非常感谢主持人对我的介绍

Thank you so much for that introduction.

我的名字是Marcela Hastings

My name is Marcela Hastings.

这是一篇SoK论文 我将讲解我们的调查结果

And this is a SoK paper. So, I'm going to be talking about the results of our survey.

我们考察的工具是安全多方计算

So, the tool that we were looking at was secure multi-party computation.

这是一个密码学工具 允许互不信任的参与方根据自己的输入联合计算任意一个函数

This is a cryptographic tool that allows a group of mutually distrustful parties to compute an arbitrary function on their joint inputs

计算过程不泄露输出结果以外的任何信息

without revealing anything beyond the output of the computation.

安全多方计算实际应用中最著名的实例是代买甜菜拍卖系统

So, the most famous example of this in practice is the Danish beet auction.

在这个场景中 售卖方是丹麦种田菜的农民

In this setting, we have sellers, who are Danish beet farmers,

而购买方只有一个 即丹麦唯一的一个甜菜加工公司

and a single buyer, the only beet processing facility in Denmark.

售卖方为甜菜出价 表示他们希望按照这个价格售卖甜菜

The sellers have prices at which they're willing to sell their beets.

而购买方希望得知市场出清价（即保证供求关系平衡的售卖价格）

And the buyer wants to find a market clearing price.

但售卖方不希望泄露自己的具体出价

But the sellers might not want to reveal their bids,

如果常年泄露出价 则其它人就会得知自己的甜菜种植能力和做生意的能力了

because it could reveal information about their businesses and their farms, especially over several years.

因此 他们使用安全多方计算协议 在不泄露售卖方出价的条件下计算市场出清价

So, they use secure multi-party computation to find an appropriate market clearing price without revealing the value of their bids.

另一个例子是波士顿妇女劳动委员会与企业的合作项目

Another example is collaboration with the Boston women's workforce council,

此项目研究员工性别是否会影响到其实际的工资

which is studying the gender and racial wage gap.

公司不希望、从法律角度也不能够对外泄露自己雇员的收入或金融类信息

Companies might not wish to reveal or might be legally prevented from revealing financial information about their employees.

但通过安全多方计算 他们可以在不给出具体数据的条件下 计算相应的统计分析结果

But by using secure multi-party computation, they're able to compute the relevant statistics and analytics without giving up that data.

所有这些例子都告诉我们 安全多方计算已经足够高效 可以在实际场景中得到应用了

Now all of this is to say that secure multi-party computation is a tool that's efficient and practical enough to be used in real situations.

然而 我们之前看到的所有实际应用实例中

However, many of the practical examples we've seen in the past

项目方都需要组织一个密码学专家团队 针对特定用例实现专用MPC引擎

have required a team of cryptographers to implement a special-purpose MPC engine for that specific use case.

如果想让MPC得到更广泛的应用 我们需要让不懂密码学的外行也能使用这一工具

And if we want to see MPC have more widespread adoption, we need tools that are usable by the layperson.

虽然二十世纪八十年代 密码学家就提出了MPC算法

So, although algorithms for MPC have been around since the 1980s,

但是一直以来MPC协议的效率都非常低 无法在实际中使用

they were assumed to be too inefficient for practical use,

直到2004年“公平参与”编译器的提出改变了这一现状

until the Fairplay compiler was presented in 2004.

“公平参与”是第一个通用MPC框架 可以通过MPC执行任意函数的计算

This was the first general-purpose framework, which could take an arbitrary function and execute MPC on it.

在接下来的10年 这一框架掀起了学者们针对MPC协议性能优化的浪潮

And it started the beginning of huge and fruitful field performance improvements over the next decades

直至现在 无论从算法角度还是从实现层面 MPC协议得到了巨大的优化

rapidly advanced the state of the art, both algorithmic and in the implementations.

在过去十年间 学者们提出了很多端到端MPC框架

And the number of end-to-end frameworks exploded in the past decade.

在本工作中 我们重点考察通用端到端MPC框架

So, in this work, we're looking at general-purpose end-to-end frameworks.

端到端框架的架构如幻灯片所示

And so, end-to-end framework looks something like this.

框架一般实现了两个阶段的功能：编译器、执行器

They have two phases, a compiler and a runtime.

这是因为大多数MPC算法只支持有限的原子计算 如模质数下的加法和乘法

This is because most MPC algorithms operate on only a limited set of primitives, like addition and multiplication module prime.

开发者很难应用这些有限的原子计算模型实现所需的计算函数

And it's hard for a developer to reason about interesting computations using that very limited model.

因此 我们重点考察包含一个编译器的框架

So, we looked for frameworks that have a compiler,

编译器的输入是用高级语言描述的函数 编译器会把函数编译成算法可执行的协议

which takes a function description in a high-level language, and converts it into a representation that can be executed by the algorithm.

接下来 执行器会具体执行编译好的协议

Then, the runtime phase is the actual execution of the protocol.

协议会在多个参与方上同时执行

It is run simultaneously by multiple parties,

每个参与方都以编译器的输出结果和自己的秘密值作为输入 计算得到函数输出结果

and who each take the output of the compiler and the private inputs of each party, and compute the function output.

当然了 不同框架的架构各不相同

And of course, the architecture varies a lot based on different frameworks.

但是它们基本都是这样的结构

But they're all taking this general shape.

在本综述中 我们想回答这样一些问题：

So, in this survey, we wanted to ask some questions about

目前都有哪些框架？

where we are, where the frameworks are?

谁在使用这些框架？

Who can use them?

这些框架是否可以实现实际的计算过程？

Are they ready for real computations?

这些框架是否可以支持所需函数的计算？

Can they express interesting functions?

这些框架是否可以在实际中使用？

And can they be used for real situations?

为了回答这些问题 我们调查了9个端到端框架和2个电路编译器

In order to answer these questions, we surveyed 9 end-to-end frameworks and 2 circuit only compilers.

我们记录了这些框架的不同特性

We recorded different features of them,

包括框架所实现的协议 框架所支持的数据类型和运算操作

based on the protocol that they implement, the data types and operations that they support,

以及其它一些实现的具体细节

and other details of the implementations.

我们通过多种可用性标准对这些协议进行评价

And we evaluated them on a variety of usability criteria.

为了收集这些数据 我们在每个框架上实现3个样例程序

In order to collect all this data, we implemented 3 sample programs in every framework.

我们把每个框架的完整构建环境及其对应的样例程序都放在了开源的代码仓库中

So, we collected our sample programs plus complete build environments for every framework that we used, and put them into an open source framework repository.

结合这些框架的使用经验 我们为各个框架补充了相应的文档

This also has additional documentation based on our experiences using the framework.

构建代码仓库的总时间约为750人小时

We estimate this repository took about 750 person hours to produce.

此仓库是开源的 我们仍然在积极地维护这一仓库

And it is open source and we're actively maintaining it.

非常希望大家能看一看这个代码仓库

So, I encourage anyone to check it out.

现在 我想简单介绍一下 完成这些工作后我们所得到的一些结论

Now I'd like to talk a little bit about what we found after doing all this work.

总的来说 几乎所有的框架做得都不错

Overall, almost frameworks are in good shape.

各种框架在不同的安全模型下实现了不同的协议

We have a diverse set of threat models and protocols that they implement.

根据具体用例的不同 框架也提供了一些协议和安全模型的选项

So, depending on their use case, there are different options.

我们可以在几乎所有的框架上实现我们的样例程序

We are able to implement our sample programs in almost every framework,

这意味着对于绝大多数场景来说 框架的高层语言具有较好的可表达性

which means that the high level languages are expressive for many situations.

总的来说 大多数框架都是开源的、可编译的、可用的

And overall, most of them were open source, and compilable, and usable.

然而 我们发现了两个重要的改进方向

However, we did find two major areas for improvement.

第一个方向是：框架的工程局限性较高 例如 框架的系统构建环境过于复杂

One is that there were significant engineering limitations, for example, complicated build systems.

第二个方向是：框架的可用性较差 其根本原因主要在于框架缺失相应的文档

And the other is significant barriers to usability, which are mostly rooted in a lack of documentation.

在具体讲解这些问题之前 我想先从宏观层面介绍我们的发现

So, before I go into those issues, I'd like to give you a higher level picture of what we found.

我们考察了9个框架 最下方是2个电路编译器

So, these are the 9 frameworks we looked at, and the 2 circuit compilers at the bottom.

大家可以从表格中看到框架支持的参与方数量、支持的安全模型

You can see the number of parties they support and the threat model.

我们有两种安全模型

We have two threat models here.

在半诚实模型中 攻击者会正确执行协议

in the semi-honest model, an adversary is going to execute the protocol correctly,

但是攻击者会尝试得到其它参与方的输入

but we'll try to learn something about other parties’ input.

在恶意模型中 攻击者不会遵从约定执行协议 以错误的协议执行结果中推断信息

In the malicious model, the adversary may not stick to the protocol and will try to induce an incorrect output.

我们还对不同的协议进行了简单的分类 这里需要简单解释一下

We also created a loose taxonomy of three different protocol families, which I'm going to talk a little bit about.

第一类协议是乱码电路协议 最初由姚期智于二十世纪八十年代提出

The first is garbled circuit protocols; these are first introduced in the 1980s by Andrew Yao.

自姚期智提出此协议以来 学者们持续不断地对协议进行改进

And works have been essentially continuously developed since then.

理论密码学家从不同角度对乱码电路进行了优化

Well, theorists have produced a wide variety of settings in the garbled circuit model.

从实际中我们发现 几乎所有的框架都实现了半诚实两方协议

We found in practice that they're almost all implemented as semi-honest two-party protocols,

一个参与方对电路加密后将结果发送给另一个参与方

where one party garbles or encrypts the circuit, and sends it to the other party,

另一个参与方根据输入对电路求值

who evaluates it based on their input.

在乱码电路中 需要把函数表示为布尔电路的形式

And these typically represent functions using Boolean circuits.

第二类协议包含很多不同的协议 我们称这类协议为基于电路的多方计算协议

The second type encompasses a wide variety of protocols, which we call multi-party circuit based protocols.

这些协议拥有两个共同的特性

Now, these are protocols that have two things in common.

第一个特性是 需要把函数表示为代数电路或者布尔电路的形式

The first is that they represent functions using a circuit either arithmetic or Boolean.

第二个特性是 数据需要用线性秘密分享的形式表示

And the second is that data is represented as linear secret shares.

线性秘密分享意味着协议支持任意数量的参与方

So, the linear secret sharing means that these protocols can typically support an arbitrary number of parties,

参与方协同工作 依次对门电路求值 将秘密分享输入转换为秘密分享输出

where the parties work together going gate by gate through the circuit to convert secret shares of the inputs to secret shares of the outputs.

然而 在基于电路的多方计算模型下 不同协议将输入转换为输出的方式不太相同

However, within this model, the way that they do this conversion can be executed in different ways,

可以基于信息论安全模型下转换 也可以基于密码安全模型下转换

may be information theoretic or cryptographically secure models.

我们认为这两类协议可以涵盖大多数框架的基础协议

Now, we expected these two protocol families to cover the majority of implementations.

在理论层面 大多数理论密码学家用非常有限的运算操作定义MPC协议

However, so in theory, most theoreticians define MPC protocols again on a very limited set of operations,

设计的运算操作只包含模整数下的加法和乘法 或者逐比特加法和异或运算

either addition and multiplication mod an integer, or bitwise, and an XOR.

这两类运算操作都是图灵完备的 任何函数都可以用这两种运算操作表示

And these are universal representations. So, you can express any function using that.

然而在实际中 我们需要为代数模型下的除法、比较等公共函数定义更优的子协议

However in practice, you can often define optimized sub-protocols for common functions that you have, like division or comparison in an arithmetic model,

这样我们就不用把所有函数都表示为基本运算操作 从而提高函数表达的效率

that are more efficient than reducing all the way that these very low level primitives.

我们发现有3个框架实现了特定的子协议

So, we found three protocols that implement certain sub-protocols.

我们称这类协议为混合协议

And we call these hybrid protocols.

在最下方 大家可以看到2个电路编译器

And then at the bottom, you can see our 2 circuit compilers.

当选择使用适当的框架时 人们着重需要考虑高层语言对协议的抽象能力

Now, one major thing that people need to consider when they're choosing an appropriate framework for their use case is the level of abstraction of the high level language.

不同框架都实现了协议的抽象 我们对这些高层语言展开了考察

So, we saw a variety of these. And I want to walk through a couple of them.

我们测试的其中一个样例程序是内积运算

The example we're going to use as one of our sample programs which is the inner product.

内积运算是指逐位计算乘机 再对各个结果求和

It takes the sum of the pairwise product of two vectors.

Frigate是一个电路编译器 它的高层语言是非常传统的C语言风格抽象

So, Frigate is a circuit compiler, and it uses a pretty traditional C style abstraction.

大家可以看一下幻灯片上给出的内积运算代码实现

You can see an implementation of the inner product here.

初始化结果变量 用循环语句依次取出向量中的每一个元素

You take a result, and then you loop over your vectors for each element in your vector.

计算各个元素的乘积 并对乘积结果求和

You multiply them together, and add them to your sum.

实现过程非常直观 通俗易懂

Now, this is pretty straightforward and makes a lot of sense.

然而 如果你熟悉MPC 你可能会知道在线性秘密分享模型中

However, if you're familiar MPC, you might recognize that in a linear secret sharing based model,

我们可以通过一轮交互并行处理所有的乘法运算

you can paralyze all the multiplications into a single round of communication.

如果你希望得到优化后的协议 你可能就需要使用PICCO等框架了

So, if you want to have that optimization, you might use something like PICCO.

PICCO会对内积运算的乘法进行了并行优化

PICCO recognizes that you can do this optimized multiplication step.

它们实现了一个针对内积运算的自定义算子 所以PICCO是一个混合协议框架

And so, they implement a custom primitive, this is one of our hybrid languages, for the inter product operation.

大家可以看到 可以用这个非常简单的自定义算子求两个任意长向量的内积结果

And you can see you just use this very simple custom infix operator to execute an inner product on any size vectors.

即使你不熟悉密码学 也可以很方便地直接使用自定义算子

Now this is really good if you're not super familiar with the cryptography.

你也不需要关注底层到底做了什么

And you don't really care what's happening under the hood.

然而 如果你是一个密码学家 你想实现一个比内积更复杂的函数

However, if you are a cryptographer, and you're working on perhaps a more complicated function than the inner product,

你可能会希望对生成的电路做更深度的控制和修改

you might want to have more control over exactly the circuit that you're going to have produced.

这种情况下 你可能会使用ABY这样的框架

In that case, you might prefer a framework like ABY.

ABY是一个端到端框架 在C语言上实现了一个函数库

ABY is an end-to-end framework. It's implemented as a library in C.

大家可以看到 我们用一个share类管理秘密数据

So, you can see, we have a share type that holds our private data.

我们随后放置一个乘法门

We put a multiplication gate in.

ABY会帮助我们实现乘法的并行优化

And ABY does implement these parallelized multiplications.

我们需要一个乘法门对整个向量逐位计算乘法

So, you need a single multiplication gate to do pairwise multiplication across your whole vectors.

随后 我们把向量展开 对所有乘法运算结果求和

And then, you break apart your vector representation to add up the individual elements.

这可以给我们更大的自由 实现我们想实现的函数

Now this gives you a lot more freedom to do exactly the thing that you want to do.

但如果你对密码背景不熟悉 你可能就不想具备这些自定义的能力

But if you have less cryptographic familiarity, you might not want this much power.

这就是我们考察的前后端高层语言的范围

So, that's the range of front-end high-level languages that we saw.

我们下一个想讨论的内容是这些框架的一些限制

The next thing I want to talk about is some of the limitations.

正如我前面提到的 软件工程是这些框架中最主要的问题

As I mentioned, software engineering was a major issue.

大家一定要记住 大多数框架都是在学术研究场景下开发的

You have to keep in mind that most of these frameworks were developed in academic settings,

因此 这些框架在工程落地时会有很多的限制

and they are therefore subject to the engineering constraints of such a setting.

在我讲解下面内容的过程中 大家要把这一点牢记于心

So, keep that in mind as I go through the next couple of things.

最主要的痛点是构建系统 系统的整个构建过程非常复杂

One major pain point was the build systems, which tended to be extremely complicated.

你需要从源代码层面编译特定版本的OpenSSL库 这就要花费很长的时间

You often had to compile a specific OpenSSL version from source, which takes a long time,

或者你需要建立一个自定义的证书认证机构 从而建立秘密通信信道

or set up a custom certificate authority to ensure private channels and your communication.

光编译每一个框架平均就要花费我们1-2周的时间

On average, it took us one to two weeks just to compile the existing frameworks.

这个过程苦不堪言 但是大家很幸运 我们已经把编译好的环境放在了Docker仓库里

This is very frustrating, but luckily for everyone else, we've put this all in our Docker repository.

所以大家不需要再重复一遍此项工作了

So, no one should ever have to do this again.

在系统构建之上 要使用这些软件框架项目 我们还需要很多的软件开发工作

Beyond these build systems though, these are significant software projects that require a lot of software development.

正确实现密码学协议已经很困难了 但在这之上 开发者还需要实现很多支持系统

In addition to implementing cryptographic protocols, which is notoriously difficult to do correctly, programmers have to implement a variety of supporting systems,

例如分布式通信、用高层语言实现与其它通信系统交互的功能

like distributed communication and interfacing with other communication systems in secure languages.

这方面的结论虽然比较细节 但仍然令人沮丧

This results in just small, but frustrating things.

例如 在ObliVM中 我们无法撰写一个返回结果超过32比特的计算函数

For example, in ObliVM, we weren't able to write a computation that returned more than 32 bits.

我们可以通过进一步的代码开发来解决这个问题

This is a problem that could be solved with more hands on keyboards and some time.

由于框架都是在学术层面上开发的 框架在实现层面上都不太完美

But given the constraints of the academic setting, not all the frameworks have these kinds of engineering perfection.

另一个比较严重的问题是可用性 尤其是文档比较匮乏

The other major issue we saw was usability, and especially in documentation.

如幻灯片所示 我们定义了5类文档

We defined 5 types of documentation which you can see here.

一半框架都缺失了至少3类文档

Half the frameworks had no more than 3 of these.

我这里不详细介绍每一类文档的细节

I'm not going to go through all these different types in detail.

我想给大家展示几个例子 从而证明语言文档的缺失极大地影响框架的可用性

But I do want to give you a few examples of places where there were limited language documentation that made it more difficult to use the frameworks.

语言文档指的是描述如何使用高层语言的文档

A language documentation is anything that describes how the high-level language works.

CBMC-GC是一个电路编译器 可以将代码编译成乱码电路

So, CBMC-GC is a circuit compiler that compiles a subset of GC.

大多数人都熟悉C语言

So, most people are familiar with C,

假设我们要实现这样一个简单的程序 把两个数直接相乘 这个代码感觉上是正确的

and you assume that a simple program like this, which multiplies two numbers together, would just work.

然而 我们会得到一个编译错误：我们忘记返回一个值

However, we get an error that says did you forget to return a value.

实际上 在CBMC-GC中 计算过程中的所有秘密输入的变量名都需要以input开头

It turns out that in CBMC-GC, all private inputs of the computation have to have variable names that start with input.

这根本不算是一个问题 但是并没有文档说明这一点 我们有必要告诉大家这一点

And this isn't an issue, but it wasn't written down. So, we had to sort of figure it out.

另一个例子来自于ObliVM

Another example is from ObliVM.

这是一个将类Java语言作为高层语言的端到端框架

Again, this is an end-to-end framework that consumes a Java like language.

与前面相同 我们的程序是计算两个数的乘积

And again we have our program to multiply two numbers together.

但我们碰上了解析错误

And it encounters a parsing error.

事实上 Alice和Bob是高层语言中的保留关键词

It turns out that Alice and Bob are reserved key words in the language.

因此 我们不能把这两个词作为变量名

And so, you can't use them as variable names.

Wisteria是开发编程语言的人所撰写的端到端框架

Wisteria is an end-to-end framework that was developed by programming languages people.

此框架使用函数式语言来描述计算函数

And so, it uses a novel functional representation.

此框架包含了一个详尽的语言指南 告诉大家如何使用函数式语言编写计算函数

And they include an extensive language guide for people who might not be as familiar with that functional style.

然而 语言文档没有考虑到解析器的限制要求

However, the language docs, this tutorial doesn't account for the limitations in the parser.

开发者需要在代码中放置很多的括号 编译器才能编译通过

So, developer has to put a lot more parentheses and then the tutorial would imply.

EMP-toolkit是一个我们非常喜欢使用的框架 这是一个基于乱码电路的框架

And then, EMP-toolkit is a framework we really liked using. It's a garbled circuit based framework.

然而 我们发现平均600行代码才会有1行注释 并且没有单独的代码解释文档

However, we found an average one comment for 600 lines of code with no additional documentation in separate places.

这些问题都会导致框架难以使用

So, these are all things that make it more difficult to use these frameworks.

然而 有一些框架的文档工作做得很好

However, there were some frameworks that did a really good job with documentation.

我真挚地感谢这些框架的作者

I just like to thank them here.

对于那些维护一个较大开源项目框架的开发者 我想给出两个重要的建议

We do have two major takeaways for anyone who maintains a large open source project like this.

第一个建议是 即使针对不同方面的很简单的文档 也会大幅提高框架的可用性

First is that having multiple types of documentation, even if it's mediocre, can drastically increase the usability of your framework.

不同类型的文档指的是 可能有一个文档解释框架的架构

This could be something like one document that explains the architecture of your framework,

另一个文档是带注释的样例程序 演示一些高层语言的特性

and then a commented example file that demonstrates some features of your high-level language.

第二个建议是在线资源 例如提供一个邮件列表或在GitHub上开启问题追踪

The other thing we recommend is an online resource. For example, a mailing list or a GitHub issue tracker.

这是一个持续生成、持续更新框架文档的好方法

These are a super sustainable way to produce documentation for your framework.

问题追踪就像一个在线问答平台 这样你就不用通过邮件重复回答相同的问题了

It creates a living FAQ. So, you don't have to repeatedly answer questions by a private email.

问题追踪也是用户之间相互交流的平台 他们可以互相回答遇到的问题

And it also allows a place for users to interact. So, they can answer each other's questions.

如果你不再想维护你的框架 用户仍然可以相互讨论 解答相应的问题

And if you end up not maintaining your framework, users can still talk to each other and find solutions to their issues.

即使有这些工程和可用性方面的问题 MPC框架的实现情况还是非常乐观的

So, looking forward even given these engineering and usability challenges, MPC is in pretty good shape.

我们可以在框架上实现很多样例程序

We were able to implement lots of example programs.

总体来说 实现过程还是很顺利的

And overall, things look pretty good.

社区也发现了框架可用性的问题

These usability challenges are acknowledged by the community.

IARPA HECTOR项目正在赞助下一代MPC框架的实现

IARPA HECTOR program is funding the next generation of MPC frameworks.

在赞助中 它们专门提出了可用性评价标准

And they have specific usability criteria included in the grant.

我们强烈建议后续的开发者们可以与编程语言研究者合作

We do recommend that future developers consider working with programming languages researchers.

大多数框架都是由密码学家实现的

Most of these frameworks were developed by cryptographers.

因此前后端语言的设计可能不是很规范

And therefore the front-end languages are perhaps not as principled as they might be with

编译器经验充分的开发人员介入 会对语言设计有更好的帮助

if somebody with extensive compiler experience helped develop them.

插播一条广告 我们仍然在积极维护我们的代码仓库

And then, one last plug, our repository is actively maintained.

我们随时准备接受新框架、已有框架的提交请求

And we're accepting pull requests from new and existing frameworks.

如果你在维护其中一些框架 或者想在学术项目中使用这些框架 建议看看此代码仓库

If you maintain one of these or would like to use this as a starting point for your academic project, we encourage you to check it out.

最后一条广告 我们在寻找MPC的落地项目

And then one last plug, I am looking for projects related to MPC in practice.

如果你是潜在合作方 如果你有一个有趣的项目 请稍后与我们联系

So, if you are a potential collaborator, and you have an interesting project, please come talk to me later.

非常感谢

Thank you.

非常感谢

Very much.

如果有问题的话 请用麦克风提问

Any questions, please come to the microphone.

你好 首先非常感谢你们所做的工作

Hello, first, thank you very much for all this.

社区迫切需要你们所做的工作 这个工作非常令人激动

This was very much needed for the community. And I think it's a pretty amazing work.

我知道你们完成了这一概览性的工作

So, I know that you did all these great compendium of things.

你们也得到了一些好的结论 一些不好的结论

And then you got the good things and the bad things of each.

但你们没有给出类似这样的结论：这是正确的框架

But you didn't really give a sort of like, this is the framework,

社区应该在这个框架的基础上继续构建

like this is the one we say the community should continue build it on top of this one or something like that.

类似这样的结论可能会非常重要 因为如果没有这样的结论 最后的情况可能就是

And I think it's important if you have some word on that, because if not like, what people end up doing is like,

哦不 这里有12个标准 我们要尝试构建一个标准 然后我们就得到了第13个标准

oh no, there are 12 standards, let's try to standardize this. And of course, you end up with 13, right?

是的

Right.

你能给出类似这样的结论吗？

So, you have a word on that?

当然可以 在论文中 我们明确给出了建议 推荐使用哪些框架

Sure, so, in the paper, we do give stronger recommendations as to which frameworks we recommend.

If you're looking for a specific recommendation for a setting, there are sort of 4 that I would say stand out above the rest.

For garbled circuit frameworks, Obliv-C is a good general purpose framework.

And EMP toolkit is really good if you have a little more cryptographic expertise.

And then, SCALE-MAMBA is the best of the multi-party, or it's actually a hybrid protocol.

So, it's the best of these linear secret sharing based protocols.

It's really extensive, and it's actively being developed.

And then, if you are a person who has actual security needs, you might be interested in Sharemind.

This is the only framework that has paid developers working on it, who are in academics.

So, those would be the strong recommendations I have.

谢谢

Thank you.

So, I'm actually absolutely going to develop a programming language in which Alice and Bob are reserved words. That's cool.

OK, thank you very much again.

Thank you.