MITOCW | 23. New Directions in Crypto

TADGE DRYJA: So today will be some sort of future technologies, future developments that might not be around yet, but are interesting things to look for.

And if you're interested in researching these kinds of things, you want to make a EmenG out of this or want to make a-- who knows?

PhD probably is a little overkill for a lot of these things.

But if you're interested in this kind of research, here's some things we'll talk about today-- block slash block filters or committed bloom filters, sharding, accumulators, and UTXO commitments.

OK, so, first one I'll talk about is block filters.

So I don't think I ever really talked about what a Bloom filter is.

I'm still not really going to explain how they work, but the basic idea, the sort of high level-- here's the prototype.

Here's the function prototype.

Here's the interface that Bloom filters have.

So you make a filter from a bunch of objects.

And in this case, objects are just bytes, right?

Or some string of bytes.

And Bloom filters usually use hash functions under the hood.

A lot of times, they don't use cryptographic hash functions.

They can use sort of faster functions, where there may be collisions.

But in this case, it's not like a security problem.

So you've got a bunch of objects.

They might be addresses, which are 20 byte pubkey hashes, or your UTXOs, which you can represent as a 36 byte TIXD and out point, or TIXD and index.

So they're small, right?

So either 20 bytes, 36 bytes, sometimes 32 bytes.

You put these list of data objects and make a filter, and you get a filter out.

And the filters are usually a kilobyte sometimes.

But the idea is it's sort of a mix of hash functions.

And then you say, OK, I want to match filters.

Usually, a different person does this and says, OK, I've got a filter that someone generated.

And I compare it against this object and see if there's a hit, all right?

So I see, OK, that matched.

This object matched the filter, or this object did not match the filter.

And this returns a true or false.

And so what's interesting about these is you can have false positives.

So it may be that this object was not in here.

It was not used when creating the filter, but it still returns true.

So it's sort of matching against multiple different hash functions, seeing, hey, do any of these bits match?

And it says, oh, yeah, this object may have been in this filter.

However, there's no false negatives.

So if you did put, say, address A into this filter and then matched this filter against address A, it would always return true.

There's no way you can put something into the filter, and then it doesn't show up when you try to match against it.

So this is useful for lots of things.

Bloom filters are definitely not restricted to cryptocurrency, Bitcoin, or anything like that.

They're used all the time in various databases, all the time in things like that.

The current way they're used in Bitcoin is for SPV filtering.

So we defined SPV, like, months ago.

The basic idea is I'm a client.

I don't want to download and verify the whole blockchain.

I want someone else to do that.

I assume the miners are doing the right thing.

I assume the rest of the network is doing the right thing.

And I just want to know about my data.

So I'm not going to download the whole block.

I'm not going to verify all the signatures or keep a UTXO set on my own.

I'm just concerned with my UTXOs in my wallet.

So what I do-- and this exists today.

You can do this.

You make a Bloom filter of all your UTXOs and addresses.

So you say, OK, here's all my addresses that I'm hoping to receive money on.

I've got 20 of them, 30 of them, 100 of them-- however many-- although Bloom filters don't really work once you have too many.

But so the idea is if you have five addresses, let's say, you start a wallet.

You don't have any money.

You're pretty sure you don't know of any money yet existing.

But you say, OK, I made five addresses.

I told these addresses to people.

They might have sent me money.

That'd be nice.

So you make a Bloom filter of all these addresses, right?

So let's say you've got, OK, address A-- dot dot dot-- A through E. And you make a filter.

OK, so you say I've got my filter F. I then send that filter F to a remote server.

So there's the cloud.

There's some Bitcoin full node out here.

It receives filter F. OK, so this got filter F. And it knows that this filter F is specific to me, right?

So it sees that, hey, I'm a client.

I'm an SPV client.

I connect to a full node, and I say, hey, I sent a message that's called load filter.

And I say, hey, load filter, here's my filter F. Only send stuff to me that matches this filter.

99% of all the whole thing-- everything going on in Bitcoin, I don't care about.

Here's a filter, and only send me messages that match this filter.

And so when it sends a transact-- well, they send INV messages, right?

When it sends an INV message with inventory, saying, hey, I found this thing that you might be interested in, normally, a full node to another full node-- so let's say there's communication between two full nodes, Full2.

These guys will talk to each other about every transaction they see, right?

So if they see a transaction, it's valid.

There's nothing wrong with it.

They'll just send it to each other to propagate transaction throughout the network, so they can get mined later.

However, if a filter has been loaded, it says, oh, OK, I will only send you INV messages that match this filter.

So if there's a transaction that doesn't have any of these five addresses as an output, they're just not going to send it to you.

Similarly, when a block comes out-- this is the big one.

Normally, blocks get propagated the same.

They just send you the block.

Here, we do something called a Merkle block.

I don't send you a regular block.

I filter everything within the block and send you only the things that match to that filter.

So generally, it gets very small.

So the Merkle block might just have one transaction in it.

And it has the sort of Merkle proof up to the root.

So the server sends only the matching transactions in the block, which can drop from a megabyte to less than a kilobyte.

And then the client says, oh, cool.

There's a transaction where I received money.

Great.

Also, let me update my filter to include this new transaction that I received.

So if that gets spent later, I want to know about it, even if it doesn't send to one of these five addresses, right?

So if you're only matching on addresses, you can only sort of get money.

But if you're matching on these UTXOs, you can lose the money as well.

And you don't want to lose money, but you sort of want to know when everyone else thinks you lost money.

So this works today.

This was implemented 2012-ish.

The history behind it was the first Android Bitcoin wallet.

Andreas Schildbach wrote it, and it didn't do this, right?

It just downloaded the whole block and then threw away most of the data and only kept-- it wasn't a full node, and then it didn't keep a UTXO set.

But it did download everything.

And then they were saying, OK, this is really slow.

We want a decentralized way to do this kind of thing, where instead of just connecting to a server-- so the other model I explained, again, weeks ago was, you just have some server.

And you tell it the address and say, hey, here's my address.

How much money do I have?

And it sends you transactions, and you maintain your wallet that way.

This is nicer because it's decentralized, right?

Every full node can do this.

And by default, if you download Bitcoin 0.16 or whatever recent versions-- not even recent-- since like 0.7?

I don't know.

Most of the versions will have this capability, where if a client says, hey, here's a Bloom filter.

Load it.

Your full node will load that filter.

And then every block that comes in or every block that's requested, they will match against the filter.

The filter match function call's not too heavy.

It involves a bunch of hash functions.

It's not too slow, but it's slower than doing nothing, right?

It's slower than just sending it directly.

OK, so this is nice, right?

You can sync the entire chain in way less data and having SPV security.

Problems-- it's really bad for privacy.

You're sending a Bloom filter, right?

So it's this thing that's created from your list of addresses.

But in practice, it's got about the same security as just telling them all your addresses, right?

So it's sort of like, oh, I'm sending a hash of my address, instead of my address.

Well, yeah, but I know all the addresses in existence on the Bitcoin network.

I can just try to match it, try to hash and stuff like that.

When you do Bloom filters, there's this sort of false positive rate that's sort of a knob you can twist.

And you can say, oh, I'm going to make a Bloom filter where 10% of the time, when you perform this match filter for any given object, I'm going to create a filter where 10% of the time, it'll just return true.

So I can dial in a false positive rate.

So I can say, OK, I'll make it 1%.

And then when I get these matching transactions from the full node, yeah, I'll get an extra few transactions that don't match my filter.

Or they match my filter, but they don't actually match anything I'm looking at.

And that will improve my privacy, right?

Because then the full node doesn't see what's-- he doesn't know what's truly mine.

In fact, you don't even know what the false positive rate is.

When you receive a filter, you don't know what the false positive rate is.

You just see, OK, these things match, and you send them.

Another strategy is OK, I'm a SPV client.

I connect to a bunch of full nodes, and I can give different filters to each one.

I think the initial software did this.

And because you can sort of put some randomness into your different filters to hope-- why did they do this?

It actually makes it worse.

It actually makes it worse because if these full nodes collude-- collaborate-- whatever.

If these full nodes share the information of the filters, it makes it easier to determine, to sort of filter out the false positives.

Because they'll have different false positives because they had a different filter.

And so if they collaborate, if they work together, they can say, oh, well, I got some false positive transactions.

I did, too.

We can filter out the ones that one of us had as a false positive and not the other, right?

So we can detect the false positives we're sending to the client.

So privacy is really bad.

There's a paper written I think 2013 or 2014, where they basically broke the whole privacy argument for this Bloom filter based SPV.

And they said in practice, you can get like 90 something percent of the addresses and UTXOs that people are sending for the software.

And their recommendations were like, yeah, I don't see how you make this work privately.

There's just no privacy here.

It's slow for the servers, so when you're running-- I don't know if you can see.

When you're running a full node-- OK, I should close that.

Don't even know what that is.

When you're running a full node, you can see here's all the nodes that are connected to my full node and running downstairs.

Most of them are other full nodes.

Oh, OK, these are not.

I don't know what those are.

Those are not true.

All these 0.9, 0.99s, they're not actually nodes at all.

They don't seem to ask for anything.

And then some people put their version message.

They put an address.

Hopefully, someone will send them lots of money.

It's not going to happen.

That's weird.

There's like no SPV-- well, OK, bitcoinj.

So that's a Java implementation, which does filter load.

And so we can look-- hm.

A fee filter, but no filter load.

Well, that's weird.

Someone's sending me lots of different fee filter messages.

There's lots of weird stuff going on in the Bitcoin network.

But I weirdly don't have any SPV filter load things going on right now, which is unusual.

I don't know.

It changes a lot, too, based on-- so also, you can sort of track your hourly data usage.

And this server basically is only for-- all this traffic is the clients connected.

You know, Bitcoin.

And so I'm doing, like, a gigabyte every hour, which is a lot for a home connection, but not too bad for here.

But in December, everyone was interested in Bitcoin.

And so you had lots of people downloading it, running it.

And it would be something like 10 times this, where just tons of people were installing it, downloading it, getting a whole blockchain, and then probably after losing interest, deleting it, but whatever.

And a lot of SPV clients doing filter loads can slow down your server.

It can take CPU time.

Right now, yeah, OK, so 6%, 2%.

It's pretty low CPU usage generally for Bitcoin, even with that many-- however many that was-- 30 or 40 different other nodes connecting and downloading stuff.

Basically, this is like receiving transactions, verifying the signatures, and sending them out.

And granted-- but this is per core, right?

So if I have 2%, that's 2% of a single core.

It's really not much.

And then I guess Cryptokernel's only using 1%, so even less.

So yeah, it's not much, but when you have a lot of SPV clients, it can start using a lot of CPU.

OK, so how do we improve this?

This is a new-ish idea.

It's actually about two years old.

And it was kind of interesting.

It was just a random anonymous internet person posted on the mailing list.

I think his email address was some inappropriate swear word or something.

Anyway.

But whoever this person was just said, hey, why don't we do it the other way?

Why don't we do it backwards?

And instead of having the client create a Bloom filter and send it to the full node, have the full nodes make Bloom filters from all the transactions within a block.

And then the client will just ask for that filter.

The client can then perform the filter match function on their own UTXO set.

And then if they do find a match, they request the entire block.

Right?

So this is a different model.

I don't know.

Does it get the idea where, OK, so you're a client.

All you do is request filters.

So you say, filter please.

So you have some kind of filter request.

And then the full node just says, OK, for every block in the blockchain, I'm going to create a filter, right?

I take all the objects in the block, which are basically all the addresses used in every transaction, all the UTXOs spent in every transaction.

I concatenate that, so there's going to be 5,000, 10,000-- a lot of these objects.

Put it into a really big Bloom filter, generally bigger than the ones used in this method.

Because usually, a wallet won't have thousands of addresses or thousands of UTXOs.

It's possible, but in this model, usually, you've got 20, 30, maybe 100.

But in this case, you're going to have thousands.

Make a larger filter, create the filter, and store it for each block.

So maybe it's 20 kilobytes or something.

And maybe in this case, they're only like 1 kilobyte.

So you have a filter.

And then the node will request these filters for every block.

So it's OK, this block, get the filter.

Get the filter.

And then perform the matching on their own, right?

So they've got the filter.

They see, hey, does this filter match any of my addresses?

So is there anything in this block that may have paid me or anything in this block where my transactions may have been spent?

And if they get a true, they just request the whole block.

They just download the whole 1 megabyte block or whatever it is.

And there may be false positives, right?

So they might be downloading the block for no reason.

They download the whole block, see there's nothing of their address.

Yeah.

AUDIENCE: How does that work if we increase the block size to a ridiculous amount?

[INAUDIBLE] TADGE DRYJA: Wait, ridiculous amount block size?

Yeah, I guess.

AUDIENCE: [INAUDIBLE] TADGE DRYJA: Well, I mean, they're not going to actually-- I don't think any of these have actually 32 megabyte blocks and that no one's using them.

Even Bitcoin now, right, where the actual block usage has gone down substantially.

You've got like-- I don't know-- 500k or something average now?

It's low.

Occasionally, you have lots of little transactions saturating the mempool and then full blocks for a few hours.

But it's gone down, and it's not even at full usage.

So and Bitcoin cash, yeah, you've got 8 megabyte max size, but-- AUDIENCE: [INAUDIBLE] TADGE DRYJA: Right, right.

And they're going to do 32 megs full sized.

But still, the actual blocks are like 10k, 20k, whatever.

I mean, yeah, but if you did have actually 32 megabyte blocks, a false positive would be a big problem for a light node.

Because now you have to download this 32 megabyte block, look through the whole thing.

Actually, there was nothing of interest.

It was a false positive.

OK, try again.

But yeah, you can download 32 megs.

It's not the end of the world.

And if you're only downloading one out of every 100 maybe, if you have a 1% false positive rate, it's not too bad.

That's like 32 megs a day or so.

But anyway this model is-- and not only that, you can request all the filters, match them, and then download from someone else, the full block, right?

You can request a block, download it from someone else.

So this full node, they know you requested all the filters.

They don't see anything else other than that.

And then another full node just sees you requesting blocks and thinks nothing of it.

Because that's totally normal.

This is a lot nicer model for privacy because the full nodes don't learn anything other.

At most, they learn you downloaded this block and not this other block.

So maybe you have transactions in this block, but not this one.

But that's a much bigger sort of needle in a haystack problem, where OK, here are the blocks they used.

What are the commonalities between these sets of blocks they were downloading?

Possible to maybe weed things out, right?

If the blocks are small, and there's only a few transactions, and they download the entire blockchain from a single node, and that node can track, OK, which blocks are being downloaded?

What are the common transactions or addresses in these?

It's possible.

But it's a lot better for privacy than the current model and a lot better for CPU, right?

So privacy-- great.

Great improvement.

The server has much lower CPU, because it can pre-compute all the filters for every block, right?

So as soon as it downloads a block-- or a few seconds later-- there's no rush.

Compute a Bloom filter for it, store it on disk-- because it doesn't change-- and then when anyone requests it, you've already got it on disk.

So you just read it off the disk, send it over the network, you're done.

The current model where you don't write these to disk because they're sort of client specific, so client connects in, sends you a Bloom filter, you have to keep that in RAM, and then match all the things against this specific filter for this specific user.

Whereas in this model, you make a filter for the block, save it, you're good.

Yes.

AUDIENCE: Is the filter for the block [INAUDIBLE] given you're just sending the addresses and not [INAUDIBLE]?

Or is it just less data?

It's just not the full block, right?

It's just the-- TADGE DRYJA: Yeah.

So if it were the addresses and the UTXOs, then it would be really big.

It would be-- oh, maybe like 40% of the whole block size.

So the idea of the Bloom filter is you squish it down.

So the Boom filter itself might only be like 20k.

And so yeah, the basic way a Bloom filter will work is you take sort of a bunch of hashes and populate a bit field with them.

So the thing is, if you keep adding objects to the filter, the filter will eventually just be like FFFF.

And everything will match it.

So you need to sort of decide how big the filter should be when you start creating and adding objects to it.

So with this, you can get it down to about 20k, and then it'll have a pretty low false positive rate, but not tell you exactly what the addresses and UTXOs were.

So it's a nice trade-off to have.

Yeah, so I mean, the perfect sort of easiest Bloom filter would be, here's a list of all the addresses and all the UTXOs-- basically a block minus the signatures, which is sort of what you can get with segwit, where you say, hey.

In segwit, you can say, hey, give me the block without all the segwit data.

Because I'm just looking for things.

I don't want to actually validate the signatures.

So if you did that with regular-- yeah, it drops it by about 50%.

But the Bloom filter drops it substantially more.

So if you did this, it'd be better-- lower CPU for the server.

It's also harder to lie and omit things.

So in the current SPV model, if someone is running a client-- it says, hey, here's my Bloom filter.

And the full node responds.

The full node can easily just omit things.

So there was a transaction that did hit the Bloom filter and did match.

And address A was present in a transaction, and the full node just doesn't send it.

There's really nothing the client can do to detect that kind of thing, which in general, it's not the end of the world.

In normal Bitcoin usage, if you don't hear about a transaction, maybe you'll hear about it eventually.

The worst they can do is sort of lie about you-- they say you didn't get paid, but you actually did.

Not the worst thing in the world, although in Lightning Network, that can change a little bit in that you want to know about the transactions that potentially close the channel and immediately respond to them.

So that's more of a security problem with Lightning.

So with this, it could be harder to omit things, especially if you commit the Bloom filter into the Coinbase transaction.

So if this committed filter becomes like a consensus rule, and you say, OK, everyone makes a 20 kilobyte Bloom filter.

Everyone takes the hash of that and puts it into an op return in the Coinbase transaction the way they do a segwit.

Then it's a consensus rule.

Then it becomes essentially impossible for the full nodes to lie or omit anything.

Because you can say, hey, I've got the headers.

Give me the Coinbase transaction and a Merkle proof for it.

And now I've got the Coinbase transaction.

And now, hey, give me that filter that matches this committed hash in the Coinbase transaction.

So they would have to do valid proof of work to lie or omit.

Whereas now, they can just easily omit anything they want.

Oh, lying also.

So it becomes harder to lie because it's in a block, right?

So this is operating on the block level.

The current SPV does not operate on the block level.

So you can get unconfirmed transactions over the wire that match your filter.

I think this is a really bad idea.

I'm not 100% sure why they put it in.

But there's still people who like it.

But the whole idea of SPV is that you're verifying the proof of work, right?

You're verifying that the miners validated this.

And you think, well, the incentives are such that miners don't want to mine invalid things.

They won't get paid.

So if it's in a block, I'll accept it as OK.

For mempool transactions, if it's just an inv message for a transaction that's not in a block yet, there's no SPV security at all.

And it's trivial to send an invalid transaction to an SPV client, this currently.

So if you say, hey, here's-- and it's not only is it trivial, but you can also try to figure out what their addresses are, and lie to them, and say that, hey, you just got thousands of coins.

So that's the current problem with SPV usage.

You're basically telling the full load your addresses, and you're accepting transactions without proof of work.

So the full load can say, hey, here's a transaction that sends you 5,000 coins to an address that I think you have.

Because I tried to figure out your address from your filter.

And it's got an input that doesn't actually exist, right?

So I'm just saying I'm spending 5,000 coins from here.

And the from part isn't actually a thing.

But since you're an SPV node, you don't know that.

So there's like a-- let's see.

Lie to SPV is like a branch on Bitcoin [INAUDIBLE].

OK, yeah.

So there's a branch that Peter Todd made called Lie to SPV.

I don't know.

Quick and dirty hack to lie to SPV wallets.

They can't verify amounts, so yeah.

So you can just sort of detect-- well, if you want to look at it.

You can detect their addresses.

And then I think that one just sort of opportunistically, if it finds a match, just multiplies the amount that they're receiving by, like, 100.

And there's no way that they can validate that since it's in mempool.

So this also makes it on the block level, which is what SPV really should be.

So that's nice.

Downsides-- mainly that it's going to be higher network traffic for the client, right?

So even at low false positive rates, the entire rest of the block is essentially a false positive, right?

So they're like, hey, there's one transaction in this block I want to get.

I have to download the whole block to get it.

So yeah, more network traffic for the client, which is a downside, but it helps with privacy.

So there is current development.

Lighting Labs-- basically [INAUDIBLE] working on-- it's called neutrino.

It's a variant of this.

So, and then hopefully, something like this will eventually get into Bitcoin Core itself and replace the current server side Bloom filter code.

Hopefully, but something to work on.

Any questions about the Bloom filter stuff?

Cool, OK.

OK, other issue, sharding.

So this is mainly being worked on in the context of Ethereum.

And it's sort of their sort of holy grail of scalability.

It's common in the database world, where you've got d data objects, n servers.

So in the case of Bitcoin or these blockchains, you just store d times n, right?

Every node stores all data.

Instead, store something closer to d itself and shard the data over all the servers, so that each server holds like d divided by n, right?

So if you have 10 servers and a gigabyte, have them each store 100 megs.

And then you still got all the data.

Of course, if they actually store exactly d over n-- and you need to coordinate it so they all store their own little shard-- and then if any single node goes down, well, you're stuck.

So this is sort of the limit.

And there's no redundancy there.

But you can have different redundancy ratings.

So you could say, OK, well, any-- you could have [INAUDIBLE] you're coding.

So if any five nodes or any 20% of the nodes disappear, we're still OK.

So in the database world, this is a well studied problem.

But in the context of Bitcoin, Ethereum, and blockchains, it's more difficult.

It's difficult here because you're in this adversarial environment, where people are trying to break your system at all times.

People want to create transactions that are invalid.

Because that can be worth a lot of money.

I can say, hey, you don't know about this shard, but I'm telling you that on this shard, I have a lot of money.

And I'm sending it to you, so give me your house or whatever.

So the idea is to split a single UTXO set into multiple smaller sets, and that part is OK.

But you need communication between the shards, right?

So you could say, OK, well, making sort of a bunch of different UTXO sets, each node can choose their own UTXO set that they're keeping track of.

And then you have some kind of merge mining between the UTXO sets, but you need swaps between the shards.

It's kind of an interesting thing to think of.

We already have multiple UTXO sets, right?

So as of this morning, coinmarketcap.com tracks 1,614 different currencies, 10,000 markets.

There's supposedly $434 billion going around.

Is this sharding, right?

And it's sort of a joke, but in a real sense, it has taken scalability pressure off of Bitcoin and off of any individual currency because there's so many of them.

So if you didn't have Dogecoin, maybe there will be more Bitcoin transactions.

AUDIENCE: And how does this play into Bitcoins that are just running on Ether?

TADGE DRYJA: Right, so in the case of ERC-20 or ERC-721, it actually is worse.

Because now you've got sort of multiple UTXO sets all being managed by a single UTXO set.

So if you want to keep track of how many-- I don't know.

What's an ECR-20 token?

If you want to keep track of how many Pied Piper coins there are, you need to download the entire Ethereum blockchain.

So that's sort of the opposite of sharding in that now any single UTXO set you want to keep track of, you need to

keep track of all of them.

However, in this case, if I want to keep track of my Bitcoin UTXO set, I don't need to download Dogecoin.

So that's great.

And if people want to swap between Dogecoin and Bitcoin, they can do so-- well, Dogecoin doesn't have segwit support, right?

So it's a little harder.

But Vertcoin, Litecoin-- a lot of different coins have fairly easy swaps.

But it's more than just swaps.

We need actual fungibility between the shards.

Because if I can say, oh, I'm going to use Litecoin, and I can just swap to Bitcoin whenever I need to pay someone who accepts Bitcoin, right, maybe.

But maybe Litecoin drops in value 20% with respect to Bitcoin.

And then I tried to pay.

And this is sort of getting ahead of the real use cases of these things.

Because well, Bitcoin also drops 20% randomly against whatever asset you're trying to buy.

And so but Bitcoin does tend to be a bit more stable than most of the smaller market cap coins.

I mean, you can sort of think that generally, if you have a bigger market capitalization, you're going to tend to be less volatile.

And we've seen that in Bitcoin, where it still seems ridiculously volatile, right?

It's gone down 50% since January.

And most currencies in the developed world don't do that.

But if you actually compare it to in 2011, it dropped, like, 95% in a month or two.

Yeah.

AUDIENCE: It's more likely to be liquid supply [INAUDIBLE] TADGE DRYJA: Yeah, so sometimes, there is a currency that has these really inflated market caps.

So you'll see one where someone makes a coin and says, OK, I'm making a million coins.

And I'll sell you one for $100.

And really, I still have all of the coins, and one other person has one of them.

But we can sort of do the math and get a market cap of $100 million that way.

So a lot of the coins do that to sort of inflate.

Because a coin market cap is literally a ranking.

And it even says rank.

So if you click EOS, it'll say rank.

Where does it say there?

Yeah, rank five.

So it's the fifth best, presumably.

And yeah, to what extent-- how many actual people hold these things?

They just sort of made them up.

But there's all sorts of problems with this.

But the idea is, if you really want sharding, you want the swaps between the shards to not really have counter parties and to maintain the same value.

You want fungibility between the shards, so that you can quickly and easily say, OK, well, I've got something on shard A. I'm going to pay someone who's using shard B. And I don't want any friction.

I don't want any exchange between there.

So this is hard.

There's a lot of cool research going on here.

And if it works, it's a real scalability improvement.

This is sort of the holy grail.

I don't really see much in-- in Bitcoin, they're sort of like side chains.

But those weren't really talked about as a scalability improvement.

And it's mostly the Ethereum crowd that are like, this is our real sort of holy grail for scalability.

So it's interesting to look at.

I haven't kept up.

I've read a little bit about their most recent sharding ideas.

So it's cool if it works.

But there's a lot of sort of different assumptions that go in.

Yeah.

AUDIENCE: What are some of the current plans to do sharding?

TADGE DRYJA: They sort of like-- a lot of them hinge on fraud proofs.

So the idea is, you've got, say, five different chains going along.

And you don't validate the other four.

You say I'm going to only validate this one.

There's four others going on.

And if, in my chain, something bad happens, like a transaction with an invalid signature gets confirmed, or transaction has more coins coming out than going in-- something like that-- you provide a small fraud proof.

You provide a proof that says, OK, you don't need to know everything that's going on in this chain.

But I'll provide enough data to convince you that this specific transaction is broken.

And then I try to broadcast to the other people in those other four subchains.

And then they know, OK, something's going on wrong here.

Let's freeze-- let's not accept any cross shard swaps from that one.

And then the other-- this chain will have to reorg.

So the idea is as long as you have-- and it sort of makes sense.

As long as you have some number of people checking it that can then broadcast it between the different shards, you can assume that they're doing OK.

Yeah.

AUDIENCE: Problems temporarily arise from [INAUDIBLE].

So you'll have a situation where [INAUDIBLE].

And then you wait two days, and then the transaction [INAUDIBLE] which relies on data from two years ago for that shard.

All of those shards have disappeared.

So you have a choice.

Do you do it to your real or just accept that that money is gone?

TADGE DRYJA: Yeah.

Or just accept that, oh, well, it's two years ago.

So I'll assume it's OK.

AUDIENCE: Yeah.

TADGE DRYJA: Yeah.

Which is probably more dangerous.

Yeah, so sort of availability and liveness are other issues here, where if in Bitcoin or Ethereum-- well, it's more of an issue in Ethereum.

In Bitcoin, there's a lot of copies of the full set.

So if you're not sure about something, you can pretty easily get the entire blockchain, even though it's like 180 gigs, and go through it.

There's so many copies of it out there.

In Ethereum, a little bit less so.

Well, there's more full nodes.

But full is sort of redefined.

And so it may be harder to get the full thing.

And in the case of sharding, if you divide it too finally, it may be that you lose data, and no one has it.

And then you're in real trouble.

OK, so that's sharding.

OK, accumulators-- this is something I'm actually looking at.

I'm not going to go into the details of what I'm working on, but accumulators in general.

This is a cool-- so accumulators, as I will describe, are nothing new.

But if you read the papers-- so one of the first papers was called One Way Accumulators in, like, '93.

And if you read the paper, a lot of the words jump out.

It's like, hey, this might be useful for Bitcoin.

It's like, set membership, and timestamping, and signature aggregate.

It's got a lot of stuff in there that's like, hey, this could be useful.

So an accumulator is basically a cryptographic set.

And there's some set operations that you can do and then provide proofs.

So, the simplest-- well, not even the simplest.

The simplest would just be add and prove.

But sometimes you can add.

Sometimes you can remove.

Sometimes you can prove something's in there.

Sometimes you can prove something's not in there.

And if you can do all four, that's even better.

So the idea is, you take an accumulator.

And you add an object to it.

And in general, objects are going to be strings of bytes or just numbers, right?

And then it spits out a new accumulator.

Essentially, it modifies the accumulator in place.

So if you delete-- you say OK, I've got an accumulator.

And I want to delete this object.

Well, it'll modify in place and return a different accumulator.

And then maybe I want to prove that this object is in this accumulator.

And it will return a Boolean.

Like, yep, that worked, or no, it didn't.

So the simplest example, which I think is kind of fun, composite numbers.

So accumulate prime numbers.

So to add, multiply.

To delete, divide.

So really, if you're going to do this, you start with 1.

1 is not a prime, I guess, but whatever.

So let's say you've got-- so yeah.

1 is not a prime, so that works.

So you cannot prove any prime exists within that accumulator.

Anyway, but let's say you start with 1, and then you added 3 to the accumulator.

So you multiplied by 3, and you get 3.

Now I want to add the number 5 to the accumulator.

So I multiply by 5.

I get 15.

And now I want to add 7 to this accumulator.

So I've got my accumulator, which is 15.

I add the number 7 into it, and I get 105, right?

I just multiply by 7.

I get 105.

Wait, is it called fundamental theorem of arithmetic?

I think that's what it's called, where everything's a product of primes.

It's got some cool name.

So everything's a product of primes.

And everything has a unique factorization.

So 105 is 3 times 5 times 7, right?

There's no other ways around it.

And if you want to delete, you can say, OK, well, I'm going to delete the number 5 from this accumulator.

Just divide.

Now I get 21.

And then I want to prove 7 is in there.

So I can say, hey, 7 is in this accumulator.

It was added, but not deleted.

And I can do that, right?

I tried to divide 21 by 7, and it worked.

I want to divide, and I want to make sure there's no remainder.

I want to see that it divides evenly.

So I get a true.

Yep, if I divide 21 by 7, I get 3.

3 is a natural number.

It works.

So this is kind of cool.

It is not really [INAUDIBLE].

Oh, but this works even if you do modulo.

So you could do modulo some big prime and have just formality based accumulators.

So anyway, you get the idea, right?

I think in this case, you can also prove things are not in there.

But this is limited to prime numbers.

Anyway, but the idea is, keep adding things to it, removing things from it, and proving that things are in it.

So there's RSA accumulators, which are some of the most well-known.

And that's you've got some RSA number, which is basically a product of two large primes.

And the accumulator itself is of constant size.

And the proofs are also of constant size.

So we call this a proof.

You're just giving the number itself in this case.

So it's efficient, but the RSA accumulators use trusted setup.

So the idea is you need to find some composite number n, which is p times q, where p and q are prime, where nobody knows p and q.

Or nobody knows or you trust that the person who does know p and q not to screw around with the accumulator.

Because the person who-- if knowledge of p-- I think it's actually knowledge of p or q-- will let you create proofs that, hey, this object is in the accumulator when really, it isn't.

So that's not so much fun for Bitcoin.

If you need trusted setup, people don't really like that.

There's all these other accumulator assay ideas.

Some are one way where you can't delete.

You can add things to the accumulator, but there's no way to remove it.

Sometimes you can batch things, where, OK, if I want-- so you can see in the composite number accumulator, you could batch things.

Where if I add 105 to the accumulator, I'm performing one operation that essentially adds three objects, right?

3, 5, and 7.

Some can be batched like that, some cannot.

Some have trusted setup, some don't.

So there's different tradeoffs for all these different use cases.

And in the case of Bitcoin, the idea of an accumulator would be put the UTXO set in it, or put the STXO.

Like, spent the transaction outputs into it.

And then prove, in the case of UTXOs, prove that it's in the accumulator.

Or in the case of STXOs, prove that it's no longer in the accumulator.

Right?

Provided proof of non-inclusion, that, hey, it's not in this STXO set.

So, well, with the STXO inclusion-- so let's say you did it that way.

You'd have headers, and then you've got this STXO accumulator.

And what you can do there is say I've got a transaction.

It's got some inputs.

I prove that this input exists in the headers, right?

I provide you an SPV proof.

So I say, OK, at some point, this was created, right?

This input.

Maybe a year ago, I showed you, OK, there is this header.

Here's a Merkle proof that this was in a block.

So this exists.

Then I also prove somehow that it doesn't exist in here, right?

It exists, but it was never spent.

So now you can accept that, oh, OK, he gave me an SPV proof.

He gave me a non-inclusion to the STXO set proof.

So I know this transaction, this input still exists and can be spent.

So what this would do, if you get it working, you don't need to store the UTXOs anymore.

You just store the accumulator, and then everyone provides proofs that, hey, I've got these coins.

But so right now, if you store the UTXO set, it's a couple of gigabytes-- 3 or 4 gigs.

And when someone sends a transaction, you just look in your UTXO set, right?

So you see this input.

You say, hey, does that exist in my UTXO set?

OK, it does.

Cool.

We'll now verify the signature, verify everything else about the transaction.

If it doesn't exist in my UTXO set, I'm like, hey, you're trying to spend something that isn't there.

So what would be cool is if you got rid of the UTXO set and only used an accumulator.

Because the accumulators are constant size.

So even if the UTXO set is 20 gigabytes, well, I've got this 10 kilobyte accumulator thing on my hard drive.

And I just modify that in place.

And now I don't need to store a bazillion gigabytes.

And I basically can have the same security as a full node.

I'm still a full node, but I just don't store the UTXO set.

I just require either SPV proofs or STXOs-- different proofs from the people trying to spend the transactions.

So this is really cool.

Constant size, the proofs are small.

Sometimes the proofs are really small.

And then the wallets track the proofs.

So some questions.

Proofs can be like constant size, or sometimes they're log n.

If the proofs are o of n, then it's not useful, right?

Because you might as well store the UTXO set at that point.

Also, what is n?

Is it the number of transactions, blocks?

Can you aggregate these operations?

So there's a bunch of questions there.

Another really big problem-- so I was talking to Peter Wool, who's sort of one of the premier researchers on Bitcoin.

He had been talking to people at Stanford about a lattice based accumulator that did not need trusted setup.

And he was very excited about it because it was like, hey, there's no trusted setup.

You have constant size proofs.

The accumulator itself is pretty small.

CPU wise, it seems doable.

The thing that sort of killed it was you couldn't batch operations that accumulated.

And the other thing is we need some kind of bridge node.

So the idea is, normal transactions, right?

They just say, here's my input.

I've got a couple inputs.

I've got a couple outputs.

I don't provide any proofs or anything.

I just point to what I'm spending.

And you look in your UTXO set and see if it's there.

With this new idea with accumulators, you're going to have to stick proofs on.

So really, it's an extra data structure, probably per input.

So you say, hey, I'm a node.

I run an accumulator.

I don't keep the whole set.

So for all your inputs, please provide proofs.

And wallets can maintain these proofs and attach them to their transactions.

And then the nodes will verify them.

However, right now, most wallets don't have these proofs-- have no idea that this is a thing, right?

So if you're the first node to do this and say, hey, I'm getting rid of my UTXO set.

I'm only going to verify proofs that these UTXOs exist.

Most of the wallets won't provide those proofs.

And so as a new node, say, hey, I got rid of my UTXO set, but no one's giving you proofs.

OK, I'm just stuck, right?

I see a block come out.

I can't validate it.

So you're going to need some kind of transition, right?

If you started from scratch and say, OK, the responsibility of every wallet is not just keep your private keys and keep track of what your UTXO is, it's also to keep track of a proof.

So that when you want to spend it, you give it to someone else.

If you started that way, great.

But if you want to transition, what you're kind of going to need is a bridge node.

And the idea of a bridge node is you've got-- so here's an accumulator node, where it requires proofs.

Here's a old node that just has regular UTXOs.

So when the old node sends a transaction, this gives TX proof, right?

So this basically has proofs for everything, like all proofs.

So the bridge node can provide proofs.

You'll need one bridge node, right?

So if these accumulator nodes talk to each other, they can send the proofs along to each other and stuff like that.

The old nodes cannot send proofs because they're not aware of it in their software.

But if you have the new nodes, you can say, OK, well, when I receive a transaction, I verify the proofs.

I also keep the proofs and give them to other peers who want to verify them.

But you need some kind of bridge between these two networks.

Yeah.

AUDIENCE: You would just keep that running until everyone switches?

TADGE DRYJA: Probably forever.

So yeah.

So the issue with the bridge node is like, well, if you're an old wallet, you don't to keep track of these proofs.

So, how much of a problem is this?

And the lattice based accumulator, that was sort of what killed it, was that you couldn't batch operations.

Sorry, and the proofs changed.

That's not the case with the sort of silly prime number accumulator that I showed.

But in the lattice space one, the proofs changed every time an accumulator operation happened.

So every time an ad happened, your proof had to change.

So you didn't just need to add something and modify the accumulator in place.

You had to modify your proofs in place as well.

And so if you're a wallet, and you've got three UTXOs, that meant that every block, you'd have to do a couple thousand operations, right?

Because every block, you're going to have a few thousand adds and a few thousand deletes to this accumulator.

Every time one of those happened, you'd have to modify your proof for each of your three UTXOs.

So, a little ugly but doable, because you've got three UTXOs.

You basically 3x the number of operations, set operations in the block.

The bridge node, on the other hand, has to keep track of something like 70 million UTXOs.

Right?

That's about how many there are right now.

And so since there's no way to batch it, that means you're going to have maybe 10k times 70 million every block.

It's just not going to happen.

And then it's like, well, maybe you can sort of try to shard the bridge nodes.

And this bridge node only keeps track of this portion of the set.

But it just looked really daunting.

And that was why he was sort of like, yeah, I don't think this lattice-- I think it's really key that either there's some kind of batching operation, where we can consolidate all the operations within a single block to one set operation for the accumulator, or try to make it so that the proofs don't have to be updated or something like that.

So this is still an unsolved problem, although I'm working on a fun way that I think might work to do this.

And I don't want to-- because this is videotaped.

It's going on the internet and stuff.

I don't want to talk about it.

But if you have questions, we can talk about it at office hours or something.

So if it works, it'll be cool, although in some cases, it might-- so in the case of that bridge node, it was the bridge node that really killed it.

Because it was just like, oh, you're going to have to do billions of operations per block.

But in other cases, it's like, well, you can have accumulators that seem good, but aren't actually faster.

Because verifying the proofs takes a long time or something like that.

So it's sort of like with the range proofs or something like nimble-wear where the o of n is great.

But you've got these constant factors that, in practice, end up meaning you're not actually faster, right?

Because the Bitcoin UTXO set, well, it's 4 gigs.

Not even, right?

It's 3 and 1/2 gigs now.

So in practice, there's probably a lot of cool cryptographic technologies.

And you can write a whole paper and have all these cool things.

And then if you actually implement it, it's like, well, it actually goes twice as slow as just using the regular.

Also, this is super optimized.

One of the biggest sort of code engineering things that has happened in Bitcoin Core is OK, how can we make database updates to this?

How can we make different caching, different flushes to disk, and editing level DB itself.

So making this UTXO set database operation faster is a big thing in Bitcoin.

So it's pretty optimized.

And even if you've got something that seems like in computer science terms, hey, this is log n instead of n, like, yeah, but what about the constant factors?

So that's another issue for these.

OK.

Other things-- I was going to talk-- well, yeah.

I'll do last one, and then we can maybe end a little early and talk about projects if you want.

UTXO commitments, which is somewhat in the same region as accumulators-- a little bit different idea.

And this exists in some coins.

Like in Ethereum, Joe Bonneau was saying that you basically have a tree of all the different contracts that exist, all the different addresses in Ethereum.

And the root of that tree appears in every block header.

So you can make these proofs that coins exist based on a block header.

So it exists in Ethereum.

It doesn't exist in Bitcoin, but people have been talking about it for years.

The simplest would say take a hash of the UTXO set and put it in every Coinbase transaction.

And make that a consensus rule, so that when you mine a block, you have to take the hash of the UTXO set that you're aware of.

Put it in the block so that everyone can see it.

And if everyone disagrees, they're going to invalidate that block.

This is really simple.

You probably want to do a little bit fancier than this.

A bit more useful-- instead of just taking a linear hash of the entire UTXO set, make it a Merkle tree, right?

And that way, yeah, sure, it's very little extra hashing.

Well, no, wait, twice as much hashing, right?

But anyway.

2x the hashing, but now you've got the ability to make small proofs.

And then you can prove an output exists at a given block height pretty easily at SPV level security.

So currently, you can prove that a transaction exists at a block height.

And you can also prove that a transaction was consumed at a certain block height, right?

So if you have all these different blocks, 2, 3, 4, 5, you can say, hey, here's a Merkle proof that transaction 1 was included in block 2.

And then you can prove here, OK, here's transaction 2 that consumes one of the outputs from transaction 1.

But you can't prove just given-- but also, you can omit this proof, right?

So if you want to prove that-- so at block 6, how do I prove that the outputs from transaction 1 still exist at block 6?

You're stuck, right?

You basically have to go through blocks 3 through 6 and look at the whole thing.

So with current SPV proofs, we can prove inclusion, right?

You sort of think of this as an accumulator, where you've got-- a Merkle tree is kind of an accumulator.

I can prove inclusion, and then I can prove transaction 1 is in block 2.

I can prove that transaction 2 consumes transaction 1 in block 5.

But you have to rely on me to give you that proof honestly, right?

If I'm trying to cheat you and say, oh yeah, I've totally still got money, I can't prove that it hasn't been deleted.

Whereas with a UTXO set commitment, if every block, there was a total commitment to every UTXO, I could just say, hey, at block 6, my output from transaction 1, it's still in there.

Well, it wouldn't be in this case.

But I could prove it at block 4, right?

And say, hey, look, transaction 1, it's still in there at block 4.

No longer there at block 6.

Depending on how you construct the UTXO commitment, you may be able to provide non inclusion proofs, which is also really cool.

So for example, if you had it be a Merkle tree, if it was just in order insertion-- so this is TX1, this is TX2, this is TX3, and you make a Merkle tree that way, then you cannot prove that something's not in it, right?

So when TX3 gets deleted and now TX4 goes here, and TX5 comes here, you can't prove the TX3's not there anymore in any real way.

However, if you sort them so that every time so now you sort them canonically, so like by hash, just like greater than, less than, and then the order gets all weird, right?

So now TX5 is here and TX6 is here because TX5's hash is closer to 1 and stuff like that.

Then you can prove non-inclusion.

So then you can say, hey, TX3 is not here because TX3 would be between 5 and 4, right?

Just based on what the hash looks like.

And I can show you 5.

I'm going to prove for it.

I can show you 4 and approve for it.

I can show that these two things are adjacent, right, in the tree.

And there's no 3 there, and it would be there.

If it existed, it would be right here.

And I can show that it's not there.

So then I have non-inclusion proofs in the UTXO set, which could be used for some kind of fraud proof.

So if someone makes a transaction spending something, you can say, no, look, here is the last block.

Here's the UTXO set hash.

And here's a proof that this input doesn't exist.

So this must be an invalid transaction.

That would be really cool, too.

Because then you could propagate that on the network and sort of prove fraud.

So this is an idea that's been around for a long time.

I think maybe the reason it hasn't yet been implemented is, no one can really agree on exactly how to do it.

A lot of people are like, yeah, we should do that.

That would be cool.

Yeah, you can prove [INAUDIBLE].

So, one thing you could do would be to skip years of initial block download.

So I don't really care what happened from 2009 to 2015.

I assume it was fine, right?

I'm going to start my synchronization in 2016 and just synchronize the last two years.

So I'll go get this UTXO commitment from end of 2015.

I will then not just get the commitment, but download a UTXO snapshot from that time.

And then I'll check that it matches the committed UTXO set hash.

And then I'll just start from there and then become a full node, where I didn't check the first six years or something.

And its interesting mix of SPV security and regular full node security, and I think personally that's probably OK, right?

If you only verify the last six months of signatures, well, if everyone's been wrong for six months, we have bigger

problems, right?

If there's some erroneous transaction in 2015 and the entire network has been just extending for three years without reorging, like, wait, what?

So on the one hand, you don't want to do this for today's blocks, right?

Because then the miners have full control.

And the miners can just make up transactions.

And if you really just say, oh, well, the miners are doing the right thing, then they get a lot more power.

But if you say, well, other people are validating it in real time.

And I assume that that's the case, and I'll validate six months' worth of transactions.

So I'll pick up any errors in the last six months and be able to report them.

But I will assume that after a certain amount of time, I'm pretty sure it's been OK.

Everyone else has been looking for this.

So that's a slightly different model than full nodes.

But in practice, not really.

Because if you look at the code for Bitcoin today, there's assume valid, which doesn't check signatures.

It sort of has a block hash and says, OK, anything before this, don't check signatures.

The developers themselves have said, yeah, every signature before here is OK, so you have to check the signatures, which is a little nicer.

Before they had check points, where it would just not validate anything before a certain block hash.

So you sort of already have things like that to speed up the initial synchronization process.

But a UTXO commitment would make it a lot more sort of decentralized.

Because right now, it's just the programmers are like, well, this is last year.

Everyone's validated.

Let's just hard code this into the code, into the client.

Anyway.

So this is an idea.

The issues-- timing is probably one of the biggest issues.

If it's consensus critical, then the miners need to put this into their blocks.

And miners also need to verify it when they get a block.

And adding even a second to creating and verifying a block can centralize mining a bit.

Because the idea isl I want to be able to immediately start mining as soon as I see a block.

Because a larger miner doesn't need to verify the block that just came out, right?

Because they created it.

If they created a block themselves, they know it's fine.

They build on top of it immediately with zero propagation delays, zero verification delay.

Smaller miners or miners receiving that block have to make sure it's correct before mining on top of it.

So adding even a one second creation verification delay is something that a lot of the programmers are like, mm, if you can get it down to half a second, maybe we're OK.

We want this functionality, but not at the cost of increasing that.

That's the worst time to add something.

If you can defer it, if you can say, oh, well, you need to verify this, but you can do it after the fact or something, then it's fine.

So this I ran up against this because I had this fun idea, where you could half aggregate Schnorr signatures within a block and do it non-interactively.

And it was really cool, and I'm pretty sure it works.

But it added, like, three seconds to block verification, so that killed that idea.

So one of the issues, yeah, timing.

Another issue is it does encourage more SPV level verification, where you can sort of see that a lot of people will now use this method to not run a full node.

Because they're like, well, I can just get these compact proofs for all the UTXOs, whether inclusion or non-inclusion.

So why run a full node?

I don't need to verify it because other people are.

You don't really want to encourage that because there's so much of that already.

Also, the biggest is probably there's got to be a better way to do this.

So there's all these different ideas on how to do it.

And they never seem to settle or converge on a single way.

So the three main ideas are sort of hash based UTXO sets, elliptic curve based commitments, and RSA based commitments.

And there's a lot of overlap with accumulators, in that you're sort of-- the UTXO commitment could be the sort of route or the hash of the accumulator itself.

That you want to prove that something's in there or prove that something's not in there.

And the EC one is the current one that Sip was looking at was looking at, but it never made sense to me.

Because it's trivially-- you can provide invalid proofs really easily.

So it's all trusted.

But his idea was, well, let's say you have a node that you synced up, and you want to sort of port that to somewhere else.

You can do that very compactly with an elliptic curve based UTXO commitment, which is cool, but you can't trust the inclusion or non-inclusion proofs.

And there's RSA ones that also-- yeah, so there's some overlap there.

Anyway it's a pretty cool idea.

It does exist in Ethereum, which is just hash based that Joe Bonneau sort of explained the tree thing they used.

But that doesn't provide for non-inclusion proofs, which would be fun.

So yeah, so this is another research area.

Any questions about UTXO commitments?

There's a bunch of-- Bram Cohen's Bitfield thing.

Have you seen that?

Yeah.

So that was a really interesting case of-- in computer science terms, it's o of n.

But he got it so that it's actually one bit per spent transaction output.

So the idea would be maintain a TXO set, right?

So for every transaction output ever created in order, you have just bits, right?

So if eight transaction outputs are created, that's a byte.

And you leave it all as zeros when they're unspent.

And when they have been spent, you just set it to 1.

And the thing is, yeah, there's 70 million, but if it's that-- well, there's more.

There's hundreds of millions.

But if it's bits instead of bytes, that's pretty small, right?

It's maybe 100 megabytes if you have 800 million outputs ever.

And then you can quickly see on your hard drive, OK, this output has been spent.

And you can provide proofs and stuff.

So the Bitfield thing was a kind of an interesting one.

OK, it's o of n, but 1 bit per object.

And you can sort of quickly sort between them.

So these are some of the things that-- I think that's it for today, right?

I was going to talk about covenants, but that's another story.

So these are some of the current research issues in Bitcoin, blockchain, cryptocurrency.

And there's lots of other topics.

But these are some of the big ones that I'm aware of and I know people working on.

But yeah, there's lots of ways to improve privacy, scalability, functionality.

It's certainly not like-- and sometimes people say, oh, well, Bitcoin was made in 2009.

And it's like blockchain version 1.0, and we need to make blockchain 4.0 or whatever.

And to some extent, yeah, Bitcoin is annoying to change, right?

It's difficult to change, and you've got the idea of a bridge node, and can't we just start over?

It'd be so much easier.

In some cases, yes, it would be nicer to start over.

But that's sort of the challenge.

It's like, wait, can we make this system better, but maintain backwards compatibility?

And so, in a lot of cases, like the accumulators or the client side Bloom filter checking, it's not a fork at all, right?

The miners don't have to change anything.

If you don't know about this new change, you don't have to change anything.

And so there are totally optional ways to improve some of these things.

And that's a fun challenge, right?

You're given this system.

It's sort of like trying to improve an airplane while it's in flight, where you're like, OK, I'm going to make it faster, make it better while it's flying.

We can't even land.

So yeah, that definitely makes it harder.

But other things are-- but what's also interesting is that there's not that many things in it that really require starting over from scratch.

So I know at the New York City Bitcoin Core Developer Meetup in March, Bram Cohen, who made BitTorrent-- he's making his own coin, I guess, now.

And it's kind of interesting because usually, the Bitcoin people don't like alt coins or are sort of wary of them, and don't invite alt coin people to their events and stuff.

But Bram Cohen invented BitTorrent, so everyone sort of owes him in some spiritual way a couple thousand bucks for all the music and movies they've downloaded, right?

So it's like, oh, Bram, he helped us out.

So he comes to these things.

And he was asking at the thing in March, OK-- asking a lot of people-- if you were going to start Bitcoin over from scratch, what would you change?

All the people who were programming this for years and dealing with all the annoying problems in it gave, if you started from scratch, what code do you change?

Or what sort of basic infrastructure things do you change in the system?

And there weren't that many, right?

A lot of people were like, oh, there's the little things.

Like, oh, get rid of the little silly op zero in multi sig.

Get rid of this, or change a couple of these things.

But not a lot of really fundamental changes people would want to have made.

Some people were saying, hey, maybe instead of pointing based on TXID, be able to point height and index, so you can have a lot smaller there.

Some different opcode things, but what was interesting is that-- and also, it's a super biased sample.

And it's like, if you're asking all the people who work on Bitcoin, well, yeah, they sort of like how Bitcoin works, or at least, they've grown accustomed to it.

So you might want to ask other people if you want to sort of think out of the box.

But there weren't a ton of major changes that people wanted to have made.

So it does seem that it's a fairly well thought out base of a system.

And then I'd say the Ethereum design is the other design that's very different, right?

The account based versus UTXO base.

And a lot of the people who work on Bitcoin think that account based is worse.

A lot of people maybe in Ethereum case think UTXO base is worse.

They do have tradeoffs.