

[SQUEAKING]

[RUSTLING]

[CLICKING]

PROFESSOR: Next thing I want to do, now that we have Fiat-Shamir-- so now what do we know? We have this protocol. Again, I'm always thinking of this protocol as parallel repetition-- parallel repeated. And we know the Fiat-Shamir sound on it. Now what's really nice about it, that this gives us-- so now I wanted to say two things in the hour or so remaining.

One thing I want to say is that it gives us a NIZK. Non-Interactive Zero-Knowledge proof. And the second-- and so I want to explain what that is. And the second thing I want to say is to mention a little bit what was the question here in the beginning, which was why this contradicts all this idea with Fiat-Shamir or not, something weird is going on, so I want to talk about that. And I want to end with explaining why this technique is more general.

OK. So let's first see why are we getting a NIZK. OK. So-- OK. So now, what did we get? Once we apply the Fiat-Shamir, we have a non-interactive protocol. So let's look at this protocol and see what this protocol becomes when we apply the Fiat-Shamir paradigm to it.

So let me just write it down. Actually, I don't even want to write explicit-- like everything explicitly. Let me just say this protocol had, like, a public key associated with it. Now it also has a hash function associated with a hash key, which we use for the Fiat-Shamir hash function.

And now, essentially, the verifier, he sends alpha, which is all the commitments per repetition, like the first message. And then he tells you beta-- he doesn't send it, everybody computed, but that's evaluating, with a hash key, alpha. By the way, I abused notation a little bit, and actually, every time you-- let's see. There's also a G here, like the instance, the graph.

Really, what you have is the graph in alpha. I kind of omitted the graph of the rotation because it's kind of cumbersome and it's kind of fixed in advance. The alpha is the one that's not fixed, but the G is always there. When you apply Fiat-Shamir, you apply it to the first message and the instance.

OK, so that's the beta and gamma. This is-- that's one message because we applied Fiat-Shamir. So this protocol, which consists of three messages, alpha, beta, gamma, now, once we applied Fiat-Shamir, we have not only the public key in the CRS, we also have the hash key corresponding to the Fiat-Shamir. And now, this alpha, beta, gamma are sent all at once. There's no interaction. That's what Fiat-Shamir does.

But beta, the prover's hands are tied and it should be the evaluation of G and alpha. The hash function evaluated on G and alpha. So this is completely non-interactive. And it's sound. Because we just worked hard to prove that it's sound if we use this hash function under circle circle FEG. Yeah?

AUDIENCE: Is this a NIZK argument now, then, because we used [INAUDIBLE] security?

PROFESSOR: Good. We'll see. You can make it a NIZK proof or argument, so we'll talk about that in a second. So this is one message. It's sound, and it's complete. Now what happened to the zero knowledge? Let's see what happened.

Here, we said, we had zero knowledge. We could simulate it. What happens here? And what I want to argue is that it's-- still, you have zero knowledge. Even though before we set future actually routes you that shows it's not zero knowledge, this non-interactive version, you can prove that it's zero knowledge; however, the definition of zero knowledge here is a bit different because we have a common random string.

So now, this common-- or common reference string, we denote by CRS, it consists of the public key, used for the commitment-- for the message α , and the hash key used for the CRS.

Now, let me-- I want to argue that this protocol is non-interactive zero knowledge, or is a NIZK, Non-Interactive Zero Knowledge. But to prove that, let me define first what non-interactive zero knowledge is because the definition is similar to the interactive setting, but slightly different. So the definition is the following.

A NIZK for a language L is the following. It says that for any PPT V^* or poly size-- it can be non-- it can be poly-- but, OK, let me-- for any PPT V^* , there exists a simulator, a PPT simulator, a Probabilistic Polynomial Time simulator, such that the simulator can simulate everything that the verifier learns in the real world.

So what if the view of the verifier in the real world such that-- the view of the verifier in the real world, which is what? It's the CRS. So every NIZK is a-- it's non-interactive and there's a CRS. So, the verifier learns the CRS. And he learns what he got. He learns such that-- sorry. For every x in L -- I forgot to say, for every x in the language, he learns this. And he learns the proof of the actual prover.

So I guess what the actual prover gives, when he gets CRS, x and witness. This is what the verifier learns in the real world. He sees the CRS and he sees the actual proof produced by the prover. And the prover is given CRS, x , and a witness. Instance and witness.

And zero knowledge says that the simulator can simulate this. So I forgot to say security parameter-- there's also a security parameter. Oh. Actually, you know what? You don't need security parameter because the CRS you can think of including-- this includes also security parameter. You can always assume the CRS includes a security parameter.

So here, what the verifier learns in the real world, he sees the CRS, which includes the security parameter, and he sees the proof that the verifier gave. And now I want to argue that the simulator can simulate this entire thing. Typically, we also allow auxiliary information. So for any x and for every auxiliary information aux , the [INAUDIBLE] also sees aux , and still he cannot tell the difference.

So when we talk about zero knowledge, there's just zero knowledge, and then there's auxiliary input zero knowledge. This is stronger. We usually want this, so I'm just writing it, but there's also the-- without auxiliary information definition. But again, what is zero-- NIZK? Non-interactive zero knowledge. It says that for any efficient V^* , there exists an efficient simulator so that for every x in the language, it has a witness, and for any auxiliary information that the simulator may have or the verifier may have, what the verifier learns in the real world can be simulated by the simulator. Yeah?

AUDIENCE: Here, we don't even need to V^* , right? It's non-interactive?

PROFESSOR: Yeah, sorry. There's no V -- yeah, there's no V . Whatever you can learn. You're right. Thank you. For every L , there exists a simulator for every x , for any auxiliary, that's it. That's the definition. Because V doesn't send anything. OK.

Now here is-- note what-- the reason this is still zero knowledge is that now the simulator, he gets to choose the CRS. The simulator, he outputs-- this simulator outputs a CRS and some fake proof π that should be indistinguishable even given aux .

So really, it's-- and the point is, that simulator, he has the power to choose the CRS. This is a power that the cheating prover does not have. And that's why, if you could prove that-- the fact that Fiat-Shamir works for-- the fact that you have-- there's no contradiction with the fact that it's zero knowledge and Fiat-Shamir-sound because the fact that it's zero knowledge, the fact that it's Fiat-Shamir-sound-- or in other words, the fact that it's zero knowledge does not mean that you can use the simulator as a cheating prover.

The simulator succeeds in simulating because he has the power to fake the CRS. The cheating prover does not have that power. Again, remember before we said, oh, if you have Fiat-Shamir, then the simulator, because it can simulate the view, he can also simulate a view in the case of cheating, so he can actually break Fiat-Shamir.

Here, he can't break Fiat-Shamir-- because, I mean, the prover can't-- the simulator can't because he has an additional power that a cheating prover does not have. He can fake this-- he's the one who gets to choose the CRS. And that's the only reason why we can actually get a NIZK here, because he's going to choose the CRS to his liking. In an indistinguishable way, of course.

But, I mean, this CRS and this CRS are indistinguishable. But he can program it. That's the idea. So let me just tell you why it's zero knowledge, and then you'll see-- because you know why it's Fiat-Shamir-secure. I'll tell you why it's zero knowledge and then you'll see how I'm abusing the power of the simulator.

So the way-- the reason why it's NIZK, it's a non-interactive zero knowledge. So I need to come up with a simulator. I need to simulate the Fiat-Shamir protocol. And actually, if you think about it, how can I simulate it? I need to come up with an-- think what I need to do. I'm the simulator. It seems like a hard task for me. What do I need to do as a simulator?

So what does the sim do? Sim needs to-- he needs to generate CRS somehow. I don't know. OK. I can just generate public key and hash key. But then, I need to find α , β , γ accepting. So I guess, first note-- look, the simulator is given-- wait, he's given also x . I forgot to write x . He's given the instance x and aux .

Now note, I'm the simulator. x can be in the language. x cannot be in the language. I have no idea what x is. So in some sense, if I succeed, I should also succeed for some x that's not in the language because I can't break the language. Now, to succeed for x not in the language, I need to generate β that's bad. Otherwise, I can't succeed. But how can I generate β that's bad?

Remember, the encryption? It's a problem. The reason it's not a problem because I'm going to break the encryption. I get to choose it. I choose the encryption. I chose it with a secret key. And actually, let me be-- actually, for this specific hash function, I actually don't know how to prove that it's an ISIC. It may be. I don't know, but I don't know how to prove it. But if you slightly change that Fiat-Shamir hash function in a way that doesn't affect soundness at all, then I can prove that it's an ISIC. So I'm going to slightly change the hash function, the Fiat-Shamir hash function.

OK, I'm going to slightly change it, just add a constant to it. I'm going to argue that it doesn't change the soundness, but now I can use it for an ISIC. So let me show you how this is done. So how do I generate this alpha, beta, gamma? So the first thing I'm going to do, I'm going to say, you know what, I'm going to just generate beta completely random. So I'm going to actually use the way-- remember, last time we showed why the single-- the one-bit protocol is zero knowledge? I'm going to use that idea.

Or in other words, I'm going to first use a simulator. First, I'm going to use an honest verifier, zk simulator. This is just short for honest verifier. So I'm going to use a simulat-- I want to first argue this protocol, alpha, beta, gamma, even repeated in parallel-- repeated in parallel, so it's honest verifier, zero knowledge. What does it mean, honest verifier? The verifier just sends beta completely in random, so $b_1, b_2, \dots, b_\lambda$, completely randomly. Then it's your knowledge.

So that, we kind of saw last time. So first of all, I want to argue that we have honest verifier of zero knowledge. That's the first thing. The interactive protocol-- the interactive but parallel repeated-- the interactive parallel repeated protocol is honest verifier, zero knowledge. Why? Well, it's very easy. What I'm going to do-- I'm the simulator. What I'm going to do, I'm going to first-- I need to generate a transcript.

What I'm going to do, I'm going to first generate the b 's-- b_1, b_2 up to b_λ at random. Now I know what I-- now I know, if b equals 0, I'm going to just commit to-- I'm going to commit here just a completely random Hamiltonian cycle, have nothing to do with the graph. If I need to open-- if b was 1, I'm going to just really commit to π of g . It won't have any Hamiltonian cycle in it, but I'm going to commit to π of g .

So that's going to be my simulation. It's kind of a little bit of what we did-- it's similar to what we did last time, but it's even easier because the verifier is completely honest. And let me repeat it. How do I generate this transcript repeated λ times in parallel? I need to generate alpha, beta, gamma that looks like real. I'm going to first generate the beta. So b_1 up to b_λ . That's the first thing.

Then I'm going to generate alpha. I'm going to take it out of order. How do I generate alpha? Well, if the b corresponding to-- let's say, how the first alpha. If the b was 0, I know I need to open to the Hamiltonian cycle. I'll commit to a random Hamiltonian cycle. If the big b was 1, I know that I will need to open the π and the non-edges. Then I'll just commit to π, g , honestly. There's not going to be a Hamiltonian cycle in there, but I know I'm not going to be asked to open it. And that's what I do bit by bit.

So because I know beta, I choose alpha to be kind of-- I exactly choose alpha to hit the bad on purpose. OK, so that's-- so I can choose alpha-- so I can choose alpha, beta, gamma to hit the bad. But now what do I do with the hash function? How do I-- I didn't choose the hash function yet. Remember? Now I'm going to choose the hash function.

So now I want to be able-- now I want to choose hash key such that you know eval the hash key and alpha-- actually, g of alpha, but I'm going to omit g for simplicity-- is equal to that beta, which is the bad beta. How can I do this? Is it-- so can I choose hash key subject to this? I can choose the public key, the secret key, everything. I can break the encryption, but can I choose a subject too?

So actually, to tell you the truth, I don't really know if you can do it, but let me change slightly the hash function, and now I'll be able to do that. How do I change the hash function? My hash function is going to be the original hash function plus some fixed kind of lambda. So the hash function outputs some strings, 0, 1 to the lambda, small lambda. I'm going to just add a random, completely random lambda bit string to it.

Now, I don't like this letter because it sounds-- I'm going to add z in 0, 1 to the lambda, a random lambda bit string. Now I'm in business. That's easy now. Because now I want to make sure that-- OK, so now I need this to be equal to beta. No problem. I'll choose hash key honestly. I'll compute this, and I'll just-- in the CRS-- so I told you how I simulate alpha, beta, gamma. Now I need to simulate the CRS.

What does the CRS consist of? It consists of the public key and the hash key. I'll choose the public key randomly. Now the hash key consists of this part and this part. It's now two hash keys because it's the original hash keys, but I also add a fixed z to it. That's my new hash key. My new hash function does the previous hash function and add z . z is fixed. It's part of the description of the hash function.

So now, how do I generate the CRS? So for CRS-- so to generate a CRS, I'm going to choose a random public key, a random hash key, and z , such that this equation is satisfied. I'm going to choose z , which is equal to-- I chose alpha, beta, gamma. I chose already hash key, so this defines this, the eval. So z is just going to be beta minus or x or-- I'm over strings-- so x or this eval. And this is a good simulation.

So now let me go back to your question, which was-- OK, so the question was, so this is going to be it's zero knowledge. It's computational zero knowledge. It's not statistical zero knowledge because the alpha contains actually the-- when I simulate it, I don't simulate it with a real π of c . I simulated with junk. So it's only computational zero knowledge, but it's statistical. And it's not even statistical soundness because I'm using a fake g , so I don't get anything. It's computation is zero knowledge and computational soundness.

There is a way to get statistical if you play with it, but this way I explained it, everything is computational. Yeah?

AUDIENCE: Well something in the final hash key, is obviously then you have that randomness. But in the real case, is it will be just--

PROFESSOR: Good, good, good. Great, great question. So you're saying-- you're saying, wait, so I changed the-- I changed the hash key on you. So before-- let me erase this. Before, the hash key was exactly what we wrote there. That was the hash key. Now, I changed the hash key. Now the hash key is what?

So let me first-- now, the hash key is going to be this and z , random z . So the hash key consists of the a public encryption-- FHE public key-- public key for fully homomorphic encryption and encryption of a g that's in the analysis is going to be-- OK, in the analysis, we're going to see what it's now changing only because of the z . But some g of fixed size like 0 to the t , and the random z in 0, 1 to the lambda. That's it.

Now, I still want to argue-- I still want to argue that this g is not bad. I want to argue that-- I want to say that when I do-- for every α , when I do this fully homomorphic encryption, it's not bad. So for which g -- for which g star, when I do this-- so now-- OK, now what I want to argue-- let me change this a little bit. I guess now I want to argue that when I do this and I add z , it's not bad because I add z .

So I want to argue that this is not equal to this minus z , or I guess x or. We're doing everything x or. So minus plus-- don't worry about it. We'll get it later. So I want to argue that this does not equal β , x or z because my hash is not just this. It's this x or z . That's my hash. I want to argue that this x or z is not β , which is the same as saying that this is different than β x or z .

So now what I'm going to do, I'm just going to-- here, I'm going to x or-- in the analysis, I'm going to say sure. Let me tell you, if you change g star-- if you take g star to be the decryption of bad x or z and then change it a little bit-- x or was just 1, like 0001-- then I can argue that you get statistical soundness.

So in the analysis, I'm going to say, OK, you started with this g , whatever it was. Suppose it was just this g , and then I can argue that for this g , it's never equal to this because exactly as before. Well, if for this g star it was equal to a -- so now I guess if it was equal to β , x , or z , then the decryption would be equal, and we got a contradiction. So it's exactly-- the fact that I did-- I added z has no significance on the proof. It's exactly the same proof.

So that's how I simulate, and now I have a simulator. So essentially, the way I proved that it's zero knowledge is by doing just the honest verifier, zero knowledge. And then what I did in that with CRS, I chose the z . I chose-- I programmed part of the hash key. If the hash key was programmed by someone else ahead of time and I couldn't touch it, I don't know how to prove soundness. OK, but I don't know how to simulate.

But because I could do it, I could simulate. So the fact that I could-- I had the freedom to program the hash key, that's what allowed me to simulate. And that's why there is no-- that's what broke the tension between Fiat-Shamir and zero-knowledge. Because in the interactive setting when there is no CRS, then a simulator really has the same power as a cheating prover. And now if there is a simulator, there is a cheating prover.

And when you think about the simulator, think about a verifier who is like a Fiat-Shamir verifier, verifier that behaves like a Fiat-Shamir hash function. Then the simulator is exactly like a cheating proof. And then we say, well, there's no cheating prover, there can't exist a simulator. But in the [INAUDIBLE], the simulator has much more power than the cheating prover. He can choose the CRS. He can program the CRS to his desire. And that's what breaks-- that's kind of what allows us to get both Fiat-Shamir and non-interactive zero-- both soundness via Fiat-Shamir and non-interactive zero knowledge.

Any questions? Yeah?

AUDIENCE: Does the random z help if β bad is not unique?

PROFESSOR: OK, good, good, good. So now the-- OK, so now the question is, what if β bad was not unique? So let me-- I'll answer that because I have a longer question-- a longer answer to that. But are there any other questions about this before I answer that question?

So let's step back and see where we are. I'm on my way to answer your question because it's about how general this technique is. So I'll get there. But before I go to how general this technique is, let's see what we did today so far.

So we started by-- we looked at this protocol, the parallel repeated version, and we said, OK, this protocol, guess what, it's Fiat-Shamir. We actually constructed a certain hash function, very specific hash function-- said that with respect to this hash function, we proved soundness. And then we proved not only is it soundness with Fiat-Shamir. Now that it's an interactive, we can also prove that it's zero knowledge, zero knowledge still maintains by playing with a CRS.

So even before talking about generalization, I just want to point this-- this was a huge open problem for a very long time to get an ISIC under LWE. Even though what you saw is only circular secure, $1w$, or if you think under FHE, you just saw the circular secure version. But even with the circular secure version, I still find it remarkable that I'm able to teach it from beginning to end in one class. I hope you understood, but it seems like from your question, you were pretty good.

So I don't know. I wish-- I could torture you with previous attempts that were much worse and much more complicated, so you will really appreciate this one. But for me, who shed a lot of tears and sweat with these previous approaches, I totally appreciate the beauty of this one.

But now-- so far, it was kind of-- the focus was on this protocol. We had this protocol. We saw Fiat-Shamir. And now we have [INAUDIBLE] and LWE. Great. But how general is this technique? And that's what I want to talk about next. OK, how general is it? And I want to point out, actually, we said-- we used this specific structure very little. What did we use?

So let me actually-- let's see what I should erase. Let me start here. We use the fact-- what did we use here? We use the fact that the beta was unique. That's one thing we used. Can we do non-unique betas? We can do non-unique betas, but we pay. So the number of bad betas can be-- let me call it d . But our assumption degrades with $1/d$ because the way we deal with-- if there is many bads, then essentially what I do is I'm going to guess which bad the prover is going to cheat with. And if I'm going to choose a random beta out of this d bad betas with probability $1/d$ I guessed his beta correctly, and then everything else kind of continues as if it was unique.

So we can deal with more betas, but we pay with the number of betas-- number of bad betas. So if it's really, really huge, then it's a problem. So we use the fact that it's unique beta, but we also used one more thing, which is that we can efficiently compute the bad beta. And that-- actually, I feel like the unique beta, as I think your intuition was, well, you can get around it a little bit. Is it unique? Well, it can be more as long as-- I'll guess it as long as there is some degradation.

But the efficiency of the beta, we really used it. And the hash function computes the bad beta. So I don't know at all how to deal with inefficient bad betas. But let me just mention that still, this result is very general. And actually, you can frame this result in a much more general framework for this specific protocol. And the general framework is the framework that's called correlation intractable.

So let me give you-- let me just start with the definition. What is a correlation intractable hash function? So correlation intractable hash function was defined in order-- in an attempt to prove soundness Fiat-Shamir. And a hash function is correlation intractable essentially if Fiat-Shamir works with it. That's kind of the idea.

So let me define a spectral relation R , so here's a definition-- a correlation intractable hash function-- hash family with respect to a relation R . so we have some relation, R . R is a relation. It's like a subset of x, y . You can think of x as, let's say, $0, 1$ to the n , and $y, n1, 0, 1$ to the n . And you can think of both n and m as like functions of security parameter. So n is like n of λ . So for any security parameter, you have a length n and m , and you have a relation.

And the hash family's correlation intractable with respect to this relation, it essentially means that for every poly size adversary, the probability that this adversary who gets hash key-- so the adversary gets a hash key. There is a hash family-- let me call it H -- that has Gen and Eval . Now, the adversary gets hash key generated, according to Gen . And his goal is to output x such that H of x is-- or such that x comma H of x is in the relation-- such that x comma Eval , hash key of x is in R .

So again, a correlation intractable hash family with respect to a certain relation R satisfies that for any poly size adversary, he gets a hash key in the family. His goal is to find an x that hashes to a y so that x and y is in R . And we say that this probability is negligible. I didn't finish to write it. So for every polysize A , there exists a negligible λ such that the probability-- so any poly size adversary, he gets a random hash key in the family, and he's able to find x at that h of x so that x, h of x is in R only with ungrouped probability.

So he always kind of evades the y in the relation. That's called correlation intractable hash family for a specific R . Questions about the definition? So first note this makes sense. So I want to say, do we a correlation intractable hash family and for which relations. So this is kind of a major open question because this is very related to Fiat-Shamir.

So Fiat-Shamir, we know-- take x out of the language. We want-- so think of α, β, γ . Just think for three messages. Everything I say now extends to more. But let's think of three. It extends the constant at least. But let's think of three for simplicity.

So we want to evade, not attach, to be correlation-- so the relation in our case-- in Fiat-Shamir, the relation is α and β are in the relation if there exists a γ that will make it accept. So the relation contains α and the bad β s. And we want to avoid these bad β s. So we want to say that any adversary that gets a hash key managed to hit a bad β only with negligible probability.

So there may-- there may exist α s, but he won't find them. It'll be hard for him to find an x or an α that hashes to a β that is bad. Here, I'm-- the idea is to take any-- take any-- think of any interactive proof of the form α, β, γ . And I want to say the relation-- think of the relation being all the relations of x , take x not in the language. I guess now this x -- this x corresponds to α here, but takes any x not in the language.

And the relation is all the α, β so that exists a γ accepting. So that α, β, γ is accepting. And I know that for every α , there's very, very few β s. And I want to say that the adversary output, α, β and the relation-- α so that the hash of α is in the relation only with negligible probability. Then I have soundness of the Fiat-Shamir.

Was that too fast? Should I repeat the connection to Fiat-Shamir? Let me repeat it once more. OK, so I want to argue that I have correlation intractable hash family for a lot of relations. Actually, before I go-- note, I can't hope to have it for any relation. The relation has to be sparse. If the relation is everything, maybe not everything-- every xy is in the relation, then, of course, a hash will hit a trivial. So the most I can hope for-- so the question I really want to ask or the most general question is, does there exist correlation intractable hash for every evasive-- or also people sometimes call it sparse-- relation?

That's the question. And by evasive or by sparse, I just mean that for every x , there are very few y 's that are in the relation. So sparse means-- forgot an e -- sparse means this. This means that for every x , the probability of a random y -- that xy is in R is negligible. Yeah?

AUDIENCE: So Fiat-Shamir is for x not in the language, right?

PROFESSOR: Yes, yes. Yes, yes, yes. Good. OK, good. So now let's go to-- yeah, so now let's go to Fiat-Shamir. Good. So I want to ask, if there exists a correlation intractable for every such evasive or sparse language relation. Now, if there is, I'm done. Why? If you can prove that there is a correlation intractable hash for every sparse language-- now I'll tell you-- now here's the Fiat-Shamir. Take x not in the language. Take the correlation intractable hash. You had said there is a correlation intractable hash. Take that hash function.

Now what I'm going to do-- now, I want to-- now take x in the language. The evasive relation that I look at is this relation. α, β is in the relation if there exists γ . Of course, it's evasive because it's a proof. It means there's only negligible amount of β 's I can answer. So it's evasive. And therefore, the probability that you hit a bad β is negligible. I'm done. Proof.

So if I have a collision hash that works for any-- so for any invasive relation, this holds for any R , then I'm good. So if for any R -- yes, for any R , you don't hit it, then I'm done. Fiat-Shamir is sound. OK. So then, this was actually defined in an earlier work, in 2004, maybe. I don't really remember, but a while, back.

And then people say, OK, so we need to construct this evasive function. So we proved that the random oracle model is-- the random oracle is an invasive function. But beyond that, they were stuck. And what this result that the Canetti-Lombardi week's results show-- so let me now state the result in more general terms. What it shows that there exists a correlation intractable hash for every relation that is-- for any relation, that is-- they call it searchable-- I'll explain-- in time t -- so searchable in time t just means that-- searchable means that for every α there is a unique β . It's a search problem. It's like for every output there is a unique β , and this β can be computed in time t .

So what they show is there exists a correlation in hash-- correlation intractable hash for every-- that is correlation intractable for every relation searchable in time t . And this hash-- the time to compute the hash grows with t . So essentially, what they say is for every polynomial t -- because the hash grows with t -- so for every polynomial t , there exists a hash function that depends on t , and its correlation intractable with respect to all relations that are searchable in time t .

And again, relations searchable in time t means that for every α -- let me denote the relation by this and not x because x often denotes input, and I want to think of the relation as α, β instead of x, y because then there's a confusion between the x here and the x instance. So it just means that the relation is a set of α β s such that for every α there exists a unique-- at most a unique, maybe less, but at most-- at most unique β such that $\alpha \beta$ is in R , and to go from α to β -- or but if there is no β -- is computable in time t .

OK, so a searchable relation means for every α , there is a unique β , at most unique. Maybe there's no β , but if there is a β , it's unique, and from α to compute the β corresponding to it. And the relation can be done in time t . And what they showed is that there exists a single hash family. So first note, of course, of course, if you fix a relation that's searchable in time t , of course, there's a hash that's correlation intractable with respect to it. Just compute the β and add one.

Again, let me move there. So you'll have this. So we want to show a correlation intractable hash with respect to any relation that's searchable in time t . And first, I want to note, I want one hash that's correlation intractable for all relations. Of course, if you first choose a relation-- if you first-- of course this-- what I'm saying now is a trivial. For any t time searchable relation, there exists a correlation intractable hash running in time t or close to t . That's easy.

What is the hash? The hash of α will just be the β that you're searching for in the relation plus 1. Just avoid it somehow-- plus whatever, 1. Just move out of the β . That's it. That's easy. But what they show is something much stronger. They show the oppo-- they show the reverse order of quantifiers. They show that there exists one hash family, and this hash family is correlation intractable for all t time searchable relations. That's the difference.

OK, so that's kind of what I want to end with. I want to prove it to you because we saw this proof. It's exactly the same thing, but let's just see it in this language. But before we go over the proof again, or just remind the proof, so this is a much stronger-- this is a pretty general statement. And this statement is used in many follow-up works. For example, this can be used almost to prove the soundness of GKR, for example or sum-check. Let's do sum-check for simplicity because all the ideas are there.

How can you do-- how can you prove Fiat-Shamir is sound for sum-check. Let's just recall, just for a minute, what the sum-check is, just so you see how-- it's not immediate, by the way. It requires more crypto. But I want to give you a sense a little bit of the why this is very useful. So it's not immediate, but it definitely uses this fact.

OK, so let's see why. As an example, let's look at the sum-check. So suppose I want to prove that sum of g_1 up to g_m is equal to some value v . And I want to do it in the Fiat-Shamir. So what is the relation? Let's see. It's many rounds, so it's harder. But let's just look at the first round for a second, or the first three rounds. So we have a prover and a verifier.

Suppose this is false. So this is actually-- it's actually false, but the prover is trying to cheat me. So now the first thing the prover does, if you remember, he gives a univariate polynomial. Now this univariate polynomial. Is wrong. I'll put star because it's wrong. Why? Because I know the value here is actually not v . He's proving that it's v . He should be giving-- the value that he should be giving is g_1 , which is sum of g on b_2 up to b_n . That's what he should be giving.

And I would check that when you take this and sum it only over b_1 , you get v . But of course he's not going to give me this because when you sum this over b_1 , it's not v . Because the real answer, this is not v . So he will give me another one, star. What are the bad betas?

The bad betas-- so now beta bad are all the elements t in the field such that g_1 star is equal. Now with this, now we can cheat. If he managed to give me g star that's equal to gt , now he kind of continues with the good. Now he has a true statement. Lucky him. So these are going to be really bad for me. But there are very few of them, like the degree, the univariate degree of this g .

So as long as I don't hit-- if the degree is d -- as long as I don't hit this d bad, I'm good. And it's pretty easy to compute this thing because I'm going-- so computing g_1 may be hard, but I can hard wire it. It's like a trapdoor. I have this statement. I can use g_1 as a trapdoor. And now, once I have g_1 as a trap door, giving g_1 star, computing all the inputs that collide with my g_1 , there are algorithms. There's efficient algorithms for this. So now it's efficient.

So all I need to avoid-- so it's not searchable. It's not 1 beta like we have here. It's d betas. But to go from 1 to d , as I said, you just guess. You lose a factor of d . It really extends pretty easily. So you apply Fiat-Shamir, done.

It's not that easy. The reason it's not that easy is because now a t comes out from the Fiat-Shamir, and the next g depends on this t because the next g is going to be the next g star. The next real g , g_2 , is going to be sum g , the real g here, t_1 , and then b_3 , da, da, da, da. So I can't hardwire g_2 ahead of time. I don't know what t_1 is. Things are not as simple. So there's more-- it's not a trivial, but the first part is trivial. And then you need to add to it, throw on it a little more crypto and do some more work. But this technique is used to prove GKR.

This technique is also used to prove the result of to do with Zhengzhong Jin where Zhengzhong Jin is going to present next class where they use Fiat-Shamir to prove the existence of a batch non-interactive argument. So he'll explain what that is next time. But again, this technique-- so this technique is pretty general. OK? Question before I just remind you why this holds? Questions? Yeah.

AUDIENCE: So you said get all the keys and pick a random one?

PROFESSOR: OK, exactly. So in the analysis, you can think of it-- yeah, I'm going to-- so I have a cheating prover. And it's a bit complicated, but yes, in the analysis, I'm going to say, look, I have a cheating prover, so it means he cheats with respect to 1 random-- with a random 1 here with probability 1 over d . And then I'm going to get a contradiction because he's going to cheat with anyone with negligible probability. For each one, fix it. With that, he's going to cheat with that with negligible probability. So then you're union bound kind of, and say, OK, so for the probability that he cheats with any of these is d times negligible. So as long as D is polynomial, you're good.

AUDIENCE: What will our hash function look like?

PROFESSOR: It's the exact same-- OK, so actually, let me-- so the question, what was our hash function look like. I'll tell you exactly what it is when I prove this, and you'll see it's the exact same hash function. So I say here-- and that's the hash function that I'll use up there. I say here that there exists a hash function that's correlation intractable with respect to any t time searchable relation. What is this hash function? I'll tell you.

So the hash function-- so to say a hash family, I need to say what's gen and what's eval. So here is the gen. Help me out. What does the gen do?

AUDIENCE: Computes a public key?

PROFESSOR: Exactly. He computes a public key for the FHE. And encrypt something. Nobody cares. We're never going to decrypt it. It encrypts with public key, just let's say the all zero string. So it's not exactly t . It's a little more than t . It's the t -- we'll see what-- encrypts t prime, like $000t$ prime. t prime is going to be very similar to t , but a tiny bit more. t plus the decryption plus 1, we'll see.

But that's gen. It outputs a public key and encryption of a bunch of zeros. How many? A little more than t . We'll see the exact number. That's what gen does.

What does eval do? Eval, given this hash key-- so this is a hash key-- and given alpha-- or-- yeah, alpha-- what it does is simply computes the eval of the FHE so FHE can be evaluated, a new alpha, and the ciphertext. And let me call this g hat because it's kind of describes a circuit.

And g hat, which is what it does-- it essentially computes g of alpha encryption. That's what this eval of FHE does. It takes-- this is a universal circuit. It takes the input, a circuit g , a description of a circuit g and outputs g of alpha. And this eval does this under the hood. So he outputs whatever is encrypted here, the all-zero circuit, whatever, a description of-- this is-- let's say this is all of the all-zero circuit. So in that case, it will be encryption of 0. That's what it does.

That's the hash function that I'm going to argue is correlation intractable for every relation, for every t time searchable relation. So now give me your favorite t time searchable relation, and I'll show you why it's a correlation intractable with respect to it. So, OK, you give me your favorite relation. Let me say a relation is a bunch of alpha and f of alpha. It's searchable. So there's some function that for every alpha, there is a unique beta. Find the unique beta.

If in the more general case where there's d -- think if there's d f 's, so there's a few more. But for 1, you can think of just 1. OK, so now I want to argue there is no way you can find. So I'm going to argue you can't find $F1$. But if there's more, eventually, you also can't find $F2$ and $F3$. But let me focus on the first F . Let's say if there's only one, I'm going to show you on the first F . So you cannot-- there's no way. Why? Because suppose-- if the adversary given hash key could find alpha so that the hash of alpha is f of alpha, then you could also do it in the case where I replace alpha-- where I replace g with a fake g .

And my fake g that I'm going to replace it with-- so suppose there exists an adversary that breaks. So there exists an adversary such that the adversary, given hash key, outputs alpha such that eval of h , key, and alpha is equal to f of alpha. That's what we want to avoid. We say no, you shouldn't be able to find something in the relation. If you can-- if you can do that with probability greater than some epsilon, then I claim-- then a succeeds-- in the same thing-- succeeds with probability close to epsilon. Also, by semantic security-- also with HK star, a different HK.

And what is HK star? It generates the same public key, but it generates an encryption of g star, not of g . So here I encrypted all zero. g was all zero. I've encrypted a new g , a g star. What is g star? g star-- an input alpha output, the decryption-- I know, weird-- it has secret key hardwired in it. And it outputs the decryption of the bad function and alpha, x or with one.

And we show that actually if this is what you put there, you really-- you never-- it's impossible to output f of x . This is never-- it's never the case. So we want to say that g star, that the eval g star α is never f of x -- f of α . It's just never. Why? Because if it was, then if you decrypt this and decrypt this, you would get the same value. But if you decrypt this, you get g star α . And if you decrypt this, you get this, and they're not the same. Yeah?

AUDIENCE: And what is f over? Does the adversary-- do you get to pick f ?

PROFESSOR: No, f -- sorry. Let me-- so we have a t time searchable relation. t time searchable relation, it means that for every α , there is a unique β . So I call this β --

AUDIENCE: Sorry, sorry, so is this over all remaining?

PROFESSOR: Good. Exactly. So I want to say-- I say I constructed this hash family that just has an encryption, and he does computation under the hood. And I'm saying this hash family, it's correlation intractable with respect to every t time searchable relation. Why? Fix your favorite t time searchable relation, your favorite one. The fact that it's searchable means that for every α there's a unique β , and it's computable in time t . So I call f the function that computes this β .

And now I say, look, there's no way an adversary can find α for which the hash function takes him to β , to f of α . Because if he could, he could also do it where the hash key, which encrypts 0, actually encrypts this g star, another g . What is g star? Now g star depends on this f . But it's under the hood. Nobody can tell the difference.

And this g star, it decrypts this f , even though f is not a ciphertext. So it's a very weird thing to do, but it decrypts this f and changes something, changes a bit. And now I want to say, the hash gives you encryption of g star α . And encryption of g star α cannot be f of α . There's no way. Because if it was-- any encryption, actually-- because if it was, if you decrypted this and decrypted this, you would get the same thing. When you decrypt this, you get g star α . And g star α is not decryption of f of α by design.

So once you change the hash key to this, you statistically can't-- an all-powerful adversary cannot hit f . So there is some kind of-- you use computational assumption. But then once we move to g star, we don't care about the assumption anymore. We're in statistical land. Even an all-powerful adversary cannot cheat.

AUDIENCE: Sorry, I don't mean to ask too many questions.

PROFESSOR: No, it's fantastic. The questions are great.

AUDIENCE: Is this still fine if the adversary picks the relation after he knows the hash key?

PROFESSOR: Exactly. Yeah, yeah, exactly. You're right. So think about this. The adversary gets a hash key, public key, and encryption of something, encryption of 0. Now he gives me f . He tells me f . With this f , I can cheat. I can find-- I can-- really, OK, can you distinguish between encryption of the all 0 and encryption-- assuming circular security? Yeah, because this is kind of some secret key. But let's-- so I'm using semantics. This g star depends on f and on the secret key. But assuming you can't break semantic security circle or security, then even if he chooses f , he cannot distinguish between encryption of g star and encryption of 0 assuming circular secure FHE.

AUDIENCE: I guess what prevents him from just picking FHE evaluation of α and-- why can't f just be the FHE evaluation? Why could-- he's just said--

PROFESSOR: OK, good point. Hold on. You're saying why-- no. OK, OK, OK, sorry. No.

AUDIENCE: It's kind of a weird--

PROFESSOR: Yeah, yeah, yeah, yeah, yeah. No, no, no, you're right. OK, sorry, sorry, sorry. He can choose f after he sees the hash family.

AUDIENCE: The hash family, not the specific key?

PROFESSOR: The hash family. No, once you have the specific key, then you're right. You'll just be the hash key. Sorry, I-- yeah, there's one hash family. After he sees the hash family, he can choose the relation. But of course for the specific hash key, he'll-- then--

AUDIENCE: I see.

PROFESSOR: And yeah, exactly.

AUDIENCE: And that should just follow from the fact that the ciphertext is randomized, right? Once you have an encryption of the g^h , predicting the evaluation should be very, very [INAUDIBLE].

PROFESSOR: Exactly, exactly. And just going back, this t prime is the time-- the size of g^* . So it's a little more than that because you also need to decrypt. But it's very, very similar to f , just a-- question?

AUDIENCE: This is the same kind of issue as to why it mattered that in some trek, the later bad relations depended on the verifier messages.

PROFESSOR: Exactly.

AUDIENCE: If that wasn't the case, then this wouldn't be--

PROFESSOR: Exactly. Exactly. Exactly, that was the same issue that came in [INAUDIBLE] that the things-- the t dependent-- yeah. Yeah. Yeah, great, great, great question. Anybody else? Yeah?

AUDIENCE: I still don't see how we can modify g^* through-- if f is not unique.

PROFESSOR: If f is what?

AUDIENCE: Is not unique.

PROFESSOR: OK, good. So if f is not unique, what I'm going to show-- suppose α has many f 's, f_1 up to f_d associated with them. Then I'm going to say, look, first let me argue that an adversary cannot-- the probability that he outputs f_1 is negligible. So for every-- the probability that the outputs α and f_1 of α is negligible. This I kind of convinced you. But that was also for f_2 and for f_3 .

AUDIENCE: Oh, so you use the remaining g^* .

PROFESSOR: Yeah, I'm just using--

AUDIENCE: [INAUDIBLE].

PROFESSOR: Exactly. In the analysis-- exactly, so I'm saying-- exactly, in the analysis, I'm going to change my g star. Exactly, fantastic. Yeah?

AUDIENCE: So is a similar idea that you would define [INAUDIBLE] or relations that have many betas instead of few betas?

PROFESSOR: Exactly. Doing the many betas, the only difference is it's exact same proof, except that I'll need to choose one of them. I'll degrade with d because I'm going to show for each one the probability that with that one you succeed is negligible. And then I'll just do a union bound.

AUDIENCE: I guess my question is a little different. I guess in CLW, they say [INAUDIBLE] there exists CI hash function for all relations that are searchable in time t . And I was trying to ask how to extend the R definition to handle cases where data isn't unique.

PROFESSOR: Yeah, exactly. So yeah, essentially think of R . Instead of searchable, it has f_1 . So for each α , there is f_1 of α up to f_d of α . These are the betas corresponding to it. So now that's your relation. That's-- yeah? OK, great. OK, we're out of time. Guys, thank you so much. Have a great, peaceful, fun Thanksgiving. And I'll see you, not the week after, but the week after after.