypted outputs. I shuffle them. I receive one from A, one from B, one from C.

Actually, I receive the whole set. I shuffle the order. And then I use my key to decrypt one layer. It's still going to be encrypted, because I just see, here's a bunch of encrypted data.

I decrypt it. It's still encrypted with the next person's key. And then I hand it to them, and they shuffle, and decrypt. So the final user gets the outputs.

And the final user-- so in this case, user C, can decrypt. And now, he's got my output.

But it's been shuffled around with everyone else's output, at every step of the way.

So they can't tell who's went to whom. And then they have this final transaction, which has all the outputs in some random order, because everyone contributed to randomizing the order of the outputs. So this is really cool.

As long as two parties are honest-- if there's only one honest party, it doesn't work, because then every dishonest party colludes, and says, well, we know everything but the honest party's output. So we can figure out which one it is. But if there's two users actually doing this, then you can't determine the order.

So that's pretty cool. I think it's being used for Join Market. Do they use something like this? I don't know.

So there's cool techniques for this kind of. Thing OK, real world, though, some people use this. Join Market exists. I don't know how popular it is.

Problem, which people use this. Who uses this? It's got a limited anonymity set of the people who really want anonymity, which is not the anonymity set the people who want anonymity want, which is kind of confusing. But the idea is, I don't want to just be in this group of people who want anonymity, because those are the people I don't want to be associated with.

I would rather associate myself with the people who don't particularly want anonymity. This is a big problem. People don't care about privacy. People don't want to do these kinds of things.

And this costs money too, in this case. If you're doing these transactions, this isn't really a payment. This is just superfluous transaction to turn your money around to get some more privacy. This is one of the big issues with, OK, we want anonymity.

But if anonymity is opt in, it's not very useful, because then it's sort of like Tor, where if I'm using Tor, everyone's like, why are you using Tor? And so at this point, encryption is now sort of ubiquitous. And it's like the standard. But yeah?

AUDIENCE: Given that most transactions happen on exchanges, why not just lobby a few exchanges to try and implement this?

TADGE DRYJA: Well, wait, most actual bitcoin transactions are in and out from exchanges?

I don't know-- AUDIENCE: If that's true.

TADGE DRYJA: There's a lot. But I don't know if it's a majority. I think it's a decent percentage. But I think it's less than half. But, yeah, there's a lot.

It'd be awesome. So exchanges are actually uniquely positioned, where something like CoinShuffle, you could really easily do an oblivious withdrawal. Where you say, I want to withdraw my coins. And I'm not going to tell you where to. But I'll give you this encrypted thing that gets shuffled around with different users.

And then at the end, I sign off on it. And the exchange can say, well, we're fulfilling all our customers' withdrawals. But we don't know where the coins are going. You could totally do that. That's not a conversation that happens at any exchange. Yeah?

AUDIENCE: I also think that the bulk of exchanges are now getting caught up in some regulatory, in Japan, soon to be in the US, for know your customer, anti-money laundering. So for the exchange business model, they're going to have to give this Coinbase to the FBI some data.

TADGE DRYJA: Yeah, and whether that data is, here is this user's name, and address, his house home address,

and his social security number, and here's what he bought and sold.

Versus, does that also have, and here's where the addresses where he withdrew his coins to? I don't think the IRs-- the IRS thing, I think it was mostly we want to get people for not reporting their gains.

AUDIENCE: That's the IRS point, but the other part-- TADGE DRYJA: Right, FinCEN.

AUDIENCE: Financial Crimes Unit, FinCEN would want the addresses.

TADGE DRYJA: FinCEN would want to map bitcoin addresses to human names, so they could see who did what. IRS just wants to know who sold and didn't report gains. By the way, this weekend, if you sold any coins last year, you got to tell the IRS. I've actually got to pay a little bit of tax on that.

Yeah, so anonymity is tricky. Like, you could try to go through exchanges, probably not the best. They're not going to do it. It's hard to do even research on this kind of stuff when you're a company.

There's a lot of sort of chilling effects and stuff. That's sort of why I like working at MIT, in that if I want to research crazy anonymity stuff, totally fine. Whereas if I'm working at a VC-funded company in San Francisco, and I'm like, hey, we're making these anonymity protocols. Are you sure you want to do it?

It can be an awkward conversation, in some cases. So people don't care about privacy.

It's sort of an externality. If you say, I'm reusing addresses. But that only hurts my privacy.

That's actually not true. In the simplest sense, you're reducing the anonymity set for everyone else. So if I I say, OK, where are James' coins? Well, I know they're not at 1Gmaxwell909, because that's Greg Maxwell's coin.

So when you use publicly identifiable addresses, and vanity addresses, you're actually reducing the anonymity set for everyone else. It's sort of an externality, where the people who say, I don't care about privacy are not paying the cost of harming the people who do care about privacy. So this is a tricky problem.

And there's no solution. But one cool thing, one way forward, is, well, everyone likes cheaper transactions. Everyone likes saving money.

So can we make it cheaper to improve anonymity? And so privacy and scalability, in some cases, are at odds. So in things like ring signatures, which I don't think I have time to go-- but Monero is a sort of privacy focused currency that uses ring signatures where, where you can point to-- a ring signature is, I can point to two public keys.

And say, OK, there's public key A and public key B. I'm signing message M on one of those two public keys. But

I'm not telling you which. And you can verify that, OK, one of these keys signed. But I don't know which.

And the signer had to have one of the private keys, or both. If you have both keys, you can obviously make a ring signature. It doesn't have to be two. It can 5, 10, whatever. You pick a bunch of public keys, sign with one of them.

So that's pretty cool. That expands the size of signatures, I believe, with like 0 of n.

So if you have 10 keys you're pointing to, and you make a signature on one of them, the signature actually gets bigger.

So for Monero, they use all these different systems. And scalability is one of the biggest problems with there-- I mean, it's a problem with bitcoin as well. It's a problem with everything.

But it's even worse in Monero because of the choices and the different algorithms they use. So in some cases, privacy and scalability are at odds. But in some cases, it works together, where you can say, we just want to have less information about these transactions. So if you have less information to store, there's less information to link the users to their coins.

So an idea here is aggregate signatures. So, currently, when you sign, you have this input.

And you say, OK, here's user's A signature, here's user's B signature. And you're doing this ineffective CoinJoin thing here.

The goal would be we want to aggregate these signatures. So we don't have a signature for this input. We just have a single signature on key C, which is just, somehow, the combination of A and B's signature.

And then you can save space, because there's only one signature that stays the same size, but still prove that both A-- so this is not a ring signature. This is aggregate is A and B both signed, and produced a single signature together. And that can validate the whole thing.

OK, so how can we make this signature? We know pub keys A and B. They're both signing the same message, which is really important. There's other terms for these things.

There's multi-signatures, aggregate signatures, key aggregation. And no one agrees on-- like, I've had discussions with people. And I'm like, wait, that's what you call it? I was calling it something else. And there's not really good terms for this.

But in this case, what we want is we're signing the same message. So that makes it easier.

We're not trying to combine different signatures on different messages from different pub keys and combine it to the same signature. That's even harder, although that is possible in some ways.

But in this case, A and B are signing the same message with their two different keys.

And they need one signature. And the signature is going to be R and S.

So the equation that we had, that we've done a couple of times, the signature is k, some random number, minus the hash of the message, and k times gr, times the private key. And to verify, r minus the hash of the message and r, times the public key. Now, if you share the private keys with each other, it's kind of easy.

Alice just say, here's my key, Bob. And then the Bob can compute everything. You can do that with ECSA. But you really don't want to share private keys, because as soon as you give someone else your private key, maybe they don't sign this aggregate signature of the thing you want to sign, they sign something else, giving them all the money. So you want to be involved in this process.

So the simplest sort of multi-signature system for Schnorr signatures-- first, you want to share an r value. So this r is also going to have to have contributions from both users.

Who comes up with what here? If c is the combination of both Alice and Bob's public key, the message we have already agreed on. But this r value is the question.

So the idea is, Alice comes up with k sub a, computes r sub a, and gives it to Bob.

Bob makes k sub b, computes r sub b, and gives that to Alice. Then both of them can say, OK, well, we know the real r that we're going to use for the signature is the sum of r sub a and r sub b.

Then, they want to compute their own s's. So for Alice, s sub a is going to be equal to k sub a, that only she knows. She shared r sub a, but not k sub a, with Bob. Minus the hash of the message, and this aggregate, this summed r, times her private key.

For Bob, it's s sub b, equals k sub b, minus the hash of the message, and r, times little be. And then they give each other a really-- only one party needs to do this, right? Either Alice gives s sub a to Bob, or Bob gives s sub a to Alice. But they don't have them both do it, because then the final step is you just add the s's.

So this aggregate s is s sub a, plus s sub b, which is k sub a, plus k sub b, minus the hash of m and r, times a, minus the hash of m and r, times b, which is, you can call this k. That's the discrete log of r, because they summed it. And then you can factor out the a and b here.

And so this is the sum of their private keys. This is the sum of their k values. And then this single verification step will work. And that works. Awesome. OK, questions about this?

Basically make sense? There's all sorts of minefield caveat sort of watch out things.

For example, if they learn your k value, they will learn your private key, once they get this. Normally, you make deterministic k values, where you compute k as the hash of your message being signed and your private key. That's dangerous to do in this case, because they might get a k value from you, and then give you a different r value, and get another k value, and find your private key.

So there's all sorts of watch out things. So the idea is, OK, now cool, we've got output, which needs a signature from a. We've got an output which needs a signature from b.

We'll use those as inputs, and just have this sum c signature. And that shows that both parties signed.

So we're good. Now, we have way less data. These signatures are like 65 bytes. So instead of having it twice, you just have one that covers the whole transaction. Great.

And it doesn't have to be two. You can see how this would extend to three, four, or five different people. They would all have to compute their own k value, give it to everyone else, everyone computes this combined r value, and then does their own s's, and then sums them all up. And this would work with any number of participants.

Problem-- you've got a bunch of coins, that 40,000 coins from before. And then one day, you see, wait, I'm user A. User B came up with a user A and B signature and sent it all to address E. I never signed anything. I didn't do this process with him.

How did he steal all my coins? This is bad. So any idea of how you could do this with the equations we?

OK, so what you do is you say, hey, here's key A on the network. And, normally, you see pub key hashes. But a lot of times you see pub keys. And, anyway, you definitely see the pub key once people try to spend it.

So maybe you have to do this quickly. But you can do this real quick. So you say, OK, I'm going to make Q, a random private key, compute q times G to be big Q. And then I'll compute key B, which is q minus A. Then I will send some coins to B.

And now, note that I don't know the private key for B. The private key for B is going to be this little q, minus little a. I don't know little a, so I don't know what little b is.

But that's OK. I don't send too many coins to key b. And, anyway, I'll get them back.

I don't know b. I can't sign with it.

However, I want to spend from b and a together. I don't know a little b. I don't little a.

I don't know the private key for either. But I do know the private key for both, which is sort of confusing.

But the idea is, well, C, is going to be a plus b, which is a, plus q minus a, which is just q. So in this case, I can observe a key, make this rogue key, and then spend from both of them, without knowing the actual key that I wanted to steal from. So this is a huge problem. You can't have this. This would make signature aggregation completely insecure. What are some ideas of how to fix that?

So the interesting thing is in all the literature about multi-signature, this problem sort of still persists. And the new papers about how to use this in bitcoin are the ones that are actually addressing it. So the general idea was, well, make b sign. Make b prove that he actually can sign with key b.

Before you start doing these things, have Alice and Bob talk to each other. And Bob says, OK, here's a signature with key b. And Alice says, OK, here's a signature with key a. And, now you've proven that they actually can sign, and they're not doing these kinds of rogue attacks.

So that's the straight straightforward way to do it, where, OK, if you don't know b and can't sign, we're not going to continue with this process. That's interactive, though.

So make b sign a message before combining keys. Easy, that's what people sort of thought about for years.

But the whole point here is to aggregate signatures. And so if you require b to create a signature and post it into the transaction, you've just eliminated all the gains, all the space saving, for this technique. So we can't have that. We want it to be non-interactive.

We want any existing keys to be usable in this system, without pre-committing to anything.

You can also make it interactive, where-- well, anyway, there's actually a better way to do it. You de-linearize the signatures. So you redefine the signatures.

You still send to, for example, key a or key b. Your outputs are still pub keys. But when you require a signature, you don't require a signature from that pub key. You require a signature from that pub key, times the hash of that pub key.

So you say, OK, I'm sending to a. But when normally I have sig a, signature from key a. Instead, I say, no, I want a signature from a, times the hash of a. So anyone who knows the private key for a will know the private key for this,

because this is public.

Everyone can see what the hash of a is.

And then so it's just little a, times the has of little a. This is a scalar multiplication.

So the scalar multiplication works the same for the-- sorry, this is the point multiplication.

And it works the same for the scalar down here.

So this doesn't hurt. It doesn't make it any harder to sign. You just have to perform a hash operation and a multiplication, which is quick. But what it does is it prevents this kind of attack.

So instead of signing with a plus b, sign with a, times the hash of a, plus b times the hash of b. Since, in this case, in bitcoin, you can see what public keys are being signed and aggregated, so you say, OK, well I'm going to try to do this technique again. c is a, times the hash of a, plus b times the hash of b.

The private key is going to be a, times the hash of the pub key, plus little b times the hash of pub key big B. I know b, which is q minus a-- sorry, I don't know b. I know q. And I constructed sort of this b is q minus a. But I don't know it.

So c is now going to be a, times the hash of a, plus q minus a, times the hash of q minus a. That's what b is. I can't cancel out this little a anymore, because it's got a different coefficient. Maybe it's easier if you sort of move them on the other side.

I've now got these coefficients in front of my private keys that are not the same, and won't cancel out anymore.

Before, the system is times 1. So the idea was a, times 1, plus q minus a, times 1. Since the coefficients are the same, 1, I can factor them out, and now apply these, and remove the a.

So if the coefficients are the same-- if it was a, times the hash of a, plus q minus a times the hash of a, I'm good. I can still factor out that coefficient and cancel out a. But now, the coefficients are different. And so I can't do that subtraction. And now, I'm stuck. I cannot sign with this little c, just from knowing q. Questions about this?

AUDIENCE: So how do come up with q?

TADGE DRYJA: In this attack, q is just any random number. But the idea is here, q being any random number doesn't help you, because you've sent money to b. You're trying to sign with q minus a. But you don't actually know the private key, q minus a. So you're stuck.

Even if you did, in this case, it wouldn't help, because you still have that either.

So this prevents that kind of straightforward attack. You can't get rid of the a times h term.

This actually is not enough. There's a paper called Wagner, Wagner's paper, some guy at California, UC Berkeley, I think, a generalized birthday problem. And it's a hard paper to read. But if you're working on these kinds of things, it's a good paper to read, because it comes up again and again with these kinds of attacks.

Collisions-- I don't think we ever talked about. So the idea of colliding, for example, a hash function, or in this case a public key, usually it takes half the work that you-- the reason it's called a birthday paradox is because-- do people know this birthday thing. Like, how many people need to be in a room before two people have the same birthday?

And it's like 22 or something.

It's really low. And it's surprising, because you think, well, maybe 365 divided by 2, so like 180 or something. But it's actually 22, I think, because of the way like for every new person added, there's a possibility they collide with every already existing person's birthday.

And so the collisions are more common than you think. So in the case of colliding a hash function, even if the hash function is 2 to the 256 bits, you only need 2 of the 128 attempts to collide and find two that are the same. And that's if you store all the old hashes.

And so keep coming up with new ones after the square root of the number of attempts, you'll find it. And then there's some techniques with cycle finding, where you don't have to store them all, things like that. So usually, you think, OK, 2 to the n over 2 time to find a collision. But that's a collision between two things.

And Wagner's attack is sort of, yeah, find a and b, such that a equals b, kind of the normal collision. Now, find a0, a1, up to ai. So find i things on this side, and j things on this side, such that the sum of a equals the sum of b. And in the case of elliptic curve points, it would actually be the sum. You'd do addition.

In the case of hashes, it might be xor, or some other operation. But the idea is if you have a lot of different things that you can pick and choose on both sides, you can potentially find collisions with much less time. And in the case of these elliptic curve points, it probably would be practical. You'd need dozens, 20, 30, 40, different keys.

But you could potentially say, OK, there's a key with a lot of coins. I'm going to find a set of a whole bunch of keys, such that that I can cancel that key out, even with this multiplying by a different coefficient, because I've got so much search space. It's a cool paper. You should look through it, if you're interested.

But that's a problem. You can sort of make progress, is the basic idea. I can keep getting closer and closer to canceling this thing out.

So instead, we have this improved delinearization. And this is talked about in the paper from like January. You take the hash of all the keys concatenated together, in some specified order. So you sort the keys some way. And you say OK, z is the hash of a and b, or if there's more keys a, and b, and c, just all concatenated.

And then you sign with a, times the hash of z, and a 0, plus b times the hash of z, and a 1. So you make the coefficients distinct. But you also make the coefficients commit to every single part of the set. Every key in the set needs to be in this coefficient.

And the MuSig sort of explains why that works. So the attack in this case is, well, I've got these different coefficients. But the coefficient here only depends on this key here.

So maybe I can find a bunch of different keys with different coefficients, such that these coefficients will cancel out, or these coefficients will sum, or multiply, or sum up to hash of a. So it seems hard. I can't find different hashes that will equal the hash of a. That's a hash collision.

But if I have any number of different things, I may be able to. And so the idea is, if you do this, now every time I'm adding or removing a key from my attempt, I have to change z.

So I can't leave this here, and start adding things, and try to work on it that way, because every time I add a c or d, z changes.

And so I have to start over from scratch. So anyway, that's my general intuition of how the delinearization works for MuSig. And then, yeah, it used to be like z, and then put a in here, because now that makes it unique. But you can actually just number them.

You just need them to be distinct. And since it's hash, function just say like 0, 1, 2, 3, 4, is good enough. OK, any questions about this?

I realize it's kind of complicated, and the software is kind of crazy. But the idea is that it prevents a bunch of these attacks so that you can securely use aggregate signatures.

Questions?

And you wouldn't have to program it. Yeah so you could say, OK, it saves face. The first use case will be in single wallets. It's interactive. So you have to do that thing where I come up with different k values, I share the different r values. I come up with the s values. I add those up, sure.

If it's all doing it yourself, where it's just there happened to be two different pub keys that you control both of, you can basically skip all of that. And you can just say, no, I'm just going to add the two points here. And I'll use the same k value. That's just me.

So this happens a lot in bitcoin wallets, where you're making a transaction with multiple inputs. They're all controlled by you. But you might have two coins here, and three coins there, and you want to send someone four coins. So you've got to use both of these inputs.

Currently, in bitcoin you're going to have multiple signatures. With this new signature aggregation, you can say, OK, I'm picking these two inputs. I'm signing for all of them with a single signature. And in the case of it all being years, you don't even have to do the two operations and add them, because you know the all the keys. So that's going to be the first use case.

And that's much easier to program, because it's all just within the same computer. And this is easy. That's cool.

And then a cooler use is, OK, use it with CoinJoin, where users A, B, C, and D all contribute.

They all have an input with the same amount of coins, great. And they're doing CoinShuffle to shuffle the order of these outputs.

So A knows, oh, mine is G. But I have no idea about E, F, and H. I don't know the mapping other than my own. But, hey, my output is in there. So whatever everyone else is doing, I don't care.

But I'm good with this transaction. And they can contribute their own k value on their own s value. And then, finally, you put the signature at the bottom. And it verifies that all these participants have signed.

And this is really nice because it's smaller. This would be 250 bytes worth of signature data. Now, it's just 65 bytes sorts of signature data at the bottom. So you save on fees. You save on space.

Everyone likes this. Yeah, it's cheaper than a solo transaction. And so that potentially helps scalability and privacy, in that if people say, I want to do something cheaply, I will connect to other people and aggregate my signature with theirs.

And so we'd save some space and save some money. And maybe I don't really care about the anonymity thing. Maybe I'm entering into a transaction, and there's some people doing shady deals in this transaction. But I don't care.

My thing gets in and out. And I save money. So the extension of this would be, what if you had just one giant

transaction in every block, which is all the inputs and all the outputs, and there's a single signature? That would be really cool.

There are techniques to do that. Maybe next time. So the issue is it's interactive, in that you have to know about all the other inputs and all the other outputs.

So if it were non-interactive, then I could say, oh, well there's this transaction where D is sending coins to H. And there's also this other transaction where C sounds going to G. And they've got their own signatures.

I'm not either of these people. I didn't do anything with these signatures. But if I could take those two things and combine the signature non-interactively, then I could just basically take all the transactions in my meme pool, squish them into one transaction with one signature, and put that in my block. That would be really cool.

But the issue there is, one, the signatures have to be interactive. And, two, they're on different messages. So in the case of C sending to G, it's a message of C to G. And then D sending to H, It's on that message.

There are techniques, though. So BLS signatures do allow you to do this, so non-interactive aggregation of signatures. Which would be really cool, because then a block would only need one signature. And then BLS signatures themselves are a single point, not a point in a scalar. So it would just be 32 bytes.

Well, you'd probably use a different curve. You'd have to. But, yeah, the idea of having these tiny signatures that's satisfy proof for the entire set is really cool.

So this is a cool idea. So you saying there isn't software? Or it's not on GitHub?

OK, so the idea is, there is a bunch of software for this that mostly SIPPA-- or Peter Wool, his name on the internet is SIPPA. And he told me what that stood for. And it was some like super cheesy thing he made when he was in middle school.

Anyway, he has been working on this kind of software. So has Greg, that guy who five years ago wrote about this stuff. A couple of people have been working on this.

It's not publicly available code, which is kind of weird. But the reason is that they're working on Schnorr signatures and aggregation. And it was on GitHub in like a branch of lib sec p, the repo they're using. And then all these altcoins were taking the code, and putting it into their altcoins, and saying, hey, we support Schnorr signatures.

So for example the rogue key attack, it was the simple version, which didn't have this multiplied by the hash of the pub key. And they were like, wait, no, don't, we're working on this software. But we know all of these problems with it.

You can't actually use this for signature aggregation. We're just sort of playing around, and trying different things. And so they actually pulled it from GitHub.

And so I have seen it. You can see the old stuff. Yeah, the three years ago stuff that doesn't have any of this delinearization, or this thing that prevent Wagner's attack, all of that stuff is not in there.

AUDIENCE: That's something that sounds really cool. But that's something [INAUDIBLE].

TADGE DRYJA: No that's not what they-- OK, maybe that's a cool attack. But they were like, oh, shoot, no. But it said in their repo, do not use this. This is research code.

Do not use this in production. There are known vulnerabilities.

But people are like, whatever, people didn't care. Well, yeah, there's some esoteric edge case vulnerability, but I'm going to solve that, because xyz. So they actually pulled the code.

They are working on the code. I mean, the code works. I've seen it. But they are very sort of like, we don't want to put it publicly until we're sort of saying, OK, here's our final version that we want to put in bitcoin, because they know as soon as they sort of put it out there for review, all these altcoins will take-- I mean, you've implemented this in-- yeah. Not in vertcoin, in crypto kernel. And you do use the u sig?

AUDIENCE: I will do now.

TADGE DRYJA: Oh, OK, cool, let me know. So they know that people are going to use this, because it's a cool signature scheme. And I've talked about this multiple times before.

Once you have this Schnorr signature equation, there's all these other cool things you can do with your addresses and things like that.

And so the idea here is scalability is one that, OK, now we can combine things, make it smaller. And also, we can hopefully help privacy this way, in that people who are not particularly concerned with their anonymity will say, yeah, I still want to use it, because I can save $1 on fees. So, yeah, I'll join this transaction with a bunch of other people.

And then you've got the people who actually want anonymity are like, great, these are exactly the people I want to be associated with, the people who don't particularly care about anonymity. So it seems like a win-win.

Right now, all the software requires a regular old signature. So, for example, if these are outputs that are currently being used, and you do this, I'd validate the output one at a time.

And I say, OK, I'm looking at this output, looking at this input, there's no signature at all, fail. OK, the whole transaction fails. So you need a new output type, which says, OK, I'm using this new aggregate signature. So, actually, look at all the inputs. Don't validate them at all, and just validate at the end at the bottom.

So that's going to be some different code. And you can't retroactively say, OK, all the existing outputs we can now spend with this new signature scheme. You can't really do that. So we'll have to make new addresses.

And the new addresses will be bc1v, instead of q, new address type. And then we send to that. And now, you're allowed to spend from it using aggregate signatures. So all that software is mostly there.

But it might take a while. You've got to get everyone on board. I'm curious to see what people will complain about. Segwit was a little bit more like, there's a lot to complain about.

Even I don't like parts of Segwit. There's a lot of parts I don't like and won't use.

But this seems like a sort of clear win, where it's like, great, you can aggregate signatures, do all these cool things. I'm guessing there will be people saying, this is bad. And saying, no, because Satoshi's vision is one signature, one input. A coin is a chain of signatures.

And this coin has no signatures.

I'm guessing. I don't know what'll happen. Or maybe it might be smooth, and everyone is sort of like, yeah, this is obviously better, great paper, great math, great idea. Let's all activate it and use bitcoin this way. Who knows?

So we'll see. It's pretty cool software, though. And you can use it in other projects. So Crypto Kernel is using Schnorr signatures, probably a bunch of different coins who are sort of more agile, in that it's easier to make changes, will probably start using this.

Ethereum is still ECDSA though, right? There's no coins that are actually implementing this yet. Monera uses ED25509, which is essentially a Schnorr signature. And I think Ripple can use ED25509. So there's some different schemes that are more similar to this, but still don't have the definitions for aggregation.

So, anyway, so this is cool stuff. Next time, I will talk about-- next one, I'm not looking forward to making slides, because it's like a really complicated one that I don't even understand. Like, I sort of get it, but it's like, wait, how? OK, but how to mix different amounts.

In this case, hey, it's great. But in many cases, you can see, I said the reason it doesn't work is because it's really obvious. But what if you could hide the amounts, but still enforce that the amounts were correct?

So that's the idea of confidential transactions, which I'll talk about next time. And then it leads to MimbleWimble, which is even more complicated. I'm not going to promise that I'll explain the whole thing. But I'll touch on the ideas.