[SQUEAKING]

[RUSTLING]

[CLICKING]

YAEL KALAI: OK, so let's start. Thank you, everyone, for coming. This is going to be an advanced class on cryptography.

Those who do not have background in cryptography, it's OK. It's not actually hardcore cryptography. This is actually a class on the evolution of proofs in computer science.

So this class is really about proofs. That's what we're going to learn, and how proofs evolved with the years, starting from the mid-80s all the way to today and how we think about proofs differently today. So this is what this class is about.

Actually, before I go into the content of the class and what I'm going to be teaching, first of all, quick question. How many students here are not MIT students, but cross-registered students? OK, a bunch.

So do you guys have access-- so I hope I made the website public. It's on Canvas. But now I think it should be public.

So you should have access. If anyone has problems with accessing the website, just let me know. I'll try to fix it. So this class is meeting once a week weekly from 1:00 to 4:00 PM. In terms of expectations or what-- so the expectation's that you'll enjoy it. I hope you will. I think it's really, really beautiful material.

But beyond having fun, I want to ask-- I have two kind of requirements from you guys. So one, I want to have Scribenotes for the classes. I have written notes. I can give it to you to help you out.

So we have about-- actually, we have 11 classes total in this class. But we're going to have a bunch of-- one day, we're going to have a guest lecture by Oded Regev, who's an-- he has a beautiful result. He's going to come here on October 6 and present it to us. So that won't need to be scribed. And then one day there's a crypto day, which also won't need to be scribed. Well, there's more-- anybody wants to sit

But for the rest, we'll need scribe notes. I'm happy if you want to do it in pairs to the scribe, unless we'll run out of people, and then we'll need to keep it for one. So that's one requirement.

And then I was told that I need to do something more. That's not enough. So we'll do a P set sometime in the middle of the course. It won't be anything too difficult. It's mainly to make sure you're on board.

But I really do recommend that before class, just every time before-- so here's a real homework. Every time before you come to class, in the website, there's reading. I really recommend you look at the reading and try to read.

So there's a beautiful survey by Justin Taylor. There's a link on the website. The survey is on proof system. It's more on the applied side of proof system. This is a more theory class. But he does have really beautifully written sections that are very theoretical that I'm going to link to.

So I recommend, or you can think of it as informal homework before class to do the reading. I think it will be very helpful for understanding this class. And also, after the lecture to look at the reading, and I think it will be very, very helpful.

So any questions about bureaucracy stuff? So let me tell you a little bit more about what the class is actually about. So proofs is something that has been studied in mathematics for thousands of years. And usually when people think about what a proof is, a proof, you write kind line by line, and you can verify it line by line to make sure the proof is correct. That's how most people think of what a proof is.

Starting from the mid-80s, in computer science, in particular in cryptography, the way we thought about proofs started changing. This came about because of the idea of zero-knowledge proofs. So there was a hope to construct proofs that reveal no information about why the claim is true.

And around '85, Goldwasser, Micali, and Rackoff constructed the first zero-knowledge proof. By the way, this class is not about zero knowledge. Actually, it's not about crypto, per se. It's just about coming up with new proof systems.

Do you have a question? No? OK. But they constructed-- they wanted to construct these proofs that reveal no information. And to do that, they needed to change the model. Turns out they could not do it using what we think of, a classical proof.

And the way they change the model is by defining a new proof system, a new model, which is called interactive proofs. We're going to talk about this today. But they defined this notion of interactive proof.

So we're going to start this course by talking about what interactive proofs are. And we're going to construct interactive proofs. In the '80s and '90s, when people thought about interactive proofs, usually the model is we think of the verifier as being efficient.

For us, efficient means polynomial time. But of course, in practice, it has a more specific meaning. But for now, think of efficient as being polynomial time. And the verifier can be all powerful.

So the time complexity of the prover was not even a consideration. Actually, they called the prover Merlin, like he's a wizard who can do whatever the hell he wants, like infinite power.

What we're going to talk about after introducing this model and giving some protocols, we're going to talk about the model of double efficient g proofs. And here, we say, no. The proof is not all powerful. We don't have all-powerful machines in the world.

Every machine is bounded. So what we want, yes, the verifier should be very efficient, but the prover should not be all-powerful leader. So we care about the prover running time, too. That's what's called doubly efficient interactive proofs.

We're going to construct double efficient interactive proof. We're going to show the GKR protocol, a specific protocol, and then we're going to show application of this protocol. And we're going to see application to really nice models such as PCP, which is probabilistically checkable proofs, MIP, multiprover interactive proofs.

And in particular, we're going to show how to use this to get a-- reprove a celebrated result, which is the IP equals P space theorem of Shamir. So if you don't know any of these buzzwords, I'll explain everything. I'm just kind of doing a very quick overview for those who heard these buzzwords a little bit so you'll know what to expect.

So up to here, no crypto at all. This is not at all about crypto. It's just about proof systems. Then we're going to show how to use cryptography to get our proofs more succinct. And then we're going to leave the idea that-- the classical idea of a proof, that if the theorem is false, you can't cheat. It's impossible to cheat.

There's no fake proof. There's no false proof. There's none-- if the theorem is incorrect, there's no proof, period. We're going to use crypto to say maybe there are false proofs, but they're very hard to find.

So the only way you can find a convincing proof to a false statement is by breaking a hard cryptographic problem. You can factor large numbers, you can break lattice assumptions, and so on. So we're going to use cryptography to make our proofs more succinct.

But then what's going on here is that our proofs we can make them succinct, but they're still interactive. And then we're going to show how to use this beautiful Fiat-Shamir paradigm, which is a paradigm for reducing interaction.

And we're going to study the security of this paradigm and show how to make this paradigm secure. So these are results that have happened very recently. Actually, one of the students here, Alex Lombardi at MIT, was-- Vinod's student was one of the people who showed the security of this paradigm.

So we're going to show it. We're going to use it then to show how to get our doubly efficient interactive proof, how to make it non-interactive anymore. So make it completely non-interactive. We're going to also show how to use this Fiat-Shamir paradigm to do batch proofs non-interactively in a very succinct way.

So taking a bunch of claims and giving one short proof from them. And finally, we're going to show how to use that to go all the way to what's called a SNARG, which is a succinct, non-interactive argument. "Argument" is just another word for a computationally sound proof.

So this is kind of very, very high level of what you should expect to learn in this class. If you didn't understand a word what I said, then this is what-- the goal is that you will understand this by the end of this class.

Questions before I start? OK. I need a volunteer to scribe for today. Oh, fantastic. Thanks.

So with that, let's start. So let's start with the notion of interactive proofs. And before I go to interactive proofs, let me just-- even before interactive proofs, let's look at the class NP, non-deterministic polynomial time. Even this class is about proof systems.

So what is-- this class is trying to-- by the way, there's another chair here. For those who don't have a chair, there's one right here. Great.

Great. So even before we go to the interactive setting, let's even think of the class NP. What is the class NP? If you think about it, it's all the languages that have a proof, that's the witness, that can be verified by polynomial time.

So we want to think-- the framework for this class is the following. We want to ask-- we have-- the verifiers are polynomial time, or even-- they're very efficient. What can you prove to these verifiers?

So that's exactly what NP captures. NP is all the languages that you can prove membership in the language to a polynomial time verifier. So NP is exactly the set of languages, membership can be proved in poly time. So for a-- or can prove to a poly time. Maybe I should say, can prove to a polynomial time verifier.

So a polynomial time verifier can verify membership in the language. So far so good? But NP is not that rich. So we want more, more than NP.

So one thing we can add-- so what is the class IP? The first thing that-- so IP, interactive proofs and two components. So in NP, the proof is classical. It's the class of languages for which a membership can be proved using a classical proof. A proof that you can verify just running a Turing machine.

Interactive proofs are different in two ways. First of all, we allow interaction between the prover and the verifier. They can talk back and forth. Importantly, we allow randomization.

Now note. What do we mean, randomization? The verifier can be randomized and there can be some chance that the theorem is false and he'll accept. But that's very, very small.

We can make it kind of-- if the communication complexity is, let's say, n, we can make-- you'll accept the false statement with probability something like 2 to the minus n. So we can make it very, very small. But there is a nonzero chance that you'll accept a false statement. So in that sense, we weaken the model.

Now note, if there is no randomization, if we force the verifier to be deterministic, if the interaction-- so again, the interactive proof is a kind of interactive protocol between prover and verifier. If the prover knew what the verifier was going to send, if he was completely deterministic, then the prover can just generate the entire transcript and give it as a proof.

So it will really be NP. There was no point of this interaction. So really, the power of this IP comes from the fact that the verifier is interactive.

It comes from the fact that the verifier is interactive, is randomized. Sorry, it's randomized, and the prover doesn't know what this randomness is going to be. So if I'm trying to prove to you a statement, I send you my first message. I don't know what question you're going to ask me.

You're going to answer some random question and I'll need to give an answer. And then I don't know what the next question. If I knew all your questions in advance, then there's no point. It's like it loses the power. So really, the power comes from the fact that there's interaction and the prover cannot predict ahead of time what questions he's going to get from the verifier.

So any questions? So I can identify this formally yet. I'll define it formally in a bit. But I hope I convinced you that without randomness, this model is really useless. There's no point.

How about just-- how about without interaction? Let's say I had only randomness. Do I get more?

And actually-- so what if we allow only randomness? So proofs. So let's talk about non-interactive proofs where the verifier, V, is randomized.

By the way, I forgot to say, please stop me at any point and ask me questions. I like questions. Questions are good for me to-- can't slow me down. It gives you an opportunity to digest. So it's a great thing.

So please, just at any point, stop me with questions. Questions? No. OK.

So good. So what about if we-- so we said if we get rid of randomness, we get nothing. What if we get rid of interaction and we just allow randomness? Do we get more? And actually, we do get more.

And actually, this complexity test has a name. This is MA for Merlin Arthur. "Arthur" is known as a verifier, "Merlin" is known as a prover. So the prover sends a message. Arthur, the verifier, can toss coins and based on that, decide if to accept or not.

But anyway, the non-interactive randomized version is believed to be-- I mean, it may be more powerful than NP. We actually don't know. But it seems there's some more power.

And actually, let me give you an example. This is an example that will kind be useful for us later. So it's not only an example. But let me give you an example where randomness can be helpful for us.

So suppose you want to verify-- let's say you have two n by n matrices, A and B. These are n by n matrices. Let's say over 0, 1, or over a finite field.

And let's say you want to verify-- suppose you want to verify that some matrix, C, is equal to A times B. So you're also given, let's say, C. So you need to verify task of matrix multiplication.

We can do it efficiently. And in general, the best known algorithm is something like order O of n to the 236 maybe, something like that. That's the best known matrix multiplication algorithm. Why are you looking at me?

Oh, the number? Yeah, close. But it turns out that using randomization, you can do it more efficiently. So let's show how you can verify that a matrix, C, is a product of two matrices, A and B in time order n squared.

So kind of the best can do because you need to read the input. So how will you do that? Here is the idea. So here's a randomized algorithm for verifying this.

So let's assume that f-- first assume-- this is kind of without loss of generality, that F is much bigger than n. If not, you can take kind of an extension field of F embedded in a much bigger field. So just let's assume-- those who are not comfortable with extension fields, let's assume it's just much bigger.

Then what do you do? Here's my algorithm. What I'm going to do? I'm going to choose a random element in the field.

I'm going to consider x, which is just 1, r, r squared up to r to the n minus 1. So all the powers of r. I chose a random r in the field. I look at all the powers of r. And then what do I do? Yeah?

STUDENT: Check whether it's Cx [INAUDIBLE].

YAEL KALAI: Exactly. And then I'm going to check if C times x, I guess transpose, because I wrote it, is equal to AB times x. That's my algorithm. So note-- yeah.

So note this is just a vector, and vector times matrix-- so to multiply a matrix by a vector is time n squared, as opposed to n to the 2, 3, 6. So this is more efficient. Why is it OK?

So of course, it's complete. If these are equal, I'm going to accept. But if they're not equal? So why is it sound?

So maybe you can convince me. Let's say they're not equal. Maybe you can still convince me that they're equal. That would be a problem. So-- yeah?

STUDENT: You could move the term over and write it as a polynomial that opens that r. And since the size of the field is much greater than n, it's less probable that in a random point [INAUDIBLE].

YAEL KALAI: Fantastic. Fantastic. So let me just repeat what you said a bit more formally. So suppose they're not equal. There's one row that they're not equal in. Let's focus on that row.

So you can think of-- when I do C times x, let's say C1, the first row of C. When I do C times x, you can think of this as I take-- I look at this, the row, as a polynomial. So if you look at the first row, maybe I'll write it as C, 1, 1 up to-- or C 1, 0 up to C1, n minus 1.

Let's look at this n row. When I multiply by x, which is 1 r, r squared, this is nothing but sum C1i x to the i. Or, sorry, r to the i.

So now what do I check? So now if this row-- let's say it is different than the row here. Then I have two different polynomial.

So if I write the row there, let's say, as I don't know, D0, or D1, 0 up to D1 n minus 1, let's say this is the row corresponding to A times B. If these are different-- if these are different, then I have two different polynomials that agree on a random out-- two random degree n minus 1 polynomials.

So the polynomial is defined by this row. These are the coefficients of the polynomial. There are n coefficient. So it's a degree n minus 1 polynomial. So when would I accept a false claim?

If there exists two things that are equal that are different. There exists a row in C that's different than a row in A times B. So there exists a polynomial of degree n minus 1 here and a different polynomial degree n minus 1. And these two polynomials are equal in this r.

But what is the probability that two distinct polynomials of degree n minus 1 are equal? It's just-- this happens with probability at most n minus 1, the degree divided by the field size.

Two polynomials agree. Two distinct polynomials agree, and at most, or the fraction. Now they will, at most, n minus 1 points, and which is n minus 1 over F fraction of the field. So I'll reject this probability.

And if we're not happy with this, F is not big enough for us, then we do everything over a bigger extension field. So our rejection probability will be high If the claim is false. Any questions about this? So far, what we saw is that red--

STUDENT: Ma'am?

YAEL KALAI: Oh, yeah. Oh, yes. Please. Thank you. Thank you.

STUDENT: Is that the Schwartz-Zippel lemma we're doing?

YAEL KALAI: This is not a Schwartz-Zippel lemma. Schwartz-Zippel lemma is similar, but in a multivariate setting. So this-- good. So let me-- that's a good question. So thank you.

So I just want to say-- oh, by the way, I should tell you guys this is being recorded. I hope it's OK with you. You're not being recorded. It's supposed to-- I'm supposed to be the one being recorded. But I think you should know that there's a cameraman behind you.

But actually, I don't think I'm going to put this-- I mean, it's in the website for us. Well, now the website is public, so it will be public.

[LAUGHING]

I can make it private if you prefer. If people come to me and ask for it to be private, I'll make it private. So good.

So the question was, was this Schwartz-Zippel lemma? And that's exactly-- Schwartz-Zippel Lemma is a multivariate version of this. This is for the single variate case. So the fact that two univariate polynomials of degree d distinct are-- they're equal and a random value-- they're equal and at most d element, d being the degree, is kind of-- I don't remember the theorem, basic something of math--

[INTERPOSING VOICES]

YAEL KALAI: You said Lagrangian interpolation? Yeah. But Schwartz-Zippel is exactly that, but in the multivariate setting.

STUDENT: So it's a generalization of that?

YAEL KALAI: Exactly. Exactly. The Schwartz-Zippel is exactly a generalization of this theorem. And it will come up, actually, later in this class. So yeah, we'll talk about that. Any other questions? Yeah?

STUDENT: Are there any examples where having randomness but not interaction gives you more than a polynomial time speed up or something.

YAEL KALAI: Good. Good. Good. Very good question.

So the question is, do we have examples where randomness gives you more speed up than just kind of pathetic? This is not very interesting. Or in other words, let me rephrase your question, is the class MA different than NP? That's the question. We don't know.

STUDENT: And so MA said that the verifier has to be polynomial?

YAEL KALAI: Yeah. Yeah. So yeah. So let me explain.

There's a kind of-- there's this class called as MA, MA, MA, as many as you want. And the idea is M means Merlin speaks. Merlin is the prover. He's the wizard.

"A" means Arthur, the King, he's the verifier, answers. M speaks, A answers. M speaks, A answers.

So there's these classes, MA, MA, MA, MA. Here, MA means M, the Merlin, speaks. Arthur kind of uses randomness and then kind of sends randomness.

Well, he doesn't speak-- Merlin doesn't speak again. But Merlin speaks. Arthur chooses coins. And then there's a verdict function that decides if to accept or reject. And we don't know if MA is different than NP. Yes?

STUDENT: So this actually is a-- the polynomial identity testing, right?

YAEL KALAI: Right. You're saying this is polynomial identity.

STUDENT: Yeah.

YAEL KALAI: Yes.

STUDENT: So I give you a circuit. It's supposedly not a degree polynomial. I claim that it's identically zero circuits. You can check it clearly with this multivariate [INAUDIBLE]?

YAEL KALAI: Right, right, right.

STUDENT: So you don't want to [INAUDIBLE].

YAEL KALAI: Right. But-- oh, you're saying-- sorry. You're saying to test whether a circuit is--

STUDENT: Identically zero, let's say.

YAEL KALAI: But wait. Oh, you're giving us just as a circuit.

STUDENT: Circuit, yeah.

YAEL KALAI: As a circuit, testing if it's identically zero. But it needs to be low degree, because otherwise--

STUDENT: So somehow, like, it's syntactically low degree, right. Low depth or something, right? Low depth arithmetic circuit.

YAEL KALAI: Low depth, like log, log depth arithmetic circuit. Then you're saying it has polynomial degree.

STUDENT: [INAUDIBLE].

YAEL KALAI: It has polynomial degree. You can test it by taking a large extension field. You can test. You don't know how to predict if it's, yes, good.

STUDENT: If you don't know how to derandomize it.

YAEL KALAI: How to randomize.

STUDENT: If we do randomize it, then there are circuit lower bonds, which means that you can randomize everything probably, right? More or less.

STUDENT: There's like-- yeah, you get some circuit lower bounds. But it's like--

YAEL KALAI: Yeah. Yeah.

STUDENT: Although, that's a problem in [INAUDIBLE], promising [INAUDIBLE]

YAEL KALAI: No, why? Why is it not an MA? So you look at the class--

STUDENT: It is an MA.

STUDENT: I'm sorry.

STUDENT: But it's [INAUDIBLE].

YAEL KALAI: Ah. So, Right. OK. Yeah, yeah, yeah, yeah, yeah. You're right. It's in BPP. Yeah.

STUDENT: So again, I just wanted to make the point that what Yael described is essentially the only-- the canonical problem in MA [INAUDIBLE].

STUDENT: OK.

YAEL KALAI: Yeah. Yeah.

STUDENT: Up to details, which I think that they're going to depart.

YAEL KALAI: Thanks. Thanks, Vinod. Yeah.

OK. So now let's go ahead and just define this class of NP-- of IP, sorry, interactive proofs. And then we'll show how an actual-- and we'll actually show why we believe this class is more powerful than NP will give a protocol that solves a problem that seems pretty hard.

So let's see. So here is the-- so what is an interactive proof? An interactive proof, IP. This is a protocol between a possibly all-powerful-- I'm not a prover, P.

So P can be run as long as you want. And A polytime verifier, V, such that-- here's the requirement from the protocol. We want to say that A-- so interactive protocol, let's say, for-- sorry, between a prover, P for a language, L.

For some language, or some problem, L. So the problem corresponding to this membership in L, that's the problem. Is x a member or not a member?

So an interactive proof should satisfy the following. So it's a protocol such that it should have two properties, completeness and soundness. Completeness says that if x is in the language, the prover should convince you to accept.

So for every x in the language, if you look-- if you denote, let's say, by t, the transcript, so let's say P and V, which has secret randomness. The prover doesn't know what it is ahead of time. And input x, let's run this protocol. We get a transcript.

So it's an interactive protocol. It actually-- you can think of it, the protocol consists in two phases. First, there's interaction between the-- they talk back and forth. Let's denote the transcript by T.

I often denote like PV. That's kind of the protocol. The prover has r. Both of them know the input x. They generate a transcript.

And at the end, the probability that V, with his input-- he gets the transcript, the randomness in x, the probability that he accepts-- outputs 1. When we say "outputs 1," we usually mean he accepts, is large.

Let me call it least C. And C is going to be a completeness parameter. So protocol between P and V for language L with completeness, C, and soundness, S.

STUDENT: Yael, do you mean L, x, and L?

YAEL KALAI: Yes, I do mean L, x, and L. Oh, thank you. Thanks, Sally.

So if it's in the language, he's accepted with high probability. There's some completeness parameter, C. And most of what we see in class today, this is 1. You accept with probability 1. And [INAUDIBLE] says this.

[INAUDIBLE] says that for any x not in the language-- for every x that's not in the language, no matter how you try to cheat, you'll fail. So for any cheating prover, or for any P star, because it can be any. Any prover that tries to convince me to accept, if the probability for the same t is above, the probability that the verifier, given the transcript, the randomness, and x, outputs 1 accepts is at most S.

So that's-- it's in the language, it should be accepted with high probability C is high. If it's not in the language, you should accept with small probability S is small. OK? Question about the model? Yeah.

STUDENT: Is there any restriction on S and C, or are they just--

YAEL KALAI: Great question. What are S and C? Are they constant? So that was my next remark, which means that the question was right in place.

So often, if you look at the literature, there won't be a mention of S and C. And C will be taken to be 2/3 and S will be taken to be 1/3. That's kind of a canonical-- so when people say there's an-- here's an interactive proof, often, you think of S as 1/3 and C as 2/3.

And this seems very arbitrary. And you should be alarmed. So where do these constants come from?

STUDENT: I don't know. I think it's possible to just repeat the procedure to--

YAEL KALAI: Exactly.

STUDENT: [INAUDIBLE] paper defined [INAUDIBLE]

YAEL KALAI: Exactly. So the reason why we don't care so much about these constants is because if you have an interactive proof with a gap between C and S, so for example, 1/3 and 2/3, what we're going to do is repeat the protocol. So let's say that it's 1/3 and 2/3. Let's repeat the protocol, let's say, n times.

And then we're going to accept if you're closer to C and reject if you're closer to S. So in other words, if more than C plus S over 2, like right in the middle, if more than C plus S over 2 are accepted, you're going to accept. If less, you're going to reject.

In this case of 2/3 and a third, let's say, more than half repeated many, many, many times, you should accept if it's in the language, you should expect to see about 2/3 acceptance. If it's not in the language, you should expect to see 1/3 of them being accepted.

And if you repeat enough times, by the law of large numbers, you're going to get closer and closer to the expectation. So you're going to be either very close to the 2/3 if you're in the language or very close to the 1/3 if you're outside the language. And there's very strong concentration.

So if you repeat it, let's say n times, then now you're going to be accepted-- or the soundness and completeness gap will be something like 2 to the minus n versus 1 minus 2 to the minus n. So you can really very fast make these things go to 0 and 1, which is why often we don't really care about these.

And you can think of them as very close to zero and very close to 1. So if it's in the language, acceptable probability almost 1. If it's not in the language, you reject. You accept probability only 0. OK? Great.

Any questions about what an interactive proof is? OK. I just want to mention one thing. This notation, like PV and input x, is overloaded.

We use this notation to denote the transcript. We also use this notation to denote the verdict function, whether we accept or not. And also I'll use that in class, and also wherever you read, you'll see both of them used.

So it's good to get used to. So this sometimes denotes the entire transcript. Sometimes, it just denotes the acceptance or rejection bit of the verifier. And it'll be clear in the context.

So now the question you should ask is, OK, is this model more expressive than NP? What can we do with this model? What's the power of this model? So let me actually go to the punchline and say there's a celebrated result by Shamir from '92, I believe.

And he showed-- look. Oh, this model is very, very powerful. And actually, you can prove any P space computation in this model. Namely, if I want to prove to you any language in P space, take any language in P space, I can prove membership in this language to a verifier that only runs in polynomial time.

So that's Shamir-- so P space, we believe, is much, much more expressive, stronger than NP. We actually don't know how to prove it, by the way. But we believe that's the case.

So we believe this is a much stronger kind of proof system. Actually, before-- I'm going to show you-- actually now, I'm going to give you an example of the power of this proof system. But before I go into the technicality, I want to step back for a second and think a little bit about the story here, which is a very nice story.

So this introduction of interactive proofs, this model came about for the goal of doing zero-knowledge. Nobody thought at that point about whether you can prove more languages, not more languages. The only thing you want is to hide.

I just want to hide why a theorem is true. I want to hide the proof. I want to convince you, but hide. That was the only goal that kind of introduced this model.

And it's kind of interesting how very fact. Like, also, who cares about zero knowledge? This is a really interesting model. Let's see what we can do with it.

And this kind of opened our eyes, just the idea that we can shift from the traditional way of thinking about proofs to this interactive and randomized land, and brought about a lot of really, really beautiful and fascinating results. Including the fact, this is the first time that realized, wow, we can prove a lot of-- to a polynomial time verifier, much more than what was believed to be-- what we believed we can do with a classical proof.

So this was kind of really, I think, eye opening and very interesting. So now I'm going to convince you-- try to convince you of why this proof system, this model is so powerful by giving you an example of things, of a problem that I can prove to you using an interactive proof that we really don't know how to prove using a classical proof.

So let me give you the example. So here is the example. So the protocol is called sum-check protocol. And I'm going to give you an interactive-- this is an interactive proof for the following problem.

So take any degree fix, or-- OK, given a polynomial, f, let's say, from f to the m, so a multivariate polynomial, of degree d in each variable. I want a proof that sum of f b1-- let me just make sure how I wrote it down.

OK, sorry. So take any-- fix any subset H in F. You can think of H as 0, 1, you can think of a bigger set. I want you to convince me that sum f of h1 up to hm for all h1 up to hm in H is equal to some value beta.

So there is a fixed polynomial f. Let's say I know this polynomial. And either it actually has a succinct circuit, or maybe I have oracle access to it. So I'm a verifier.

I'm a verifier. Let's say I have oracle access to this polynomial, or there is some succinct way to compute it. So I can compute this polynomial easily. Let's think for now I have oracle access to it. So I can query it in some point, and after that, get back an answer.

I want to prove you-- you're going to prove to me that the sum of this polynomial and h to the m points. On all the points, there's some set, h. You can think of h as 0, 1.

You can think of it bigger. And I want you to sum this polynomial on all the elements in the set h to the m. All of this is done over some finite field, F.

And there's a result, beta. I want to prove that this sum is beta. I'm going to show you an interactive proof for this language. Now, before you guys are like, seriously? Why do I care about this language? Is it interesting? Where did I come up with this thing?

Let me tell you, I think this is the most important protocol in all the literature on proof systems. We all use this protocol all the time. It turns out to be super, super influential. I mean, every single thing we'll use from now on relies on this protocol.

So it turns out, and I'm going to convince you that later, but that this is a very, very important, actually, problem. If you have interactive proof for this problem, it kind of gives you rise to interactive proof for many, many, many other problems.

Actually, we use this. If you give me an interactive proof for this problem, I can use it to get an interactive proof for all 3D space. In kind of a black box way. So really, this is the heart of coming up with interactive proofs. This is the most important interactive proof.

So with that, let me show you the interactive proof for proving this. Any questions before we jump in? Yeah?

STUDENT: So just to clarify, h is like a subset of F, like one dimensional. So it's not like--

YAEL KALAI: Exactly. You can think of H as 0, 1, for example. You can think of H as 0, 1, 2, I mean, if 2 is in the field. Yeah, it's a one dimensional thing. And you sum over-- you sum here over h to the m, size of h to the m elements.

STUDENT: So any of the two h's can be equal, or it doesn't have to be distinct from F?

STUDENT: Wouldn't it still [INAUDIBLE]?

YAEL KALAI: It can be. You sum over all of them. You sum over also like 0, 0, 0, 0, 0, 1. Yeah. Yeah, h1 and h2 do not need to be distinct at all. Yeah, exactly. Over all possible-- yeah. You sum over h to the m elements, each one over all the possibilities.

Good? Other questions? Thank you. Yes?

STUDENT: So degree d in each variable just means like any variable is then just exponential?

YAEL KALAI: Exactly. So yeah, let me-- good. Great. So degree D in each variable means-- f is a polynomial. So you can write it as a sum of monomials.

Each monomial-- each variable has a degree. So I was saying, if you look at all the monomials, each variable has a degree at most D in each of the monomials. Great questions. Thank you. Any other questions?

Great. So let me give you the protocol. So let's do it.

By the way, I want to tell you, I get such satisfaction from your questions. So if you just want to make me happy, which is a great goal on its own, you should ask questions. OK. Ready for the protocol?

OK. Prover convinces a verifier that the sum f on h1 up to hn is equal to beta, where the h's are all elements in the set H, which is in the field. We have some fixed field that we're working over.

How would he prove-- what would he do? So here is the idea. The idea is the prover will first send this sum, this polynomial. But where the first variable is open.

So he will compute-- Let me get those. He will compute sum only over h2 up to hm, f of x, this is a variable, h2 up to hm. And he will send this over. Let's call this g1 of x.

So he sends a univariate polynomial where he sums over h2 to hm, but leaves the first variable open. What is the degree of this polynomial? d. Very good.

So this is a degree d polynomial. The verifier checks that g1 is of degree at most d. If not, he rejects, because he knows that this is a degree, d polynomial multivariate polynomial. So he knows that each variable is of degree at most d.

So if the same thing it's gotta be, it's like you're cheating him. So he's going to immediately reject you. He will also check that sum g1 one of h, I guess, 1, let me call it h1.

That if you take g1 and you sum it over all possible h's in H, what does he expect to get? What would he check for?

STUDENT: Beta.

YAEL KALAI: Exactly, beta. Because the claim is beta. So he checks that this is a degree at most d, that when you sum over h's, you get what was claimed.

If this does not hold, then you reject him. You say, Bob, I reject you. I'm not going to continue.

If this holds, then he chooses a random t1 in the field. This is often a notation, left arrow, that we use for choosing at random. So he uses a-- he chooses at random a random element in the field. Not in h but in the big field.

You need to think of this field as being big. Well, the bigger the field, the better the soundness will get. So it chooses a random t1 in the field. Good. Then what happens?

The prover-- in some sense, this reduces the problem from checking that this sum is beta to checking sum-- let me write it here. To checking that sum of ft. t is fixed. Or t1, we called it. And then h2 up to hm is this fixed, I don't know, g1 of t1.

We kind of reduced the problem to this problem, which is of one dimension less because it's-- the sum is only over m minus 1 variables. Yes?

STUDENT: When we see the verifiers polynomial time here. What are we polynomial in?

YAEL KALAI: Good. What are we going to polynomial in? Very good question.

When I said the verifiers polynomial time, polynomial in what? So he's going to be polynomial-- we think of f. I said F can be-- has oracle access to it. So polynomial and what?

So we're going to analyze his runtime in a bit. But his runtime will depend polynomial on n, which is the number of variables, and on the degree. And we think of a-- today, and actually-- throughout the course, and it's common in this field, we think of f, that he can add, and multiply things in order one time. It's a constant.

And typically these fields are going to take polylog. But we don't want to chase after these polylog terms. So we're not going to consider them, really. Sometimes we do, sometimes we don't. Depends. But beyond-- so if you want to be very pedantic, then you should log f here.

STUDENT: Also h, maybe.

YAEL KALAI: Yes, thank you. Yes, of course. Thanks, Surya. The size of h and log F. That's what's going to happen.

But you write that here, the polynomial time doesn't really apply here. Because you say-- I said that he has oracle access. So what polynomial over?

So this is what he's going to be after. When we're going to use him for applications, you'll see it will correspond to polynomial time. But we're going to go over this when we analyze the protocol.

So let's just continue. So let me actually start it. Let me even say-- before I continue the protocol, let's continue the protocol with a high level idea of what we're trying to do here. The idea is the following.

We want to catch a prover cheating. That's the goal of a prover. We want to catch you if you're cheating. So suppose a prover is cheating here. How will we catch it?

The way we're going to catch it is we're going to tell them, you know what? Give me this g1. Now he cannot give me the actual true g1, because the actual true g1 will not sum up to beta. Beta is the false thing.

So if a prover cheats-- I'm trying to-- let's come up with this protocol together with the goal of trying to catch the prover. So how do we catch a prover cheating?

So the idea is we'll tell them, you know what? Give me this sum but with this open. I'm going to check that this something-- that the sum of g1 is equal to beta and then its low degree. If it passes, then this g1 cannot be the true, true g1. It can't actually be the real sum of f, because the real sum of f is not beta. He's cheating. So this must be a cheat.

Now when we send him t1 and we consider-- so now we send him t1. And we reduce the claim to claiming this. But with very high probability, g1 is also a cheat, because g1/t1 is a cheat. So g1/t1 is not really the sum, because they're different.

g1 is different than that. So we know that if they're different, they agree only on d elements. Two distinct polynomials in d elements.

What's the probability that this one is one of the t? Very small, d over all f. OK, so I think of f as really large. Great. So this is also false, actually.

It gave me g1, but it's actually not the real f. He's going to claim it's the real f, but I know it's not. Now I'm going to tell him-- next thing I'm going to tell him, OK, give me g2.

g2 of x is sum-- it should be sum f t1. I'm going to only sum over h3 up to hm. The x2, the 2 is going to be open, and then h3 up to hm.

So we started by getting rid of summing over h1. We gave him the univariate polynomial, or we chose a random t1. Now we're getting rid of summing over the second coordinate.

We're asking for a univariate polynomial g2. What do we check? So the verifier checks that g2 is of degree at most d. And what else does he check?

STUDENT: The [INAUDIBLE] of g1.

YAEL KALAI: Very good. So sum g2 and h-- maybe I'll write h2-- is equal to what?

STUDENT: g1-- g1 of t1.

YAEL KALAI: Exactly. g1 of t1. So essentially, we checked this polynomial exactly as we did before. So now, note, if, in the Alice case, of course that's the case. g2, of course, is degree at most d, because f is degree at most d.

And when you sum over all of h2, you should get exactly g1 and t. That's how we-- great. If this does not hold, he immediately aborts and say, I reject you. If he's happy, he's going to send a t2 at random from the field.

What am I going to reply with? What should the prover reply with? So let's see. g3 of x is what?

STUDENT: Sum over h4 hl plus t1 t2 and then x.

YAEL KALAI: Very good. Very good. f of t1 t2 x and h4 up to hm.

Yeah. Again, we take, essentially-- what happened here? We reduced-- after t2, we reduced the problem to proving a sum-check over sum f1 t1 t2 h3 up to hm. And we need to check whether this is equal to g2 of h2. Of t2, sorry.

So again, each time we kind of layer-- get rid of one variable in the sum check. We start with an n variate sum check, and each round, we get rid of one variable.

OK, last one. What do we check here? We got G3. What do we check? What is--

STUDENT: g3, h3 [INAUDIBLE].

YAEL KALAI: Exactly. So first, that g3 is low degree-- is degree at most d, and sum g3 is g2 t2. And we continue in this way until-- we continue, continue, continue until we got rid of everything.

So in the last around-- what will happen in the last round? In the last round, we will send gm minus 1 gm of gmx, which is f of t1 up to tm minus 1x. That's what he should check.

That's what he should send. If you're honest, that's what you send. You peel off everything. Finally, you're left with gm, which is f, and t minus 1x.

And then the verifier is done. He'll choose a random tm and he will check whether this is indeed f of t1 up to tm. We assumed he has oracle access so we can check if this value is t1 up to tm.

So finally, this is this, and verifier-- I'm out of room, but maybe I'll write it here. V chooses a random tm and checks that gm of tm is equal to the oracle. He has f on t1 up to tf. That's the protocol. I'll give you a minute to appreciate the beauty while I erase the board.

STUDENT: Do you have a different set for each variable than just--

YAEL KALAI: You're asking, can I have a different set?

STUDENT: Yeah.

YAEL KALAI: I can have a different set, yes. Yeah. So the question was, do the h's have to be identical? And the answer is they don't have to be identical. You can take a-- we didn't use that anywhere. And yeah.

Yeah. Somehow, we always use it when it is. But that's a very good point, because it's like-- it seems more general. It seems like it can be useful, but somehow we don't use it.

But that's a very good point. Yeah, you can-- the H does not have to be the same h. It can be the sum over h1 up to hm.

STUDENT: So in particular, h could be equal to f, right?

YAEL KALAI: OK. h could be equal to f. Yes, h could be equal to f.

STUDENT: How do you do the summation over-- in g1 up to h1?

STUDENT: Sorry?

STUDENT: The first line, the summation over g1 of h1.

STUDENT: You will take time proportional to that.

YAEL KALAI: Yeah.

STUDENT: But let's say that's OK.

STUDENT: I think your soundness goes away also.

STUDENT: Say again?

STUDENT: Your soundness goes away?

YAEL KALAI: Yeah. We'll talk about soundness in a minute. Actually, if f is degree d, the soundness won't decay. I think-- actually, A-- let's talk about soundness and then we'll see.

Actually, the soundness won't-- in general, usually when h is big, in applications, what happens often is when you take big h, the f turns out to be degree h, or h minus 1. So when we use this protocol-- so as I said, this problem.

You say, what do I care about this problem? Who cares about this problem? So it turns out we care a lot. It comes up all the time.

The way we use it in-- like, the way it comes up is the f that we use usually is of degree h minus 1 in each variable. We use it in a way that turns out to be the degree of f. And the soundness decays with a degree.

So if the h is big, the soundness-- if the degree is big, the soundness is big. But actually, if you're promised that the degree is small, then we're good. We'll analyze it right now. Yeah?

STUDENT: Can you say something about what's so special about polynomials here? I see that they only agree on the coordinates, and you can do the summation thing. But is there something more general happening?

YAEL KALAI: Yes. Yes. Great question. Love the question.

So what's so fascinating about polynomials that comes up here? So why is this so interesting? Why is polynomials so important that this comes up everywhere?

So here's my take on this question. Everything can be-- OK. First of all, polynomials are tough. They're very, very useful because of their power of error correction.

So in other words, what happens, if you cheated once, we're going to amplify your cheat. That's kind of what we do here. You cheating once, we're going to-- for a random point, t, if you're low degree, then-- we're assuming that this prover gets kind of low degree, low degree, low degree all the time.

If you give low degree, low degree, low degree and you cheated, then the cheat must correspond to a cheat at a random point. Like at the end, a random point t1 to tm, you should be different. And we'll analyze this. This is just intuition.

But that's kind of the beauty of polynomials, that if you're wrong, you're wrong almost everywhere. Because if it's not the correct polynomial, then it's different from the correct polynomial almost everywhere. That's the Schwartz-Zippel lemma for the multivariate case. For the single variate case, it's just a Lagrange interpolation, I guess it's called.

But OK. Now you're saying, OK, but that's just polynomial. Why am I saying it-- what's so special about polynomials? Why are they so general?

OK. If you happen to have a polynomial, good for you. You won, because it has such error correcting guarantees, you're lucky. It's your lucky day.

But what if it's like-- why is every day a lucky day kind of thing? Why does this polynomial come up everywhere?

And the answer is that you can actually embed many, many problems into polynomials. So there's a way-- there's a technique, which we'll call low degree extension. We'll see that pretty soon. If not today, then tomorrow-- then next week, where you can convert many problems into polynomials and then use the sum-check.

So that's often what we do. We take a problem, we embed it into a polynomial. We kind extend it to look like a polynomial of low degree. And then we use sum-check to it. So that's how we will come up.

Any more questions before we analyze the subject protocol? Questions on the protocol itself, or? Yeah?

STUDENT: So sorry, just to follow up. So is there some version of sum-check you can run on some generic error correcting codes, or do you need not only the error correcting properties of polynomials, but also their average structures.

YAEL KALAI: Oh. You're asking-- what you're asking, you're saying this sum-check, do I really need this specific kind of polynomial code or can I do it using arbitrary codes kind of thing. So actually, this is a great, great question.

There is a paper by [INAUDIBLE] from several years ago, I don't remember exactly, where he shows how to do a sum-check over more general codes. Not any kind of code, but more general than that. Things like tensor codes, or-- Rachel, do you remember? You looked into it.

STUDENT: Yeah, he did tensor codes.

YAEL KALAI: Tensor codes.

STUDENT: Oh, he did multiplication property [INAUDIBLE].

YAEL KALAI: Yeah. So he generalized it, not to any code, but he gave a generalization of the sum-check protocol to work for a larger family of codes.

STUDENT: But when we're trying to parse the question, what does it mean for the subject protocol to work for a more general approach?

YAEL KALAI: OK. Good, good, good. So you can think of it as I'm proving something about-- I don't remember exactly how his formalization was, but it's like I'm proving something about a code word, and I want kind of--

STUDENT: Sort of reinterpret this.

YAEL KALAI: You can reinterpret this, because you can think of this f as being an error correcting code. So does it work only for this kind of Reed-Muller? This is called the Reed-Muller. Does it work only for Reed-Muller? Can you do it for arbitrary codes?

And it proves it for arbitrary codes. So this is kind of-- any polynomial is an error-correcting code. It's a type of error-correcting code. For a multivariate case, it's called Reed-Muller. For univariate case, it's called Reed-Solomon.

And so the sum-check protocol for-- you can think of it as a protocol for Reed-Muller codes. And then you can ask, is there a analog for other codes? And actually, there are analogs out there. Yeah. Any other questions? Yeah?

STUDENT: Yeah. [INAUDIBLE] like making the size page the same as the side of--

YAEL KALAI: F.

STUDENT: If I hear correctly, in that case, wouldn't there be no point of doing this, because it's only of size of h and log f.

YAEL KALAI: OK, you're right. You're right. So if-- let me go back. So the question is, what should this h be? What is this h?

And the answer is this h can actually be all of f. Of course, if it's all of f, the verifier runs in time, which is like the size of f, because he checks sum over h. If it's all of f, he runs them all of f.

But maybe f it's not that big. Maybe f is polynomial. Even if f is polynomial size, still, checking this sum is much bigger than f. It's like h to the n, like f to the n, and it can be big.

So the hardness does not come only from f, it comes from m. It's exponential in m. It's the m that kind of makes it very hard.

And so h can be as large as the entire field. And you can run some check over that, too. It's still an interesting protocol. Yeah?

STUDENT: But if we do that, wouldn't we need to do this as an extension of f or something to get the--

YAEL KALAI: Good. OK. Good. So the question is, wait. Do we get any error-correcting guarantees here, because-- if we're doing over all of f. And I think the answer is yes, we do need to extend it further.

Because as we'll see now when we analyze it, the soundness error goes down with something like d over f. h does not come into play there, but we'll see that. Let's see that together.

You're right. I agree when you say that, yeah, that should, right? Things should degrade with age. But actually, they don't. So we'll see that now when we analyze it.

These are great, great questions. So thank you. Any other questions before we go to the analysis?

OK. So actually, before we do soundness, let's just first do a sanity check on the communication complexity, and the runtime, and so on.

So first, let's see this. Protocol has m rounds. Each round, we get rid of one sum in the sum-check.

So the-- so the communication complexity, our round complexity is m rounds. m back and forth. m, 2m, I don't know if you want to call each one a round is 1, 2. So let me just say order m.

The communication complexity is what? So each time one polynomial is sent and one field element. So it's really-- a polynomial is a degree, d. So it's just d field elements each time times m.

So fine. We can do times log f, if you want to. So that's the communication complexity. What about the verifier runtime?

So the verifier, he gets a polynomial-- degree d polynomial. So d essentially field elements, of d plus 1 field elements. And he just sums them up. He checks that it's a degree d and sums them up.

That's all he does. So it's really essentially the same. For m rounds, he has a degree d polynomial. And he sums them-- I should put here maybe probably log them, because the sum-- I'm assuming addition will take polylog f.

Again, as I said, often, I think maybe even in the notes, just in Taylor's notes, this is admitted. It's like, the field thinks of it as one operation. So just so you know. If you read things, often you won't see this part. Question?

STUDENT: The verifier of them, like, h, that's [INAUDIBLE]

YAEL KALAI: The verifier also needs to choose the t you said?

STUDENT: No. Like, you'll see the sum over h.

YAEL KALAI: Yeah, exactly. So sum over h is like h additions.

STUDENT: Yeah.

YAEL KALAI: OK. Yeah. Oh, sorry. Yeah. So OK. Good.

You're saying he needs to compute-- it's still-- yeah. So the verifier, he gets a polynomial. He needs to evaluate the polynomial in h.

So yeah. So for m rounds, he gets a degree d polynomial. He needs to evaluate it. Oh. Sorry, this does not have h.

Yeah. He needs to evaluate it in H points, the degree d polynomial, and then a log F. Thanks. Good.

Thanks a lot. Good catch. That's awesome. OK, great.

So that's the verifier. As I said, the prover, we care less about. But just-- if you want to just-- and by the way, this is assuming the verifier has access to f.

He knows f. f is kind of-- he has oracle access to it. Because, of course, then he needs to check equality with f. So this, we just assume he has oracle access to f.

And then the prover, what does he need to do? Well, he needs to do this computation. So he runs in time like h to the f. He needs to do all the sum. He needs to run-- so the polynomial. So let's call it T sub f is the time to run-- to compute f.

So he needs to compute f. He needs to do the sum over all f's and he does that kind of m times, right? Something like this. M polylog H-- polylog F, sorry.

Yeah. So that's the prover's time. OK. Yes?

STUDENT: Doesn't the fact that it required us [INAUDIBLE], as [INAUDIBLE] of using a prover?

YAEL KALAI: No. Good, good, good, good. Great question. No.

OK. So even if V has random oracle access to F, he can compute this entire thing, but it will take him time, H to the m, because he needs to compute the sum. So the goal-- he needs to check-- so let's say computing F once, it takes him long time. He can ask.

But to check the sum, he needs to ask this H to the m question. Now think of H to the m as huge. Even if h is 2, think of m as being big. m is n variables. So it's like exponential. That's it. Great question.

STUDENT: But should I think of the description of f as polynomial, or?

YAEL KALAI: Good. So you can-- there's two options. You can think of the description of f as being polynomial. That's one option. Or that you can compute f. And it's like described by a polynomial size circuit.

Or you can think it can be large, but the verifier has access to it. Somehow, the verifier has a trusted oracle that computes F for him. OK? Great. So now the-- yes?

STUDENT: The break from t sub f.

YAEL KALAI: Good. Yeah. This is just-- sorry. This is just the time it takes to compute F. Thanks.

Yeah. Because he needs to compute. The prover supposedly doesn't have-- he needs to actually do the computation. OK, great. Yes?

STUDENT: Where in that-- I do not see take into account the prover finding g sub 1 of x? Don't they have to combine a bunch of terms in that order of d?

YAEL KALAI: Oh, yeah. But he runs in time F anyway. F is much bigger than d. So the degree is also always much smaller than f. So I'm kind of lumping it in. Yeah.

Thanks. Great question. Yeah?

STUDENT: So if I want to really foster [INAUDIBLE] like a multiplicative subgroup or something for h so I can evaluate a polynomial faster, can I do something else to it?

YAEL KALAI: Again, you're saying if-- you want to be-- you're saying, let's say the verifier wants to be even more efficient than this. Then what can he do?

STUDENT: Can I choose H to be a special structure?

YAEL KALAI: OK. But you're changing the problem. What you're saying is-- so the problem is H is given to us and you want to prove. Now you're saying maybe if H has a special structure, maybe we can be even more efficient is what you're saying. A good question. I haven't thought about it.

STUDENT: I think [INAUDIBLE]

YAEL KALAI: Yeah, you can probably-- yeah. Yeah. Probably you can-- so Vinod was saying you can probably batch things, or probably-- yeah. I think if you can choose H-- if H is of a certain structure, then I think you can do better.

But this is kind of-- the verifier doesn't choose it. That's what's given to him. Great. OK, great. Any more questions? Yeah?

STUDENT: So why would an oracle [INAUDIBLE]? Is it the size of all these [INAUDIBLE]?

YAEL KALAI: Oh, OK. It's degree d, but it's m variables. So in general, degree d polynomial with m variables can have size like m to the d, which can be very big. Great question. d to the m, sorry. Great question.

OK. So let's go to completeness and soundness then. So what is the completeness of this protocol? Let's say the prover is on it, the statement is correct. What probability will the verifier accept?

STUDENT: 1.

YAEL KALAI: 1, exactly. Because you'll just be honest. You'll always give the correct polynomial, always the low degree, will always sum to what it should be. And you're always going to accept it. That's easy.

The hard part is soundness. So let's do the soundness. And then, after soundness, we'll take a break.

So for soundness, here's the idea. So let's look at a-- suppose there's a cheating prover that generates a false claim. So suppose that the claim is false. Namely this sum of f is actually not great.

OK, so we start with the false claim. The first thing we know is that-- so fix any-- so you fix a claim that's false. Fix any P star.

He tries to cheat. Let's show that he fails. He won't succeed. Or he succeeds with a very small probability, if the field is large enough. So let's see.

The first thing I want to point out, which I mentioned before, is g1 must be false. So it must be the case, no matter how we cheat. And it gives me a bad g1.

It cannot-- if he's accepted, if the prover is accepted, then g1 cannot be sum of-- let me call g1 x is not some of f h2 up to hm, because I'm going to check that it's equal to beta. But the sum of f is not beta.

So the g that he gives me cannot be the correct f, like the correct g. So in other words, he starts the protocol by a cheat. I mean, the first message is not the correct polynomial.

Because the correct polynomial, if you sum it, you'll get the real beta, not the fake beta that he claimed. However, at the end of the day, if you're accepted, the final check, when you check-- then you check with respect to the honest f and that verifies.

So at the end of the day, when you look at it with g to the-- like last one, gm to the tm, that's actually honest. So somewhere in the protocol, you had to move from a bad g to good g. So again, here's the claim.

So I want to argue if-- so I want to argue there must exist a round such that-- a round, let's say i. So to prove-- to cheat, there must exist a round i such that gi is false. So gi is not sum of t1, ti minus xhi plus 1 up to hm.

This is what it should be, but it's not. And yet at some point-- but gi plus 1 is actually equal to-- oh, I forgot the f. Is actually equal to sum f t1 up to ti.

So to cheat, you must go-- so you start with-- you start with an incorrect polynomial. We know that's for sure, because you-- assuming you're accepted. If you're accepted, g1 is false.

Now either now g2 is true, and then I'll show that happens with very small probability, or g2 is also false. If g2 is false, either g3 is true, I'm going to show that happens with very small probability, or g3 is false.

Until we're left with GM, which, again, let's say it's still-- I'm going to argue that if gm minus 1 was false, gm is also false with very high probability. But if gm is false, you're not going to be-- if gm is actually not f, then for random t, the probability that you'll get f is very, very-- you'll be consistent with f is very, very small, because these are two different polynomials.

And they agree only in d points. So what's the probability that he chose one of these d's, one of these points, d over f? So let me be a little more formal now.

So I want to argue that-- first, let argue that-- so first, to cheat, there must exist some round where we go from false to true. Because the final, final check must be true. Otherwise, I'm going to check you with a true.

So there must exist a round that you go from false to true. Let's say, what's the probability of going from false to true? Good. So what's the probability?

So let's denote this event-- let's denote this event by Bi. Now let's see what's the probability that this event happened. This is kind of a bad event, an event that you go from an incorrect-- a false to true.

For the verifier, it's a very bad event, because then he accepts a false statement. What is the probability that this bad event happens? So the probability of Bi, I want to argue, is at most. d over F. Let's see why.

So when is it-- let's see. When is gi plus 1 true? We're sending here-- the probability, let's fix-- we fixed our t1 up to ti plus 1. That's fixed. So fixed.

And I don't care how. t1 up to t minus 1, this defines. what the prover gives gi in the protocol. The cheating prover gives in the protocol.

Now the verifier chooses a random t, a random-- good. So this defines gi, which is-- OK. This is my gi. Now the prover, the verifier, the probability over ti that the verifier chooses, this ti tells you if gi-- this defines gi plus. 1.

But what is-- what do we know about when-- if we know that, because the verifier is accepted, we know that sum gi plus 1 of h is equal to gi, which is the wrong polynomial. Sorry, and ti.

And this is the wrong polynomial. So what is the probability that for-- so in other words, gi plus 1 is the wrong polynomial. What is the probability that for a random t, it will be the right one? d over f.

Because two polynomials that are different agree on, at most, d points. I choose a random point. The probability that it's one of d's is the d over f. So what is the probability that you cheat?

This is the probability that there exists some i such that Bi holds. A bad event holds. Namely, there exists an i that you go from true-- from false to true on that i.

By the union bound, this is, at most, m number of rounds times the probability of one-- of a specific Bi bad event, which is d over F. So just note h actually doesn't come up in the soundness. This is the soundness.

So the soundness needs m times d to be significantly smaller than the field size. That's all that's important for soundness. Yeah?

STUDENT: So is the point that you can pick a dh that you actually care about. And then you have some potential with F and then this will be a really big field for [INAUDIBLE].

YAEL KALAI: Exactly. Exactly. The point is, if you're not happy, if F is not large enough, if it's not much bigger than m to the m times d, then you just make it bigger.

You can take extension fields. You can take-- so you just kind of artificially increase F so that F will be much larger than m times d. And hence, you'll get a small soundness here.

STUDENT: [INAUDIBLE].

YAEL KALAI: Yeah.

STUDENT: So the applications, looking ahead, what happens is you're given h, right? And the degree happens to be h, as it here, or h minus 1. And that's why it doesn't make sense for h and F to be the same. But you can--

YAEL KALAI: Exactly.

STUDENT: You can disentangle these.

YAEL KALAI: 100%. So let me repeat it because I'm recorded, you're not. Yeah, because that's a very good point. The point is for sum-check in general, it makes sense to consider any h. Even h, which is of size F, is interesting for the sum-check protocol.

In application, because-- as long as the d is small, of course. In applications, often, it turns out that the-- when we use the sum-check, it turns out that the d is often equal to the size of h. Because we do this low-degree extension, which makes the field as large as h. And then if we take h, which becomes the degree to be f, this is meaningless.

So for application, often, indeed, h is small. But for the sum-check protocol on its own, you can take h to be the entire field. Yes?

STUDENT: Is it true that any finite field, it's easy to just put in another bigger kind of field?

YAEL KALAI: Yeah. OK, good, good, good. Yes, exactly. Yeah.

So there's-- you can take-- so this is a fact from mathematics. You can take any finite field and embed it into a bigger finite field. And how big is the bigger finite field? Kind of as big as you want. So you can take any-- loosely speaking, you can take any f and embed it into a field f to the n.

And you can take n as big as you want. So you can kind of-- that's what's called an extension field. So you can extend it to as large a size as you want, while still keeping F as a subfield.

So operations in F kind of behave exactly as they did before. So without ruining this sum or anything like that. Yeah?

STUDENT: These are not the same [INAUDIBLE], right? Is it really the case that there is actually a cheating strategy that succeeds whether or not--

YAEL KALAI: Yes. Yeah. This is good. This analysis is really tight.

So I mean, if I'm malicious, I can always hope for the best. If it happens to be I'm malicious, I gave you the wrong polynomial, look, you may find-- so I want to go from the wrong polynomial to the true polynomial. That's what-- to cheat, right? That's what I want to do.

Now the my wrongful polynomial and the true polynomial, they agree, actually, on d points. Any two polynomials can agree on d points. So as long as the verifier-- I'm lucky enough that this is one of the d points, I'm in luck. I got a good polynomial and I'll continue with the good polynomial till then.

Now I'm la-la land. I'm happy. But the problem is the chance each one-- the chance that you give me a t that's one of the d points, is only d over F. And a union bound over everything, it's n times d over F. And if F is much, much bigger than n times d, then the chances that the prover will succeed-- will be in this la-la land is very small.

Any further questions about the sum-check protocol? OK. So let's take maybe a five-minute break to stretch and so on. And then we'll show-- we'll start showing applications for this.