Hello, everyone.

I'm very happy to be here to present our work about ObliVM.

That is a programming framework for secure computation.

And this is a joint work with our teammates, Xiao Wang, Kartik sitting there, and Yan Huang, and Elaine Shi.

And so, I'm very happy that my previous talk have intensively explained what is secure computation, aka, secure function evaluation.

I just give you a brief motivation that,

for example, Sheldon and Amy here wants to find a good partner for each other, and they all believe in genomic dating.

And so, they can run an analysis to see whether they match or not.

So, the key security question is they don't want to leak their sensitive genomic data to each other.

So, and we have been talking that secure computation is a good solution to solve this problem.

For example, we can abstract this problem as following,

given two parties Alice and Bob, having their secret input x and y,

so, they want to jointly compute a public function f(x,y), get some results.

And then, the computation over these two secret inputs should not leak anything other than the result z to each other.

So, this is secure computation.

And so, you already know there are Yao’s garbled circuits, and there are GMW protocols that can solve the secure competition.

So, what is our focus?

Our focus is that how we can make secure computation really practical?

For example, one developer wants to develop a secure computation application.

So, they don't want to write in a circuit format.

They want to write in a C language, write in a Java or Python language.

So, there is a gap between the source program and actual secure computation protocol.

So, this is what our ObliVM framework do.

So, our ObliVM is a tool to translate source programs into the actual secure computation protocols.

So, OK, this is an overview.

And then what is the main question?

As Kartik just introduced, a question is the gap between the programmers’ favorite model, for example, this is a Python like language.

And this is what the problem that is really want to write.

But actually, the most secure computation protocols are treating with these circuit models.

So, there is a big gap between the high level language programs and the circuit.

So, our question is how can we translate the left part into the right part, OK?

So, thank you very much Kartik. You have motivated this well.

So, the key challenge here is that how we can make the dynamic memory accesses not leak information.

So, this is not a trivial challenge.

But this is another trivial task.

So, actually our solution is that we want to translate the problem in a RAM model language into the oblivious counterpart.

So, here oblivious means that memory accesses and instruction traces do not depend on secret input.

So, in this way, the oblivious program can be easily transferred into a circuit.

So, we can see, in this transformation, the later part is relatively easy, and the first part is really challenging.

So, this talk will focus on the first part, and how to do that.

So, a very trivial, not very trivial, but a solution that was proposed first in 2013 was to use oblivious RAM.

So, for oblivious RAM, or ORAM, that can compel an arbitrary program into as a biggest counterpart and based on this idea last year we compile an arbitrary program into its oblivious counterpart.

And based on this idea, last year, we presented our SCVM framework that does this work to simulate generic ORAM.

And we can show we can achieve asymptotic performance gain over all previous solutions.

However, this solution is generic and easy to implement.

But the problem is that it might not be the most efficient way.

So, over the past few years, we have observed many customized secure computation protocols.

There is a long list of all this work and unfortunately I don't have space to list them all.

So, they are very very efficient, basically I can say they are more efficient than our last year's work.

But the problem is that they incur very high desire effort.

For example, we have talked to Nina Taft. She is our collaborator.

And we had a private conversation about their paper on private matrix factorization publishing in CCS 2013.

And she mentioned to us that it spent a team of 5 researchers about 4 months to implement everything.

So, that is basically more than 1/2 research a year. So, that was a big effort.

So, our question is that can we make it better?

So, can we build a generic framework, but also achieved customized performance?

So, this is about our work ObliVM to achieve this goal.

So, we want to allow non-specialties, so for example, non-cryptographer,

to implement some secure computation protocols while achieving the customized performance.

So, how do we do that?

So, the key idea is that within ObliVM, we provide several programming abstractions.

For example, I list many here, oblivious data structures, and MapReduce, and loop coalescing.

I will introduce one of them later.

But for more details, I will refer you to the paper.

And I also would like to mention that the GraphSC paper that was just presented by Kartik is also a program abstraction dedicated for parallel computing.

So, we provide program abstraction.

So, I will give you a intuition about what it is.

So, let's think about the distributed community.

我估计绝大多数人都听说过MapReduce

I think most of you should heard of this, the MapReduce,

这是谷歌于2004年在OSDI会议上发表的论文

that was published by Google in 2004 in OSDI.

And before that paper was published, parallel computing or distributed computing was considered a hard task.

But using MapReduce, the developer only needs to encode their computation into a mapper along with a reducer.

开发者不需要关注分布式计算方法 MapReduce框架会实现整个分布式计算过程

So, the developer doesn't need to worry anything about the distribution. But the MapReduce framework can do everything for you.

So, the development effort using MapReduce is very low, comparing to previous work.

So, we want to take a very similar approach here.

So, we want to provide some abstractions to allow programmers to encode their tasks into these abstractions,

so that they don't need to worry about the underlying cryptography primitives, while achieving the same performance.

So, here is our goal so so here is our solution so how can we provide these program abstractions so we want to - we want to implement a language support so for example we want to implement a new language for our developers so when T when developing this language we have two goals in our mind the first the first one is that using our language the expert our cryptographers can implement different different programming abstractions very easily and on the other hand for those non non specialists so for example they do not know very much about the problem of cryptography then they can use these abstractions well very easily to build their own applications so with this goal with these two goals in our mind and we our solution is to support new language features that was never supported before you buy any previous system and so I list a couple of them here and I will refer you to the reader for more details I can and further feed further we will open source our compiler here so that you can know even more details about the compiler so uh so I think these features are really cool for example we can use random time tandem functions so and to actually implement the biggest Ram protocols in in the sauce level rather than as a back-end primitive and ok so so so having all of these than what programmer can do for example they want to we want to implement a sparse graph algorithms then we have this arsenal about our programming abstractions then for example we want to implement a sparse directress shortest distance algorithm then we can peek oblivious data structure that is our previous heap and then you pick a loop called as you know abstraction to implement them okay then using these tools these tools we we actually implement three three different sparse graph algorithms and everything looks fine right but we have a very expected result is that that is our implemented algorithm made a theoretical breakthrough so for example all these three algorithms have better asymptotic complexity than the state-of-the-art so this is very surprising to us and and for more detail about all those algorithms I would also like to refer you to the reader okay so how do we actually do that so I will you know as I just promised several minutes ago that I will introduce you one program as direction that is loop coalescing so a big a big this is a this is a deep detail so insecure computation it is very challenging to implement a secret loop in the contacts in secure computation because the loop itself will be at an odd number of iterations of the loop really being information so here we allow to write the programmers to write upon the loop for example we can we allow the loop to have a secret art but we ask the programmer to provide a puppy pound after loop so so this is a nicely loop and the interesting thing is that in the inner loop from line for to line silent so the the pound M is not the pound for one pound for each iteration of the outer loop but that is a total bound for the entire execution after an acid after tunity loop so that we don't to repeat anything so for example if we provide a pound a natural way is to make n times M iterations but here we only make n plus M iterations and then our some hardware analyze these codn't and automatically transform that into the following one that that it looks like a state machine so that we do not incur any extra actual complexity after algorithm and so without dealing with all these techniques so what can we achieve so I I have mentioned that for the max factorization is spend about one more than 1/2 research a year so what can we do now using our PBM so that is very surprising we only do that in one day only only one PhD student can implement everything in one day so you may wonder whether it is efficient or not furthermore for example that I might not be efficient so but the thing is that we can achieve 10 to 20 times faster so that is even better so this is what we can do today so I so so I believe rpm is a very valuable thing for our secure computation developers ok so let's see and let's look at a detailed breakdown of the performance game so this is this is we we give the result of the Dijkstra's algorithm and more result will be shown in the paper so so a BVM although I focus on the programming language part of a BBN actually it has a has a highly optimized back-end called rpm as C that is already open source and you can visit our website for the link and we implement a state-of-the-art circuit circuit orang that is optimized that optimized for secure computation and this single thing we compare our system with path path previous solution publishing in CCS 2012 so the circuit itself can you can gain us about 50 times performance beat up and the language and compiler can give us about two thousand and two thousand five hundred speed-up and further we also do some other packin optimizations and those details can be found in our paper so there are also gives us seven times speed up so in total we achieve about 1 million times speed up so that's a huge huge thing so so give you a even more intuitive solution so in 2012 in that same paper so they did binary search over 1 gigabyte of data base and one only one single query spend about modern 12 hours to compute and what about today so using our oblivion free market only only requires seven point seven point three seconds for each query ok we also compare our ICBM in framework with with a you secure solution for example that claim playing program running on top of SATs it's machine or something like that and we compute the slowdown so so the Sotiris are very moderate so for example for the distributed GWS so the slowdown is only 100 and 113 so so so so we can expect that the in the future this number will be even slower ok so so there are several collaborations with that that our our PBM has been updating all these kind of these process and we with just one one one task of genomic and analysis competition again in March in March and that was two months ago and now we are to for future directions we are adopting our p VAR b vm framework to more cryptography tasks like follow homomorphic encryption and something more so thank you very much here is my talk now I like to take questions so the the example he gave right at the beginning of binary search what would that look like like what would the programmer have to write so we see example are you right in the beginning you gave an example of binary search and you said oh so so this is this is the sort of left the left part of this figure is what programmer need to write for the loop coalescing so you mentioned you mentioned the binary search example right yeah I think that's a good point so so the thing is that let me go back to there oh and here so so this is actually what programmer need to let me see if they can tell ya this program can be compiled you know oblivion compiler so the compiler will handle the memory access so yes the compiler can figure that out which part in you to be placing or amend which part do not need to do that okay so is it a potential use for oblivion to have it just output like C code instead of a circuit so that you can take basically an arbitrary program and turn it into one that doesn't have side channels oh yeah that's a very good question so yeah so it's a very interesting theme that our Oblivion compiler actually emits Java code yeah that and then the tower run in the Java code will actually produce the produced a circuit so I think that's a very good that's a very good question that and I think that will be an interesting future direction to see how our RP VN compiler can be customized to avoid such channels yeah thank you Oh you actually you can come to the microphone we don't have channel if you just compare to generic Iran and but you can alternatively just call our obliviously no structure we have binary search trees implemented in our oblivious in the structure and those if you use our obliviously the structure abstraction it will cut a log log n factor during the compilation change the compiler how do you compare efficient circuits thanks Helene yes until II and III thank our speaker again