

[SQUEAKING]

[RUSTLING]

[CLICKING]

**Yael T. Kalai:** Let's start. So first homework assignment is out. As I said, it's due right before Thanksgiving. If anybody needs an extension, just let me know. And the instructions are on the site. Essentially, each one should write like their own solution so make sure you understand. It's OK if you talk among one another as long as it's fruitful. And please type it up in LATAc and submit it through Gradescope. Questions? Yeah.

**Audience:** Is there a Gradescope code for this class? In GradeScope, I thought you go into GradeScope online to do that.

**Yael T. Kalai:** Oh. Why can't you-- don't you use Gradescope for other classes?

**Audience:** Yeah. But I'm mostly in this class.

**Yael T. Kalai:** Oh, no, it is. It is now. When did you check?

**Audience:** Oh. It is like a while ago.

**Yael T. Kalai:** It's recently. Yeah. If there's issues, let me know. Yeah. This class is assigned. There is a GradeScope site for this. Any other questions before we start? OK.

So just a quick recap, so last time we talked about the Kilian-Micali protocol. It's an interactive argument. It's a succinct, interactive argument which you can use to prove membership in any NP language. And if you recall, the idea was we started before that was kind of our entrance to cryptography.

So we said the idea of the Kilian-Micali protocol is that you take any PCP, which is very, very large but very efficiently, if you just have RAM access to it, you can verify it very efficiently. And the idea is you take this very, very large PCP and you shrink it to digest it down using cryptography to this kind of succinct digest. This is done using hash function but, in particular, Merkle hash or hash function that are succinct but can be opened locally. So I'll recall what this is in a second.

And once the prover sends over a squished form of the PCP, then the verifier now behaves like he's a PCP verifier. He tells the prover, open this PCP and the location that the PCP verifier would want to open. And he gets back the openings. So that's what the Kilian-Micali protocol is.

But in order to-- when I said it squishes down and then there's opening, really, what we need is, A, we want to take a hash function. So if you remember, a hash function is-- consists of a key generation algorithm that generates a hash key and an eval algorithm that take a hash key and an input and generates a digest, or a hash value.

Last time, we talked we-- showed how to get a hash value-- how to construct from the discrete log-- we showed how to construct a hash function from  $2^{\lambda}$ ,  $2^{\lambda}$ . It takes two  $\lambda$  bits. The  $\lambda$  bits actually was  $z$ ,  $p$  to the  $2^{\lambda}$  to  $z^p$  to  $\lambda$ . But anyway, one can encode it to have to be 0, 1 to the  $2^{\lambda}$  to 0, 1 to the  $\lambda$ .

But this is actually not enough for the Kilian-Micali protocol. What we actually need is a collision-resistant hash function. First, it needs to be much bigger because we're taking a gigantic PCP. We want to squish it way more than half. This takes-- make-- squishes the size-- reduces the size from  $2\lambda$  to  $\lambda$ . So it squishes it by a factor of 2. We actually want to squish it a much bigger factor.

So we want to take eval-- what we use actually is an eval algorithm that takes strings of size  $2\lambda$  to the  $\lambda$  to  $\lambda$ . So note here the  $\lambda$  is upstairs. It's very different than  $2\lambda$ . It's exponential difference.

So we take a hash function that goes-- takes gigantic strings to  $\lambda$ . And moreover, another property that we want is we can open locally. Remember, we said the way the Kilian-Micali protocol works is that the verifier asks, you committed a gigantic PCP. Open it in location 3, 7, and 15. Now, how do you open it?

So we also want an opening algorithm. So really, what we needed when we talked about Merkle hash, or hashes with local opening-- in addition to having a key generation and eval algorithm, we also have an open algorithm that generates a succinct opening and which can be verified in the verification algorithm that verifies it.

So last time, when we constructed this Merkle hash-- I'll recall it today quickly. And we ended with how-- proving that it's collision-resistant. So that's where we're going to start today. We're going to just wrap up the Kilian-Micali protocol by explaining the thing that was left is to argue that this Merkle hash construction is, indeed, collision-resistant. So that's where we're going to start today.

So once we do that, once we argue that we have this Merkle hash, we prove that it's collision-resistant, then we're done with the Kilian-Micali. You actually proved that it's sound under this assumption. So where are we heading today after we finish the Merkle hash?

Now we actually showed a succinct argument for all of  $np$ . This is great. However, it's interactive. And interactive proofs or arguments are really-- they're not very useful. And the reason they're not very useful is for many reasons.

A, there's latency. We need to talk back, forth, back, forth. But another reason, perhaps much more important, is that it's per verifier. So if I want to just prove to the world some statement, I can prove it using-- I'm not going to go to each person in the world and give them an interactive proof. So it's really verifier-specific. Interactive proofs are verifier-specific. It convinces the specific verifier you talk with. And we want a prove that convinces everyone.

So that's where-- the segue we're taking today. How do we construct, actually, non-interactive, one that will-- to reduce interaction? So what we're going to show is how to eliminate interaction. And this is going to be done using a really beautiful paradigm called the Fiat-Shamir paradigm. So I'm going to explain what it is. It's very, very simple.

And then we're going to introduce the random oracle model, which is a model that was invented in order to-- an attempt to analyze or to understand the security of this Fiat-Shamir paradigm. It's an ideal model. We're going to prove the soundness of the Kilian-Micali protocol that we saw last time. We're going to actually show that it's sound, even if you apply the Fiat-Shamir paradigm to it, as long as you rely on this random oracle model and this ideal model.

And then we're going to-- probably this we're going to only get till next time. But we're going to try to-- we're going to talk about the soundness of the Fiat-Shamir and the standard model. So again, we're going to talk about eliminating interaction using Fiat-Shamir, talk about the random oracle model, show that, actually, the random oracle model of this paradigm works well.

Often, we'll see exactly when it works well, when it doesn't. And then we're going to talk about, if we have time maybe-- start talking about security in the standard model because the random oracle model is not a-- it's an ideal model. So is it secure? Is it sound in-- when used in actual reality? That's the plan. Questions? Yes?

**AUDIENCE:** You shouldn't be able to do  $2$  to the  $\lambda$  because your thing-- if you're hashing something that long, you'll find a collision?

**Yael T. Kalai:** So of course, you can find a collision. So we define it-- so the question was about this  $2$  to the  $\lambda$ . We'll see the construction. I'll explain. We get completeness for  $2$  to the  $\lambda$ . Now you're talking about soundness.

Soundness says that any poly adversary or  $t$ -time adversary-- poly  $\lambda$  adversary or  $t$ -- poly and  $t$   $\lambda$  adversary cannot find a collision. Now, if this  $t$  is  $2$  to the  $\lambda$ , then you're not going to be secure. So the assumption will-- so there's-- this is for completeness.

For soundness, the assumption is-- for collision-resistant, the definition is that no poly  $\lambda$  time or poly  $t$   $\lambda$ , depending on if you want standard collision-resistant or  $t$ -collision-resistant, which is a slightly stronger assumption, cannot find collisions. Great. Thanks for the question. Any other questions? I like questions-- just reminding you guys. Yeah?

**AUDIENCE:** So during this whole thing, I'm trying to imagine how big are the parameters. So  $n$  inside of the input is something that's-- normal people can read, or [INAUDIBLE]?

**Yael T. Kalai:** Exactly. So let me talk a little bit about the parameters. That's a great question. That's a good thing to start with. So we want succinctness.

Now, when we say succinct, really, what we want is-- so we get some things by using cryptography. Once we use cryptography, things grow with the security of the-- that we want. So things will grow with  $\lambda$ , which is the security parameter.

What the security parameter is depends how paranoid we are in the world. You can think about it as 256 bits. Sometimes, it's more, depending on also which cryptosystems you use. But that's a good-- it's independent of the input. You want as succinct as you can get. And if you want to use cryptography, it's like security parameter.

Now, the input-- it should-- in some sense, it should not depend on the input. But of course, the input is not going to be more than  $2$  to the  $\lambda$  because you think of  $\lambda$  is such that  $2$  to the  $\lambda$  is more than the number of molecules in the universe. So nothing is that big in our world.

So you think of the--  $x$  can be bigger. Actually, the input can also be small. But then it's-- sometimes, you don't need cryptography. It's so tiny, it's like cryptography is useless. You can just give the witness.

So usually, we think in our head of the input as being-- can be much bigger than  $\lambda$ . It can be poly  $\lambda$  and maybe even be  $\lambda$  to the log  $\lambda$ . But it's not as big as  $2$  to the  $\lambda$ . That's-- doesn't exist. But it can be anything that's real-world number, in the sense. And so that's what we should think of.

**AUDIENCE:** So when we do something like this, we're trying to reduce the amount of-- or the size of the proof from still a reasonable polynomial size thing, but just to a 256 [INAUDIBLE]?

**Yael T. Kalai:** Exactly. Exactly. So you can think of it-- you're trying to-- when you do-- and let me just repeat it also for the mic. When you're trying-- when you do this interactive argument, you think of-- you take an  $x$  that's still real-world.

You can think of it polynomial  $\lambda$ . But it can be more. But as long as it's a real-world thing and you reduce it to 256 bits, that's the practical idea you should think of. And the more theoretical idea is to say, you can take any input  $x$ , but not more than  $2$  to the  $\lambda$  because that's [RASPBERRY]. But something-- anything less, and we make it as small as poly  $\lambda$ . Great. Any other questions? Yeah?

**AUDIENCE:** And to verify that  $\rho$  is the opening-- is that what it is?

**Yael T. Kalai:**  $\rho$ ? Yes, yes, yes. Sorry, yes. I'll go to it now. Good. So let me just finish-- wrap up what we-- where we left last time, which is the Merkle hash. So we use Merkle hash in the Kilian-Micali protocol.

Actually, we don't need to really remember the Kilian-Micali protocol for today. We're not going to actually really use it. But let me still finish the Merkle hash because it's a very important primitive. It's used everywhere in cryptography. So it's very important to know.

So what is the Merkle hash? It takes any hash function, not local opening, just a regular hash function that generates a hash key and has an eval. Algorithm evaluates, just generates hash value, such that it takes two  $\lambda$  bits to  $\lambda$  bits. Actually, you can make it more. It can be even  $1 + \epsilon \lambda$  bits to  $\lambda$ . You can play with it. But let's say two  $\lambda$ , two  $\lambda$  bits.

And we construct from it a hash function with local opening. So let me tell you exactly what this hash function with local opening-- what it looks like. So the gen, the hash key, is exactly the same as before. We take the same hash key as our underlying-- so first, we started with a hash. We started with this hash family.

What is it? Last time, we saw we can take-- for example, I gave an example with the discrete log. But you can even-- if you don't remember, it doesn't matter. Think of it-- someone gave you a hash family with this domain and range.

Now I'm going to do a domain extension. I'm going to make the domain much bigger and get local opening. How do I do it? So this is the Merkle hash construction. The idea is-- so the gen is, as I said-- just generate the same hash key as the original gen, no change to the gen algorithm.

The eval algorithm, on the other hand, of course, changes because the domain is much bigger. It takes strings of size, at most,  $2$  to the  $\lambda$  bits. And what does it do? So what it does is the following. It takes as input  $x$ . Let's suppose, for simplicity, that  $x$  is exactly  $2$  to the  $l$  times  $\lambda$  bits.

You don't have to assume this. This is just easy to assume. And you can always get it with padding. Take any input and pad it so it will be of this form. And now hash it. I'll say a word if it's not of this form. But if it is of this form, what do you do?

You take your input  $x$ . Just partition it to blocks, each of length  $\lambda$ . So you have  $2^{\lceil \log_2 n \rceil}$  blocks. You have an input  $x$  of length  $2^{\lceil \log_2 n \rceil} \times \lambda$ . You partition it to chunks, each of length  $\lambda$ . So you have  $2^{\lceil \log_2 n \rceil}$  chunks.

Let's call the chunks  $x_1, x_2, x_3, x_4$ , and so on, up to  $x_{2^{\lceil \log_2 n \rceil}}$ , each of them-- up to  $x_{2^{\lceil \log_2 n \rceil}}$ -- sorry, each of them of length  $\lambda$ . And now what you do-- you take two blocks-- this is two  $\lambda$  bits-- and hash it using our building block hash that takes  $2\lambda$  to  $\lambda$ .

You take the next two  $\lambda$ . You hash. You take these two. You hash. So in other words, you take that  $2^{\lceil \log_2 n \rceil}$  blocks to  $2^{\lceil \log_2 n \rceil - 1}$  blocks. Every pair, you hash. Every pair, you hash. Every pair, you hash. Every pair, you hash. You got  $2^{\lceil \log_2 n \rceil - 1}$  blocks.

Then you take these blocks. And again, every pair, you hash. Every pair, you hash. Every pair, you hash  $2^{\lceil \log_2 n \rceil - 2}$  blocks, and so on and so forth until you arrive to one block at the end.

What do you output? You output the top, the final-- it's going to be somewhere here-- block, which is often called a root. You output the root. And you output the depth of the tree. That's the output of the hash.

So the output is the root. The hash value here-- it's  $\lambda$  bits and  $d$ , which you also think of it as being  $\lambda$  bits. Even if it's less, just encode it using  $\lambda$  bits.

So if you're mad now at me because I promised you  $\lambda$  bits and you see here two  $\lambda$  bits, apply this hash one more time, and you'll get your  $\lambda$  bits. But having two  $\lambda$  bits is also fine. Yeah?

**AUDIENCE:** Why are we bounding the input length to  $2^{\lceil \log_2 n \rceil} \times \lambda$  instead of just arbitrary?

**Yael T. Kalai:** Good. Good. Good. Good. Good. Great. Because last time-- great question. So last time, when I mentioned, I said, just take star here, anything. So why am I bounding it out to  $2^{\lceil \log_2 n \rceil} \times \lambda$ ? So this construction bounds  $2^{\lceil \log_2 n \rceil} \times \lambda$  because as you see, I take every  $\lambda$  I can take. That's all it gives me. So that's all I get.

Now you can say, wait, what if I want more? Maybe I want  $2^{\lceil \log_2 n \rceil} \times \lambda$ . You can actually get more than  $2^{\lceil \log_2 n \rceil} \times \lambda$ . For example, you can bootstrap this. So now you can take  $2^{\lceil \log_2 n \rceil} \times \lambda$  to  $\lambda$ . You can create a Merkle hash of this Merkle hash.

But actually, you never-- it doesn't come up because you never need to.  $2^{\lceil \log_2 n \rceil} \times \lambda$  is-- think nothing is more than  $2^{\lceil \log_2 n \rceil} \times \lambda$ -- we choose  $\lambda$  so that in our world, nothing is more than size  $2^{\lceil \log_2 n \rceil} \times \lambda$ . But you can think of-- you can get-- if you-- just as a mathematical curiosity, you can apply it.

So once you have from  $2^{\lceil \log_2 n \rceil} \times \lambda$  to  $\lambda$ , you can chop your things to  $2^{\lceil \log_2 n \rceil} \times \lambda$  and do this-- use this as a building block to a bigger Merkle hash. So you can amplify it. But we're not-- for us,  $2^{\lceil \log_2 n \rceil} \times \lambda$  to  $\lambda$  is good enough. Great. Great question. Any other question? Yes?

**AUDIENCE:** Why does the depth need to be included in the output? Should it not just be a function of the length of  $x$ ?

**Yael T. Kalai:** Good. Great, great question. Why do I insist on including  $d$ ? Is it not a function of  $x$ ? It is a function of  $x$ . It's a deterministic function of the length, not even of  $x$ -- of the length of  $x$ . Yes. However, nobody knows what  $x$  is.

So at the end of the day, I want collision resistance. And so the property we want from this hash is that it's collision resistance. What does collision-resistant means? It means that you cannot output a hash value with two different openings.

Now, when you open a hash value, nobody knows what the  $x$  is. Actually, you may not even know-- well, if you-- when you open, you know. But when you give someone a hash value, there's no  $x$  inherently-- a part of it.

If you don't include the depth-- actually, if there's an attack, it's not collision resistance. It's really not. It's not just a-- why is it not collision resistance? For example, just as an attack, let's say I publish this root, even-- this is my root. This is my input. My input is  $x_1, x_2, x_3, x_4$ . And the output is root.

Now let me break the collision resistance. I'll give you two different openings. One is  $x_1, x_2, x_3, x_4$ . You do the hash? You see you're happy. It's a good opening.

Now let me give you another opening, just this and this. That's also a good opening. So I open it in two different ways. That's not okay. That's a collision. If you include  $d$ , however, then-- so that's exactly what will allow me to argue collision resistance. It's this  $d$ . Great question. Thank you.

So this is the construction. Questions? Let's see. How do we open? So now this is just so far great. We managed to go from 2 to the  $\lambda$  to  $\lambda$ . I didn't argue collision resistance yet. But at least we did domain extension. So we extended the domain. But I also want-- especially, I need it for the Kilian-Micali protocol and for many, many other uses in cryptography-- I want to be able to do local opening.

So remember, what does local opening mean? I may not want to open when-- with-- I may not want to give you my entire input. I may just want to give you-- to convince you that this hash value in some location  $i$  of the input-- the value is a big  $B$ . But I don't want to give you the entire-- because the entire input is gigantic. You can't even hold it.

So I want a succinct opening. I want to convince you in a very succinct way that what's sitting here in this bit is 0 or is 1. So how do I do it? So this construction is actually really nice. It gives you a local opening in a very nice way.

So what's the local opening? Suppose I want to open this-- the bit here sitting here. What I'm going to do-- I'm going to actually give you this entire block. It's only  $\lambda$  bits. So it's not that big.

I give you this block. What do you do with it? So let me help you-- convince you that this block is, indeed, the block related to the root. Namely, it's the block that, indeed, sits here next to-- how do I convince you?

I give you the sibling. So this block has a parent. I give you the sibling and the parent. This parent has a parent. I give you the sibling and the parent, and so on and so forth, until you get to the root. I give you all these blocks.

So in each layer, I open two blocks, which are the two siblings. So I open two siblings. You get this. I open the two siblings. You get this. I open the two siblings, and so on. Yes?

**AUDIENCE:** This is a constant factor. So I don't know if it matters. But do you strictly need to return the ancestors? You can only return the siblings, right?

**Yael T. Kalai:** Good. Good. Good. Right. Right. Right. You're totally right. You're saying, well, if you really want to be-- what you're saying, which is completely right thing-- this is wasteful because actually, all I need to give you is the sibling. So why waste communication?

If I want to open this, I'll give you this and this sibling. I don't need to give you this. You can compute this on your own. So compute this on your own. I'll give you this. I don't need to give you this. You have these two. So compute this on your own.

And then I'll only give you this and then-- and so on. You're 100% correct. I actually just need to give you one. I wrote two because it's just easier to say. But yes, exactly, one is enough. And you can compute the-- fantastic.

So I denote it by-- so what does the open do? For each layer, it gives you two blocks for layer  $j$  denoted by  $z_j$ . So  $z_j$  is two blocks. And the leaves-- it's two blocks.  $j$  equals 0. It's in the leaf-- layer 1, layer 2, up to layer  $d$ . It's actually just the root.

So you can actually go to  $d$  minus 1 if you want. But I'll just-- in my mind, I want to include the root. The reason I include both in the root is now when I analyze it, it's-- notation-wise, it's just very, very easy. But really, it's enough, like you said, to have the sibling. And the roots, of course, you don't need to give.

But let's just think, this is how I open. And now how do I verify? I just take each two sibling, compute the hash, and make sure it corresponds to the relevant  $z$ 's-- indeed, the parent.

So again, you gave me all these things. I compute these two. I check that it's consistent with what you gave you, or if you didn't give me, fine. I'll compute it on my own. I get this. I'll compute this and give in. And then I'll check that-- all I need to check at the end is consistency with the root. That's the verification. Question about the design, the algorithms?

So this is where we left off, actually, last time. And what I want to do now is do the collision resistance. I'm happy that, actually, we got a chance to do it again because it's a very important primitive. So it's worth having it ingrained in your mind.

So why is it collision-resistant? So to prove collision resistance, what do we want to argue? That an adversary that's given a hash key cannot produce a hash value-- namely, root in  $d$ -- with two different openings.

So suppose there exists a poly-size adversary that does find a collision such that the probability that  $A$ -- he gets a hash key. Hash key is from  $\text{gen}$ . And somehow, he manages to output a hash value, which is root in  $d$ . That's just a hash value. And he gives me some index  $i$ . He managed to open it in two different way.

He gives me an opening that says  $i$  is 0. And he gives me another opening that says the  $i$ -th bit is 1 and such that for both of them, such that for every  $B$ , for both 0 and 1,  $\text{ver}$  will accept--  $\text{ver}$  and hash key root in  $d$ ,  $i$ , and row  $B$  outputs 1.

So suppose I have an algorithm that gives me a hash value, an index, and a valid openings, both a 0 and to 1. Valid-- I mean that the  $\text{ver}$  will accept both of them. Oh, sorry--  $B$ .

So ver takes hash key of value, an index  $i$ , and a  $B$ , big  $B$ , saying whether index  $i$  is 0 or 1, and a proof, an opening. And suppose he accepts both. Suppose he accepts both with probability at least, I don't know,  $\epsilon$ . So suppose there's this and exists  $\epsilon$ .

So suppose there exists  $\epsilon$  such that for any  $\lambda$ , the probability that  $A$  managed to find the collision like this is at least  $\epsilon$ . I'm going to argue then you can find a collision in the underlying  $h$ , in the small  $h$ , with probability  $\epsilon$ .

So if you have an algorithm like this, then there exists an algorithm poly size such that  $B$ -- such that the probability that  $B$ -- he gets hash key-- the probability that he outputs, I know  $x_0$ ,  $x$ , and  $x$ , prime such that the original  $\text{eval } hk, x$  equals  $\text{eval } hk, x$  prime, and  $x$  is different than  $x$  prime, is at least  $\epsilon$ .

But we assume that this is collision resistance. We started with an underlying hash function that's collision-resistant. So the  $\epsilon$  has to be negligible. That's what we're going to show. So we're going to use if an adversary can find a local opening that collides for the Merkle hash, we can use them to actually break the defined collision in the underlying  $2^\lambda$  to  $\lambda$  hash function in the little hash function.

So how does  $B$  work? What does  $B$  do?  $B$  takes a hash key. He just runs  $A$ . So let me-- so we claim that there exists a  $B$ . So let see.  $B$ , an input hash key, what does he do? Just run  $A$ . So get root  $D$ , I, row 0, row 1, which is just  $A$  and hash key.

Now, he got row 0 and row 1. What are row 0, row 1? Each row is this kind of two pairs of siblings. I'm going to assume that both exist because it's just easier for me. That's why I wrote both of them. So each opening-- so row 0 is for each level you have  $Z_0, j$  from 0 to  $D$ . And row 1 is-- let's call it  $Z_1, j$ .

Now, what do we know? In the leaves, so we know that  $Z_0, 0$  is different than  $Z_1, 0$ . The two leaves are different because, in one leaf-- suppose verifier accepts both of them. In the case that these are both accepted, I'm going to argue that I found collisions. So now suppose row 0 and row 1 are accepted. So ver accepts both of them.

Now, if ver accepts both of them-- so we know that the leaves are different because, in the leaves, in one case, I have here a 1, in the other case, I have here a 0. So they have to be different. So I know the leaves are different.

I also know that  $Z_0, D$  equals  $Z_1, D$ . These are both a root. Now, of course, I said, we don't need to include it because-- but I'm just putting here because I want-- the reason I want to include it is just for clarity to say, look, we have  $D$  layers of information on each opening. The leaves, they must be different because one opens to 0, one opens to 1.

The root, the left one must be the same. It's the root. It means that there must be some kind of two adjacent layers on which one of-- below they disagree and above they agree. We know here they disagree, here they agree. They disagree. Do they disagree here, disagree here? At some point they're going to agree.

So we say there must be a layer  $J$  such that  $Z_j 0$  is different than  $Z_j 1$ , But  $Z_{j+1} 0$  is equal to  $Z_{j+1} 1$ . Yeah? OK, they disagree here. How about one layer above? Do they agree? If they agree, great. Then that's going to be our  $j+1$ . They disagree? Let's go up until, at some point, they'll agree because, Hamiltonian, do they agree. Yeah.

**AUDIENCE:** So just for the video, should it be  $0 \leq j < D$  and  $0 \leq j < D$ ?



**Yael T. Kalai:** Oh, thank you very much. Thank you. Great. Thank you. Yeah, it's not just for the video, it's also for you guys so I won't confuse you more than that. Thank you. 1,  $j$ , 1,  $j$  plus 1. Thank you. OK, yes. Is it--

**Audience:** Is that [INAUDIBLE]? Because  $j$  plus 1 is-- oh. You missed  $j$  plus 1.

**Yael T. Kalai:** Did I miss--

**Audience:** You missed  $j$  plus 1.

**Audience:** On the bottom.

**Yael T. Kalai:** Yes.

**Audience:** You need a 0.

**Yael T. Kalai:** Oh my God. OK. No, the 0, 1. Ah! OK, did I get it right? Yes. OK, so there's a layer  $j$  such that, in layer  $j$  they disagree, in layer  $j$  plus 1 they agree. That's it. This is a collision. Why? What is layer  $j$  plus 1? One of them is the father, is the parent. So we have two that they disagree. But both of these they agree on. In particular, they agree on this.

So then, so we found a collision. This must mean that the hash eval hash key of  $Z_0j$  is equal to eval hash key of  $Z_1j$ . Because this value is one of this, and one of this depending if it's the left or the right depending on the tree, and they're both equal. So done. I found a collision with the same exact probability. So it's--

Now, as I said, this-- I wrote here, suppose if this poly lambda, I could have-- if we assumed that the underlying hash function is  $t$  secure, namely you can't find collisions even if you run time poly  $t$  of lambda, then we get that the Merkle hash is poly  $t$  secure because  $B$ , all it does really is just run  $A$ . It doesn't run much more than  $A$ , it just runs  $A$  and then it does something very trivial.

So the complexity of breaking the Merkle hash is really the same as the complexity of breaking the underlying hash. So if the underlying hash is  $t$  secure, Merkle hash is  $t$  secure. Questions?

OK. So where are we? So we proved the Merkle hash. As we saw, the Merkle hash was used to give an interactive-- a succinct interactive argument for all of NP. So now, if you're willing to-- you're happy with interactive protocols, thank you very much. I hope you enjoyed the semester, and we're done.

[LAUGHTER]

If you're not though-- but, as I said, we're actually-- we're not happy with interactive protocols. It's really interesting, actually, to think of the evolution of this because before all this interactive proofs and arguments and so on, we had just mathematical proofs, which were not interactive, but they were very, very long. And people were not happy. It's like, oh, they're so long.

And then interactive proofs were defined. I think they were defined only for the sake of zero knowledge. They were defined just to get zero knowledge proofs. I'm going to actually mention that-- we're going to talk about zero knowledge later today.

And this motivated the interest-- to introduce interaction. And then people are like, wow, you-- oh, what a great-- forget about zero knowledge. Forget about cryptography. This is such a great model. Using interaction we can prove a lot more.

And we saw the GKR protocol. You can do-- you can prove, actually, any circuits that are really, really huge as long as they're kind of have that, you can do P space or any computation that has poly depth. Your verifier can be any depth D. You can-- the verifier runs in time D. Great. Not-- it doesn't run in the side, so it's much, much more powerful. Wow. Wow. Wow.

And then people are like, actually, with cryptography, you can do much more. Great. Yeah! But now, but wait, there was a big price. We actually don't want interaction. So where are we. Are we back at NP? NP is very-- we're back in the mathematical proofs that are way, way too long?

So I said, no, we're actually going to circle back in a very interesting way and get succinct non-interactive proofs. So how do we circle back?

So this idea of how to circle back actually was put forth by Fiat and Shamir. This is in the mid '80s, in '86. They proposed a really, really nice paradigm. Actually, they didn't think about-- this is, remember, this is '86. This is before interactive proofs. It was just when zero knowledge was introduced. Interactive proofs just came about, but now there was no PCP, no interactive argument, none of that. This is in '86, Fiat and Shamir.

They proposed a method of actually reducing interaction-- of constructing signature schemes. Now, I stand behind what I said, that this class will not require a lot of crypto background. You don't need to know what signature schemes are.

But what they did, their idea is, there is a primitive called identification scheme. I want to prove to you that I'm Yael. How do I prove that I'm Yael? I have a public key corresponding to my name. And I'm going to prove to you that I know the secret key. Me knowing-- whoever knows the secret key has my identity, essentially, that's how things work in the digital world. So I'm going to prove to you that I know the secret key. This proves that I'm me.

This proof is interactive. Identification protocols are like three-round protocols. I give you a message alpha, you send me a question beta, I give you an answer gamma, you verify. That's how it works.

Now, they said, let's convert this identification scheme into a signature scheme. Instead of interacting to make sure it's you, every time you sign-- you want to send a message, use this message, take this identification protocol together with the message and eliminate the interaction. I'll explain exactly how they do it. But they do it in a way to construct signature scheme. That was their goal.

But now, after several years after, we use it all the time to actually forget about just the specific application of identification scheme to signature, that's one application. But actually we can show-- we can use it to reduce, to eliminate interaction, not only from identification scheme, but from any interactive protocol that is public coin.

So let me explain. So now, take any protocol. So this is a paradigm for eliminating interaction from interactive protocol that are public coin. So not all protocol, but ones that are public coin.

So what is a public coin protocol? Take any protocol, let's say, a proof. So I have a prover and a verifier. And suppose for a second-- so suppose for simplicity that it's three messages.

So the verifier-- the prover sends a message alpha. The verifier sends beta, which is random. Public coin means the message of the verifier is completely random. Let's say it's  $\lambda$  bits. If it's less, then just pad it to be  $\lambda$ . And if it's more, just call that  $\lambda$ . Make that the security parameter. Just increase the security parameter.

Now you send gamma. And maybe it's more round. So maybe then he sends delta, let's say random bits, and then you say epsilon, and you can continue. Let me stop here for the sake of--

Now I'm going to eliminate interaction. How do we eliminate interaction? It's an interaction, it's very important. I mean, the entire-- if you remember the sumcheck in GKR-- I mean, it's tempting maybe to say, oh, tell the prover, compute alpha, beta, gamma, just compute the transcript on your own. You simulate the verifier's messages, then he'll cheat.

So how do you get rid of the interaction? It seems really, really hard. Actually, it's so easy. I mean, their idea is trivial. So what is their idea? They say the following.

How does the verifier-- how does the prover get this beta? The verifier chose it. Do you know what? Let's not have the ver-- now we don't have a verifier. It's not interactive. There is no verifier. So how do you compute this beta?

Let's compute it as a hash function applied to the transcript so far. So let me explain. So how-- here's how you convert it to P Fiat-Shamir, V Fiat-Shamir, this is associated with some hash family. There are some-- it's associated with a hash family. So the Fiat-Shamir paradigm, you can take any hash family and use that to reduce interaction.

How do you use it to reduce interaction? Now, there are some hash key that's chosen once and for all. Everybody knows it. Either the verifier sends it once, or if you think about public-- everybody can verify, I don't know, the US government publishes this hash key, everybody uses this hash key. There are some fixed hash key, everybody uses it.

Now, once we agree on a hash key, what does the prover do? He computes alpha. Let's say he wants to prove that  $x$  is in some language. So let's say he says, OK, I'm going to prove to you that  $x$  is in the language. Let me prove it to you.

OK, I compute the alpha. Now I'm waiting for a beta. But there is no verifier. Fine. I'll tell you what the beta is. Beta is just-- has to-- is going to be eval of hash key and the transcript so far,  $x$  and alpha. That's currently what's in the transcript, the statement and the first message alpha.

Now the proof is like, OK, this is the beta I got from the verifier. He's interpreting this as the beta he got from the verifier. And now he computes gamma, my answer, the prover's answer. Now he's waiting for a message for-- a query from the verifier, some delta. But there is no verifier, so it's going to compute the delta as eval of the hash key using the transcript so far. So  $x$ , alpha, beta, gamma.

Once he has delta, he's going to go back to being a prover and compute epsilon. And then you can continue. Next message, he goes-- computes a hash value of all the transcript so far, computes the next prover's message. Here another verif-- and we go-- that's how we continue.

Any questions about-- so what did we go? We started from an interactive protocol. You can have many rounds. And now all we need is to agree on a hash key. And once again, we're going to hash key-- the prover just gives you-- this is one message. That's it. Yeah.

**AUDIENCE:** Is it important to include beta in the transcript when you're hashing?

**Yael T. Kalai:** Good, great question. Is it important to include beta? Actually, you don't need to include beta at all because the verifier can compute beta. So you only need to know beta so you can compute gamma. You actually don't need to include it. Great question. Great, great question.

So note, for example, if you think about Kilian and Micali protocol, you may not remember it, but let me just write it here just to refresh your memory. Why not? You don't need to actually understand this, but since it's such a nice protocol-- and so, how does it go? There's a prover and a verifier. The verifier sends a hash key.

So he wants to prove that  $x$  is in  $L$ . He does a Merkle hash of the PC. So he computes a PCP.  $\pi$  is a PCP for  $x$  in  $L$ . And he sends you the Merkle hash. So he sends you root and the depth. That's just the-- so he takes the PCP and he does a hash with local opening. For example, Merkle hash actually can do any hash with local opening that you want. This is the most common one.

And then he gets the randomness from the PCP verifier. So this is  $r$  from  $r$  corresponding to the PCP verifier. This determines a few locations to open. So now he opens. So he gives-- so this randomness corresponds to some locations,  $i_1$  up to  $i_l$ , and then you just open  $\pi_{i_1}$  with open 1 up to  $\pi_{i_l}$  with open  $l$ . That's the protocol.

Note, this is completely random. It's the randomness of the PCP verifier. So we can eliminate interaction. This, if you think of it, this is just of a protocol where you send a hash key, and then you have alpha, beta, gamma. So let's eliminate interaction. So how do we eliminate interaction? Well, there's this hash key. We can use this hash key to do Fiat-Shamir, but we can also use another hash key, hash key sub Fiat-Shamir.

And now the-- well, this can be-- now you give alpha. Beta is going to be simply-- you don't have to include it. But it's going to be, in our head, it's going to be eval of hash key Fiat-Shamir of  $x$  in alpha. And then maybe hash key is also part of the transcript. And this gives beta. Now you compute the gamma, which is the opening.

So you can take any-- so now if-- what does the Fiat-Shamir gives you? It gives us a way-- remember, Kilian and Micali was very succinct. You can take  $x$  any-- let's say NP language,  $N$  can be very large. It can be  $\lambda$  to the 1,000, I don't know, it can be whatever. It can be even super polynomial  $\lambda$ . It can be almost as large as  $2$  to the  $\lambda$ .

And look how succinct the protocol is. You just send a hash key, the root, which is  $\lambda$  bits, and the opening, the PCP is all  $\lambda$  bits, poly  $\lambda$  bits, so very, very succinct. And now you can even just make it-- so what is this-- you can all send it all one round.

So what does it give you? What does it tell us? That we can take any NP statement and give a proof of size poly  $\lambda$ . Doesn't matter what size  $x$  is. That's amazing. Without cryptography, we need poly [INAUDIBLE], these gigantic witnesses. Yeah.

**AUDIENCE:** I guess for the collision resistance security property and probabilistic encoder hash keys, you can imagine some kind of weird hash family where there exist hash keys that are really bad. And so I guess here it's fine because the verifier chooses the hash key. But you might imagine if the US government publishes a hash key or anything else, what if it happens to be a bad one or it was chosen to be a bad one?

**Yael T. Kalai:** OK, two things. First, you're saying, who chooses this hash key of the Fiat-Shamir, how do we know that we trust it? But you know, it's much worse than that. Is it secure? Let's say it's chosen randomly from the hash family. Is this sound?

So again, what did we do here? We started with an interactive protocol. We proved it's sound. Let's say, Kilian and Micali is sound. We proved it. Assuming we have collision-resistant hash, we proved that it's sound. It's sound assuming the prover gets the messages of the verifier randomly. That's what it sounds. We used it to prove soundness.

Now, what am I saying? What are we doing now? I'm saying, oh, forget about it. It's not random. Actually, you compute it on your own from some hash key that someone chooses even honestly from-- let's say we agree-- we all agreed on some hash family and someone honestly chose a hash key. But now you know this hash key. Now you compute. Who says it's sound? I mean, is it still sound? Maybe it's not sound.

**AUDIENCE:** Maybe the hash key always outputs 0, but at the start--

[INTERPOSING VOICES]

**Yael T. Kalai:** OK. So first of all, of course, there's-- one can think of-- this hash functions need to have some property. Collision resistance, at the minimum, if you can find collisions-- for example, here, you're screwed. I mean, it can't be like that. It can't be trivial. So you need some property.

Now the question is, what property do you need? What property do you need from this hash family to argue soundness? Yeah.

**AUDIENCE:** I'm not sure if it's pseudorandomness or true randomness, but the output's still random.

**Yael T. Kalai:** Very, very. OK, good. So it seems like, well, what do you need? It needs to be random, like in that protocol. We want to say, take any-- what is our goal? Our goal is to say, take any protocol that is sound, public coin, and we want to convert it to non-interactive. That's what we're trying to do.

Now, I'm saying, what property do you need? Well, look here, well, our guarantee was that it's sound if this is random. So we need this hash to be somehow random. Stands to reason. So this is written out. But, look, hash functions are not random. It's a hash function.

By the way, let me say, this paradigm is used all over the place. It's a very popular paradigm. It's used a lot in signature schemes and also in proof systems. It's used all over the place. And which hash function is used? Some off-the-shelf hash function, SHA256, like these engineer-type hash functions that are optimized.

So is it secure with SHA256? It's definitely not random. So we actually don't know. So let me be honest that is it secure? I don't know. Yeah.

**AUDIENCE:** Yeah. I think the specification [INAUDIBLE] is already shown that by length extension [INAUDIBLE] is definitely not now indistinguishable.

**Yael T. Kalai:** Yeah, right. Yeah.

[INTERPOSING VOICES]

**Yael T. Kalai:** Right. Right. Right. So actually, let me-- good. So let me just say a few things. OK, maybe I'll cut to the chase. I do know-- let me tell you, it's not necessarily secure. Actually, unfortunately, we have counterexamples. So I said this is used in practice all the time. It's one of the great inventions of cryptography. Is it secure? No.

[LAUGHTER]

So what do I mean by no? OK, let me-- is it always secure? No. So let me say-- let me take back my notes said. Is it always secure? Is it always the case that when you start with an interactive protocol and convert a public coin, and you apply Fiat-Shamir, is it always secure? No.

Moreover, let me tell you, we have protocols so that no matter which hash family you use, for all hash families, the protocol becomes insecure. Let me tell you even more than that. That will really you'll be mad. The Kilian-Micali protocol that we built up, ah, ah, ah, even that's not secure. The Fiat-Shamir will be secure on  $n$ .

But this, let me actually put an asterisk there, it's what-- OK, so let me tell you what's known about the insecurity of the Fiat-Shamir paradigm for this protocol. What is known is there are some contrived-- so the Kilian-Micali protocol says, take any hash function that's collision-resistant and take any PCP, we get soundness.

There are examples of specific hash functions or specific PCPs that are sound, of course, because it's-- I mean, there are examples of collision-resistant hash functions and PCPs such that this, of course, is sound-- it's sound for any collision hash function PCP, but such that no matter which hash function you use for the Fiat-Shamir, this will be insecure, not sound. No matter what you use. Yeah?

**AUDIENCE:** Could you do a little more in detail what breaks?

**Yael T. Kalai:** Yeah. I'll say a little bit what breaks in a second. Yeah.

**AUDIENCE:** [INAUDIBLE] or it's natural?

**Yael T. Kalai:** OK, good. Same question. So the example for why-- so why it breaks? The example for why it breaks is very contrived actually. So that's why, is the message here, don't use Fiat-Shamir. We have a break. No, that's not the message. You should use Fiat-Shamir because, in practice, it works very well. I don't have a single instance that people use Fiat-Shamir in practice and it broke. So it's a great paradigm. Use it.

Can we come up with a general proof of security? No, because we have counterexamples. So what did we learn? So maybe I'll start by saying-- I'll say very, very high level, not in detail, but about the counterexamples. How can we build counterexamples? And as opposed to going to the Kilian-Micali protocol, because there you need to work a little harder to fit it in-- I'll say a word about how you fit it in there too.

But the idea is the following. Let me try to give you a proof that is-- or an interactive protocol that is sound. But when you convert it to-- when you apply Fiat-Shamir, soundness breaks.

So the idea is the following. It can be a very contrived protocol. So what I want to do is construct a contrived protocol that is sound. But when you apply Fiat-Shamir, no matter which hash function you use, it will break.

And intuitively, the contrived protocol is the following. Take any sound protocol, for example, Kilian-Micali. Whatever-- or GKR whatever sound protocol you like. And I'm going to change it-- I'm going to tweak a little bit that the soundness will still remain. I'm going to tweak it so that the soundness will still hold. But on the tweaked protocol, when you apply Fiat-Shamir, it will fail.

So start with PV, for example, GKR, for example, Kilian-Micali, whatever you like. And I'm going to tweak it a little bit. How am I going to tweak it? I'm going to say, tell the verifier, you know what? Accept, if you accept it before, but accept also if something very weird happens. But don't worry, it will never happen.

When else are you going to accept? So remember the verifier, let's say, he sends a random beta. If the prover, let's say, ahead of time, guessed your beta, he told you, I know how you're going to compute your beta. So if the prover, ahead of time, send you a message that will tell you-- say, I can predict beta, and will predict beta ahead of time, then you accept him because this is random.

What's the probability that you can pick my beta? If beta is random 0, 1 to the lambda, there's no way you can predict it. So if by any chance you predict I'm going to-- you won the lottery, I'll accept you. I don't care if x in the language or not. It only adds a negligible probability of-- because the probability that you could predict beta is 1 over 2 to the beta.

Now, what happens in the Fiat-Shamir paradigm? Oh, I can tell you, oh, I know, beta, I can predict it. And I'm going to tell you exactly how I predict it. Beta is going to be equal hash key and this message. That's the idea.

Now, to implement this idea, it's harder. It requires some work. I'll tell you why this idea is not trivially implemented, because when we do Fiat-Shamir, we say we start with a protocol. And now the question is, does there exist a hash of Fiat-Shamir hash family for which it's secure?

Now, we saw with the protocol, this protocol can say, OK, alpha is hash key, for example. And then you accept, if you originally accepted, or if eval of hash key of alpha, which is also hash key, is equal to beta. Now, this will not happen, but in the Fiat-Shamir it will happen. I'm going to just send the Fiat-Shamir hash key. That's what I do.

But then someone will say, OK, so you give me a protocol. What is the length of your alpha? It's lambda bits. I'm going to use a hash function that the hash key is longer. And now you can't send it because here I'm like, I'll send the hash key. So then you need to use a hash to squish this hash key in. It's more complicated.

But the basic idea is, this does not work if somehow the verifier accepts you if you convinced him that you can predict his beta. That's really-- and so we can construct contrived protocols that achieve this. Or we can construct a very contrived hash function that do this weird game inside the hash function, like the hash function becomes trivial if you feed it something that kind of-- so it requires more work, but it's all very contrived.

So I hope I convinced you by-- with these examples, the counterexamples are so contrived. So when you say, oh, we found that Fiat-Shamir is not secure, and, you show this example, the reaction should be, come on. Nobody uses these protocols.

But what it does tell us, you can't apply Fiat-Shamir blind. What it does tell us is, for all of us who are obsessed with trying to prove the paradigm, there's no general proof because we have counterexamples. At least that. Question?

**AUDIENCE:** Do you need to start with the protocol that's negligibly sound?

**Yael T. Kalai:** Great. Great question. That's my next one. Yeah. The answer is yes, but I'll talk about that next. The question was about negligible soundness, but we'll get to that next. So any questions though before-- OK.

So but, still, people use the Fiat-Shamir paradigm. And there's a question of is it sound? For natural protocols, is it sound? And the answer is, actually, it would have been nice if we can come up with nice protocols and say, at least a large class of protocols, and say, yeah, if you use this hash function, this specific hash function, which we know is collision-resistant under discrete log or under some other assumption, then your scheme is sound. That would be very nice.

We're still not there yet, but we made a lot of progress in recent years. And I want to tell you about the progress. But before that, the first kind of way we tried to analyze-- so the Fiat-Shamir gained popularity very quickly. And because it's so simple and so efficient, it became very, very popular. And people really wanted to understand its security. Is it sound? Is it not sound?

And then in '93, Bellare and Rogaway introduced an ideal model called the Random Oracle Model. And this model said, you know what? By the way, this was before we had any counterexamples. This model was introduced by Bellare and Rogaway in, I believe, '93. This was before we knew any counterexamples for the paradigm.

And they-- I mean, except for some, which goes to your question, Leo, but I'm going to get to it. And they wanted to ask, can we prove security of the Fiat-Shamir-- what do we need about the hash function? What properties do we need about the hash function to prove security?

And the first property they said, you know what? What if the hash function was completely random? So when you use Fiat-Shamir, we use a specific-- there's a circuit that computes the hash function. It's like a Turing machine, a polynomial time Turing machine or a polynomial size circuit takes a hash key and, OK. They say, you know what? What if the hash function is completely random?

So the Random Oracle Model says, suppose we apply Fiat-Shamir paradigm with truly random function  $H$ . So this is completely, completely random. So what do I mean by completely random?

So now, I guess, let me, instead of choosing a function from a family, we don't choose a function, and we just say, suppose they have a truly random-- they have access to a-- both of them-- to a truly, truly random function. And they can ask. So when the prover computes  $\alpha$ , he asks-- he has Oracle access to this function. And he tells the function, give me the function applied to  $x$  and  $\alpha$ , and he gets back  $\beta$ .

And every time this hash, this Oracle, he gets a query, he chooses a completely random-- so think of this as a database of all poly of size  $2$  to the  $\lambda$  of all the-- just random, random elements. So really, really truly random function.

So now the question was, OK, if this is truly random, now can we get security? At least, that's the minimum. And moreover, it seemed like the answer should be yes because, in some sense, what was their intuition?



The intuition is, what does it matter if you're talking to a verifier that gives you random queries or you're talking to, in terms of the prover. So the prover is trying to cheat. What does it matter if he's talking to a verifier to give him random betas, or he's talking to a random function that gives him random betas? Both of them give random betas. What's the difference?

So it should at least be secure in the Random Oracle Model. Let's first establish that. So is it secure in the Random Oracle Model. No. And this goes to Leo's question. And I'm going to tell you why it's not secure in the Random Oracle Model.

And actually, let me take a little detour and tell you-- I'll show why it's not secure in the Random Oracle Model via a little protocol. This protocol actually is going to be useful because, actually, later, we will show that the Fiat-Shamir is sound for this protocol. It's kind of weird. But let's-- this protocol, by the way, is very important protocol.

So let me quickly do-- let me do a little detour. I think it's a really interesting detour. So I hope you enjoy it. It takes us a little bit into the land of cryptography, but not too bad-- a little further than we went so far. So far we just talked about hash functions.

So what I'm going to say now is going to be a little bit of a high level. So a protocol that I want to talk about is actually a zero knowledge protocol. So, so far, we just talked about verification, verification, succinctness, and efficiency. But, as I mentioned, interactive protocols were actually invented for the sake of zero knowledge.

So what was the goal? The goal was, forget about succinctness now. It was not at all about succinctness. All we wanted is a prover that has, let's say, a witness for some-- there's some language. And let's say, the language we're going to be interested for today, just as an example, is the language of Hamiltonian cycle. That's just an example.

So this language consists of graphs. So this is the language that consists of all graphs. Graph is just consists of nodes and edges such that this graph has a Hamiltonian cycle such that  $G$  has a Hamiltonian cycle. What this means is just, you can just-- there's a kind of a cycle of all the nodes. So it means that there's a cycle of all the nodes.

So let's say you start with node  $i_1, i_2$ , up to  $i_n$ , and then go-- up to  $i_n$  so that there's an edge here, edge, edge, edge, edge, edge, and edge all the way back. That means, if you can-- if there is some such a setting of the nodes for which you can set them in a way that there's edges that complete a cycle, then we say the graph has a Hamiltonian cycle. This language is NP complete.

Now, so now going back, what is the zero knowledge proof? A zero knowledge proof is a way I want to convince you that a graph  $G$  has a Hamiltonian cycle. Now, I can give you the Hamiltonian cycle. I can give you-- see? Here's the nodes. You can look, there's a cycle there.

But I don't want to give you any information. I don't want you to learn anything. I don't want you to find the Hamiltonian cycle. I don't want to reveal any information, but I want to convince you that it has a Hamiltonian cycle. How do I do that?

So this kind of-- so there's a celebrated result. It started with Goldwasser, Micali, and Markov in the mid '80s and where they defined this notion of zero knowledge and constructed some protocols. But later, very shortly later, by Goldreich, Micali, and Wigderson showed that, actually, any PNP language has a zero knowledge proof.

Let me give you one specific zero knowledge proof for the Hamiltonian cycle language. And this is due to Blum. So I'll show you the proof. It's very, very nice. And it's going to be useful for us to show why the Random Oracle Model fails, and then also to show the security of the Fiat-Shamir. So this is actually-- we'll get back to this protocol.

So what is this protocol? There's a prover and a verifier. The idea is the following. The prover will-- he knows-- he will first-- he has a  $G$ . The first thing he's going to permute-- completely permute the nodes. So he's going to choose a random permutation  $\pi$ . Let's say  $n$  is the number denoted by  $n$ , the number of vertices in  $G$ . He's going to choose a random permutation of the nodes.

Now he has the permuted graph in his head. So he has  $\pi$  of  $G$  in his head. Now this, it has, this  $\pi$  of  $G$  has a Hamiltonian cycle because  $G$  has and I just renamed the nodes. Now I'm going to-- I want to give you just this Hamiltonian cycle.

So I want to give you the nodes here that form a Hamiltonian. So I want to give you-- I would like to give you the nodes that form the Hamiltonian, just the Hamiltonian cycle. But I can't give you, that will reveal information. So I don't want to do that.

Instead, I'm going to give it to you in a safe, hidden. This is what's called, in cryptography, we call this commitment scheme. So I'm going to give you a commitment of the cycle  $C$ . So think of this, for all the possible edges, and grid 0, 0, 0, 0, only in the cycle I'm going to put 1. That's what I mean here.

Because, so this is-- think of it as a matrix of  $n$  by  $n$ -- sorry, yeah,  $n$  by  $n$ . For any possible edge, for any possible nodes,  $i$  and  $j$ , I'm going to put 0, like there is no edge, only if it's in the cycle I'm going to put an edge there. Yeah.

**AUDIENCE:** So a cycle is on the label of the permuted graph.

**Yael T. Kalai:** Exactly, on the permuted graph. The reason I permute the graph is to ensure zero knowledgeness. Even though when I send the commitment, really, think of it-- we're going to later talk about how to do this commitment. But for now, think of it, there's a safe. I'm putting my commitment inside the safe. I leave the key to myself and I give you this box. So you learn nothing. So here, really, you learn nothing.

But now-- but you're not convinced either because you just got a safe. What are you going to do with it? So now here's how I'm going to convince you that, actually, this graph has a Hamiltonian cycle without leaking any information.

So you, the verifier, you're going to give me a random bit  $B$ . This is going to be random, random bit. Now, if  $B$  is 0, I'm just going to open this. So if  $B$  is 0, if  $B$  is 0, open. I'm going to open the safe. And now you can see that what's sitting there is a Hamiltonian cycle.

Now, what did you learn? Well, because I permuted, it's just a random cycle. Actually, it has nothing to do with the graph. I can even first just choose a random cycle, commit to it, and then post choose this permutation to align with this cycle. So really, because I chose a random permutation, it's just a random cycle. So you didn't learn anything. It's just a random cycle. You and yourself can put a random cycle in a safe. So really, no information.

What if  $B$  equals 1? If  $B$  equals 1, I want to-- first I want to give you  $\pi$ . So I give you  $\pi$ . But now I can't open this. Because now, if you know  $\pi$ , and you see the cycle, you know what my Hamiltonian cycle is.

So what I'm going to do, I just want to convince you that this cycle only has edges that were in this graph. So I'm going to open only the non-edges. So instead of opening everything, I'm going to open non-edges in the permuted graph.

So note, here we have-- so I have here-- so there's a question, how do I open? So think of it, I commit to  $n$  squared bits. For any possible  $ij$ , I have a bit saying, is it 0 or is it 1? Is there an edge or is it not an edge? Think of it as I'm giving you  $n$  squared safes.

Now, if  $B$  is 0, I open all the safes. I open all the safes and you see, oh, there's only-- there's just a cycle. There's everything is 0 except for  $i_1$  and  $i_2$  are-- there's one, and then  $i_2$ ,  $i_3$ , that's it. Everything else is 0.

If you send me a 1, then I open something else. I don't open everything. The opposite. What I do is, I give you  $\pi$ . And for every non-edge in  $\pi$ , I open the non-edge safe and I show you, see, there's 0. So all the safes I opened are just going to be 0.

So you really didn't learn anything. I just opened zeros. You can simulate that in your head. What do you learn? In some sense, you know that if you accept me, I'm going to open to just zeros there. And the  $\pi$  doesn't reveal anything. It's just a random permutation.

So note, you, the verifier, didn't learn anything. If you open to 0, if you send me 0, I just open random cycle. Has nothing to do with the graph, nothing, just random cycle. That's not helpful for you. If you open 1, I gave you completely random permutation. You know  $\pi$   $G$  on your own, and I opened to zeros. All the non-edges are open to 0. You learn nothing.

Still, you are somewhat convinced that this protocol-- that the graph is Hamiltonian. Why are you somewhat convinced that there's a Hamiltonian cycle? Why are you somewhat convinced? Because if there is no Hamiltonian cycle in  $G$ , I cannot open to both 0 and 1.

Why is it the case that I cannot open? Let's see. I'm achieving-- let's say the graph, is there is no Hamiltonian cycle. Now, I sent you these  $n$  squared safes. Case one, there's indeed only a Hamiltonian cycle there. Indeed, when you open all these  $n$  safes, what you see is everything is 0 except for Hamiltonian cycle.

If that's not the case, I cannot open to 0. If I open to 0, you catch me. Now, suppose I did succeed in opening to 0. Namely, there's just a Hamiltonian cycle there. That's it.

Then I claim there's no way I can open to 1. Why? If there is a Hamiltonian cycle there, I know the Hamiltonian cycle is not on the non-edges because the non-edges are 0. If I open on one, the non-edges are 0. So the Hamiltonian cycle is on the edges. But there is no Hamiltonian cycle. [LAUGHS] It can't be on the edges.

So either-- I'm achieving prover-- either I can-- I may be able to open to 0. I may be able to open to 1, but I can't open to both. So with probability  $1/2$ , you'll reject me. Half isn't that great, but it's something. So at least now we know, with probability  $1/2$ , I'm going to be rejected.

You like the detour? OK, let's go back to the random Oracle. But before-- yeah.

**AUDIENCE:** So do you have to compute the whole graph, or just a cycle?

**Yael T. Kalai:** So what-- no, what I commit to is, actually, I have  $n^2$  commitments. Everything is 0 except for a random cycle. I do-- I only commit to zeros. Why is it important that I only commit to zeros? Because when I open-- ah, OK. I want to make sure that there's a cycle here.

But when I open to 1, I don't want to reveal anything. And the way I don't reveal anything is I show that all the non-edges in  $\pi$ , in  $\pi$  of  $G$ , if you open them here, you'll get a 0.

**AUDIENCE:** Oh, OK. So it's a matrix. So it already kind of has everything-- with the cycle, like outlining the cycle.

**Yael T. Kalai:** Exactly. Exactly. What I commit to is a matrix-- what I commit to is a matrix that only has a Hamiltonian cycle and that's it. Everything else is 0. Yeah.

**AUDIENCE:** So in the second case, the verifier cannot use the permutation to figure out what edges are not in the Hamiltonian cycle for the original graph.

**Yael T. Kalai:** Right. So in the  $B = 1$  case you're talking about? Yeah. In the  $B = 1$  case, the verifier-- what does he learn? He learns  $\pi$ , the permutation. That's just a random permutation. And then I'm just telling you, I'm reassuring you that I didn't lie.

What do I mean by reassure you? I open-- everybody, you can compute  $\pi(G)$  on your own. That's not a secret.  $G$  is given,  $\pi$  is given. You can find  $G$ . You see all the non-edges. That's not a-- and all the non-edges, I'm going to open only the non-edges and show you zeros. Because the point is, if I was honest, what I should do, is put ones only on edges, and not only on edges, only on the Hamiltonian and the edges that form a Hamiltonian cycle. That's what I should do if I'm honest.

And what I show you, I don't show you that because that will give information. But at least what I show you. Then all the non-edges here, I committed to 0. And that gives you no information.

But if, indeed, the non-edges are 0, and there's a Hamiltonian cycle, that means the Hamiltonian cycle must be edges. And then it means that  $\pi(G)$  has a Hamiltonian cycle. And that can't be. If  $G$  doesn't have a Hamiltonian cycle,  $\pi(G)$  doesn't have a Hamiltonian cycle. And that's why I can't succeed in convincing you in answering both questions. Yeah.

**AUDIENCE:** Is there a formal way to say what it means to be zero knowledge? Like, [INAUDIBLE].

**Yael T. Kalai:** Yeah, good. Good, good, good. So the question. Yeah, actually, I planned not to go into the definition, but it's so beautiful, so I will.

Yeah. So the question was, what is the definition actually of zero knowledge? And the definition of zero knowledge says that, what does it mean, you don't learn anything. So the definition says that-- so this is the definition of zero knowledge.

What it says is, the verifier-- actually, it says, it's stronger. It says, for every verifier, not just the honest verifier, even malicious verifier. You can talk about honest verifier zero knowledge, the honest verifier, who just does what it should do and just listens, doesn't learn anything.

But actually, there's a stronger notion of malicious. So let's say even for a malicious verifier, you want to say what he learned. He could have simulated on his own. So the definition says, for every-- so you-- OK, for every verifier, if you look at the transcript-- and for every  $x$  in  $I$ , for every  $x$ , if you look at the transcript between the honest prover and the verifier, the prover has a witness, but the verifier doesn't.

What he learned from this transcript or what the verifier maybe-- sorry, let me write it differently. Let me say the view of  $v$  star while interacting with  $p$  and input  $x$ . So whatever  $V$  star learned, he could have simulated on his own.

So it says there exists a simulator. And this is efficient, an efficient simulator, such that the simulator efficiently can generate this transcript on its own if he has  $V$  star. He needs  $V$  star because, of course,  $V$  star depends what he sends and so on. But he could generate it on his own. Yeah.

**AUDIENCE:** Can the stimulator depend on  $V$  star if we're then using [INAUDIBLE]? So then, is it the same simulator for everyone? Or is it--

**Yael T. Kalai:** Oh, my God, there are so many-- OK, there's so, so many notions of zero knowledge. So for example, first of all-- I'll answer your question. The question was, is simulator only gets  $V$  star as Oracle access? Does it get an input? Is it universal? Is it for there exists a simulator for every  $V$  star, or is it for every  $V$  star there is a simulator? Moreover, let me continue. Is  $V$  star all-powerful? Is  $V$  star supposed to be PPT or a poly size?

And the answer is, there's many definitions. But yeah. So originally, all the protocols we had used the verifier as a black box. However, in 2001, I think was it, Boaz Barak was the first to construct non-black box zero knowledge, where the simulator used verifier is a non-black box. And it was a really breakthrough result because what he managed to get is a new zero knowledge proof that we didn't know of before. It was kind of constant round.

Now you can say, wait, this is also constant round. What do you mean we didn't know before? But this is only soundness half. Turns out, if you want negligible soundness, you need to repeat this protocol sequentially. If you repeat it in parallel, it's not necessarily zero knowledge. Actually, yeah, this one actually is not zero knowledge if you repeat it in parallel. We'll talk about that.

And so, using non-black techniques, he managed to show the first kind of constant round zero knowledge-- public coin zero knowledge proof. So there's many, many variants is the point. But this is a common variant to think about.

And also, so this is kind of poly size, or  $V$  can have auxiliary input. So for every  $x$  in [INAUDIBLE] or  $V$  star is poly size. So we can have auxiliary information about the  $x$ , even if it has auxiliary information about  $x$ , still, so  $V$  star is-- may have auxiliary information about  $x$ , still the two are indistinguishable. Yeah.

**AUDIENCE:** What if the prover for the commitment where he created matrices that ensure that there's a cycle in that matrix that commits to that?

**Yael T. Kalai:** Wait, again, what if the prover commits to a cycle or no?

**AUDIENCE:** He commits to a cycle, but the cycle is not necessarily with--

[INTERPOSING VOICES]

**Yael T. Kalai:** It has nothing to do with  $G$ . Just commit to any cycle.

**AUDIENCE:** Yes.

**Yael T. Kalai:** OK, good.

**AUDIENCE:** He commits to a cycle.

**Yael T. Kalai:** OK.

**AUDIENCE:** [INAUDIBLE]. And he'll be able to [INAUDIBLE] it's a 0, right?

**Yael T. Kalai:** Yeah, he'll be able to-- exactly. He'll be able to answer 0, but he won't be able to answer 1 because that cycle must touch an edge. Because-- sorry, that cycle must touch a non-edge. Because if it only touched edges, there is no cycle. So that cycle, you give your  $\pi$ .

Now, the prover can-- the cheating prover can give whatever  $\pi$  he wants. But the point is that the graph, the permuted graph, doesn't have a Hamiltonian cycle, because if  $G$  doesn't have a Hamiltonian cycle, the permuted  $G$  also doesn't have a Hamiltonian cycle. So this graph doesn't have a Hamiltonian cycle, and that means that the cycle that's here, I don't care how you generate, whatever the cycle was, it must touch a non-edge.

**AUDIENCE:** [INAUDIBLE]

**Yael T. Kalai:** So the Hamiltonian cycle says, there's an edge between  $i_1, i_2$ . There's an edge between  $i_2, i_3$ . One of them cannot be edge here. It can't be that all of them are edges here because they're-- so because it doesn't have a Hamiltonian cycle. So one of them cannot be an edge here.

And now, what the verifier is asking from the prover, he looks at all the  $i_1$  and  $i_2$  all the  $i$  and  $j$  that do not have an edge. And he asked them, open that edge. I want to see that it's 0. I want to see that I gave you  $n^2$  safes. So for every  $i$  and  $j$  there is a safe. And it should be 1 if there is an edge, and 0 otherwise-- I mean, if there is an edge in the Hamiltonian cycle and 0 otherwise.

Now, what the verifier is asking the prover, for every  $ij$  here that does not have an edge, I want you to open the safe corresponding to that edge, and I want to see a 0 there. If there's a 1 there, I'm not happy. I'm going to reject you. Because this, there should be 1's only on the Hamiltonian cycle. And these are the only places that have an edge in your graph.

But the problem is, on one of these  $ij$  that don't have an edge, you'll see an edge because the Hamiltonian cycle is not all on edges. And then I'm going to catch you. Yeah? Great. Any more questions? Yeah.

**AUDIENCE:** I'm just curious. Is there also a notion for a proof of knowledge about--

**Yael T. Kalai:** Yeah.

**AUDIENCE:** --just not just edge, so a slightly stronger that requires the prover to actually know the Hamiltonian cycle? So here the proof, would the second part say that, that only guarantees that if it's G is not a-- doesn't have a Hamiltonian cycle, that the prover cheat, prover cannot compute this [INAUDIBLE]. Isn't necessarily a converse, is it possible for the cheating prover to not know the Hamiltonian cycle, but still contains the verifier?

**Yael T. Kalai:** Good, no.

**AUDIENCE:** Where the G is--

**Yael T. Kalai:** Good. Great question, great question. So the question was-- or let me just answer the question. The question was, wait, we just proved here soundness, that if  $x$  is a NAND language, on one of the answers you'll fail. But actually, doesn't this protocol give you something stronger, was the question, and the answer is yes.

So what do we mean by something stronger? Let's say  $G$  does have a Hamiltonian cycle. Doesn't this protocol tell us, not only that  $G$  has a Hamiltonian cycle. If you succeed, let's say you succeed probability 1 or more than half. This protocol tells us not only that  $G$  has a Hamiltonian cycle, it means that  $P$  must know the cycle. You can actually find-- you can extract the cycle from  $P$ .

And the answer is yes. You can actually-- this is what's called proof of knowledge. It means that  $P$  knows. Now, you can say, what does it mean that  $P$  knows? What are we-- this is a theory class, mathematics. What does know mean? So what we mean by  $P$  knows, what we mean is that we can actually efficiently extract the cycle from  $P$ .

And how do we extract the cycle? It's actually very easy. We ask him. He committed to something. He knows how to open both. That's an assumption. So first we'll ask him, open only the cycle. And then give me the  $\pi$ . Once I know the  $\pi$  and the cycle, I can undo-- I know what  $\pi$   $G$  and I can see-- I can actually find the Hamiltonian cycle.

So it's more than just a soundness. It's a proof of knowledge.  $G$  must know. I can actually use him to extract the Hamiltonian cycle from him. Any other questions?

So let's take a five-minute break. And then, after the break, I'm going to show you why this protocol is not secure in the Random Oracle Model. But we'll fix it. Don't worry. OK. Let's take a break.