NEHA NARULA: OK, let's go ahead and get started. So Tadge is in California today, so he asked me to give today's lecture, which will be based on a research project that we're working on at the DCI. This is actually really connected to confidential transactions and confidential assets, which I think that you guys covered in a previous class. So Pedersen commitments will play a very important role in the work I'm about to present.

I want to present this work that we've been doing. This is joint work with Willy Vasquez, who was an MIT M.Eng. He graduated and moved to UT-Austin, and Madars Virza, who is a research scientist with the DCI.

And this project is called zkLedger. It's about privacy-preserving auditing for distributed ledgers. So to start with, I am not sure how much work you guys have done on permissioned blockchains so far. How much have you covered with respect to permissioned blockchains? Nothing at all? OK.

Should we start with what is a permissioned blockchain, or do you guys feel like you have a handle on what that means or what that is? In its most generic, abstract sense? Yes?

AUDIENCE: Does it just mean that different people have access to see parts of the chain?

NEHA NARULA: Yeah. So, good question. OK, so these words that we use-- permissioned versus permissionless-- usually refer to who has right access to the chain.

So in cryptocurrencies like Bitcoin and Ethereum, theoretically, any one of us could spin up our own ASIC, get really lucky, mine a block, and put the transactions that we are interested in into the chain. Now of course, no one's going to accept that block if it violates the majority consensus rules of the chain. But assuming that it follows the rules, it's well formatted, it has the appropriate proof of work, et cetera, any one of us could submit a block to the Bitcoin blockchain. Right? There's no permissions on who is allowed and who isn't allowed to write to the Bitcoin blockchain.

Permissioned blockchains are a different story. These are more like distributed databases. The idea behind permissioned blockchains is that you choose a set of actors ahead of time, and only those actors are allowed to write to the blockchain. So you can't just have anyone willy-nilly spinning up a mining node, or spinning up a node in the system, and submitting transactions and things to the blockchain.

Usually, permissioned versus permissionless doesn't say anything about reads, and who can read the blockchain. We're really just talking about rights. So an important use-- permissioned blockchains, I think, are not really the focus of this course, in part because we take the position that the technology behind them is essentially distributed databases. And there's lovely courses out there on distributed databases. And there's a lot to learn about them, and they're very interesting, but they're not necessarily very closely related to cryptocurrencies and the designs of the past decade.

So that's the idea behind permissioned blockchains. Now, as I was saying, I think that there's a lot of use cases for permissioned blockchains. They're very useful. Distributed databases, in general, are very useful. And so let's talk about one of them.

And I'm going to interchangeably use permissioned blockchain and distributed ledger, here. Distributed ledger, permissioned blockchain, I'm sort of saying the same thing. So permissioned blockchains-- any questions, before we dive into the use case in zkLedger, about permissioned blockchains? Get the idea? It's a whole bunch of people working together to construct a ledger.

So let's start with the structure of the financial system. And this is setting the use case for zkLedger, and what we were trying to do, and what we were trying to address. So to start with, there are dozens of very large investment banks that are trading securities, currencies, commodities, derivatives. Many of these things are traded on exchanges, regulated exchanges like NASDAQ or the New York Stock Exchange.

But up to 40% are actually traded in unregulated fashion. They're done in a way that's called over-the-counter, meaning that there is no regulated exchange sitting in the middle facilitating these trades. Instead, the banks, the entities, trade amongst themselves in a slightly more ad hoc fashion.

This represents trillions of dollars of the economy. And just to give you a sense of scale, the time frequency we're dealing, with we're not talking about high-frequency trading. That's not the use case that zkLedger is targeting. We're thinking like 10s of trades a minute.

These are things that are done electronically. It's not like people are-- people might be calling each other up to arrange these trades, but oftentimes they're done electronically. But we're not considering high-frequency trading.

So a useful abstraction for thinking about a table of these trades is a transaction ledger. So each row in the ledger is a transaction, transferring assets from one bank to another. So this should be very familiar to everyone.

It looks a little bit different than things like Bitcoin, because since this is a ledger designed for the existing financial system, we know who the entities are. They have names. They're regulated. They have companies behind them. We're not talking just about arbitrary public keys.

So each row in the ledger is a transaction. So here, Citibank is transferring a million dollars. So that's another thing that's very different than cryptocurrencies. Here we have multiple different assets being recorded in the ledger.

So we're not talking about a ledger with a native asset, necessarily. This is not a cryptocurrency. Really think about this like a table in a database recording transactions. Those transactions might be in dollars, or euros, or municipal bonds, or securities, or shares-- they could be anything, right? So we're just recording transactions.

So Citibank is transferring a million dollars to Goldman Sachs. JP Morgan is transferring money to UBS and to Barclays. OK? So as with cryptocurrencies, I think, cryptocurrencies have inspired people to reconsider the case of a distributed database as a ledger that multiple entities can write to.

And part of that consideration involves thinking about digital signatures. So maybe these banks have public, private key pairs. And perhaps they use their keys to indicate consent on the transaction, right? This is something that's done under the hood right now, when you think about the protocols we use to exchange data.

But it's not necessarily done explicitly when creating these distributed databases. So they can use their keys to attach signatures to the transaction to indicate a consent to transfer for security. Now, this ledger might be maintained by a third party, or it could be a distributed database, a permissioned blockchain that is run by the transacting parties.

So there are some important financial invariants that you want to maintain here to verify that transactions are happening correctly, that they're valid, that assets aren't created out of nowhere. So one of them is consent to transfer. And we use these signatures for that.

Another important thing to verify, another important invariant, is that the person doing the transacting actually has the assets to transfer. That they're not making up assets out of nowhere, or they're not double-spending-- that there isn't something incorrect happening. And then, that assets are neither created nor destroyed. That we have conservation of assets.

So these are important financial invariants to maintain. This looks a lot like what the invariants are that we're trying to maintain in a cryptocurrency like Bitcoin or Ethereum, except this was with multiple assets, and we know who the people are involved.

Now, if this ledger were being maintained by an exchange, then the exchange would be responsible for verifying and validating that all of these transactions are correct. If this ledger is being maintained by the participants, it's a permissioned blockchain, then everyone can verify and validate that these invariants are being maintained by looking at the blockchain, by checking the signatures, right?

So that's great, except for the fact that privacy is really, really important. And quite a bit of data, and quite a bit of sensitive information, is actually leaked by looking at these transactions. So if we want to be able to maintain financial invariants, if we want to be able to check and make sure that the ledger is correct-- and I mean, this is the whole point of blockchains, is that everyone can verify, and validate, and reconcile real time. Then in the act of looking at those transactions, you're actually finding out really sensitive information about banks' trading strategy-- about their intellectual property. OK?

For example, a large trade might indicate that a bank has decided to get out of a position. Other banks could learn about this, notice it on the blockchain as they're validating transactions, and try to follow suit, driving down the price. So privacy, especially in this type of context, is really important. And I'm sure you can imagine a lot of other different types of context in which privacy is important as well, right?

So we don't want to be in this situation where we have to choose between privacy and being able to validate what's going on. Right? So even if we were to use a trusted third party like an exchange, we still wouldn't have privacy, necessarily, because the exchange would be looking at everything in the ledger.

So that's the goal here that we're trying to achieve. We're trying to achieve this goal where we want to be able to publicly verify and validate what's happening in the ledger, and at the same time, obtain some amount of privacy for transactions. We don't want all transactions to be public to whoever is maintaining the ledger, be that a third party exchange or a set of banks.

So I'm just going to pause right there and-- do we get the setup, here? Privacy versus verification-- we want both. Any questions about that? Yes?

AUDIENCE: I was confused as to how you verify that people-- transactions are correct in their account balances? It's like there's no native assets supplied.

NEHA NARULA: Yeah, that's a great question. So I have dollars and euros here, right? How do you know that Citibank actually has those dollars to transfer to Goldman Sachs? Right? Where does that come from?

Notice how this is transaction number 90. I'm assuming that there's a transaction up here-- this is just a subset of the ledger. But as you'll see in a moment, we think about this role of depositors who are allowed to deposit assets into the ledger. And so I'm assuming there's a transaction up here somewhere which in a provable way shows that someone we trust to deposit assets into the ledger gave Citibank the dollars necessary for them to give to Goldman Sachs.

But you're absolutely right. If the asset isn't native to the ledger, then you do have that connection to the real world, where you do need to apply some level of trust. The question, is it auditable? Can you at least track it back up to that point? Any other questions?

OK, great. So we want to do the following. We want to be able to verify that financial invariants are maintained, that things are correct while achieving privacy. So one good technique for achieving privacy is simply to encrypt or hide the transactions in the ledger, OK? So surprisingly, maybe, we actually know how to do this. And to, at the same time, still achieve checking of the financial invariants.

Here are two systems that actually achieve this. The first is Zerocash, which was implemented inside of Zcash, an anonymous cryptocurrency. In Zcash, transactions are completely hidden.

They're completely opaque shielded transactions in Zcash.

Zcash has two transactions. Once set is public, like in Bitcoin. The other set are called shielded transactions. And in a shielded transaction, you don't know who the parties are who are involved. You don't know the amount of the asset that's transferred. But you can validate that this was correct, that financial invariants were maintained.

Zerocash and Zcash use a primitive called zero-knowledge proofs, and in particular, zk-SNARKS-- zero knowledge succinct arguments of knowledge. Those were actually developed in part here at MIT. So using these really interesting cryptographic primitives, we're able to maintain financial invariance in Zcash.

There's another system called Solidus, which is out of Cornell, which also achieves this. You can't tell who the participants are. And you can still maintain financial invariants. Solidus uses a completely different cryptographic primitive called PVORM. So you should check out both these papers if you're interested. It's definitely still possible to accomplish this, right?

However, there's still a problem here. Both of these systems hide everything. You get no insight into the economy, into the blockchain, into what's actually happening in the system.

Regulators, in particular, need insight into markets in order to maintain financial stability and in order to protect investors. Even considering non-financial use cases, it's oftentimes the case that people who might be using a blockchain, or distributed ledger, or distributed database want to be able to maintain that certain things are true and to get some insight into the system that they're a part of.

Lack of insight can actually have devastating effects, as we saw with toxic assets in 2008. One part of the problem, one contributor to the crisis, was the issue that people didn't really have confidence or insight into exactly what assets banks had on their books. It was a bit confusing.

And surprisingly, this happens to this day I think there was a case with Dow Chemical where they didn't actually know how many outstanding shares that they had issued. They just didn't know. And they had to give out a dividend, and they gave out too much money or too little money or something like that.

So here are some examples of things that we might want to measure and understand. We might want to get a sense of leverage in the system. So, the participants who are holding assets, how much do they have en masse? We want to understand exposure. We might want to get a sense of market concentration.

It's pretty useful to know things like, OK, this bank, this asset in particular is very highly concentrated. Or this asset is more distributed amongst the participants of the system. So there are things that we might want to learn about the economy represented by the assets in the blockchain. And that if the entire thing is encrypted or completely opaque, we're not going to be able to really gain this insight. And toxic assets, like I said.

So here's a specific example of something we might want to learn. Here's a bank. Here's an auditor. And the auditor might want to-- let's assume the bank is using a system like zkLedger, a permissioned blockchain. An auditor might want to know how exposed is this bank to a drop in the euro? So if the euro versus the dollar changes dramatically, what's going to happen to this bank's balance sheet?

So in zkLedger, the auditor can ask a question like what fraction of your assets are in euros? And the bank might respond, 3 million out of 100 million. So, 3 million euros, 100 million dollars. Something like that. But just simply asking this question, the auditor doesn't really get any assurance that this answer is correct. And so this is the nugget of the problem that we are trying to solve and achieve.

So this talk presents zkLedger. And it's a private, auditable transaction ledger. The idea behind zkLedger is like other systems. We also hide the transacting banks and the amounts involved. We provide integrity with public verification.

So despite the fact that you can't really see the details of the transaction, anyone, not just the participants, but anyone you want to show this to, can publicly verify and convince themselves that transactions are well-formed and financial invariants are maintained. And an auditor can interactively compute provably correct linear functions over the transactions. So things like trying to understand market concentration, or leverage, or exposure-- they can compute a set of queries over the contents of the transactions in the ledger.

OK, so I'm going to describe to you how zkLedger works. I'm going to give you a brief overview of the system model. And then I'm going to start to go into some of the more interesting pieces inside zkLedger. So the commitments that we use to hide things, the way the ledger is constructed.

And I'm going to tell you a little bit about the primitives that we use, which are zero-knowledge proofs. Not SNARKs, which I talked about earlier-- a different type of zero-knowledge proof. But I'll give you a little bit of a flavor of what those things look like. And then I'll show you the performance of zkLedger.

So whether you buy this use case or not, I think that you will see these topics come up over and over again as you're looking at cryptocurrencies. People are quite interested in zero-knowledge proofs and how they can use them to provide privacy inside of cryptocurrencies. OK, before we jump into the system model, any questions? OK, let's go.

So let's start with the system model. So here's the system model. We have a set of banks. These banks have generated public keys and secret keys, which they have. And these banks are working together to construct a ledger.

And the way that this ledger works is that they are determining transactions amongst themselves. So they're deciding, OK, I'm going to transfer this amount to that bank. And then once they've decided on a transaction, they append that transaction to the ledger.

And again, remember, this ledger could be maintained by a third party. It could be maintained by the banks themselves on a blockchain. They could be using a public bulletin board. They could be posting transactions to Twitter. It doesn't matter. We're not really concerned about that right now. The point is that there is an append-only ledger that's orthogonal to our techniques-- how exactly it's maintained.

An auditor, a third party auditor, can obtain correct answers to questions based on ledger contents. So note that auditing is interactive. It's not the case that someone can just take this ledger and compute things over it. They can't do that. There's not enough information here to reveal the answers to queries. The auditor has to actually talk to the participants in the ledger.

So you might say, well, we didn't trust the participants to give the auditor the right answer. Why would we trust them to answer the auditor at all? Well in the setting that we've set up for ourselves, if they don't answer the auditor, they can go to jail. OK? So at least you can tell whether the bank is answering you or not. If the bank were to lie to you, you might not be able to tell whether-- this allows you to be able to-- if you get an answer, you know that that answer is correct.

So the auditor will query the bank and ask a question similar to what we were asking before-- what fraction of your assets are in euros? The bank can respond. But this time, instead of just getting an answer, the auditor also gets a small part of a proof from the bank and can confirm that that proof is correct based on these opaque transaction details in the ledger.

So the auditor still needs to talk to the bank, but using the bank's response, they can compute a proof that shows that this answer must be correct. OK? So auditing is interactive. Like I said, it's possible a bank might choose not to respond. That could happen.

Also I want to point out that if the auditor is constantly auditing this ledger-- is constantly asking how many euros do you have? How many euros do you have? How many euros do you have? Then if that number changes, it's reasonable to assume that that change happened in the last transaction appended to the ledger. Right?

So if the bank continually answers the auditor's questions, this will leak transaction contents. This doesn't provide what's known as differential privacy. OK? And so the setup here is that the auditor and the bank should come to some agreement about the appropriate level of frequency of auditing such that both the auditor gets the data that they need, and the bank maintains the privacy that it needs to maintain.

OK, so as I said before, zkLedger supports any linear function over transaction values. So we can do ratios, sums, averages, variance, skews outliers. By outliers, I mean give me all the transactions that have amounts outside of this range. So you want all high-value transactions, you can provably, correctly give that. Approximation-- so orders of magnitude of my trades, without revealing the precise number. Changes over time, and also well-known financial risk measurements like the Herfindahl-Hirschman Index.

So in this talk-- oh, sorry. Yes, there are small amounts of well-defined leakage for some of these measurements. So for example, in order to compute the average, we actually compute the sum and the number of transactions. So we leak a little bit more information than just the average, but it's very well-defined.

And in this talk, I'm only really going to talk about sums. But our paper, which I'll put a point or two on the website, explains how to do more complex things. And everything's built on the sums primitive.

So let's talk about the security goals. So what are we trying to achieve with the system? So first of all, we're trying to achieve privacy. We want to make sure that the auditor and non-involved parties cannot see transaction participants or amounts. So I don't know if I made this clear before, but it's not just the case that this third-party auditor can't see who's in transactions or what they're transacting. Anyone who's not actually involved in the transaction, receiving or sending assets, also can't see inside of a transaction.

Completeness. So what do we mean by completeness? What we mean by that is that banks can't lie to the auditor. They can't say that they have 5 million euros when actually they have 3 million euros. But also, the bank can't selectively leave out things. OK? So you could imagine some ways of constructing this in which, if a bank was really clever, they could just not include some of the transactions that they had received, and thus change their answers to the auditor. But one of our security goals that we achieve is that it's impossible for a bank to omit transactions when responding to the auditor.

We also are looking to achieve integrity, so banks can't violate financial invariants. We assume that banks are malicious. But honest banks should always be able to convince the auditor of a correct answer. So we don't want to have a situation where a malicious bank screws up the ledger such that an honest bank can't respond to the auditor because the ledger doesn't make sense. So we also want to maintain that.

And then progress. We also want to make sure that a malicious bank can't block other banks from transacting-- that they can't hold up the entire ledger. And I think as I describe the design, it'll become clearer why these things were issues that we were worried about. And it's part of how we designed the system to address these problems.

So, the threat model. And the threat model describes what kind of attacks that we are trying to protect against and what kind of attacks we don't protect against. So banks might attempt to steal or hide assets, manipulate balances, or lie to the auditor. Banks can arbitrarily collude, so they can try to work together to do these things. And banks or the auditor might try to learn transaction contents. So it's not that they're passive. They might actually be trying to actively learn what's happening in the ledger.

What's out of scope for this work is a ledger that omits transactions or is unavailable. As I said, we're agnostic to the ledger implementation as long as it's available so it accepts transactions. If the ledger doesn't accept transactions, we can't provide our goals.

We also don't protect against an adversary who might be watching network traffic. So if there's an adversary who's snooping on the network and sees that two banks are communicating a lot, it's reasonable to assume that those two banks are involved in all the transactions. There are other techniques to deal with these problems. And then also what's out of scope is banks leaking their own transactions. If a bank leaks their own transactions, we throw up our hands. Sorry, we tried to give you privacy.

So that's the system model for zkLedger. That's the threat model, those are our goals, that's what we're trying to achieve. Now, let's get into the design a little bit.

So we're going to build up the design based on this example. So here is an example of a public transaction ledger. And we're going to slowly start to mask things in this ledger until we end up with zkLedger's design.

So we have four transactions here. I'm only using one asset, euros. You'll notice that the first transaction addresses the question that you asked, which is, well, where do funds come from? And here we have a special depositor, who is injecting assets into the system. This depositor might be the central bank, for example.

A lot of systems are actually designed this way. So Stellar, I think Chains, Blockchain, Ripple-- these systems are all designed with the idea of a special depositor who's allowed to create their own asset and insert it into the system. So here, maybe the depositor is the European Central Bank. Maybe it's some other bank, who provably has a lot of euros in their account and puts them onto the ledger.

So there's a depositor who's granting 30 million euros to Goldman Sachs. And then there are three transactions. And I think it's important to get a sense of what these transactions are. So Goldman Sachs is transferring 10 million to JP Morgan. And then JP Morgan, in two different transactions, is transferring money to Barclays. OK?

So let's move on from here. So like I said, the depositor injects assets to ledger, and that's the way of entering transactions. That's the depositor, right there.

Now our goals are to achieve auditing and privacy. What do we mean by that? Well, we shouldn't-- someone looking at the ledger shouldn't be able to tell these amounts except for the depositor transaction, which is always public. In zkLedger, the depositor transactions are public. But they shouldn't be able to tell after that that Goldman Sachs gave money to JP Morgan, and that JP Morgan gave it to Barclays, and what the amounts were. OK?

So in this example, we're going to try to provably audit Barclays to figure out how many euros

Barclays is holding. And again, we want to hide the participants, amounts, and the transaction graph. So what do I mean when I say transaction graph?

I think you guys learned a little bit about a system called confidential transactions a few weeks ago. Confidential transactions does not hide the transaction graph. It hides the amounts, but you can tell the source of the funds.

So here, the money that Barclays got-- so if we were trying to attain privacy in the system, the fact that Goldman Sachs got euros, and Goldman Sachs is the only company that got euros, is public, because depositor transactions are public. But we shouldn't be able to tell that the euros that Barclays got-- obviously they must have been sourced through Goldman Sachs. We shouldn't be able to tell they went through anyone, or how many people they went through, or that it was JP Morgan.

So if we leaked the transaction graph, you might be able to tell that there was an entity that the money went through before it got to Barclays. So when we say we want to hide the transaction graph, that's what we mean. We want to be able to hide the flow of funds, how they move through the system.

We hide the amounts by using a cryptographic primitive called Pedersen commitments. So Pedersen commitments, as you learned, are binding and hiding commitments to a value. And the way that they work-- and I'm terribly sorry, but I use slightly different notation than Tadge. So Tadge uses additive notation-- multiplicative notation, and I use exponential notation. These mean exactly the same thing. I will very quickly write down what they are.

So in Tadge's notation, G and H are generators in some group. And if you have some value and some randomness, then you would write a Pedersen commitment as that, OK? Now, whether it's r times G, or v times H isn't important. The point is these are two generators of a group, in this case, an elliptical curve group. And in Bitcoin, secp256k1.

It doesn't really matter what those generators are. You just pick any two. You could call this G1 and G2, if you wanted to. And so this is the way that Tadge would write it.

The way that I write it-- and I'm sorry about this, but it's good that you guys get exposed to both ways, because you'll find both ways in the literature-- is in exponential notation. So I would write g to the v, h to the r. And they're multiplied together instead of added together. OK?

So this is just purely a notational thing. It's just, instead of the multiply operator here, we're doing exponentiation. Instead of the plus operator here, we're doing multiplication.

So, Pedersen commitment. The way that you commit to a value v is everyone has chosen these generators. The generator choice is system-wide. This isn't something you pick at the time you make a transaction. The generator choice is system-wide.

However, the r, the randomness, you do choose fresh each time. You pick any randomness each time. And the way that you commit to a value is by producing this value right here, which is the generator raised to the value times the second generator raised to the randomness.

And what this serves to do is it's binding. It's very difficult to come up with, later, a different and a different r that ends up equaling the same commitment. And part of that is based on an assumption called the discrete logarithm-- the hardness of the discrete logarithm problem.

And also, importantly, these things can be homomorphically combined. So if we have these commitments to these different values, and we multiply them together, well, when you multiply things that are raised to an exponent, you end up adding the exponent, right? So by multiplying these things together, we end up with a valid commitment to the sum of the values in the individual commitments that we multiplied together. So we can homomorphically combine the commitments. It ends up summing in the exponent. And we end up with a valid commitment to the sum of the values.

Another nice property of Pedersen commitments is that they are very fast to create, combine, and verify. And perhaps surprisingly, we can actually achieve all of the auditing functions that I showed you earlier just using Pedersen commitments.

So before we move on, questions on Pedersen commitments? I know that you guys learned about this in a previous class, but they're pretty important. They're used in confidential transactions as well to prove that the sum of the outputs is less than or equal to the sum of the inputs, if you guys recall that. So any questions?

AUDIENCE: When are you making this commitment?

NEHA NARULA: Ah, great question. You're making this commitment at the time that the transaction is created. And I'll get into the details of who creates the transaction, and what do they choose, and how does that happen, and how do we make sure they don't screw it up.

AUDIENCE: So when you add a measurement to this ledger, it might add a number of commitments that people make...

NEHA NARULA: Well, so measurements happen on top of the ledger. So we can actually do all of our measurements basically just on those commitments. We don't have to add anything to ledger. OK. So, yes, like I said, we can achieve all the auditing functions with Pedersen commitments. And I'll go into a little bit of detail. OK, great.

So we can use Pearson commitments to hide the value. So again, by using these commitments, you can't-- looking at this right here, if you don't know r, you can't guess-- if you don't know v and r, you can't-- if you don't know r, you can't guess what v is. It could be anything. So that's what we mean when we say that these commitments are perfectly hiding.

So we also want to hide the participants in the transaction, obviously. And we're going to do that using other techniques. Just hold that in your head for a moment. We're going to jump to that.

But first, quickly, let's take a look at how auditing might work given that we're using these Pedersen commitments, OK? So assume that we have hidden who the participants are in this ledger through a technique that I'm not telling you about, yet. We're hiding the values in the transactions that are not inject-- not these special transactions that are injecting assets or removing assets, but just the transfers. We're hiding the values using Pedersen commitments, OK?

So let's look at how auditing might work. So an auditor will ask the bank-- in this case, it's Barclays. So remember, Barclays received euros in two different transactions, and should tell the auditor 3 million. The auditor asks Barclays, how many euros do you hold?

Barclays can convince the auditor that 3 million is the right answer by doing the following. OK? Barclays can say, hey, auditor, it's 3 million. OK. And what I'm going to do is I'm going to actually open up the combined commitments for my transactions. So I don't actually have this on a slide here, but I will write out what this means. OK?

So we have two transactions here and two commitments. And one of them is g to the 1 million, h to-- the let's just r1. And then the other is g to the 2 million, h to the r2. OK? These are the commitments in the ledger. Barclays wants to convince the auditor that it has 3 million euros.

It does so by-- when you multiply these two things together, and the auditor can't see what these two things are, but this is-- or actually, I think it's comm 2-- 3. OK, comm 3 times comm 4. Then when you multiply these two things together, you get g to the 1 million plus 2 million h to the r1 plus r2, OK?

This is 3 million. If Barclays gives the auditor 3 million and r1 plus r2, the sum, than the auditor can confirm that if they raise g to this and h to this, that it's equal to the multiplication of the two commitments on the ledger. And the auditor can-- given that these two things are equal, then the auditor knows that the bank in fact did commit to two commitments that sum to $3 million.

It doesn't know if that was 1.5 and 1.5, or 1 and 2 million, 999-- it doesn't know what those individual values were. It just knows that the sum is correct. So, questions about opening commitments? Yes?

AUDIENCE: The sub-3, sub-4-- those are the IDs?

NEHA NARULA: Yeah, those are 3 and 4. Sorry, I'm using different numbers. They're r1 and r2. But the comm 3 and comm 4 are-- this is literally a elliptic curve point. So the point is posted to the ledger. Yes?

AUDIENCE: Wouldn't 4 million and minus 1 million also do that?

NEHA NARULA: It would, yes. And we'll talk about how we address that.

AUDIENCE: So just to confirm, the bank provides both the 3 million number and--

NEHA NARULA: And r1 plus r2. Yes. And those two pieces of information are enough to convince the auditor that 3 million does in fact correspond to the commitments on the ledger. Yes?

AUDIENCE: [INAUDIBLE] so you will always get the sum and the number of transactions?

NEHA NARULA: Sorry?

AUDIENCE: You will get the number of transactions?

NEHA NARULA: Who's you?

AUDIENCE: Oh, [INAUDIBLE] as in the auditor.

NEHA NARULA: The auditor will get--

AUDIENCE: They will know the number of transactions?

NEHA NARULA: In the straw man example that I've done, it does look that way, yes. I'm going to build up to how the auditor won't know the number of transactions. Did you have a question?

AUDIENCE: So as a user, you are recording all of your randomness? You're responsible for tracking all of your randomness?

NEHA NARULA: You're responsible for tracking your randomness. And you're responsible for tracking your own individual details. You have to keep that in your own databases. Yes?

AUDIENCE: If somebody found your randomness--

NEHA NARULA: If someone found your randomnesses, then they could guess-- they could backwards-compute the values of your transactions. Yes. OK.

AUDIENCE: How do you define the timing?

NEHA NARULA: The time-- getting to that. So we'll get to that. So just in figures, showing what we did there. The auditor can look at those two commitments, compute the product for themselves, and confirm that this 3 million lines up with what they see.

However, this has the problem that it reveals the transactions in which Barclays was involved, right? Barclays is saying, hey, multiply these two things together, right?

And not only that, Barclays could lie. OK? Barclays could say, oh, actually, I only have 1 million euros. Here's the transaction that proves that I have 1 million euros. I will just give you our one. I will just point to this transaction. Don't worry about the other transactions. I'm not involved in them.

The bank, Barclays, could leave out relevant important transactions in this straw man design. OK? And importantly, this would match up, right? The auditor would say, OK, you've given me-- Barclays could just give-- could leave this part out entirely. Could just give 1 million and r1, and the auditor would say, why, yes, that does match up to commitment three. Great. Good work. And they wouldn't even know that there was another commitment out there that they should be concerned about.

So how do we address this problem? And this, I think, was the key insight in the design of our system. In zkLedger, a transaction actually has a commitment for every participant in the system. So let's talk about that for a second.

So depositor transactions still basically look the way that they did before they're public. But transfer transactions now look quite different. We don't actually have the names of the participant in the transaction. The names are implied by the values of the commitments in that participants column.

So now we're looking at a transaction as a row with multiple columns. And there's a column per participant in the system. If a bank is not involved in the transaction at all, then their commitment is to the value 0. If the bank is involved in a transaction, and it's doing the spending, then its value-- sorry, depositor transactions. Yes.

If a bank is involved in the transaction, it's doing the spending, it commits to a negative value. And if it's doing the receiving, it commits-- sorry, not it, but the creator of the transaction, commits to a positive value. So the way that you transfer $1 million from one bank to another is committing to a negative value in the spenders column-- here, JP Morgan-- and committing to a positive value in the receivers column-- here, Barclays.

And again for non-involved banks, entries commit to 0. But one thing I want to stress-- commitments to 0 are completely indistinguishable from commitments to any other value. And commitments to negative values are indistinguishable from commitments to positive values.

So someone looking at this, just looking at these commitments posted to the ledger, has no idea which ones are 0s, and which ones are negative, and which ones are positive. Yes?

AUDIENCE: Does this mean that each bank has to keep track of every transaction that is done in the ledger, and produce the random numbers, and keep track of those numbers, too?

NEHA NARULA: Great question. So, yeah, one design might have it be the case that every bank actually has to be involved in every transaction. We managed to design zkLedger so that the spending bank can produce one of these without actually involving any of the other banks. And I will describe how that happens.

OK, and so, given now that we've said, essentially, every bank is involved in every transaction, this actually makes life easier for the auditor. Because now when the auditor audits a bank, they audit every transaction. So they audit the entire column for a bank.

So here, when the auditor is auditing Barclays, they won't just look at whatever transactions Barclays tells them to look at. They will look at every transaction. OK? And this is where-- I think someone asked about timing, or how do you audit a range of transactions? The auditor can actually tell the bank, oh, I just want to audit from this point onwards. Or it can say, I just want to audit your transactions for the past month, if it wants to.

That doesn't mean that Barclays can leave out transactions, because there's supposed to be a time stamp column in here as well. So you can see when transactions happen. That's public.

So what happens here is the auditor is going to look at everything. They're actually going to multiply three commitments together, because there's three commitments in that column. And it doesn't know which ones are 0 and which ones are not 0. But the 0 doesn't contribute to the sum.

And so a malicious bank can't actually lie to the auditor, because in this case, the bank is not going to leave out commitment four. And in fact, it's also going to include commitment two. So the bank is going to have to include the r primes in here to get all of this to work out correctly. It's going to have to include everything.

So this next-- I'm actually going to skip a little bit of this, because I think we're running a bit low on time. And I do want to get into what the proofs look like. So I'm going to skip this averages part. I'm going to go back to our security goals.

What I just described to you, the ledger table construction, allows us to achieve our first two security goals. Privacy-- because these commitments are perfectly hiding, no one can see transaction participants. They can't tell who's committing to 0, and who's not committing to 0. Banks can't lie to the auditor, because the auditor audits every transaction for a bank.

But what I haven't figured out yet is, well, how do we maintain financial invariants in this scheme? How do we make sure assets aren't created out of nowhere? How do we make sure that whoever is spending actually consented to the spend? And how do we construct this transaction that involves every bank when we assume banks are malicious, and we don't want one bank to be able to hold up everyone from transacting?

So our solution to do that is to use a set of what are called NIZKs-- non-interactive zero-knowledge proofs. So what are NIZKs? NIZKs are short binary strings.

True statements have proofs. So, if a statement is true, it should have a proof indicating that it's true. False statements only have proofs with negligible probability. And proofs don't reveal why they are true.

So we're going to use NIZKs to prove things like the person who knows the secret key for this public key consented to this transfer, without indicating which bank it was that consented to the transfer. We're going to use NIZKs to prove things like this bank has the assets to spend, without revealing how many assets that bank actually has.

So zkLedger uses a set of carefully crafted NIZKs. And to be clear, NIZKs are pretty old. They were invented in the '80s and the '90s. And most of them are based on the hardness of the discrete log problem in elliptic curve cryptography, again.

So here are the properties we want to achieve with transactions. We want transaction validity. We want to make sure whoever is the spender, whoever has a negative amount in their column, actually consented to the transfer. And again, remember, in cryptocurrencies and things like that, where it's OK to reveal who the person is who's spending, we use a signature. In this case, we'll use proof of knowledge of secret key. So there's a consent NIZK.

We want to be able to prove that whoever is spending actually has the assets to transfer-- that they're not going negative. And we use an assets NIZK. And we want to make sure that assets are neither created nor destroyed, so for that we use a balance NIZK. We want to show that things work out, and you didn't create more euros than you were supposed to. The only place where you can do that is in the depositor transactions, the public depositor transactions.

We also want to make sure that honest banks can make progress. And in order to do that, creating a transaction in zkLedger is non-interactive. The banks don't have to interact to create a transaction.

This is exactly how systems like Ethereum and Bitcoin work, if you think about it. I can't stop you from sending me Bitcoin, right? If you want to send me Bitcoin, you construct a transaction that sends me Bitcoin. You submit it to the blockchain. I have nothing to do with that.

Offline, we might have agreed that the reason you're sending me Bitcoin is because I'm selling you my car or something like that. But to actually create the transaction that spends, you don't have to talk to the other participants. And we use a consistency NIZK so that the person creating the transaction can prove in a publicly verifiable way that everything matches up correctly, that all of these proofs are correct, and that all of the banks are going to be able to later respond to the auditor-- be able to respond to the auditor as necessary.

So to give you a little bit of insight into why this consistency NIZK is necessary, over here, I was speaking as though the bank knew all of the r values, right? Actually, in zkLedger, if a bank is not involved in a transaction, if their commitment's to 0, they don't know their own r value, because some other bank has created that.

So if we go backwards a little bit to here, Barclays wasn't involved in this transaction at all. And in fact, Barclays-- so Goldman Sachs created this transaction. JP Morgan created this one and this one. Barclays had nothing to do with them. They weren't involved at all. And we can't assume that Barclays knows what the rs are in its column.

Yet the way that I told you that Barclays' response to the auditor is by revealing the sum of the rs. So that's actually not how zkLedger works. They don't actually reveal the sum of the rs.

The proof is a little bit more complicated. The rs are encoded for Barclays in a way that it can't see what they are, but it can use them. And the consistency proof is to prove that that encoding was done correctly.

So again, you can see the paper for details, but I'm actually going to tell you a little bit about these proofs and what they look like, OK? So, consent. The consent proof is proof of knowledge of secret key, sk. So what this means is that if the value in the commitment in this entry is negative, then there should be a proof of knowledge of secret key to indicate that this bank has consented to subtracting from their assets.

Also, if the value is negative, you need to create a proof of assets. Technically, you don't need a proof of assets if the value is not negative, but we're trying not to reveal, remember, which bank is involved in which transaction. So I'm going to draw a transaction up here and explain a little bit about where these proofs actually go.

A transaction is a row. And there is an entry for each bank in the transaction. So there is a commitment. So let's-- negative 1 million. And then there is also these sets of proofs. So, proof of assets, in particular.

And every single entry has a proof of assets, even though, really, this is the only bank that's spending and needs to prove that it has assets. The proof of assets is constructed to look like this. It's a new commitment, either to the sum of the values in the column, or, if you're not the spending bank, a commitment to the value.

And it also includes a proof that the value in this auxiliary commitment is in range. So what do I mean by that? I think you guys should have covered this in the Pedersen commitment lecture with confidential transactions, but when you're dealing with cyclic groups, you can wrap around the end of the group. And that's the same as if you hadn't wrapped around. There are many different things that can occupy one point in the space of a cyclic group.

So we need to include proofs that your value is in a range that's expected. And in the case of zkLedger, we need to prove that the value is between 0 and a certain upper limit that is less than the order of the group. And we use the exact same technique that they use in confidential transactions to do this-- borromean ring signatures.

What's kind of exciting is that, a few months ago, there was some work done on a faster, smaller, more aggregatable version of range proofs using a system called bullet proofs. So as you'll see in the evaluation, this sounds like such a silly little thing. Oh, yeah, also prove that the value is in range.

This is like 90% of all of the space and work in zkLedger. These range proofs are the bane of our existence. They take up a lot of space. They're very slow to create and verify, but they're necessary. So making them faster is a really amazing thing.

Then the next thing is balance. And I think this one is illustrative, because it shows that a proof-- sometimes you have a proof not by creating an additional binary string, but by the way you choose things in your transaction. So the way that you prove that no funds are created or destroyed is as follows.

So these are-- remember these commitments are g to the 0. Let's say h the r1, g to the negative 1 million, h to the r2, and g to the 1 million, h to the r3. Now, given that we have these commitments, how can we prove to someone that no assets were created or destroyed? Well, what does it mean for no assets to be created or destroyed?

It means that the sum of the vs is equal to 0, right? That's what it means for no assets to be created or destroyed. How can we prove to someone that, given that all they're seeing are these commitments, that the sum of the vs commit to 0?

Well, let's just make the sum of the rs also commit to 0. So if the sum of the rs also commits to

0, then when you multiply these things together, you're going to get g to the sum of the vs, h to the sum of the rs. And if both of these things are 0, you get g to the 0, h to the 0, which should equal what? One, exactly.

So we choose our rs very carefully, so that the auditor, or whoever is verifying this, can just multiply all the commitments together and verify that they equal 1. And if it equals 1, that means it would have been very difficult to choose vs such that you could make this come out to 1 without it actually coming out to 1. So that's the proof of balance.

It's not an actual proof. It's not a short binary string. It's literally choosing the exponents in a clever way to show that we can maintain the invariant that we want. Does that make sense? Questions about these proofs? And I didn't go into proof of assets very much. It's the most complicated one. But you can check out the paper.

I see some confused faces. You don't have to ask about the proofs.

AUDIENCE: Couldn't you just choose both of the exponents? So if you're choosing rs in a certain way, can't you choose them in a certain way and then choose payments so that they sum to 0?

NEHA NARULA: Well, we're trying to prove that the payments sum to 0. That's what we want. Because that means that assets weren't created or destroyed. And remember--

AUDIENCE: You're generating-- you're creating those random numbers in a way that they sum to zero. So if you create them in a way that doesn't sum to 0?

NEHA NARULA: Then every-- ah, OK. So this is an important--

AUDIENCE: Change the transaction so that the sum of the rs and the sum of the values sum to 0?

NEHA NARULA: So basically, what you're saying-- I think what you're say-- OK, you could be saying one of two things. Number one, what if you choose your vs and rs such that it doesn't sum to 0, right? What if you produce an invalid transaction? And the answer is that everyone will reject it.

Sort of similar to Bitcoin, if you don't have a valid transaction, good for you. But it's going to be considered invalid by everyone in the system. And everyone's going to skip it and ignore it. It's not going to be part of what happens, right? So this is part of the proof of validity that all of the participants have consented to.

The second question you might be asking is, what if these don't sum to 0, but I still manage to figure out how to make this equal 1, right? That's what you-- OK, so that's the question you're asking.

Based on the assumption that the discrete logarithm problem is hard-- this is too hard to figure out. You literally cannot find-- there is an assumption here. These are two generators of the group, right? So that means the g is equal to h to the x, that there's some way to produce one generator from the other generator.

There is an assumption here that figuring out what that x is is nearly impossible. So you don't know what g is in terms of h. And given that you don't know this relationship between g and h, you can't figure out how to produce two exponents that sum to what you want them to sum to.

So is the hardness problem that these systems are based on. If you could do that, if you could figure out a sum of rs such that it gave you a 1 with the vs you wanted, then you would have broken the discrete log problem. Yes?

AUDIENCE: So who chooses the [INAUDIBLE] we know that they don't know the relationship [INAUDIBLE] they maliciously gave those two?

NEHA NARULA: Great question. So the answer is that in our system, the way that we choose g and h is by taking the SHA-256 hash of 0 and 1. Those are numbers such that-- they're sufficiently random, and they show that if you took a hash of 0 and 1, you would have no relationship between those two. And everyone uses the same g and h, right, from the beginning. Any other questions?

OK, so we're running low on time, but I do want to go through the evaluation really quickly, because-- so this system, this paper, was presented at NSDI in April, which is a systems conference. And I think part of the reason that the systems community liked it was because it's surprisingly fast. You might have heard that computing unencrypted data is too slow. It's infeasible. You can't do it.

Well, the answer is, actually, if you understand the set of computations you want to do really well, you can structure things using things like these NIZKs and the discrete log problem in such a way that you can do that computation actually pretty fast. So zkLedger is written in Go. It uses an elliptic curve as the group-- secp256k1, which is the same elliptic curve that Bitcoin uses.

We use range proofs to prevent overflow, like I said. We use the same ones as in confidential transactions and confidential assets. And it's around 4,000 lines of code. It's mostly cryptography functions. We don't actually implement our own ledger, because we're ledger agnostic.

So in the evaluation we want to understand how fast is auditing, first of all? Right? And then second, the construction I described to you, you might think is very slow, because every bank has an entry in every transaction. So how does this actually work in practice? Can we get away with such a construction?

So first, let's talk about the auditing. Auditing is really fast. So here, we are auditing four banks, and we're doing a measurement called the Herfindahl-Hirschman Index, so we're trying to understand market concentration for a given asset. And what's happening here is the auditor is keeping up with the ledger and keeping these things that we call commitment caches.

So because of the design of our system and the fact that we're using these Pedersen commitments-- which, we can actually maintain rolling sums going down of these commitments and use them when needing to answer the auditor or confirm that a bank has the correct answer. And so this means that auditing for banks is milliseconds. It's a few milliseconds to audit the banks.

And we see here on the x-axis we have the number of transactions in the ledger. The reason that auditing a bank is independent of the number of transactions in the ledger is because of these commitment caches. We can do it really fast, assuming an online auditor who's keeping up with things. And part of the reason for that is because of our design. It's very amenable to caching.

Now, if the auditor is not online and hasn't been maintaining commitment caches, then auditing is order the number of things in the ledger. Here, we have a ledger with 100,000 entries. It takes about 3.5 seconds to audit that. Still not so bad, really. And it's pretty linear, so you could imagine, well, if you had a million entries, it would be 10 times that. It would be about 30 seconds. Still pretty reasonable, actually.

A question some people ask is, well, do we have to maintain this ledger forever? How do you clean up the ledger? You can create a snapshot of the state in the ledger and get rid of the past, assuming you know the set of auditing queries that you're going to do moving forward. We don't implement this, but this should be possible. Right, so if you wanted to do 100 million rows, it would work out to be about an hour.

So here is a graph which shows how long it takes to create and verify transactions in zkLedger. And this is varying the number of banks. And again, we see this is linear. The more banks, the more entries per transaction, the more of these commitments, the more range proofs, quite honestly, is the problem. Because there's a range proof per entry.

And what we see here is that with 10 banks, it's about a second or 800 milliseconds to create a transaction, which is not unreasonable. Again, this achieves the multiple per second things we were looking for, depending on the number of banks. However, it is linear in the number of participants in the system. So something like this is not really suitable for a setting like Bitcoin, where there are thousands, if not millions, of participants in the system.

This graph gives you an idea of how all of that cost breaks down. So these are the components in the proofs. And again, I think the thing to get from here is that range proofs are the entirety of the problem. So this is the number in a transaction for k participants.

So this is each slot. There are two different commitments. The first commitment, and then the auxiliary commitment. There's two different consistency proofs. There's a disjunctive proof, and there's a range proof.

And yeah, the commitment is one elliptic curve point, which in Go is a big int-- two big ints. And so that's 64 bytes. This is something that is actually highly compressable. We didn't actually-- oh, OK. Yeah, I'll get to that point in a moment.

But note here that the range proofs are slower and larger than everything else combined. They are the entirety of the cost of the system. So a faster way of doing range proofs would really help a lot. It would make us be able to handle many, many more banks.

So this is the cost in a transaction per participant in the system. [AUDIO OUT] One thing that I want to point out is that this is all very highly parallelizable. And like I said, we're using Go's big int construction, which is not really very optimized at all.

So I think that there are a lot of opportunities to make this faster. So I think if we put in some engineering legwork, we could create a system that was maybe twice as fast as what we have now, and could handle maybe 100 banks.

So really quickly to wrap up, related work. If you're interested in this topic, there's a lot of different papers to read about and study. This is all-- confidential assets kicked off this line of using Pedersen commitments for privacy in cryptocurrencies. Zerocash was the origin of using zk-SNARKs. They don't really support auditing in any fashion.

Then there's a couple papers that are really interesting, one from Andrew Lowe, actually, who's in the economics department here at MIT, on using secure multi-party computation to attack this problem. The issue with that sort of design is that it doesn't actually have the completeness guarantee that we have.

So if the ledger really represents the assets in the system, and the assets that you're trying to audit, then there's no way for a participant to lie in a multi-party computation. They can use the wrong inputs. So those systems provide secrecy, but they don't provide completeness.

And then I mentioned Solidus before, which I think our techniques would apply to really nicely. And then there's a paper, a design for doing some other sorts of accountability stuff with cryptocurrencies, like proving you paid your taxes, or proving that money didn't go through some blacklisted thing using zk-SNARKs. But that's not really a system, it's a design. But I think that's also a really promising area to consider.

So, future work. We want to consider more applications besides just large investment banks trading securities and assets. Can we do something with clinical trials? Can we do something with supply chains?

I think that these techniques can be applied to a lot more areas. And I'd love to hear from you guys if you have ideas on that, or if you're interested in working on that and want to do a UROP, or maybe even an M.Eng., that's definitely something that's on the table.

We get surprisingly far with Pedersen commitments, but if we included more interesting types of cryptographic primitives, we could probably get more functionality. So what's the type-- we know, given that we might want to apply this to different use cases, what types of functionality might we need?

There's someone at the Media Lab here who is interested in running algorithms on research trials. And secrecy and privacy is really important. I haven't even looked at his machine-learning algorithms yet, but can we run machine-learning algorithms on Pedersen commitments, or do we need something else?

Also, another really important area of future work is to optimize the implementation. Like I said, I think we get at least twice as fast, if not more. And in conclusion, yeah, if you want to work on any of these things, please let me know. If you're interested in using cryptographic primitives to achieve privacy with interesting functions, let me know.

And this is the website for zkLedger, which at the moment is really just the paper. We have a code base which is too embarrassing to be released out into the real world, but we'll get there shortly, once we clean it up. So that's it. So thanks for listening. I hope you found it interesting.

[APPLAUSE]