[SQUEAKING] [RUSTLING] [CLICKING]

PROFESSOR: Yes, of course, if we argue-- if we want to prove to him that x is in the language, it's essentially we're telling him, if you evaluate x on circuit C, you will get 1. Or maybe if there exists a witness that C of the witness C and x and the witness gives you 1. Of course, we assume the verifier knows the description of this circuit C.

And usually, and for our purposes, you think of C as being a uniform, like a Turing machine. So it can be generated efficiently from a Turing machine. But yes, the verifier can generate-- both the prover and verifier can generate the circuit.

So if you think of three SAT, the circuit takes as input all the assignments, and it checks that all the clauses are satisfied. That's an example of a circuit. Yeah,

AUDIENCE: But also checking [INAUDIBLE], isn't that quite similar to just running that circuit?

PROFESSOR: Yeah, it's like just-- yeah. So the prover essentially computes the circuit, all the gates in the circuit, hashes it, and then gives a BARG that each and every gate of the circuit is satisfied, that the value he hashed satisfy each-- respect each gate.

AUDIENCE: It seems like because the verifier is going to also check if each of those bits, corresponding in B0 B1. Because want to be out. Seems like that among those running the gate.

PROFESSOR: So the verifier doesn't know all these values, all these bits. The verifier only knows the hash. He gets only v. He doesn't know all these things, and he can't. There are too many of them.

He knows what the circuit is, but he doesn't run in time that depends on the circuit. The circuit is too big. He just knows in his head how to generate each gate in the circuit. So he doesn't know these bits. All he knows is the hash value and the BARG, which is succinct. Yeah?

AUDIENCE: Security is going to give us some lower bound on L, right? It just happens?

PROFESSOR: A lower bound on what?

AUDIENCE: On L.

PROFESSOR: On L, yes.

AUDIENCE: Yeah, OK.

PROFESSOR: Yeah, we're going to-- yeah, yeah, yeah. Now we're going to talk about security soundness, so we'll see.

One thing I want to say before we jump to soundness-- just note, because I said the CRS of the BARG, we generated with security parameter lambda, k, which is the number of gates, and then I said n prime. We'll see. We'll see what n prime is. I forgot to-- this is n prime. It's just security parameter.

N prime will be the length of the hash key, the length of v, which grows with L, and J. That's the n prime. I'm just mentioning it because I forgot to circle back to it.

OK, are you ready to talk about analyzing this? Because that's actually the very interesting part. So let's analyze. OK, we need to argue completeness and soundness. Completeness-- I don't want to waste our time because it's just trivial. Completeness just follows from the completeness of the BARG and completeness of some extractable hash.

So essentially, if you're honest, if x is in the language, you compute all the wires correctly, you hashed all the wires. And if you hashed all the wires correctly, then you have a valid witness for all the wires. So your BARG will be accepted. If you did everything correctly, everything is going to be accepted because the completeness of the underlying primitive. There's nothing interesting going on.

The interesting part is soundness. So how do we know you cannot cheat? And let me even jump and tell you, this thing is not really sound. But--

[LAUGHTER]

But we'll make it sound. So what do we know? So soundness-- so what do we want to prove? Let's try to hope for-- so again, there's nonadaptive and adaptive. You can think of both. But let's talk about, what does it mean to be, let's say, adaptively sound? So we need to prove that for any polynomial-size cheating prover, we want to say the probability that P star takes a CRS of the BARG-- of the SNARG-- so a hash key and a CRS for the BARG-- it takes the CRS, and it outputs some x and pi such that x is not in the language.

And pi, the SNARG is satisfied v and CRS x pi outputs 1. You want to say that this is negligible. That's what we need to prove. The probability that a cheating prover gets a CRS and generates any x that's not in the language and a proof that's accepted-- you can't do that. You can do that only with the Sanger probability.

Let me just mention another more weaker definition is the nonadaptive, which says, for any P star and for every x-- so he gives x ahead of time. For any P star and for any x not in the language, when he gets a random CRS, the probability that he managed to generate a fake proof that's accepted is negligible. So there's two variants-- one when he needs to choose a given x for all the CRSes, or he can choose an x depending on the CRS. Let's talk about the adaptive because it's just stronger, and we can get it, at least in some cases.

So today, the SNARGs is in the literature-- so achieve adaptive, some achieve nonadaptive. And in general, the ones for deterministic computations are adaptive. The one for nondeterministic computations are more on the nonadaptive side.

And we don't have SNARGs for all of NP. So at the end of the talk, at the end of the class, I'll leave 10 minutes to talk about where we're standing as a community. But both definitions are relevant, and we have results in both settings-- the adaptive and the nonadaptive.

But let's talk today on the adaptive setting. So that's what we want. But the question, do we get this? So let's try to see what this construction actually gives us.

So what I'm going to argue is actually this construction does not give a soundness. What it gives us is a form of local soundness. So I'll explain what this means.

So what I'm going to argue is that-- so claim the SNARG that I just constructed is, let me call it, locally sound or locally consistent. It's also people call locally consistent. And I'll explain what it is.

We want to prove soundness. So what do we want to prove? We want to prove that this SNARG, if you manage to say that the output is 1, there's actually an assignment to all the wires that give you 1. That means it's sound. We can't prove that.

What can we prove? We can prove that if you argue that the value is 1, I can use you to extract the value of any small set of wires-- how many L. So remember, our SNARG, we're binding on allocations. These allocations, I can extract.

You gave me a V that's binding allocation. I can extract. I can get from you the L wire values.

So I get L wires values from you. What I can argue about these L wire values is that locally they're consistent. So if there's a gate, if the L wire values-- let's say three of them are connected with a gate-- I can argue that this gate is satisfied. So I can argue this form of local consistency.

So I'm going to define what local consistency is. But just high level, what I can argue is not that I have a assignment of all the wires that are consistent inside, and the output is 1, which is what I want to argue. I can argue that I have an assignment that I can have a small window to the wires, a window of size L.

So I can look at these L wires, and they look consistent. And I can look at these L wires, and they look consistent. And the output is 1.

So it seems like, OK, local now, we only need to stitch all of them together and to get a final assignment, and we're done. So the goal is we're going to say, locally we're consistent. And local consistency implies global consistency. And so we get soundness.

So the plan-- we are going to prove local consistency. So that's check. And we're going to define it and prove it.

The local consistency implies global. That turns out to be much more difficult than we thought. So when we started working on this, we were like, oh, local consistency. But we shouldn't apply global. That should be good.

So we worked really hard local, local. And then turns out, local to global was another 10 years of work. And we're in progress, in progress.

So what we're going to see today is how to get local consistency. We're going to talk a little bit about why local implies global and why not. Why doesn't it always imply global? And hopefully, it will give you some food for thought.

So I'm going to argue that it's locally sound. And let me tell you what locally sound means. Let me first define it.

So the definition-- so here's what it means. It means that there exists-- so I'm now defining what this means. It means that there exists an extractor-- what we call sometimes a local assignment-- so some extractor, a local assignment-- generator, that's another name we use-- E-- so we have an extractor that generates assignments for us. This assignment takes only L wire values on the circuit.

So we have a circuit C. This extractor or local assignment generator, it gets L wires-- i1 up to il. They're all in the wires, which we denoted by n. So it takes l wires, and it outputs some x star in an instance and wire values-- and values wi1 up to wil.

This is what our local assignment generator outputs. For any i1 up to il, it outputs some instance and y values that satisfy the following such that 1 is local consistency, which means any two wires-- any wires that are connected by a gate satisfy the gate. And the output wire is a 1. And if one of them are connected to input wires, then it's consistent with the input wire. So local consistency just means it respects all gates consistent with x1 up to xn and out and output one.

So of course, if you just read wires here and here, like very spread, they are not connected with any gates, then the local consistency doesn't say anything because you can't determine the value of any gate. But any gate for which you have-- any gate for which the input and output wires are these or the output wire or the input wires, the gate should be satisfied. That's number 1.

And number 2 is what we call nonsignaling. I'll explain. The number 2 says the wire values that he gives here, that the extractor, the local assignment generator gives, the distribution-- so this is a randomized algorithm. This extractor is a randomized algorithm.

Given these wire, he gives a distribution over x star and wire values. The claim is the distribution of x star, for example, doesn't depend on i1 up to il. So the distribution of x star in this computationally looks the same, no matter which wires you give.

Moreover, the distribution of wi1 doesn't depend on i2 up to il. So let me say this formally. Formally, for every set I and J of size l-- let me-- sorry. For any I and J subset of n-- so for any set of wires of size l-- so take any set of wires, like i1 to il. Let me call the first set I, the second set J.

And what we know is if you look at the intersection of-- so if you look at-- how will I-- one second. Let me just think of notation.

If you look at extractor an I but only restricted to the coordinate in I intersect J. So the w's-- there's many. There's w for i1, i2. Let's look only at the w's in the locations that are both in I and in J.

The distribution of them looks like-- it looks at in distinct from the distribution. If you looked at these coordinates with set J. In other words, let's say the intersection of I and J are, I don't know, some set of wires. Put the set of wires in the beginning.

So if you look at a set of wires, look at their distribution. Their distribution does not depend on what the other wires are. If you look at the distribution of these, they will look the same whether the rest of the i's were of one form or another form.

At least computationally, they look the same. They need not be exactly the same. But nobody can distinguish between the two.

And this is including the x star. So E is like when the extractor here, it means x star and the wij for I, J in i. So I include the x star.

So even with x star, the x star here and the x star here are indistinguishable. X star, I always include. Even if it's empty, I include x star. So that's a nonsignaling. The distributions are indistinguishable.

And now I want to argue that we get-- I have this extractor. I'm going to show you. I'm going to construct this extractor. It's actually very easy to construct this extractor.

So if you give me a cheating prover, if you give me any prover. Forgot about cheating. You give me a prover, I can construct this local assignment generator from him.

And then all I need to do is argue that a local assignment generate implies that I have an assignment for all, like, not just locally consistent, but a fully consistent assignment to all the wires that give output 1. And therefore, x needs to be in the language. That's the proof strategy.

So what I'm going to do first is show you how we get-- so we're going to show how, given any p star, we can generate a local assignment generate. We can give local assignment to all the wires that are consistent and satisfy this kind of nonsignaling property. And then I'm going to show how to use this local assignment generator to get soundness. Questions? Yeah?

AUDIENCE: Is another issue with obtaining this soundness comes from the left side of BARG?

PROFESSOR: Actually, the semi-adaptive soundness of BARG is enough for me. It's enough. I don't need to use-- you'll see, actually. I don't need full adaptive BARGs. I never use it. It's enough for me that the BARG only has semi even though-- OK, that's a good point.

Note, I'm going to prove adaptive soundness. And you're saying, wait, but the BARG only has semi-adaptive soundness. Is that enough? Is it enough to use only semi-adaptive BARG and get fully adaptive SNARG? Yes.

I will not need the full adaptivity of the BARG. We'll see it next. I'll use it. You'll see it doesn't come up. So it's enough.

The problem is, to get this, the local assignment, I can get. The problem why, this implies global. So we'll see that. But the semi-- it seems counterintuitive. But actually, it's enough.

Was there another question? Do you have a question? Yeah?

AUDIENCE: Can you remind me what the x star is? Is that the same--

PROFESSOR: Oh, good, good, good. This is any-- OK, so good. What is x star? No, sorry. OK. OK, yeah.

Let me call it x tilde. Sorry. Let me-- you're right. The extract-- OK. Yeah.

So in p star outputs x tilde. It's crooked, kind of a cheating x. So let's use crooked, like tilde, to denote a crooked cheating thing. And he's also going to output some tilde because his is also crooked.

And we want to argue that this crooked x is indistinguishable from the crooked x given by the cheating prover. We're going to-- yeah. Oh, yeah, so I need to say that x was-- sorry. So good.

So let's say, again, the local assignment generator, it outputs a crooked x with Partial. Assignments to some of the wires, to the L wires such that it's consistent, nonsignaling, and it's also non-- and-- sorry, and x tilde is indistinguishable from the x tilde of p star. So p star also outputs x tilde. They look the same.

Note, if it was nonadaptive, it would be fixed. So then x tilde is fixed ahead of time. Nobody will need to generate them anymore. It's fixed. And the extractor, the local assignment generator, will only give values to the wires.

But in the adaptive setting, the p star, you give him CRS, he gives you a different x star. The x tilde. Sorry. He gives you a different x tilde.

So every time he gives you a local window, it will be with respect to a different x tilde, possibly, in the adaptive setting. In the nonadaptive setting, it's always going to be with the same x tilde. OK. Any questions?

Let's try to see why we get local-- how we construct this local assignment generator. So I have a cheating prover. So fix, so proof. So fix p star fix a poly size p star. I want to get a window to the-- there's the circuit.

We have the circuit C that has x, maybe a witness. I want to use the C, the p, the cheating prover. So he's going to give me-- OK, so fixed p star. For any CRS, he's going to give me x. I'm going to get from him in x and a window to some of the wires of the circuit.

So let's see how. So here's my extractor. The extractor or the local assignment generator-- it's going to use p star as an oracle. He's going to make use of p star. He's going to get i1 up to il. What does he do?

He wants to get the values of these wires. What does he do? He's going to run p star. He's going to run p star on a CRS that he chooses.

So he's going to generate CRS. We'll see how in a second. And then he's going to run p star on the CRS to get a value and a SNARG. And then he's going to extract the wires from this hash value.

So let's see. So how does he generate the CRS? CRS is going to be a pi for CRS BARG and a hash key. So he's going to generate hash key by running gen and these location, i1 up to il.

So let me actually-- he's going to first generate hash key, and trapdoor, and gen some extractable hash with 1 to the lambda number of wires in the circuit and i1 up to il. These are the wires he wants to extract because he has i1 up to il. He wants to extract the values of these wires. So he's going to give you a hash key that's extractable on these locations.

And then he's also going to generate gen for the BARG to get CRA BARG. He's just going to do this honestly. Here, he has 1 to the lambda k and n. This is n prime. The n prime, he's going to generate exactly as in the construction.

And then he runs. Now, run p star with hash key and CRS BARG. That's the CRS he gives p star. And get v and pi BARG. That's what the p star gives you.

Now, if this is rejected, start again. So if pi BARG-- or if v and pi BARG-- so if the verifier with input-- oh, wait. Sorry. X star also, x tilde. Sorry.

So the p star gives what? Some x tilde because it's adaptive. He gives him some x and a proof. The proof is a hash value and the BARG. That's what the proof is.

Now he checks. If v-- if the verifier rejects this proof-- if it's not a good proof-- so if v, given CRS-- which is these two. This is the CRS-- given CRS, given x tilde, and given pi-- this is pi-- outputs , if it rejects, try again. What do I mean by try again?

Again, generate a hash key, generate a BARG, run. If this is accepted-- if it's rejected, try again until you get something good.

Now, OK-- until we get something good. Once you get something good, you're happy. But then, what do you do? You output.

So try again. If success, then output the x tilde. And what wire values? Extract from the v.

So remember, this V. Is hash of all the wires of the circuit. Extract. You'll get the wires corresponding to i1 up to il.

So output this, and output extract with trapdoor the value v to get wi1 up to wil. That's it. That's what the extractor-- the local assignment generator does.

AUDIENCE: Why can you do this? The slide was adaptive. So if you change the CRS, why can you fix the x?

PROFESSOR: No, it's not fixed. You're right. So what do I do? I generate hash key and a BARG. I run him.

He gives me x tilde. If I'm happy with this proof, I output x tilde. The x tilde depends on the CRS.

So given i1 to il, I'm going to get a different x tilde for i1 up to il. So every time I get i1 to il, I generate a CRS for my SNARG. I run the prover. I get x tilde and a proof. If I'm happy with the proof, I output x tilde and extracted wires.

AUDIENCE: So these are just wire values. They're not witnesses.

PROFESSOR: Yeah. No, they're wire value. Sorry. These are not witnesses. Yeah, I use W for what wires, but it's also witness. Sorry. A bit of abuse of notation. Sorry. These are wire values.

Forget about witness. There's no witnesses now, just wires. Yeah. Yeah?

AUDIENCE: If you like rejection sampling until you get the verifier to accept, then how would the distribution of x going to be the distribution of p star?

PROFESSOR: OK. OK. That's good. So I'm going to answer. Did you--

AUDIENCE: It's a different question.

PROFESSOR: A different question. OK. Let me first answer this question.

The question was-- first, there's a couple of issues. First of all, how many times am I going to do this rejection sampling/ Am I going to run forever What's going on here?

And the answer is no. Actually, I'm not going to run forever. So fix a p star that, let's say, is accepted with non-negligible probability epsilon.

Epsilon is non-negligible. I'm going to try again. At most, I know poly in lambda and 1 over epsilon times.

So I'm going to do this only polynomially many times. Think of epsilon as 1 over poly lambda. So I'm going to run in time poly in lambda. OK.

Now, given that-- and OK. So the first thing I want to argue is the probability that I will-- and so I'm going to run the most poly lambda in 1 over epsilon time. That's it. And you say, what if all of them are reject? Then I'm going to say, OK, I was defeated.

What is the probability that I'll be defeated? Negligible because each time I'm starting fresh and the probability of defeat is only-- is 1 over-- OK, I'm accepted. OK, I'm accepted with probability epsilon. If I repeat it lambda over epsilon times, by just concentration bound, by Chernoff bound, I'm going to be accepted in one of them with very, very high probability-- exponential, like, exponential in 2 to the minus lambda or something like that. So if I repeat it enough times, one of them, I'm going to the probability of BARG of me being defeated is negligible.

So but now, going back to your question, the question is, how do I know that x tilde here has the same distribution as the x tilde of p star? Yeah, that's the question. So good.

So actually, I was a bit sloppy. When I say x tilde is in the signal from this, I mean, in this thing for x tilde condition on accept. Sorry, yeah. This is condition on p star accepting.

So I can assume without loss of-- yeah. So let's think of p star either as outputting but there's no point in him giving a bad proof. He's going to be rejected. Either he's defeat or giving x with a good proof.

So I always output x tilde with probability 1 minus negligible. And this x tilde is going to be distributed like x tilde conditioned on the p star not a boarding. Yeah?

AUDIENCE: So then those two distributions are actually identical right up to the slight probability that you give up at the end?

PROFESSOR: No. OK, so the question was, is the x tilde of p star actually identical to the x tilde of the extractor? And the answer is not quite because the x tilde of the extractor runs gen with these specific i's. And x tilde is run with p star with, let's say, 1, 2, 3, up to l, I don't know, like, fixed. And the I's change.

That's the only difference. And that's why it's indistinguishable because the p star doesn't know. Because of the index hiding, p star has no idea what the index-- which indices are sitting behind the hash key. So the x tilde he gives is the same. It's indistinguishable. Nikon, did you-- are you--

AUDIENCE: I just to make sure from the last point. Each trial here will work with probability epsilon minus negligible because of the index hiding, right?

PROFESSOR: Exactly. You're right. Yeah, you're right. Exactly. Epsilon minus-- that's why the epsilon has to be non-negligible, because you lose negligible. And it needs to be negligible compared to the epsilon.

AUDIENCE: So my question was, the extract algorithm for the SEH is only required to extract when V is an eval of x, right? Is that true, or is that not true?

PROFESSOR: OK. One second. Good point. No. OK. No.

The completeness was assuming-- holds. But I didn't write it actually down. Good point, Nikon. Thank you.

The sum of extractable hash-- I didn't say soundness of the-- I didn't write the soundness of this extract. The extract has the following property. If the cheating prover gives any-- hold on. Let me think for a second.

If the cheating prover gives an opening-- if you're extract-- If the hash key-- OK. Thank you. If the hash key is generated with gen of i and if an adversary generated an opening of i, a valid opening-- so he successfully opened the i-th bit somehow with a b-- then the extract will work and give you that same b. So let me write it down because actually we need it here.

For any A adversary and for every i, the probability that A and hash key, where the hash key is generated with i-- so hash key, trapdoor, they're generated with gen with 1 to the lambda n and i. If he managed to give me a value adversarial value, but he manages to open with a b and rho-- let me call it bi and row i-- such that the verifier accepted such that there-- so the probability that he gives me a tuple that verifies-- so hash key vib rho is one. And but extraction fails is negligible. So the probability that this accepts but extract is not b, bi is negligible.

So if you open, there must be valid. And if you think about the construction, if you open it, like, everything is in the correct encryption, encryption, encryption, the encryption must be correct, and the extractor will decrypt it. So if it doesn't open anything, v can be whatever, you may not be able to extract.

But as long as he gave you the correct path, then OK. So we'll use it here. Thank you. OK.

So now let's see why is-- but this is the extraction procedure. We'll analyze it in a second. We'll see why this is used.

But again, let's just recall. So what is the local assignment generator? It takes l wires. He will generate a CRS and give it to the cheating prover.

The CRS he generates consists of a hash key and a BARG and a CRS for the BARG. The CRS for the BARG, he just generates. And the CRS for the hash key, he generates with a trapdoor with respect to the wires that he needs to extract.

He gives this to the prover. And the prover gives him a hash value that's extracted on these wires and a BARG proof. If the BARG proof is rejected, he starts all over. So we're going to do it until we have something accepted.

Once it's accepted, we're going to just extract these wires because we have the trapdoor. The extractor has the trapdoor. We're going to-- or the e here, we're going to extract the wires. And that's what we're going to output-- the x tilde and the wires.

So now we need to argue nonsignaling, and we need to argue local consistency. Any questions before I go ahead and do it? So let's see.

Let's start with local consistency. Why are things locally consistent? So I claim that local consistency follows from the soundness of the BARG. So let's see why this is.

What do we have here? OK. Oh, well. I'm just going to-- we're going to need it, but it's OK. OK, OK.

So local consistency, I want to argue, follows from the correctness of the BARG-- from the soundness of the BARG. Why? Suppose, for contradiction-- so suppose that the extractor output x tilde and L wire are values that are not consistent. So the L wire values are not consistent.

Then what does it mean they're not consistent? There's one gate that they don't respect. Now, remember-- so OK. So there's one gate that they don't respect.

So suppose E and input i1 up to il output w1 up to wi1 up to wil that do not respect some gate-- let's fix the gate-- some gate J with non-negligible probability. If there's one gate doesn't respect, choose one at random. So it won't respect this random one with probability. I mean, there exists one gate that's not respect probability 1 over poly. So suppose there exists some gate J that this does not with probability 1 over poly.

Now let's look at the BARG. What does the BARG say? The BARG tells me that all the gates are respected. So what I'm going to do in the BARG, I'm going to-- so remember, in the BARG, I have this somewhere. Now I'm going to the adaptive versus the why-- some are adaptive. Soundness of BARG is enough. Why?

If there some gate J that's not respect probability 1 over poly, in the BARG construction in the CRS, I'm going to hardwire-- sorry. OK, let me construct a cheating prover for the BARG. So we're going to construct a cheating prover for BARG that cheats. He's going to be somewhere adaptive soundness, and he's going to cheat on J and this specific J.

I don't care about what the CRS is yet. I'm going to be semi-adaptive. So I'm going to tell you, my cheating prover, he may choose the x1 star up to xk star. The actual instance is later.

But which one he's going to choose, he's going to cheat on, I'm going to tell you now. It's J. So it's going to be going to be nonadaptive and where he's cheating. That's the semi-adaptive. Yeah?

AUDIENCE: But for that local consistency, is the statement that the instructor output-- is that the same as the prover is trying to cheat on?

PROFESSOR: So you're asking, is there a match? How do we match these two? OK, so wait. Let me tell you the cheating prover.

So I'm going to construct a cheating prover-- let's say p double star for the BARG. Here's what my p double star does. My p double star, he gets CRS of BARG.

What he's going to do, he's going to run piece-- he's going to generate hash key on his own. And he's going to run p star, and the CRS BARG, and the hash key that he gives him. He's going to get v and pi BARG.

And now he's going to cheat. P double star will cheat. And here's the instances that he gives. So he's going to give the instances adaptively.

The instances are going to be hash key V and J for J goes from 1 to k, the number of gates. These are going to be the instance. So this is going to be xj star or xi. Sorry. That's a certain go I.

So that's the xi star. And the BARG is going to be this. So that's how I'm cheating. I have k instances that I chose adaptively. But I'm arguing that the j-th one, which corresponds to gate j, is actually-- it's false.

This xj star is not in the language because I'm binding on this j, and what's sitting there is false. So even if you're apart from this just does not exist of an opening. Because I'm binding on these locations, what I know from the binding condition, which is here, what I know from the binding condition-- there is no opening for the wrong value. It just doesn't exist. So there is no witness for xj star.

But look, I have p double star managed to come up with a proof even though before, in the semi-adaptive way, he gave me a j, a gate. Before I got even the CRS, he gave me a gate and nevertheless managed to produce a BARG that all my x star-- and a bunch of x stars. And the BARG is satisfied. But the j-th instance does not have a witness-- contradiction.

So the local consistency really follows from the soundness of the BARG. And all we need is a semi-adaptive soundness of the BARG because if you're somewhere in that local consistency-- not local consistent, even if you don't need to be-- even if there's one gate on which you're not local consistency only with probability epsilon, epsilon could be 1 over k. It can be very small.

The claim is on each and every gate. Local consistency holds probability 1 minus negligible because otherwise, take that gate, and on that gate, we're going to cheat in the BARG. That's the instance, in which we're going to cheat on the BARG.

So that's local consistency. And now you can ask, what about the nonsignaling? And the nonsignaling really follows from the fact from the index hiding. Are we OK with local consistency? OK. Any questions before I run-- actually, yeah?

AUDIENCE: Is the i [? changed ?] [INAUDIBLE]?

PROFESSOR: OK. Oh, sorry. I changed-- yeah, sorry. This is J.

Thanks. Sorry about that. This is-- yeah.

So I give you all-- this is an index language. So I give the instances are-- oh, ah, sorry. No.

These are all for all i. Just J is a specific J, so I don't want to-- J is the one that I'm binding on. But I give you the instances are-- yeah. All of them are i.

Sorry. J is the specific one on which I'm going to be-- I'm going to cheat on. All of them are i. Sorry about the-- yeah. Any other questions?

So now the question is, now, what about nonsignaling? And nonsignaling-- I don't want to go into a lot of depth. But let me be hand-wavy here.

The idea of why we're non-- so we want to argue that the local assignment generator-- the assignment values that he gives for any subset of wires do not depend on the other wires that is given to him. So if we look, if the extractor is given a bunch of wires and you look at the assignment and a subset, the distribution of the assignment that you get, it looks the same no matter what the other-- the rest of the wires are. And why is that?

It just follows from the fact that-- I just erased it here. So it follows from the fact that when we do a sum or extractable hash with many-- when we construct a somewhat extractable hash with many indices that were extractable on many indices, we want to have the property that even if we extract it on some indices, the index hiding still holds for the other.

So in other words, think of the way we're going to construct the somewhat extractable hash is by repetition. We're going to repeat it many, many times. So even if you saw if I extracted wire i1, wire i2, wire i3, the indices behind the other wires are completely hidden.

So here, when you look at extract and a subset of witnesses, how do you find extract these? How does the local assignment generator generate these wires? He generates these wires using the corresponding trapdoors without the other trapdoors. So he actually has no idea. This should not signal anything about the value of the other wires.

So again, let me say it again. Maybe I didn't say it quite right. But the local assignment generator, when he runs the cheating prover and the cheating prover, when you extract from the cheating prover the value of some of the wires, you need only to do that using the trapdoors for these wires.

The cheating prover has no idea what the other wires-- what the trap door of the other wires are. So these values should not depend at all about the other indices used in the other repetitions. Otherwise, you've broken the timing for these repetitions.

So the point is the nonsignaling really follows in a sense from the nonsignaling in the somewhat extractable hash. The somewhat extractable hash, when we repeat, when we do repetition, what we can argue is any cheating prover who, given a hash key, generates a value. If you extract some of the wires, the value you get do not depend on the other indices. This is a property of the somewhat extractable hash. So nonsignaling here follows directly from the nonsignaling of the somewhat extractable hash.

So I'm running out of time. I don't want to go too much in depth because I think it's not as interesting. I want to actually go and talk about why is this not sufficient and when is it sufficient. So let's call-- let's say-- OK, so we have-- suppose we have a cheating prover. We manage to construct from him this local assignment generator.

So we have the circuit. We have a cheating prover. He gave me an x.

We want to argue. It's in the language. If he convinced me, it must be in the language. That's what soundness means.

What do we know? We know when we look at any window, everything looks good. And when you look at the window that contains the output, it's 1. Does that mean we can have an entire assignment to all the wires that everything is consistent, in which case, we can then sound? So can we piece these things together?

So the answer is yes and no. OK. So in general for np, the answer is no. We don't know how to piece it together.

Moreover, we believe that you can't piece it together for any np language. We believe that there exists very hard NP languages that there's no way you can piece it together. But let me tell you why. Why is it hard to piece things together?

And the reason is, even-- let's even think, now, for simplicity, I think, it would be easier to think, let's say, even in the nonadaptive version, x is fixed. Can we argue that for this fixed x, we can piece everything together? And the reason why it's hard to piece everything together is because this local assignment generator, every time you give them a bunch of wires, think of even a valid kind of prover. Think of a valid prover.

Every time you run him, he uses a different witnesses. Maybe he has many, many witnesses in his disposal. Every time you run him, he's honest, but with a different witness. So it's actually in the language. But the question is, are we able to construct a satisfying assignment?

And the reason why this is hard is-- or seems hard is because, OK, we have our local assignment generator. We give him a bunch of wires. He chooses one of his witnesses at random-- because that's what the cheating prover does-- and gives him an assignment with this, the witness.

OK, local consistent. We're fine. We're happy.

Now we run him again. He uses a different witness. It doesn't match. How do we match things together?

So that's why in the nondeterministic case things are really difficult. And we do have some results, but for very specific languages. Let me not go into that.

Let's talk about deterministic languages. That seems to be much easier. So let me first convince you that we can do it. And then I'll show you where I cheated.

So let's first take a circuit. We have a local assignment generator. So let me convince you that the output must be 1.

So we have an input-- x1 up to xn. I'm going to call my local assignment generator. I'm going to tell him, give me the values here.

Or here, let me-- I have l, right? So let me use my l here, here, and here. I don't know. He gives me values.

Now, this must be consistent because I know local consistency holds probability 1 minus negligible. And all of them are consistent, which means these are the correct values. Now we're deterministic. We're talking about a deterministic circuit.

So if this is locally consistent, it means this is the correct value. I don't know. If it's an AND, it's an AND, whatever it is. So all of these are correct.

I can argue, when I run the cheating-- the assignment-- so I want to argue essentially any local assignment generator, every wire he gives me must be the correct one. And then I'm done because the output is the correct. And if he gives me output 1, voila. It's correct. I'm done.

So I'm going to argue, kind of by induction, that every wire he gives me must be the correct one. Why? OK. Let's try-- so I'm going to argue layer by layer. The induction is going to be by the-- think of the circuit as layered. And I'm going to argue this by induction on the layers.

Layer 1-- what does the cheating prover gives me? He gives me an assignment. Now, I know I proved it locally satisfiable, which means it satisfies the gate. So it's correct. So layer 1, correct.

Let's go to layer 2. Layer 2, now I'm going to ask him this. Let's say I'm going to ask him these. I want to argue that this is correct. All I know is that it's locally consistent.

But I know that these are correct. Why are these correct? Because of the nonsignaling, I know. If I asked, give me these wires, give me these wires with the input, it was correct. I got the correct answer.

So now when I ask the extract the local assignment generator, give me these wires with these, we said the distribution is the same up to-- or computationally indistinguishable. So when I asked him, give me the value of this wire, when I asked with the inputs or asked, give me these wires with this, it's the same distribution. If it was correct before, it's correct now.

So these are correct, and this is locally consistent. So this is also correct. So these layers are correct.

And again, now let's look at these. Is this correct? Yeah, because these still need to be correct because we argued it correct. And it doesn't depend on the distribution. It doesn't depend on the-- every time I ask him this, I get probability 1 minus negligible-- so correct. So this is correct.

By induction, the output is correct. I know it's 1. That's what the local assignment generator promised me. So it's 1 and correct. I'm done. That's the proof.

Moreover, let me tell you something. When we wrote this proof, we wrote it. We were happy with it. We literally almost submitted this thing.

A couple of days before the deadline, we noticed there's a bug. This is wrong. This does not work.

I'll tell you why it doesn't work as opposed to asking you because I think it's so subtle. But does anybody see why it doesn't work? It's very, very hard to see, in my opinion. But I mean, clearly, I didn't see it. But anybody see why I cheated here? Yeah?

AUDIENCE: Is it because we restricted to deterministic witness?

PROFESSOR: Yeah. So what? Why? Yeah, there's only the input. Actually, the fact that there's deterministic, it's helpful for us because now there's a notion of correctness. Once there's a witness, there's no-- what do you mean correct? Depends on the witness. Every wire has, like, if you use this witness, it can be this value.

Here, because it's deterministic, there's only one value for each wire. And so there's the notion of correct value. And I can argue by induction that everything is correct.

So I'll tell you why. I suffered through this. I have the desire to have you-- to torture you a little bit to see why. And so like I-- like the experience I had. But given that you only have 10 minutes left, I'll tell you.

So here's why. So local consistency holds-- or yeah, local consistency holds with probability-- or the nonsignaling, everything holds with probability 1 minus negligible, not with probability 1. If everything with probability 1-- 1, local consistency-- then this consists only 1. Then, everything would work out.

If I promised you local consistency with probability 1 and nonsignaling with probability 1, like it's exactly the same, we're good. Then I can prove.

Now, it's used in cryptography, oh, come on, negligible. It's 0. Here, It's not zero. It turns out, negligible is a pain. This is the entire problem, the negligible.

And I'll tell you why. It seems weird, but here's why. The reason is this is correct with all-but-negligible probability. So this is correct, except probability mu. This is correct, except with probability mu.

But now this is correct if this needs to be correct, and this needs to be correct. If one of them is incorrect, you're not correct. So this is correct. Is probably correct, except probability 2 mu.

So here, you have 2 mu. Now, 2 mu is also negligible. This is also acceptable 2 mu. But this-- for this to be correct, both of them need to be correct.

So here, you have 4 mu and then 8 mu. You can get 2 to the depth times mu. Now, if the depth is more than log, it's not negligible anymore.

So the problem is 2 to the depth may be much bigger than 1 over mu. So there's this exponential curse of-- we carry-- the error grows by a factor of 2 each time and eventually it can kill us. That's the bug.

So this works if it's bound to depth. Log depth, yes. More than that, not clear. OK.

So what do we do? So to fix this, there is two ways to fix this. Option number 1, which is what we did originally because we didn't have a better idea, is we said, you know what? Let's assume that this circuit is very narrow. Think of bounded depth deterministic computations-- sorry, bounded space.

So think of a distributed computation where each configuration is very small-- so like a Turing machine that has very-- the tableau is very thin and long. And then we said, you know what? Let's make L, like where you're statistically bounded on-- let's make it an entire configuration. So yeah, you pay with an entire-- with a space. The SNARG, the proof will grow with the space.

But now we say, you can read the entire thing. You can read-- so here's the tableau. You can read the entire computation. You can read the entire configuration.

And now we say, the first configuration is correct, so the probability mu. The second configuration, if this was correct, this is 2 mu, 3 mu, 4 mu. It doesn't grow exponential anymore because you read the entire thing. So I'm going to read these two configurations together. If this was consistent, this is consistent.

There's always a probability of error that I'm incurring. Fine. But it doesn't double because I'm reading the entire thing.

I said, if this was a correct configuration, then when I read this with this, this is going to be correct configuration. But I read the entire thing, so I don't double up things. So we get SNARG for any computation that's bounded space where the SNARG grows with the space.

OK. But let me tell you, we can do better. We can actually-- we don't need to grow with the space. So I'll just say a word because I'm running out of time, and I want to do a quick summary. But the idea of not running into this exponential curse idea is to say the following.

Let's do this hash in the BARG. So what did we do? This SNARG is like, hash all the wires and then BARG. The idea is don't do hash and BARG in the original circuit. Don't just do hash and BARG in the verification circuit.

Do hash and BARG and an extended circuit. Change the circuit. Do hash and BARG on a slightly more elaborate circuit.

So what is the more elaborate circuit? Let me give you an example. One example you can do-- take each kind of layer and add a Merkle hash in the circuit. So now before you add a verification circuit, change it a little bit. How do you change it?

You compute each one, and then you compute a Merkle hash. Compute the next layer, and compute a Merkle hash. Compute the next layer, compute a Merkle hash. It's weird. Why would you do that? Do that.

Moreover, also compute for every wire like the opening of the Merkle hash. And add all these Merkle hashes, and I add all these openings to the circuit. Now, that seems a ridiculous thing to do. Why would you add to the circuit that you're trying to verify that it's correct for each wire, Merkle hash all the wires, and add wires that check opening? Like, what? This sounds really weird and not a very-- a weird thing to do.

Sorry, you don't need to check opening. You just add Merkle hashes. And for each wire, add Merkle hash to the wire. For each layer, add a Merkle hash, so on, and so forth.

Why is adding Merkle hashes helpful at all? So I'll tell you why it's helpful, because now we have the local assignment generator. And now in the local assignment generator, what we can do-- we said, the first layer, each one is correct.

Now we're going to argue, this implies that the Merkle root is correct, that this must be correct. Now, when we argue it, we do pay exponential with the depth. So this is a computation.

And the error does grow, but the depth is long. Merkle hash is a very low-depth computation, so it grows. But polynomially, it's OK.

Now, when we read things here, we're going to read them together with a root. We're not going to just-- we're going to say that when we ask the local assignment generator, what is the value? Give me the value here, here, here, we're going to also say, and give me the value of the root and the paths corresponding to this and this.

And now the thing is the following. If the root was correct, both of them have to be correct unless you broke the hash function. So you don't need to get this independence. Before, we were like, oh, we needed this to be correct and this to be correct, and things doubled.

Here, we can say, if the root is correct, we know that both of them have to be correct. So it's enough to argue that the root is correct. So it's a little bit simulates the case of reading the entire configuration or an entire layer, which is what we did in the bounded space.

So I don't want to go-- I think that's all I want to say. But essentially, what I want to mention is this local satisfiability, or local soundness, or local consistency does not necessarily imply global correctness. Actually, , to use it to get global correctness, often, we need to change the circuit and do circuit gymnastics in weird ways to get it to work.

And this is still work in progress. Us as a community are still combating with this. I see smiles because my students have been suffering with me through this a lot. It's like half smiling, half crying. So that's where we are today.

So let me use the last three minutes or so to do a quick summary. So first of all, before I summarize, if you have any feedback, there's feedback forms. Please submit, so I'll improve.

Or I'll keep on doing things that worked to try not to do things that didn't work. So it will be great, great, great. if you can submit the feedback form. That'll be really, really helpful.

So with that, let me just-- I want to reflect a couple minutes on this journey and where we are and how. So if you remember, we started by saying, look, proofs has been something that mathematicians have been working with and thinking about for thousands of years. And proofs have always been very specific form of logical deductions.

And cryptography changed this. With the introduction, with the thinking about zero-knowledge proofs, they defined this notion of interactive proof, a new proof model. And this new way of thinking about proofs was a huge, huge success story for cryptography, for complexity.

And we noticed that these interactive proof systems turned out to be very, very powerful. And the main work that demonstrated that was the subject protocol. That was the first protocol that, wow, you can do something that we have no idea how to do classically.

And that gave birth to the IP equals p space theorem. Once you had the sum-check from that with a little more work, you can do the IP equals p space theorem. And what we showed in this class is, actually, we went through GKR. So we showed that if you have sum-check, you can get actually-- just by a very simple idea of doing SNARG with a sum-check layer by layer, you can get a succinct, interactive proof for any bounded depth computation that's doubly efficient. So the prover is efficient, and the verifier is succinct or very efficient.

And then we shoe the IP equals p space as a corollary. We showed that you can get PCP from it. And then we move to arguments, said, well, OK, that's where we are with proofs with statistical soundness. And then we said, well, if you're willing to rely, to reduce soundness to computational soundness, then we show that Kilian-Micali, you can get everything, like, really efficient interactive arguments by taking PCP and hashing them down.

And then there was a question, OK, we can have interactive arguments. Can we have SNARGs? Can we eliminate interaction? And then we talked about Fiat-Shamir, went through zero-knowledge a little bit because that's the first time where Fiat-Shamir proved security of it, was in the context of zero-knowledge.

But then we showed-- or Zhengzhong showed that actually you can do Fiat-Shamir also in the context of BARGs. So we have an interactive batch argument and you can-- or he constructed interactive batch argument showed you can use-- I don't know. He didn't get into detail. But it turns out, you can use similar techniques to show Fiat-Shamir is secure-- or these techniques, plus, plus show that Fiat-Shamir secure when applied to the interactive batch argument. And then you get this noninteractive BARG.

And today, to close this entire journey, we showed how to get SNARGs from these BARGs. And as I said, we have it for deterministic computation. We have it for some nondeterministic computation. Constructing it for all of NP is still a major open question. So I invite you all to work on it with me.

But I want to say one more thing about this area, which is, it's really nice because this is one example where this area has a beautiful, beautiful math, in my opinion. So it's very theoretical with beautiful mathematical ideas and applied. So this area has been a really nice bonding between theoreticians and practitioners. These things are used in particular by a lot of bankers.

And people start using blockchains, and now bank systems and so on are interested in it. So it's been a really nice example where really beautiful theory is used, like, the theory itself is really deployed and used. So I think this is an example of success story for theoretical cryptography.

OK. Thank you, guys. It was really, really fun teaching you. Thank you.

[APPLAUSE]