

[SQUEAKING]

[RUSTLING]

[CLICKING]

**ROBERT
TOWNSEND:**

So today, I'm going to give this application lecture using encryption. So Sam and I have been working on market making, combining economics and computer science, and Sam's going to present that on Thursday. Looking forward to it.

All right. So today, designs of financial infrastructure utilizing encryption, and I'm going to go through three examples today varying in detail. One is an auction, which is a familiar setup, but we'll be encrypting it. Then I have this what I call hybrid credit and insurance, having to do with insurance against balance sheet shocks in various contexts. And then finally, a market implementation using encryption.

So first, we'll do the auctions. So auctions are used frequently, but they're typically organized by a third party. And the example here is Mortgage Capital Trading's use of a trade auction manager. Worrying that perhaps this slide was a bit out of date, I checked this morning on the web, it's alive and well. So Mortgage Capital Trading is still being used, as are many of the others.

But trusted third parties like Mortgage-- the trade auction manager there are not always needed, and in fact, they're not always beneficial. So what are called these "bid-wanted-in-competition" dealers are asked to bid on a list of securities. Not necessarily securitized mortgages in the first example, but much more generally.

And there's a typical, as you could imagine, abuse, or at least potential abuse, where the auction manager is on the phone, he's the seller of the asset, so he has an interest in making the price high, so he calls buyer A, he says, I prefer you, my friend, but buyer B just offered a slightly higher price.

So if you can just make some effort, the good-- is the security is yours. But at the same time, he's saying exactly the same thing to buyer B for this example, just say there are two buyers, which has the intent of propping up the price.

And of course, neither A nor B can check on what this manager is saying to other clients, and neither can they check on the bids that are coming in in order to verify the claim of the auction manager that, in this case, buyer B was a little bit off. Not to mention, it's potentially improper to be doing that in the first place.

It's not the only example. China had, at one point, these P2P platforms, and there have been articles written about what investors were being told or not, and one of these articles pretty much documented a kind of cream skimming where AI was running over securities that could be put in the securitized pool, but some of them-- some of the prime ones were not put in the pool and were held by individual investors, but other investors were not informed that this was going on. And China eventually got rid of these P2P platforms altogether.

OK. So how do we do an auction without an auctioneer? The answer? Use the tools that we're just learning, homomorphic encryption and multi-party computation. And I'll describe, in the slides that follow, two ways of doing this, and both are instructive. And we're going to be focusing not on the seller with the asset, but the messages that go back and forth, so they're not communicating with the seller. So the seller is kind of in the background here.

First scheme, each bidder has his own server. We haven't spent that much time on infrastructure, but it's, of course, crucial in any application. So in this case, they're going to use their own servers. There are going to be two bidders. And they're going to encrypt their messages, which are their bids, and those bids are going to go-- encrypted messages are going to go back and forth among these two or more bidders.

There's no third-party server. There's no contract node. In particular, each buyer sends his encrypted bid with public-private key pairs, as in homomorphic encryption, to the others, and for the multi-party computation part, they send the results of these encrypted messages to each other.

The second scheme-- this is just an outline slide-- is with a contract node. So all communication goes to a third-party server where the contract node resides. So communication is with these encrypted messages. So the, quote, "server" doesn't see the private values. The server is doing in the code what the agents were doing in the first scheme.

It's hard not to refer to this contract as a person, but it isn't, it's just code. I think we call it a pseudo-agent in the subsequent slides.

OK. So we've got these in the first scheme, two agents A and B, and they want to see whose bid is higher and who is lower without relying on the trusted third-party seller, but without revealing to each other the exact value of their bids. And I will forewarn you from the get-go that 2 turns out to be special, but the idea of the example does generalize. And I'll show you where the special revealing feature happens at the end.

OK, so this BFV encryption algorithm, which is ring learning with error. I showed you one slide last time with the polynomials and the coefficients of the various orders of terms being drawn from integers and so on, so this is that scheme. There are private and public keys, of course. The public key is this a , Δ . The secret or private keys are the s , e terms.

But again, this is in the space of rings, which are polynomials with these integer coefficients modulo some first-- some order polynomials. So we're going to take polynomial operations of addition, multiplication, and so on, and always apply this quotient to get it down to, say, an $n - 1$ degree polynomial.

Then this error term is like a normally distributed random variable that's added on for extra noise and security. And I'll show you-- remind you, really, more on that momentarily, but we end up with these private-- sorry, public secret key pairs. a , Δ are common across the agent. That's the public part.

They're going to draw these public-private keys-- the public part is going to be published on their servers and broadcasted more generally, possibly. And the private key parts, secret-- see how I keep mixing up my p's? The secret key part is s_i and e_i .

And you'll see in a minute, when I give you the example, specifically where a and Δ enter and where's s_i and e_i enter. It's only the secret keys that are indexed by the agent, obviously, because the public key is common.

So here's the example of why we need the noise. This is just matrix-- a system of linear equations. We got a 7-by-4 matrix multiplied by a 4-by-1 column. So we're going to-- the answer is going to be a 7-by-1 column.

And the secret is supposed to be what's in red there. If this was a non-singular square matrix, we could just invert and multiply by both sides. That's not necessary. Gaussian elimination, which works row by row, can also solve the system of equations, so the red secret would be revealed.

AUDIENCE: What is the blue thing here. What is-- does that map to our alpha, delta--

ROBERT TOWNSEND: You're like seeing the blue stuff and you want to-- and we want to prevent outsiders from deciphering the red.

AUDIENCE: I see. OK.

ROBERT TOWNSEND: So we add in a 7-by-1 column vector with noise, and that, in principle, very hard to decipher the red.

AUDIENCE: The blue is not invertible, so--

ROBERT TOWNSEND: It's not, no. But Gaussian elimination can solve this system. I checked it this morning.

AUDIENCE: Yeah. So if I know the blue stuff, and I want to know what the red stuff is-- I see.

ROBERT TOWNSEND: Yeah, yeah. That's the idea. This thing with the noise, I'll just remind you, because I had a slide about MPC last time with the secret sharing. And I drew this analogy to who is the richest person. And I didn't quite pull it off, but you'll see it coming back today.

The main thing is, in that example, one person had a secret, added noise, passed it down the line to the next one, who had a secret, and added noise, and then we summed that up, and we went all the way across the first row at the top with however many players are there, getting the sum of the secrets plus the sum of the noise, which means nothing to anyone because the noise is encrypting everything.

And then we successively took the noise out. And there is a strong sense in which that's exactly what's going to happen in this auction. So it's both a technology reminder of something you've seen before briefly, and wedded with this application.

OK, so now they have a secret key, they've shared the public key, they can start to encrypt their bids. So let's say A goes first, just for the sake of being clear. So A sends-- and this was the first scheme where they each have a server, and they're just sending messages back and forth across their servers.

So A goes first and sends an encrypted message to B, which is ciphertext that A sends, and in particular, you can see the α and δ as the public key multiplying, however, the secret key, private key of agent A, and in this case for the δ , the message-- and let's just, for the sake of argument, if the auction is well-designed, say that it has an incentive to send his true willingness to pay. It doesn't really matter now, that's the bid. And the error is added. Yeah.

So intuitively, this is just gobbledygook ciphertext like a hash message, and B can't interpret it. B has no idea, partly because of the secret key and partly because of the error. And B does the same. B sends his encrypted bid back to A.

What is delta? So in a minute, we're about to set delta to 1. That kind of simplifies the algebra. I did mention last time that the problem with adding error and having multiple operations is that the error term starts to get bigger and bigger.

And so, this countervailing force here is to make delta pretty big instead of shrinking the error, but they're doing a lot of different things at the same time to try to put some bounds on the error term. But everything I'm going to show you with delta equal 1 can be done with a larger delta, it's just, quote, "heavier."

All right. So now each is holding the ciphertext received from the other one. So agent A has a ciphertext encrypted bid from B, and A adds in his or her own secret key pair in an additive way. So A adds in that term, as of a plus e_a , and B, sitting on the ciphertext from A, adds in his or her own secret error terms.

And the ciphertext for A, say, had B's encrypted message, added in his own error term on top like cross-matching. And so after adding in his own secret key pair, this new transformed ciphertext of B is of this form, a of s plus delta m_b plus e .

Now there's no subscripts on the s and e , and the reason is s equals the sum of s_a plus s_b . s_b was in there because B sent the ciphertext to A, and s_a is in there because A just added it in. So that's why it's s . And likewise for the e , the error terms are just adding. OK.

And likewise, B is in a similar position. So you could now just take the difference of these transformed ciphertext. And you get the final answer, which is just this difference delta times the difference of the bids.

Now the only piece we want out of this is to say that they each now know, because they're exchanging the ciphertext, which bid was higher, but it's also true that you know your own bid, and delta is a public key. So unfortunately, we have now revealed everything about the bids to both players, which isn't a total disaster, but in general, we can do better.

Although there is a related message here, which is when we go with more than two agents, it is still possible that a subset of them could be colluding or coordinating and revealing stuff to each other, in which case, they could, if they did that, figure out the bid of, say, for example, the last agent.

So the second scheme uses third-party server and introduces the pseudo-agent, like the contract code. And the third party here is going to be able to decipher the bids without ever having access to the secret keys of any of the bidders. So this is the same notation as before for public and secret keys, public and private keys. Now, in step 1, they don't send cipher encrypted bids to each other, they send it to pseudo-agent C, to the code suitably labeled C.

And the pseudo-agent also gets other information. It asks each of A and B to send an additional ciphertext, which consists of each the sum of the private keys, but again, it's encrypted, so these encrypted messages mean nothing, they're not interpretable, but C is now armed with them. And secrets are maintained here.

So this line has an error. And I was sorting through it yesterday. I could have tried to redo the slide, but I think the error is kind of instructive. So this says c_{tc} is the difference of these original ciphertext. Actually, they should have a prime on them. Harkening back to the first application where the agents added in their own private keys on top of the ciphertext they received, in which case, the math is entirely the same, except that these terms are not there.

And literally, what's going on here is the third-party code is just doing what the agents had done anyway. It receives the ciphertext, it adds on the error terms and the other parts of the secret key acting for each party, and takes those ciphertexts and does some, quote, "algebra" on them to figure out the difference in the bids.

So in some sense on this slide, we didn't need the rest of this if the primes had been put where they were supposed to be, but what's instructive about this line is this adding in of representation of the sum of the private keys. So in more general schemes, which I'm about to describe, at least the English version, this is the-- and this is more general than for two agents.

Why don't I just show you the next slide. So this is generalized MPC where we have more than two agents, and it goes through the sequence. Each agent individually generates a key pair as before. Each key pair has a public encryption key and a private decryption key. All agents submit their public keys to the server. That's common knowledge, that's the a , Δ .

The server combines all the agents' public keys into a single shared joint public key. This new joint shared public key is distributed from the server to all the agents. So we've kind of added these extra steps 1 through 4. And I misspoke a second ago when I said a , Δ . It's as if they each have public keys, and then you're adding up the public keys and sending back now, for future use, a common public key.

So then, with the common public key, the agents can use it to encrypt their private data, generating ciphertext. Each agent sends the ciphertext with its message to the server. This completely hides the agent's data. The server runs computations on all the encrypted data, producing an encrypted result. So that's the FHE part. All the computations are being done in this encrypted space.

The server sends the encrypted result back to each agent. Each agent then uses his or her private key that they generated up there in step 1 to partially decrypt the answer. After they decrypt it partially, they send their decrypted part back to the server. And this is, again, the number of agents without which partial decryption from all the remaining agents will still give a completely encrypted result is a parameter that can be chosen beforehand by the mechanism designer.

So this is, again, very typical of the computer science style here, which is to try to put a bound on nefarious behavior, like how many of these bidders are potentially colluding with one another. And then if you have enough agents, you can basically still keep secrets.

So the final step is the server combines the results of all these partial decryptions, and it's able to produce the decrypted result, and then that's shared with all the agents. So it is kind of a long sequence. This works in general, as I have said now, for more than two agents, and these are typically the way these MPC schemes work across multiple parties.

There is still a lot of messaging going back and forth. I mean, in some ways, the spirit of it shouldn't be surprising. If you remember the encryption lecture from last time where we went to signatures, that was Alice and Bob, and Alice is going to send a secret message to Bob, that Bob has to verify that it's Alex-- or Alice. So they went back and forth with this at least twice, sending messages back and forth to each other.

So this is in that spirit, except it's more than a signature in the sense that there are more than two agents and they're actively bidding on something. OK.

AUDIENCE: So a high-level question. So in theory price option, in theory, the rule of auctioneer is just collecting those prices in aggregate with the successful bidder. So here, I think, is it the case we just get rid of this auctioneer so that we don't need to trust anyone? But theoretically, the results are essentially the same with the [INAUDIBLE] price auction?

ROBERT TOWNSEND: All that was done here was whatever the incentives are, they've submitted their bids whether or not it was the true value, and the example was the highest bid wins. But you could imagine implementing other rules, like Vickrey auctions, where the highest bidder gets to pay the second-highest bid or something like that. All of that, if, in fact, we're going to see, you can rank-order these bids in general.

So there are two ways to try to deal with the increasing complexity that you get with more than two agents. The first uses the two-agent case as a building block and it starts making pairwise comparisons. For example, you could choose a pseudo-agent for the ranking operator applied to two agents, figuring out which of those two has the highest bid, and then you can do that for all possible pairs of agents, and from that, you could deduce the overall highest bid.

That gets into lots of pairs depending on the total number of agents, so that could be a bit onerous to do. And the second tries to economize on the number of comparisons, taking this N where N is a potentially large number and subdividing it into subgroups. So then you can compute the average for each subgroup just by adding up the bids, essentially, the encrypted bids.

And then you have bids individually, so you eliminate from the subgroup all bids that were lower than the average. And you do that for each subgroup. And so you reduce the number of potential winners squishing the dimension of C down, and you do it all over again. So this is actually just involves less comparisons than--

AUDIENCE: More generally-- so this is for auctions, I can see that we can do this, but for more general mechanisms where we have more complicated allocation rules as functions of the types, what can we implement in a way that doesn't require a principal? Do we know what the limits are of this approach? Like, if you have multi-dimensional types, for instance, you can't order them, so then what do you do then?

ROBERT TOWNSEND: Yeah, I only have a partial answer. So it's not completely satisfying, although I think it is true that what you can do depends on the context. What I'm not going to do today is have more than two or three applications and try to put them into groups where they share a common feature within a group and are different across groups. We are not, unfortunately.

But I will show you a second example, and you'll see there the ways to reduce the dimensions are different than they are in this scheme. OK.

So the second application is of borrowing and lending with insurance, and I'll give you three instances of where this comes up. Kenya, you may not recall from the second lecture, has M-Pesa, which is telecom agent, and there are two monies, essentially, in Kenya. We have the Kenyan shillings, the fiat money, and we have M-Pesa, which is this e-credit, so to speak, run by Safaricom, and there are all these dealer agents out there in Nairobi, near villages, and so on.

And you go in, say, cash in, is to take your shillings and get credits for your cell phone from Safaricom, which you can use for buying some stuff or send back home to the village, and then the recipient there could cash out, going to a different agent, and convert the e-credit back into cash.

So these agents are like dealers. And in principle, they need to carry inventory to be able to honor the demands. It's actually a pretty complicated inventory problem because they have to have inventory of both the fiat money and the e-stuff. The e-stuff is, in principle, transferable even from another dealer, but the fiat money, the currency part is not. You've got to get a hold of the physical currency to allow the client to back out.

Now it turns out, from Tavneet Suri, she documented that these agents were informally helping each other out. You're running down the street giving gifts of one object or the other to each other in order for the dealers to be able to honor, on a reasonably timely basis, the demands of the clients.

And in Indonesia, something similar is happening, but it's with commercial banks. Fortunately, they don't have to have marble pillars and big buildings, they can just be in some kiosk with a bank insignia, but these banks also complain because people have done surveys-- maybe the Gates Foundation did this survey. They're running out of stuff, too. And they're supposed to rebalance, but it's kind of difficult if you're running short to find a willing partner on the other end.

And finally, something I've mentioned, in these New York markets, you have broker-dealers. As for repo, basically exchanging treasuries for liquidity or reserves or the other way around, and as is well-known, there are potentially big liquidity problems that emerge periodically in these New York markets where people are running out of cash-- the liquidity, pardon. Without it, the interest rate on the repos goes way above the policy guidance for market rates.

These are arguably three pretty different setups. The common element here is they've all got their balance sheets. They're trying to mediate trades with clients. And so the state of their balance sheet is somewhat random. And they can be experiencing shortages.

And they're potentially able to make deals, but now we throw in another aspect to this, is you may not want to tell the rest of the world exactly what your balance sheet looks like.

If there is some gift-giving, borrowing, and lending scheme going on across these dealers, how do you induce them despite the benefit of sharing the risk-- it's a risk-sharing problem-- where the shocks are these shocks to the balance sheets? How do you assure them that they're not revealing the state of their balance sheet? So that's the motivation.

And there's summary here at the beginning. Ex ante insurance is good, so hopefully that's kind of compelling. Everyone loves to smooth, say, especially against idiosyncratic liquidity shortage shocks. The next part is probably less obvious. Revealing the shocks as they occur too quickly, in other words, can damage beneficial trade. Why is it less than obvious?

Well, if it's a static problem, then the person who's insured would basically wait till after the damage in order to enter into the insurance contract. And the insurance company knows it, so no insurance is possible.

This has to do with the timing. Even though a loss has happened, it's beneficial to conceal that from other agents, so the person with the loss knows that it happened, but the other ones don't. And that allows insurance to take place even over utilizing shocks that have already happened, and yeah, where we're going is we're going to want to encrypt these histories.

OK, so let's think about formalizing this as a mechanism design problem with this concealment, and, again, we'll simplify it. So it's an example, there are two agents, A and B. Each have utility functions subject to privately observed shocks.

So let me pause on that. I just jumped from a shock to your endowment, to your balance sheet, to a shock to your preferences. But you could well imagine, this works out for the example I had in mind with the balance sheet shocks because if your balance sheet is low and you're a concave person, you're pretty risk-averse. It's in a reduced form way, as if you're getting a shock to your preferences that make you urgent.

Two agents-- there are two periods only, and this planner, agent A in the first period could be patient or urgent to consume sending that a message to the planner. And I'll go through this in more detail, but you may remember, we did the revelation principle where, without loss of generality, we can solve these mechanism design problems by having the agents truthfully report what actually happened.

So in this case, agent A would send a truthful report to the planner. Uh oh. OK, so we'll get there. If the planner is a third party, the planner now knows this, the state of the balance sheet. And B does the same, but in the second period. This is the way, the example is going to work. First A, and then B. First period, second period.

The planner sees the messages, but makes sure other agents don't. So the planner is trusted and committed not to reveal stuff. And there are incentives to tell the truth in both periods, and in particular, in the second period. This intuition for where we're about to go, having to do with the dynamics, is that there's only two periods, there's only one good.

Pretty hard to engineer trade because more is preferred to less. So if the transfers were to vary-- say I'm low, I want stuff, you would always say that in order to always get stuff, and that makes trade in the second period very difficult. So that's what we're going to try to conceal. Sorry, by concealing what happened in the first period, we will get around this problem that we can't get much insurance in the second period.

OK. So this is a more general version of the setup. It's still two agents and two periods, but they're each, in principle, experiencing shocks. In both periods. They're getting endowments. The endowments are deterministic and known by both agents. We're going to exploit that in the sense that that stuff can be sequestered and put into collateral, put into escrow. So they know the size of the pool, they just don't know their preference shocks.

Each agent i has utility over consumption in the first state, which is a bit confusingly labeled 0, and consumption in the second date, subscript 1, with shock being drawn, hitting utility in each of those periods.

So with an i on it, this is the consumption path, although in realized time, would be stochastic path and is deterministic for any particular draws, but the draws are random. In this case, there are two parameter draws for each agent in period 0 and period 1, so this 4 tuple is the underlying state of the preference shocks, and it's being-- the way it's being drawn is common knowledge to the agents, but the particular draws are not. The correlation structure can get pretty complicated. I'll come back to that.

So we assume there's an insurance provider as a mediator or planner, collects information from the agents, decides on how to split the endowment in each period as a function of what the agents report. We're going to do the obvious thing. We're going to maximize a weighted sum of the objective functions of the participants subject to these resource constraints and to the truth-telling constraints. So nothing new there.

As I said, the revelation principle rule will apply. So agents will be sending messages about stuff. In particular, at date 0, each has a preference shock. They could be sending that truthful message to the planner. The state at the second period, due to the correlation structure of the shocks, is not simply the new shock that happens at date 1, it's also what happened previously. That's the underlying true state as far as the agent is concerned, so there's values being set for both shocks for each agent i .

So this is a little more general than the example we're going to go through. It has the common aspect of having only two periods and having only two agents, but it allows unlike, the example, that each agent has shocks in each period. You could imagine that the planner could send messages back to the agents about the other guys. There could be partial revelation.

Here, there's some, quote, "inevitable" revelation that happens just by virtue of the setup. People are going to eat at the first date, so that some of those endowments has to get distributed. And so you're going to learn something from what you get, and for that matter, you know what the other guy gets because you know how it adds up.

So you'll see that that could reveal quite a lot about what the other agent said and destroy the possibility of trade in the second period. So we're going to work on concealing that.

So this, in notation, is the example where A goes first and at 0 and B goes second at 1. And if you first look at these utility functions, they're just power functions. It's consumption to some power. Its power is either known, public common knowledge, or potentially random.

These lines try to capture that, but in a very clumsy way, because we're not saying the utility functions are the same and the realized value of utility is the same in each period. This really only meant to say that there's a common underlying utility form of a utility function, which is consumption to a power.

OK. a is the one with the unknown θ parameter value. At date 0, it could be 0.2 or 0.9. On the other hand, agent A's parameter at date one is known for sure to be 0.3. B is in a symmetric position, but not with respect to time. B's parameter at 0 is known to be 0.9. Even that's not symmetric because it was 0.3, now it's 0.9 for the known parameter.

But b is subject to these unknown uncertainty about the parameter draw, which is, again, either 0.2 or 0.9. And it's pretty complicated. The correlation structure is anything can happen with positive probability. Any of these parameter draws for agents a and b at date 0 and 1 is possible. The deterministic stuff is just that, deterministic, so it's not random at all.

I'm going to show you a computed numerical example. I don't know how general it was, and certainly, it's been on my agenda for quite some time to come back to this kind of environment and compute some stuff-- or deduce some stuff more generally.

So, again, the endowments are known. 5 for each, it adds up to 10. There's a discount rate of 0.95, and the lambdas are going to treat the agents similarly. So here is-- to be very careful with the header to this table, it's called fully revealed communication. That does not mean we got rid of the private information. It means that these messages that are sent about the shocks are basically revealed to the other party.

So agent A could be patient or urgent, and there's going to be trade-offs that induce agent A to tell the truth about things. Note that, roughly speaking, when your patient in the first-- date 0, you get less than when you say you're urgent. So the allocation is trying to accommodate the urgency. There's a little bit of a lottery here, we'll come back to those things in a minute.

So in this row, as an example of what can happen over time, agent A is patient expecting to get more in the second period, and will get 2.5, which is higher than 1.75. And there's a more complicated trade-off likewise. When he says urgent, in some sense, he's going to get less in expectation in the second period. So it's incentive-compatible, like the borrowing and lending example that we went through at one point, to tell the truth about whether you're patient or urgent.

The thing to focus on here is we're in the first row, and agent B knows it, so it's very hard to get any trade going on here. In fact, the allocation is the same regardless of the parameter drawer for agent B in period. There's movement of goods over time, but not over θ_{B1} , the shock of agent B at date 1.

There is some, quote, "exchange" and insurance going on at day 2, which is, again, something I said, but we didn't have notation to back it up, or a numerical example. Agent B here-- this is the row-- sorry-- where agent A said he was urgent.

And now it's agent B's turn. He could say 0.2 or 0.9, and the allocation is different. If he's 0.2, he gets 6.25 for sure. If he were that, he might be tempted to lie about it and say he's 0.9, and there's a good chance he will get more of the good that way. And if this line weren't here, he would definitely lie and always claim the high incoming number, which is why this got reduced to no trade.

But here, there's a lottery, 10%, 11% chance of getting 0. So if he's really-- a trade-off here is basically to take your-- not take a chance and get the deterministic allocation if you're a 0.2 guy, which is a pretty curvature person, versus claiming you're 0.9, even though you're not, and being subject to this lottery, that could give you a pretty awful outcome with 11% probability. So that lottery is enough to keep this guy honest and claiming the smaller allocation. So that's an example of why these lotteries are popping up.

So here's an example where, with fully revealed communication to the other party, it's damaging in the sense that we don't get any insurance for agent-- for either one of them, quote, but particularly for agent B in this top row.

OK, so if we could conceal the information, we can do better. And the way that goes is the allocation at 0.2-- you'll get dizzy if I'm flipping back and forth too much-- is exactly the same as it was before, but now, with 3.3% chance, when agent A says he's urgent at 0.9, the same allocation that happened in the first row could happen here.

So roughly this-- this only happens 3% of the time, but this happens-- this row happens as often as agent A could be patient at date 0. So just seeing this doesn't tell agent B in particular which world he or she is in. When the 1.75, 8.25 happens, it could be because agent A said that he was 0.2, but it could happen because agent B-- sorry, agent A said 0.9, and by luck of the draw, that same allocation happened.

So note now that we're getting some insurance for agent B. In particular, B can get more of the consumption good when he says 0.9 than when he says 0.2. So we've engineered some beneficial insurance.

How is that happening? The answer is, well, I saw my allocation, so I could well be in this row, but it's possible that I live in the second row and agent B has said point-- sorry, I did that again. Agent A has said 0.9.

So if I'm tempted to cheat and say-- like, he's getting 5.25. If he says 0.2, why not claim 0.9 and get 8? And the answer is if he's down here and he says 0.9, there's a 14% chance he's going to get 0. So that's the deterrent. That's enough to make him honest. These were all solved with linear programs, with incentive constraints being carefully enumerated and so on.

Well actually, take a deep breath because all of this was the preamble to here's the economic allocations we want-- this one. We weren't so happy with the one where there was full communication. We still got this planner. So how do we get rid of the planner and yet have all this communication going on, and in particular, have this kind of randomization going on?

Because for all the world, when the randomization kicks in, it kicks in when agent A says 0.9, but in an encrypted world, how would the code know that? Because the code is not supposed to know the realized values, so it's potentially a bit of a challenge.

So we're going to use the encryption scheme that we had for the auction, same notation. We'll use the pseudo-agent, the code, to conceal things-- well, both. That's the twist. Both to be in the dark about underlying things, but also amazingly able to conceal stuff, nevertheless, from the agents.

So they're going to enter into this risk pooling contract. They agree to it. They put their stuff in escrow, so that's now accessible by the code, like the balance sheets are sequestered for transfers, et cetera. The node, if this is Ethereum, is the one-- is the contract node.

Agent A is going to send a message as this thing gets realized about whether he or she is high need of liquidity or low, urgent or patient. We'll take the central planner and replace it by this code C.

So this is the big overview of how this thing works. The most shallow, but useful thing to remember from this is all these messages going back and forth between agents A and B, which are encrypted, but also messages going back from one of the agents after some exchange back to the code.

So there's both fully homomorphic encryption aspects in the sense that the code will eventually act on the encrypted messages received. It also is going to have these multi-party computation aspects where the agents are sending messages back and forth initially before something gets submitted to the code.

And also-- and I'll show you a bit of this, you may remember from last time that we-- with the ElGamal example of cyclic rings, we were saying certain properties of the encryption algorithm have to be satisfied, it has to be distributed, commutative, associative, and so on, and I'll show you-- I'm not going to show you every slide because this alone is going to be hard to remember, but I'll highlight where these different properties of the different algebraic properties are entering.

OK, so how does it work? They've agreed to the contract, but before anything actually happens after that, agent A, remember, goes first, is going to send two encrypted values to B, one of them is going to be the encrypted value of the urgent shock-- high, $h_{sub A}$.

And the second one, the other remaining one, is going to be the encrypted value of the low shock, $l_{sub A}$. The key here is the encryption scheme is now set for h_A and l_A , but the order in which B is sending these encrypted messages to B is not known to B, and effectively can be determined at random initially by agent A.

I'm going to encrypt the high value first, then the low-- no, no, no. I'm going to encrypt the low value first and then the high. B does not know that. So B is in receipt of two encrypted messages, call the first one m_1 , the second m_2 , but each one is a ciphertext and it doesn't convey anything about the underlying $h_{sub A}$ or $l_{sub A}$.

So he sends 1, so that's the order, and m_1 is the encrypted first number, m_2 is the encrypted second number. So B here takes those messages and sends it to C. Sorry. I guess I didn't go through this line. So A will send something else to B. Note the arrows from A to B.

The other thing that A is sending is not just these-- some order of these high and low shocks, but also sending to B an encrypted value of this vector, which is 1, 0 or 0, 1 depending on the order in which A chose to send messages m_1 and m_2 to-- sorry, the order of the actual ordering of the m_1 versus m_2 .

So for example, if agent A had chosen to send the high value, h_A first, then the encryption would be the encrypted value of 1, 0. And if he had chosen to send the low value encrypted first, he would send 0, 1. But again, these are encrypted by A, so B receiving one or the other still leaves B completely in the dark.

But anyway, B goes along with the algorithm, and actually then encrypts the encrypted value of A, so it's like double-encryption. This is what B receives, and now it's encrypted by B, and this part finally ends up with code C. So this is a bit of the MPC part.

OK, so then what happens in real-time? Well, some urgent or patient parameter is realized, agent A is going to send a message about it, m being that message, the realized value of θ of h_A , or alternatively, l_A , encrypts it, and sends it to B.

B now sends something back to A, namely sort of encrypted null reference. A encrypts the null reference. So what happens now is the encrypted value of A times the encrypted value of-- sorry, B at 0 times the encrypted value of A, and this is the second item that the code is receiving. So the code has gotten this and this.

OK, now B sends to A the difference between A's actual value and the two values that was sent at date 1 in some-- sorry, I guess it's step 1-- in some order.

So if this were the first message sent at step 1, then this is the encrypted value of A at m_1 , that's what agent A sent to B already. So B knows the encrypted value. B also knows the encrypted value here of the actual message that A is sending. So the difference here is encrypted, but it's computable.

And then A and B is encrypting that difference as the first term of a vector. And the second term of the vector is the same, except that's the difference between the actual value and the second message that was sent at step 1.

So now, this object is coming back to A. A encrypts the pair and sends it to C. Now no doubt you're completely exhausted. So the claim here now is that this level of communication back and forth between the two parties reveals nothing to the two parties, but the code is being armed with enough information to figure out, effectively, de facto, when to randomize.

To try to provide a little more insight, this step 1 in some order, agent A is encrypting the urgent or patient parameter value, and you can see the A and delta z's were messages before in the auction scheme, now they're claimed values of the-- and actual values of the actual parameter clouded up with the noise.

Setting delta equal to 1 for simplicity, we can see our first example of the encryption being commutative in the sense of we are able to reverse the ordering of the encryption from A to B, and then now B to A. In whatever order it is, we get the same answer on the right, which is a bit like the auction scheme, where we're going to basically add up A times the s's and add up the errors, leaving the message.

But again, the message is in there, but it's embedded with all this other encryption, so B does not no. Let's see what else. This is the part where A is encrypting 1, 0 or 0, 1. 1, 0, for the sake of an example is where the urgent value was sent first. And that's the 1. And then 0-- so there's a comm-- effectively a comma here to separate these two elements where 1, 0 is the key component in each-- the key part of each one of those components.

But it could have been the other way around where agent A sent the low value first, and then the high one, and then step 1. So I guess that brings me to a second thing I want to show you, which is, we've got all these cases now.

The first has to do with whether that first message, pre-play message encrypted sent by agent A to B was the high value, case 1, or case 2, it was the low value. The second branch is what's going on in the actual mechanism as it's played out where m is sent, and it could be the high value or the low value.

So in the event that agent A at step one sent the high value encrypted and that high value was also the one realized, then these objects that were the difference that B was encrypted-- that B was seeing between the pre-play and the actual values reduced to this statement, a pair in the vector encrypt 0, encrypt 1.

Or it could be that it was case 2, case B-- case 2 is where the low value was encrypted first and the actual message was low. So what they have in common is that the ordering was such that the first encrypted value in the pre-play is also equal to the actual realized message. The first and fourth rows here are exactly the same.

So the code is effectively figuring out whether the actual realized value was sent first or not. That's a big step forward. B doesn't know, but the code knows. And these two values, the second and third row, are also the same, but that's where the first value was sent in the pre-play, and the actual realized value was the second one, so the ordering is the other way around. So the code knows the ordering. Not the message yet. Just knows the ordering.

So here's a bit of the commutative and distributive properties. Basically, the commutative thing is you're exchanging the order-- I guess I said this before-- of the encryption operator from B, and then A, to A, and then B. And here is the distributive thing where you're basically taking this encrypted value of the pair, and also, it's the same thing as the encrypted value of each of the terms individually.

And then this we'll skip. That has to do with the codes remembering some of the key things, having to do with the ordering. This we'll skip. And finally, we have this line. And I hope you're happy that I'm sparing you all the algebra in between because it's already just too much. I'm just trying to give you some of the math that's happening along the way by highlighting it, although I've kept the slides as they were.

This is encrypt B, encrypt A times 1 of 1 times the randomized amount, but due to the properties of fully homomorphic encryption, we could have put a 1 here and then encrypted it. The ordering doesn't matter, so that's exactly what we did. Encrypt B is the same, encrypt A.

But now, not of 1 times the randomized amount, but encrypted of A 1 times the randomized amount. And that brings the 0 out here in front, and 0 times 0-- 0 times something is always 0, so we end up with a term that only involves the randomized amount. And it could have gone the other way, and we get the low liquidity amount.

So that's where the miracle is kind of happening, where the rabbit's coming out of the hat, and that the code is actually able to not only get the deterministic allocation to the agents when agent A said he was not urgent, but also randomized-- you have a pseudorandom number generator and the code is accessing that.

All right. So these two slides basically try to live up to my claim that I'm suffering from not really having time to go into it. We've got these messages. We generalize it from more than 2 to, say, 3, so now it's low, medium, high. And we have to decide what messages are being sent. You can do it message by message by message, or you can actually start creating more than one pseudo-agent that works on these messages pairwise.

But in a particular order, c1 is taking low, medium, c2 is taking medium, high, c3 is taking low and high. So it's a bit different grouping than we had in the earlier application.

OK. The last application is order book matching. So here, the idea is there are two dealers. This could be repo, it could be something else. Any kind of exchange, the guy with the treasuries is willing to exchange them for money and is submitting these kinds of, I don't know, almost like limit orders. 1 T for 1 M, for 2 T for 2 M. Pointwise orders, I probably should say, instead. 3 for 4.

What's the maximum treasuries that could happen depending on what gets picked by the market matching is 3, so 3 has to go in escrow, and likewise on this side, we've got the guy with the money who's potentially willing to take the treasuries, and in this order, the maximum vulnerability is 3, so 3 reserves are put in escrow. And then the code finds the common line here with this 2 for 2 and matches it. And so that's the way the trades happen.

And without burdening you with the notation, we can encrypt those messages. So each of these trades of T for M or M for T is encrypted. The code knows how to act on these encrypted messages by doing a comparison operator to find the match. So the agents don't have to reveal-- worry about revealing their orders, which could lead to front-running and so on and so forth if they were revealed.

And so this is matching with the smart contract, et cetera, et cetera. I didn't show you the notation for the encryption, but I'm sure at this point you're probably willing to take my word for it.

The other thing just to say is that with these cryptographers here at MIT and also at the company Visa, we're working on the ability to run this kind of encryption on the blockchain. That is still a limitation. All the examples I've shown you are like tier 1, tier 2, layer 1, layer 2, off and on chain. So the encryption can happen without running into validation issues and so on.

But if you're on the full blown blockchain with the validation, then you have to use MPC in clever ways. There are two applications that do it. Zama and Sunscreen in particular claim to allow encryption on the blockchain, and this paper is evaluating just how effective they are in terms of the number of multiplication operations, addition operations, and comparison operations.

So it's something of a mixed picture that certainly those companies have made real advances, and I'm pretty confident that in not too much time in the near-future, we will have fully operational encryption schemes running on the blockchain, but we're not quite there yet. And it's a big problem in practice for Brazil, for example, with its design of its wholesale CBDC, so these things are really needed. OK, so that's all for today. Thank you very much.