[SQUEAKING] [RUSTLING] [CLICKING]

ZHENGZHONG JIN: So OK. So let's start. So I'm Zhengzhong Jin. And so today I will give the lecture about SNARGs for Batch NP or batch argument.

So this lecture is about batch argument. And it has many different names, such BARGs. Or you can also call it SNARGs for Batch NP.

So batch is a very basic concept in computer science. So basically if you have multiple items and you want to collect them together, then you can call this collection a batch. So there are similar ideas in, for example, in programming. So there are concepts like batch processing. And so it's a very common concept.

So in this lecture, we'll talk about the batch argument. So it's a notion of batch in the proof system. So firstly, let me define what is a batch argument. So definition-- so a BARGs-- a BARG is a tuple of algorithms.

So it consists of a generator and the prover and a verifier algorithm with the following syntax. So at the very beginning, the generator will generate some series. So it will take some security parameter as input and generate some series. And both the prover and the verifier is given access to these series throughout of the protocol.

So instead of proving one statement at a time, the batch proof allows the prover to convince the verifier that a batch of the statement is true. So for example, so let's fix a NP language L so that L is defined as follows. And the prover tries to convince the verifier that a list of instances from x1 to xk. So the prover has this list of statements, and he wants to convince the verifier that all of those statements are in this list MP language L. So this is the statement that the prover wants to convince the verifier. And the proof is a single random message, pi. And also, the prover also has the MP witness for this x1 to xk. So let's denote the NP witness as w1 to wk. So those are the NP witness. OK.

So now, so we say such a BARG is succinct if it satisfies the following property. So succint is defined as the size of the proof is smaller than the total length of the witness. So can anyone see why we require this?

AUDIENCE: [INAUDIBLE]

ZHENGZHONG JIN: Yes. Yes. Right. Exactly. So we require this is because a trivial proof system, we could have the prover to just send over the NP witness to the verifier. Then the proof will be k times w. So [INAUDIBLE] this requires this.

And as any proof systems, we require the completeness and the soundness. So completeness requires, for any honestly-generated proof, it will always be accepted. And the soundness is defined as follows. So the soundness says, if the statement is not true, then any cheating prover shouldn't find a cheating proof. So it means, so if this is not true, then for any non-uniform PBT cheating prover, we have this property.

So the cheating prover is given the CRS signed by the CRS generator. And we require it to send a proof pi star. So pi star is the cheating proof. And we require the probability that the proof is accepted by this verifier is negligible. So I guess you already learned about the concept, the definition of the proof systems. This is just a recap of the definition.

So any questions about this definition? OK. Now, some of you might wonder why we care about such a proof system. Because such proof system is clearly a subclass. So such a statement is clearly a subclass of NP So why we care about them? So the reason is, as I just mentioned, so batch is a very basic concept. So this is just a notion of batch, the notion of batch in the proof systems.

And a similar concept has been studied in other areas in cryptography, such as signatures. So there is a signature scheme called aggregated signature. So it allows you to aggregate multiple signatures together to a short signature. Then you only need to verify the short signature to ensure all the signatures can be verified.

So there is also a similar notion studied in interactive proofs. And it's called batch interactive proof. And so such a scheme, such a BARG also has many applications. So in the next lecture, Yael will tell you how to use it to build SNARGs for all P. And in fact, there are many constructions of SNARGs for all of P up to now. So from various assumptions and-- there are different construction, but all of those constructions use these BARGs as a building block in it.

And indeed-- yeah.

AUDIENCE: Is there any hope for statistical sound BARGs, or is that a possibility?

ZHENGZHONG JIN: That's a good question. So there is a recent work shows, if you can get statistical-- so you can get statistical BARGs, then you can build some witness, some statistical witness in distinguishibility. Yeah. So we don't know the exact answer yet.

And actually, there is a technical reason why we study such a BARG, why we study BARGs. So I will briefly mention it, but I will not go into detail of this lower bound. So this is the work by Gentry and Wichs. So they show that-- I will not describe exactly the impossibility result. So they show some impossibility result about SNARGs in general. So they show it is impossible to build SNARGs for any T hard language with proof that is significantly less than log T.

So what does it mean? So here, we say a language is T hard if it takes T time to decide this language. So for example, for general NP language with witness length w, then usually you will take 2 to the w time to decipher this language. So by this impossibility result, they are saying it is a very hard to build SNARGs for general NP language if the proof size is significantly less than w. So basically, SNARGs for NP is very hard to build.

And here, the impossibility only holds for very good assumptions. So they call those assumptions falsifiable assumptions. And basically, good assumptions, such as learning with error assumption, are falsifiable. So I want to mention this because, indeed, if you want to-- so if we consider BARGs, then the language we are proving-- sorry. So if we want to consider BARGs, then the language we are proving is the following language, indeed. So the language the prover tries to convince the verifier is the language L or product k times-- so it simply contains all of x1 to xk such that all of x1 to xk is in this language L.

So what's the time to decide this language? Do anyone knows?

AUDIENCE: Would it be T times the time to decide each of those statements?

ZHENGZHONG JIN: Yes. Yes. Right. Great. So that's correct. So to decide this language, we only need to take k times 2 to the w because you only need to decide whether each xi is in L or not. And if one of them is not in L, then you know this instance is not in this language. So the point is, if you take this T into this Gentry-Wichs impossibility, it shows that it is very hard to build SNARGs for a proof size significantly less than log T. But log T is roughly the size of w So it only shows it is very hard to build a SNARG for proof size significantly less than w. But this is succint enough.

So I mean, if we can build a SNARG with-- if we can build BARGs with proof size w, that doesn't contradict to Gentry-Wichs, but it is still succinct because w is already significantly less than k times w. OK. Any questions?

OK. So in this lecture, the main theory we want to prove is the following. So there exists a contraction of BARGs from learning with arrow. And the proof size is poly of-- sorry. So the proof size is dominated by the by the relation circuit size. So roughly, you can think, if we consider 3 SAT instance, then almost this this C is almost the same as the length of w because the circuit is shallow. Yeah. And the verification time-- so the proof size is almost close to this w because the log k term is small. Yeah.

And the verification time is-- yeah. So the verification time is k times 1 of the instance plus some additional term. So the additional term-- so the first term only depends on the total length of the instance, which is k times x. So because we have k instances and each of the instances has length x, so the additional term does not depend on the instance size, the total length of the instance size. It only depends on log k, the size C, and the lambda.

So in fact, there are more structure of the verifier algorithm. So indeed, the verification algorithm can be decomposed to two parts. The first part will only take all of the instances as input and pre-processing it to output some very small state. So after the pre-processing, the online phase will take the state from the pre-processing algorithm, and it will verify the proof in a very short time. So the online phase does not depend on k. I will come to this point later. So this is the main theorem we want to prove.

So next-- yeah.

AUDIENCE: So [INAUDIBLE]

ZHENGZHONG JIN: Sorry.

AUDIENCE: The [INAUDIBLE] of pi is poly [INAUDIBLE] circuits, size of circuits is like [INAUDIBLE].

ZHENGZHONG JIN: The number of lines?

AUDIENCE: The [INAUDIBLE] of pi. X of pi.

ZHENGZHONG JIN: Oh length of pi, yeah.

AUDIENCE: So you want it to be less than k times w. So this k, like much larger than the size of-- is k much larger than length of w?

ZHENGZHONG JIN: Length of w? Oh, right. You can think of k is a polynomial in w. Yeah. Usually, yeah. Because you want the prover to be a polynomial time algorithm. Then you can only consider polynomial k.

AUDIENCE: So if k is polynomial w, then this construction is only succinct if the polynomial in C is a small polynomial?

ZHENGZHONG JIN: Right. Yes. Yeah. Right. So in fact, there is a way to generically bootstrap this to remove the dependence on the size of the circuit to make it even smaller, to make it only depend on the witness size. But that will use the things in the next lecture by Yael. So you can-- yeah.

OK. So any questions? So next I will show you how to construct these BARGs. So if I directly show you the construction, it will be very complicated. So I will show you a first initial attempt to do the construction and gradually add the building blocks to the construction.

So our starting point is the Kilians protocol. Actually, it's a simplified version of Kilian.

So initially, the prover has k-- so it has k instance and the witness. So w1 to wk-- sorry-- x1 to xk. And the prover is going to use PCP to generate the PCP for all of those instances. So he applied the PCP prover algorithm to it and obtained a PCP proof, which is a long string. So this is the first PCP proof for x1 and w1. And he does the same for the second instance and witness, et cetera-- so PCP k.

So do any of you remember what is PCP? Or do I need to recall it? So a quick recall what is PCP-- so PCP is a-- so let's say PCP for NP. So PCP for NP is a pair of algorithms. So the prover has some x and a w, which is the NP instance and the witness. And he can generate a long proof, long PCP proof. So this is the prover algorithm for PCP. And there is a PCP verifier.

So the verifier will read-- so this PCP string is some binary string. So it contains 0 and 1s. And the verifier will sample some random coordinates, Q. So let's say the size of the PCP is L, and Q is a subset of indices.

So the PCP verifier will check. It will take the instance x and the query Q under the PCP in the subset Q. And then it decides to accept or reject. So that's the PCP. And we require the-- so the combinant says, if x and w is in the language, then the proof should be accepted by the verifier. And the soundness says, for any x-- so if x is not in L, then for any PCP string, PCP star, we have the probability that the verifier accepts is bounded by some constant.

So the randomness is over Q and the-- is bounded by some epsilon. So here we can always do some parallel repetition to amplify the witness-- sorry-- to amplify the soundness. So if we do the parallel repetition lambda times, then any-- so we can amplify any constant soundness error to lambda, to epsilon lambda. So then we can set it to be 2 to the minus the order of lambda.

And the cost of doing this is to blow up the size of the query by lambda. So we can always assume with all the generality that there exists a PCP for NP, where the query complexity is roughly lambda times poly log C. And the soundness error is 2 to the-- roughly 2 to the minus lambda. So we will use this kind of PCP with this query complexity and the-- yeah.

AUDIENCE: I mean, there are PCPs with the size of QVO of lambda, right? Do we need the poly log C term?

ZHENGZHONG JIN: Oh, you mean this term?

AUDIENCE: Yeah.

ZHENGZHONG JIN: So that's a good question. So it's possible that you can get rid of the-- so for example, if you have a very good query complexity, then maybe you don't have this term. But in this work, I only-- so we will need some additional structure about the PCP. And I only know-- at least I know it holds for the algebraic construction from this written numeric code. But I don't know if it holds for general other constructions of PCP.

So all right. So I'll use this PCP. So now we have k PCP string, and we have our verifier. So--

AUDIENCE: Can you remind us how big [INAUDIBLE] are?

ZHENGZHONG JIN: Good question. So let's say this is a PCP for NP language L, and the NP language is defined in this way. So it has a relation circuit C. And the length of the L-- so length of L is some poly in C, the circuit size. Again, it's possible that you can use some more efficient PCP construction to-- I think there exists a PCP such that this L is almost near linear in the size of the circuit. Yeah.

So OK. So we will use this PCP, and so we'll use some hash function. And we will put the hash key in the CRS. So the CRS will contain the hash key. Let's say some hash key. And we are going to look at-- so this PCP are-- so they are binary strings for each coordinate. They are binary values in each coordinate. And we are going to use this hash function to hash each column of this PCP-- of those PCPs.

So that means I will take the first bit of each PCP string and concatenate them together and hash them using a hash function. So I apply a hash function here, so to the first column. And I get some hash value, h1. And I will do the same thing for the second column. So I can get h2.

So for each column, I will hash it. And so basically, for each column I will hash it to get some hash value. So if the length of the PCP is L, then I get L hash values. So under this variant of Kilians protocol, I work as follows. So I first will send all of the hash values, from h1 to hk-- sorry, h1 to hL for each column. And then I will have the verifier to send one single PCP query.

So a PCP query, remember, is a subset of indices. So this is one single PCP query for-- so I mean, it's a PCP query for one of the state, one of the PCP string. And finally, I will open the query column for these hash function, for those hash function.

So what do I mean? So this is so you can view this PCB string as a matrix. So this matrix has L columns and the k rows. And for this one, single PCP query, it will query some of the positions. So suppose it queries this second position and the last position. Then I will open the corresponding column for this query position.

So let's denote the first column as a column 1 and the second column as a column 2, et cetera. Then the third-round message will contain all of the column Q, where this Q is in this query, is in this subset. So this is essentially the same as Kilian, except that I don't use a Merkle tree to commit them. I use a hash function to directly hash each column separately.

Yeah, question?

AUDIENCE: Do we use Merkle tree to commit to the h1, h2?

ZHENGZHONG JIN: h1? So at this point, I do not use the Merkle tree to commit them. So I directly send them to the verifier. So OK.

So I send them in the first message. So by sending them, I already fixed the PCP matrix intuitively. So then after I fix them, I have the verifier send the PCP query. And I have the prover to answer the same PCP query for all of the PCP. Yeah.

AUDIENCE: [INAUDIBLE] I follow, the third message here is the point that we [INAUDIBLE], like [INAUDIBLE] OK, right?

ZHENGZHONG JIN: Yes. Yes. Good. Good observation. Yeah. So this is the initial protocol.

So right. Yeah.

AUDIENCE: Does the same thing, all those [INAUDIBLE], is that small enough for the verifier?

ZHENGZHONG JIN: Oh, sorry? You mean k, this one.

AUDIENCE: Yeah, when you open the column, is that's small enough for the verifier?

ZHENGZHONG JIN: Is that enough for the verifier?

AUDIENCE: No, small enough, small enough.

ZHENGZHONG JIN: Yeah. So good. So the size of Q is only-- you can think it's almost a poly log in the size of the circuit C. So this Q is very small, you can think. So for each row, I'm going to open-- for each row, I'm going to open Q positions. So I have k rows. So the third row on the message is almost k times Q, which is almost k. It grows with k. Yeah.

So what does the verifier do? So the verifier, it will first-- so the verifier will first verify the opening is consistent with the hash. So it will verify hash. And the verification process is-- the verify just recomputes the hash for these queried positions and see if they are consistent with the first round message. So it will hash the-- use some hash function applied to the query position and then check if it is the same as the hQ for each of the Q in this query, the subset.

And secondly, the verifier also need to verify the PCP. So the verifier will run the PCP verification for each of the statement, from x1 to xk. So it will run the PCP verification for xi and for the same query Q and the PCP answer. PCP answer for the i-th row is the-- for i-th sigma xi is the i-th coordinate of this column vector. And you take the collection of all of the such position in Q. Yeah.

So that's the initial protocol. But this protocol is an interactive protocol. So to build a SNARG, we need to make it non-interactive. So I think the last lecture before Thanksgiving, so that lecture is about the correlation-intractable hash and how to use it to build a non-interactive proof from interactive proof. So let me recall what is a correlation-Intractable hash.

So correlation-intractable hash-- so I will call it a CI hash, a CI hash for a relation L-- for a relation-- for a relation R. So R is a subset of x times y. So x here is the-- so in some subset of x and y is a tuple of algorithms. So the tuple consists of a key generator and a CI hash. And it satisfies the following property. So for any. non-uniform PBT adversary, we need the following condition.

So the adversary is first given the hash key generated by some-- let's say this is CI hash key generated by the key chain algorithm. And the adversary is required to find some input x such that the pair of the x and the hash of-- and the CI hash of x, so the pair of them is in R. So this event is bounded by some negligible amount. Yeah.

So this is what you learned from the last lecture. So this is just a recap. And yes. And also recall, so given such a hash function, how do you compile an interactive proof to a non-interactive one? So the idea is, you start with some-- so given any three-round public coin interactive protocol. So public coin means second-round message is a uniform random string. So suppose you have such a protocol. Then you can compile it using this hash function to obtain a non-interactive one.

So the proof consists of the transcript of the protocol. So the proof is just alpha and beta. So beta equals the CI hash of the CI hash key and the alpha and the gamma.

And we need to recall, how do you prove the soundness of this non-interactive protocol? So to prove the soundness, how do we use this correlation-intractability to prove the soundness? So to prove the soundness, you need to, firstly, define a relation. And the relation is the relation R. So it consists of all of the first-round and second-round message, alpha beta, such that there exists a pseudorandom message that can make the verifier accept.

So you define this relation for the interactive protocol. And then, so for any proof, for any-- so to prove the soundness, we need to consider this x, the case when x is not in L. And yes, for x is not in L. And suppose any cheating prover-- let's say we have a cheating prover with an output, a cheating proof. So let's say the cheating prover outputs a cheating proof, alpha star, beta star, and gamma star.

And if the proof is accepted, then this means-- so if this proof is accepted, then this means this beta star equals the CI hash of alpha star under the CI hash key. So I ignore the CI hash key here. And also, this-- yeah. And also the original verification passes. So here is a different verification, so under the original verification passes,

So this means this alpha-- alpha star and beta star is in this relation. And this is how you prove the soundness because-- so if a cheating prover exists, then you reach a contradiction with the correlation-in tractability because the cheating prover also outputs the input and output pair such that this such that this beta equals the hash of the alpha. But it satisfies this relation, R.

So this is from the last lecture. Any questions? So here I want to emphasize here that there is a condition in this proof. So the condition is-- So we need this for such, say hash 2 exists, we need this relation R. R, It needs to be sparse.

So what do I mean? Or evasive-- so sparse here means, for any fixed R, alpha, the fraction of the beta that can make this pair, alpha beta. It satisfies this relation is a negligible fraction. So it means, all of beta such that alpha beta is in this R. If you divide it by all the size of y, then we need this to be negligible.

So this condition is crucial for the CI hash to exist. The reason is-- do anyone know why this is the case? So let's consider an extreme case. So let's say if there exists-- let's consider extreme case. So let's say if there exists some alpha, alpha 0, some fixed alpha 0, which is in the input space, such that for all of the beta, this alpha and beta is alpha 0, beta is in this R. Then it means that then for any hash function, it cannot be correlation-intractable for such R, because any hash function will output something, some beta, and it will satisfies this relation.

So this condition, so at least we need this fraction to be less than 1. The problem is, if we want to apply the CI hash construction here, it's not clear whether we can define a relation that is sparse. So let's modify the construction using this CI hash paradigm So we will replace the second message, Q. So instead of having the verifier send the Q, we have the prover compute this Q by itself using the CI hash. And so now we apply the say hash to all of the first round message h1 to hQ-- sorry, h1 to hL. And then we put the CI hash key inside of the CRS.

So next I will define the relation R and try to prove the soundness.

So, yeah?

AUDIENCE: Can you say what changes you made again?

ZHENGZHONG JIN: Yes. Good. So the changes is instead of having the prover to send a message to the verifier and have the verifier send back a challenge, I'm going to have the prover not send this h1 to hL. So the prover directly applies a CI hash to h1 to hL and obtain some query. So the prover can proceed to the third round and finish the third round.

So the proof will consist of the transcript of the protocol, which is all of the h1 to hL and the third-round. Message. OK. Yeah?

AUDIENCE: What assumption are we making for-- we need the output phase of the CIh to be subsets that the verifier would query or something?

ZHENGZHONG JIN: Right. Good question. So, in fact, we need the PCP to be a public coin. So, yeah. That's the implicit condition. So ir the general case, the PCP query may not be uniformly random. So in fact, there is some PCP, some query sampler that takes take some uniform random string, and it will compute the actual PCP query. So, yeah.

AUDIENCE: So we're also kind of assuming that the output of CIh is kind of uniformly distributed, or pseudorandomness?

ZHENGZHONG JIN: Yeah. I mean, so in the general case-- so in fact, this is just some succinct way to describe how you generate the query. So in fact, the PCP query is generated by using some query generator or some sampler. So the sampler will take some uniform randomness R and generate the PCP query.

So what you do is you take the output of the CI hash as the randomness, and then you generate it from it to get a PCP query. Yeah.

OK, so next I will try to prove the security of this construction, try to prove the soundness. So completeness is easy to show. You just can do it yourself. I will only prove the soundness here.

AUDIENCE: [INAUDIBLE]

ZHENGZHONG JIN: Yeah.

AUDIENCE: For [INAUDIBLE] whatever indices for PCP we generate this probability 1, [INAUDIBLE] verify those [INAUDIBLE] So for the PCP itself, is complete this one?

ZHENGZHONG JIN: Oh, right. Yes. Good. Yeah. We assume that there-- yeah, we choose a PCP with perfect correctness. Yeah. Yeah. Good question.

So let's first try to prove the soundness. So the initial attempt is to define a relation R similar to the relation R as before. So we just define it as the relation R consists of all the pairs of h1 to hL and the query set Q such that there exists a pseudorandom message. So gamma here is the pseudorandom message, such that the verifier accept.

So this is the initial attempt to prove the soundness. So we'll do the same thing. So this R is the same as the relation R we defined for the CI has here, except that I just replaced the alpha, beta, gamma with this protocol. But the issue is, with this relation, it's not clear whether this relation is a sparse relation.

The reason is that this hash function must be a compressing hash function. So if we choose a compressing hash function, then for any hash value, h1 to hL, there might be multiple preimages of the PCP. So for one PCP, so let's say we have one preimage of the PCP, which is PCP1 to PCPk. Then for those PCP, you may have a Q that can make them accept. You may have a one Q to make them accept. But for other preimage of the PCP, it may have another Q that can make the query accept.

So now this relation will take the union of all of those PCP queries. So it's not clear if this relation has sparsity less than 1. Yeah. So what we will do is we will choose a special hash function. And so this special hash function is still compressing, but it will show us some information about its preimage. So now I will define this a special hash function.

So the hash function is called somewhere statistical binding hash. I will call it SSB. So it's a SSB. So such a function is defined as follows. So SSB is also a tuple of algorithms. So it has a key generator and some SSB hash algorithm with the following syntax. So the key generation algorithm will sample some key. And the key generator will will take some security parameter lambda as input, and also it will take some index I as input, such that-- and then the SSB hash algorithm will use the SSB key and it will take some message, x1 to xn, as input and output some hash.

And we require the following two properties. So the first property is called the key indistinguishability, or key hiding, so key indistinguishability. So it says that for any two indices, so i1 and i2 in this n. So the key generated by using the first indices is indistinguishable with the key generated by the second index.

And the second property is called statistical binding. So statistical binding basically says the hash function is binding for the position i. So it says for any x1 to xn and x1 prime to xn prime. So if the SSB hash of them is the same, then we have this xi equals xi prime. So it says the hash value will bind the value at the i-th position of its input, xi. So here the k is generated in this way for this index i.

Any question? Yeah.

AUDIENCE: Could you explain why that relation might not be sparse?

ZHENGZHONG JIN: Oh, right. So it's not clear whether this is sparse or not. The reason is that because this hash, so h1 to hL, are the hash values. So initially, we have this PCP1 to PCPk, and we hash them to a succinct hash value, h1 to hL. But the size of the hash is much smaller than the preimage. So for a fixed h1 to hL, there might be multiple preimages.

And for one per image, it may have some-- for example, for this fixed one preimage, it may have some query set Q that can make all of the query be accepted. But there might be some other PCP preimage for the same h1 to hQ. So let's say it's some PCP1 PCPk prime. And they may also hash to the same hash value.

So if this is the case, then for those set of PCP1 PCPk prime, they may have some other query set Q prime that can make each of the PCP1 prime to PCPk prime be accepted. But if this is the case, then this Q and Q prime neec to be-- so they are also in this relation. R. So if this is the case, then both of these, this h1 to hL-- hL and Q is in this R. And the Q prime is also in R.

So you can do this do this argument for each of the preimage of this H1. To hQ. And the issue is that there are many, many, many preimages. So it's very compressing. So the number of preimages is much larger than the size of the Q, all possibilities of Q. So, yeah.

AUDIENCE: In order for a cheating prover to take advantage of this [INAUDIBLE], they would have to find a collision like that. So why are we worried about this?

ZHENGZHONG JIN: Good question. So for example, if you take a specific hash function, it's not clear whether the prover can leverage this kind of attack. But it's also not clear how to prove the soundness. So, yeah. Sorry.

OK. So next I will show you how to build such a-- yeah, any question? OK. So next I will show you how to build such a SSB from FHE. So the construction is very simple. So do you remember what is FHE? So FHE is some public key encryption, where you can, given a ciphertext of M you can homomorphically evaluate on the ciphertext for any function F and to get a ciphertext of F of M. So that's the FHE.

So to build SSB we do the following. So from the FHE-- so for the key generation, so for the index i, we're going to generate a FHE ciphertext of i. And we define this ciphertext as the SSB key And to use this ciphertext to hash x1 to xn, the idea is we're going to homomorphically pick the i-th position under hood of FHE for those x1 to xn.

So what we will do is we will do the homomorphic evaluation of some function F. And F has x1 to xn hardwired in it. And we'll do this [INAUDIBLE] on the ciphertext of i. And this is our hash value.

So here, this f is defined as follows. So F x1 to xn, given index i, it will output of i as-- i as coordinate. So intuitively, we just pick the xi under the hood of FHE. So here, so this is a hash function because the size of the hash value is just a ciphertext. So the size of the hash value is poly lambda.

And the key indistinguishability follows from the security of FHE because the index i is hiding. So the second property, this statistical binding, you can show it directly because the hash value contains the homomorphic encryption of xi. Any questions?

So later, we will use this SSB, but we will use an additional property. So this property is called extraction. So intuitively, this hash value, it contains the ciphertext of xi. So in fact, it contains FHE. And if you know the secret key of FHE, you can even decrypt this i from the hash value. So we need the extraction.

So the extraction property says that there exists an extractor. So extractor is a polynomial time algorithm. And moreover, so in the key generation algorithm I will add a trapdoor in it. So the key generator will also output some trapdoor. And there is some extractor such that the extractor will take the trapdoor as input and also the hash value as input, and it will extract something.

And we further require the extractor has correctness, which means, for any x1 to xn, if you hash it under a P k-- so let's say S. S is the hash of the x1 to xn. Then the extracted value is consistent with the xi, with xi. So this is the extraction correctness-- so if the key, if this hash key is for the i, s position, if the hash key is generated for the i, s position.

So now we will go back to the construction. So now we have this SSB hash from FHE. We'll go back to this construction and modify this hash function we use. So the hash function we use, we will use a SSB hash. And the SSB key, we will move it to the-- we'll add the SSB key to the CRS. So this is SSB key.

So now we will use this SSB hash to hash each column of the PCP matrix to this PCP matrix. And all the other things are the same as before. Any questions? OK, yeah, maybe we also need to-- when we verify the hash, we also use SSB to recompute each column and verify it.

AUDIENCE: [INAUDIBLE], when you're doing the SSB hash, what i are you putting in? Or you just don't even need one?

ZHENGZHONG JIN: Right. Exactly. So in fact, in the actual construction, in fact, you don't even need the i because this FHE ciphertext, in the actual construction, is a pseudorandom. But you can put any index i there. Yeah.

So now if we use the SSB hash to hash each column of this matrix, what will happen? So let's say if the SSB hash is generate-- the hash key is generated for some index i star. So then intuitively, this means this SSB is binding for the i, s row of this matrix. The reason is that, for each hash value, we hash it by-- we obtain it by hashing some column of this matrix. For example, the first hash value is obtained by hashing the first column of this matrix. And now, if the SSB is binding for the i, s coordinate. So the i, s coordinate now corresponds to the i, s row of this matrix, the first coordinate of the i, s row of this matrix.

And the same is true for other columns. So for each column, we will bind the i, s position. And that means this SSB is binding for the i, s row, so for the i stars row. And intuitively, you can think h1 to hL will contain the full information about the i, s row of this matrix because the h1 to hL, they are just FHE ciphertext. So it contains the actual value of the i, s row.

Any questions?

AUDIENCE: How does that help make a relation [INAUDIBLE]

ZHENGZHONG JIN: Right. A good question. So let's continue this proof. So we want to prove the soundness of this SNARG. So how do we prove it? So to prove the soundness, we need to show for any x1 to xk such that x1 to xk is in L doesn't hold. Then no cheating prover can break the SNARG.

So we start with the statement x1 to xk all in language L doesn't hold. But what does this mean? So the key observation here is that, so if this doesn't hold, then there exists some index i. Let's say it's index i star-- some index i star such that this xi star is not in L. So it means that, if you want to prove the soundness, there is one statement xi star that is not in this language. And the goal is to just catch this i star. Then we can prove the soundness.

So we only need to use the one statement is not in this language. And that's how we use SSB. So remember, SSB can bind the i, s position, i, s row of this PCP matrix. So the i, s row of the PCP matrix is the PCP for xi. So that gives us soundness. So that's the intuition.

AUDIENCE: My belief is that the image that you're binding is the one that's not in the language.

ZHENGZHONG JIN: Good question. So the proof, so the formal proof goes like this. Good question. So the formal proof works as follows. So in the first hybrid h0, we have an adversary. So suppose we have some adversary, P star, that can break the construction, break the soundness of the construction. So what it will receive? So P star will receive the CRS, some CRS. And it will output some cheating proof pi star. And the verification of the proof will be accepted.

So the verification of the proof will be accepted. And this happens with probability greater than some epsilon. So in the next hybrid, I will-- so in the next hybrid, I know there is an index i star. And the xi star is not in the language. So this i star is fixed. So I can switch the CRS. So in the hybrid h1, I will switch the CRS to this CRS generated by SSB generation, using the i star.

So let's denote this by SSB. Sorry-- this to ki star, SSB ki star. And I can argue, if this is the case, then we still have-- so this new CRS, let's call it CRS star. So CRS star will contain this SSB ki star in it. And it will generate some pi, pi star. And it still has the property S. So this is greater than 1 minus negligible. So this is true because-- so this is just follows from the key indistinguishability because the key is hiding.

So now in this hybrid, we switch the-- so intuitively, you can think given-- so given any cheating proof, in fact, the i, s row of the PCP is already fixed. So next we will reach a contradiction using correlation-intractable hash.

So if the verifier is accepted-- so the claim is for any proof, for any cheating proof, the cheating proof has this h1 star to hL star and Q star and gamma star. So gamma star is the pseudorandom message I denoted as gamma star. So this is information threshold claim. And the claim is just if the cheating proof is accepted-- oh, sorry-- then it must be the case that this Q star-- sorry-- then it must be the case that this h1 to hL star-- Q star is in this relation R.

So R is almost the R above, but I will redefine a new R. So R, it will contain all of the h1 to hL and Q. So given h1 to hL, I can use the extractor to extract a PCP because that's from the extraction of the SSB. So this PCP is the PCP for the i star position. And then I define this R as follows. So I require the PCP is accepted for xi.

So in this definition of R, I first extract the i, s PCP. And then I check if the query Q can make this PCP be accepted for the i, s star statement, xi star. So any questions? Is this clear?

So intuitively, in the third round message, if the proof is accepted, then anything-- so in the third-round message, the prover needs to give some columns of the queried position. So for all of those columns, if you look at the i star row of it, it must be consistent with the extracted PCP because we have the SSB extraction correctness. So we define the R in this way. So in fact, it's the same as the R above.

So now to prove this claim is just-- the proof of this claim is just the argument I set up just now. So for any cheating proof, if the verification of the proof is accepted, then from the correctness of the extraction-- from the correctness of the extraction-- maybe let me write it. So if pi star is accepted, then from the SSB extraction correctness we know this PCP i star. So the PCP i star, we extracted from this h1 to hQ-- sorry, h1 to hL.

So let's say it's some PCP i star-- PCP star. If you look at the Q's coordinate, then it must-- OK, so let me write in this way. The column star-- so column star. are-- yeah Q-- so where the third-round messages are just the columns.

So that's the soundness proof. So that's the proof of the claim. Yeah, any question?

AUDIENCE: In the hybrid, how do you identify which i star because you need to know which statement was right [INAUDIBLE].

ZHENGZHONG JIN: Right. Good question. So this soundness proof is non-uniform. So it's a non-adaptive. So the soundness proof starts with a fixed x1 to xk. So for any fixed x1 to xk, you know there exists an i star. That xi star is false. So there exists a fixed i star such that xi star is false, is not in L.

AUDIENCE: So that's a given?

ZHENGZHONG JIN: Yeah, so that's given. So you can think it's a given to the reduction in a non-uniform way. Yeah.

So this is from the extracted. So this is from the SSB extraction correctness. You have this. We have the consistency between this PCP extracted and the proof in the third round. And if the proof, if this cheating proof, pi star, is accepted, then we can use the fact that this pi star is accepted. So if it is accepted, then we know this Q. This Q can make this PCP star be accepted. So then that completes this claim.

And finally, from this claim you can use the correlation-intractability of the CI hash. So pi star-- so if the proof is accepted, then this is the extracted PCP from here. It will make this a PCP verification accepted. Yeah. OK. Let's have a break. Yeah.