[译]零知识证明: an illustrated primer

原创：冉小龙 CryptoTech 1月31日

前言：

密码学一直被冠以一层神秘的面纱，它深奥的原理一直让人望而生畏，在区块链技术中，密码学又显得尤为重要，所以，尝试翻译学习一下 “Zero Knowledge Proofs” 来加深对密码学的进一步认识。

“零知识” 这个词非常吸引人，我们假设零知识是“绝对安全”的代名词，这导致了许多地方在滥用它。它被固定在各种各样的东西上，比如加密系统和匿名网络，然而它们与真正的零知识协议毫无关系。

这一切都有助于强调一个观点：零知识证明是密码学家所设计的最强大的工具之一。但不幸的是，他们的理解也相对较差。接下来，我将尝试给出一个（大部分）非数学描述 ZK 证明是什么，以及是什么使他们如此特别。

ZK 的起源：

“零知识”的概念最早是由麻省理工学院的研究人员 Shafi Goldwasser，Silvio Micali 和 Charles Rackoff 在 20 世纪 80 年代提出的。这些研究人员正在研究与交互式证明系统有关的问题，即第一方（称为“证明者”）与第二方（“验证者”）交换信息的理论系统，以说服验证者某些数学陈述是真实的。

在 Goldwasser 等人之前，这个领域的大部分工作集中在证明系统的正确性上。**也就是说，它认为恶意的证明者试图“欺骗”验证者相信虚假陈述的情况**（有点难理解）。Goldwasser，Micali 和 Rackoff 所做的就是把这个问题变成现实。他们不仅仅担心证明者，而是问：如果验证者是不可信的，你该怎么办？

他们提出的具体问题是信息泄漏。具体地说，在证明过程中，验证者要了解多少额外的信息，就会超出了陈述是真实的事实？

注意，这不仅仅是理论上的兴趣，这种理论已经有其实际的应用。

下面是一个例子：假设现实中的一个客户希望使用密码登录到 Web 服务器。这个问题的标准在 “real world” 中涉及到在服务器上存储散列版本的密码。因此，登录可以被看作是一种“证明”，即给定的密码散列是某个密码散列函数的输出 - 更重要的是，客户端实际上知道密码。（不理解没关系，接着往下看）

大多数真实的系统以相当差劲的方式实施这个“证明”。客户端只需将原始密码发送到服务器，重新计算密码哈希并将其与存储的值进行比较。这里的问题很明显：在协议结束时，服务器已经学习了我的明文密码。因此，我们只能默默祷告服务器不会瘫痪，呵呵。

Goldwasser，Micali 和 Rackoff 提出的是进行这种证明新思路。庆幸的是，零知识证明就是用来证明上述问题，同时证明除了与“这个真实的”相对应的单个信息外没有任何信息被泄漏（看完下面的讲解再回过头来理解）。

A ‘real world’ example：

到目前为止，这个讨论非常抽象。为了使事情更加具体，让我们继续，给出一个 “真正的” 零知识协议（略微疯狂）的例子。

就这个例子而言，我希望大家能想象我是一个新通信网络的电信巨头。我的网络结构如下图所示。该图中的每个顶点表示一个无线电塔，连接线（边缘）表示两个单元重叠的位置，这意味着它们的传输可能相互干扰。

很明显，这种重叠是有问题的，因为相邻塔的信号可能会扰乱我对信号的接收。幸运的是，我的网络设计允许我将每个塔配置为三个不同频段之一以避免这种干扰。

因此，要部署我的网络的挑战是将频带分配给塔，使得没有两个重叠的无线电塔会共享相同的频率。如果我们使用颜色来表示频段，我们可以快速找出解决问题的办法：

当然，绕了半天圈子，大家会发现，这是一个著名的理论，称为 Graph three-coloring 问题。您可能也会知道，有趣的是对于某些图形来说，找到一个解决方案或者确定一个解决方案是否存在都是相当困难的。事实上，Graph three-coloring 具体来说，就是给定的图形是否支持三种颜色解决方案的决策问题 - 在已知的复杂度等级 NP-complete 中。

毫无疑问，上面的例子 so easy，用手都可以找到着色方案。OK，我们将思维发散。例如，假设我的蜂窝网络非常庞大而复杂，以至于我所掌握的计算能力还不足以找到解决方案。在这种情况下，我们能想到的途径是将这种解决方法外包给具有这种庞大算力的公司。例如：google。

但是，这会导致一个问题。

假设 Google 使用大量的计算基础设施来为我的图搜索有效的着色。**在我知道他们能够找到解决途径之前，我肯定不会付钱给他们。但是，在当我付清之前，Google 不会给我一个他们的解决方案的副本。我们将陷入僵局。**

较为疯狂的解决方案

Google 的工程师与麻省理工学院的 Silvio Micali 进行了协商，他与 Oded Goldreich 和 Avi Wigderson 的同事进行了磋商，提出了如下的巧妙协议 - 一个如此优雅，甚至不需要任何计算机。它只需要一个大仓库，大量的蜡笔和大量的纸张。哦，还有一大堆帽子。

它是如何工作的呢？

首先，我将进入仓库，用纸覆盖地板，并画出我的网络拓扑图的空白表示。然后，我将退出仓库。google 工程师现在进入，按照他们预先的解决方案，使用（红色/蓝色/紫色，如上例）三种颜色的集合，随机的为我的空白图进行着色。注意，使用哪种特定的蜡笔并不重要，只要着色是有效的。

在离开仓库前，Google 工程师用帽子盖住每个顶点。当我回来的时候，我会看到：

很明显，这种方法可以完美地保护 Google 的着色方案。但它对我来说是不利的。我可能会想，Google 会用一个随机的，无效的解决方案填充图表。他们甚至可能根本没有对图表进行着色。

为了消除我的疑虑，Google 现在给了我一个“挑战”图表着色解决方案的机会。我可以随意选择这张图的一个“边缘”（也就是两个相邻的帽子之间的一条线）。然后谷歌将“揭开”这两个相应的帽子，展示他们解决方案的一小部分：

对我而言，这个实验有两个结果：

1. 如果两个显示的顶点是相同的颜色（或根本没有着色），那么我肯定知道Google正在对我说谎。很明显，我不会向 Google 支付一分钱。
2. 如果两个透露的顶点是不同的颜色，那么Google可能不会对我说谎，（仅仅是可能）。

我们肯定希望第一种情况出现，我直接就能知道 Google 在骗我（在验证过程中，我一直是处于怀疑的角色）。问题是，即使在我们的实验后，谷歌仍然可以对我说谎 - 毕竟，我只看了两下帽子。如果图中有 E 个不同的边缘，那么Google 可能会填写一个无效的解决方案。具体来说，经过一次测试，他们可能以（E-1）/ E 的概率（1000 个边缘图的计算结果达到 99.9% 的时间）成功欺骗我。

幸运的是，Google 为了进一步消除我的疑虑，允许我们继续重复执行上述的操作!

我进入仓库，重新绘制和我上述一样的网络拓扑的空白图表。接下来，google工程师进入仓库，用一个有效的解决方案重新填充图形，但是使用上述三种颜色（红色/蓝色/紫色）重新随机排序。**这里，要注意理解，我进去仓库后，绘制的还是我原先的图表，但是google工程师使用的着色方案是与第一次的着色方案不同的。**

帽子又回来了。我回来重复挑战过程，挑选一个新的随机边缘。这一次，如果一切顺利的话，我现在应该稍微有点自信，Google 告诉我的是实话。为什么呢？因为为了欺骗我，Google 必须连续两次幸运。这可能发生 ， 但发生的概率相对较低。现在，Google 连续两次欺骗我的机会现在是（E-1）/ E *（E-1）/ E（或者我们上面的1000个边缘例子大约有99.8％的概率）。

幸运的是，我们可以一直重复上述的操作过程。直到我确信Google可能会告诉我真相。

这里有一个例子，大家可以直观感受一下：http://web.mit.edu/~ezyang/Public/graph/svg.html

请注意，我永远不会完全确定Google是诚实的 - 他们欺骗我总是会有一个小概率。但经过大量迭代（E ^ 2），我终于可以提高自己的信心，以至于 Google 只能以微不足道的可能性来欺骗我 - 这足够低了，实际上并不值得担心。然后，我可以安全地把我的钱交给 Google。

在这个过程中，我们要注意理解，Google 的着色方案也是受到保护的。即使我试图通过在协议运行之间留下笔记来了解他们的解决方案，也不要紧。google 每次让我验证

之前都会随机使用新的解决方案。我每次获取到的信息都是不同的，所以我没办法将这些信息串联起来，最终获得 google 的着色方案。

What makes it ‘zero knowledge’?

我向你声称，这个协议没有泄露有关 Google 解决方案的信息。但是现代密码学的第一条规则就是**永远不要相信那些没有证据就要求这样的事情的人。**

Goldwasser，Micali 和 Rackoff 提出了三个以下属性，每个零知识协议都必须满足。它们是：

1. 完整性。如果谷歌说的是实话，那么他们最终会说服我（至少很有可能）。
2. 确定性。如果 Google 真的说实话，他们只能说服我（这个方案只对我有用）。
3. 零 knowledgeness。 （是的，这真的叫做这个。）我不知道Google的解决方案。

我们已经讨论了完整性的论点。如果我们运行足够多的时间，协议最终会说服我（错误概率可以忽略不计）。如果谷歌曾经试图欺骗我，我会以极大的可能性检测到他们的在说谎。

这里最困难的部分是“零知识”属性。要做到这一点，我们需要进行一个脑洞大开的思想实验。

时光穿梭机

首先，让我们做一个疯狂的假设。想象一下，Google 的工程师并不像人们想象的那样有能力。他们在这个问题上工作了很久很久，但他们一直没有找到合理的解决途径。所以他们决定欺骗我。

他们的想法是潜入 Google “X 研讨会” 并借用 Google 的原型时间机器。最初的计划是倒退几年，利用额外的工作时间来解决问题。不幸的是，事实证明，像大多数谷歌原型一样，时间机器有一些限制。最关键的是：它只能倒退 4min30s。所以没办法使用时间机器来赢得更多的工作时间。但事实证明，即使这种非常有限的技术仍然可以用来欺骗我。

这个计划执行起来非常简单。由于 Google 实际上并不知道图形的有效颜色，因此他们只会用一堆随机颜色对纸张着色，然后戴上帽子。如果运气好的话，我会挑战他们在一对不同颜色的顶点，每个人都会松了一口气，我们会继续这个协议。到现在为止还挺好。

然而，不可避免地，我要验证一副帽子，发现两个相同颜色的顶点。在正常的协议中，我目前已经知道 Google 在说谎了。但是 Google 这个时候启动时间机器。每当 Google 发现自己处于这种尴尬境地时，他们只是简单地修复它。也就是说，一个指定的 Google 员工拉开一个开关，“倒退”时间大约四分钟，Google 团队用一个全新的随机解决方案重新调整图表。现在，他们让时间前进，然后再试一次。

时间机器允许谷歌’修复’在他们的虚假协议执行过程中发生的任何事故，这使得我的经验看起来完全合法。由于坏的挑战结果只会发生在 1/3 的时间内，所以协议的预期运行时间（从 Google 的角度来看）只比运行诚实协议所需的时间稍微大一些。从我的角度来看，我甚至不知道额外的时间机器旅行正在发生。

最后一点是最重要的。事实上，从我的角度来看，不知道时间机器的存在，所以从我的迷之视角来看，由此产生的交互与真实情况完全相同。再强调一点，在时间机器版本中，Google 绝对没有找到合理的着色方案。

这到底是什么意思?

我们刚刚展示的是一个模拟的例子。请注意，在三维世界中，时间是不可能倒退的，也就是说，没有人可以用时间机器欺骗我，这个基于帽子的协议是正确的和合理的，这意味着在 E ^ 2 轮之后，我能够确信（除了可忽略的概率）Google 对图形的着色方案是正确的。

在上面扯淡的例子中，我们假设如果谷歌能够“倒退”我的时间观点 - 那么即使他们根本没有关于实际图的信息，他们也可以为着色方案伪造有效的运行协议。从我的角度来看，两个协议有什么区别？当我们考虑两者的统计分布时，根本没有区别。两者都传达了相同数量的有用信息。

相信与否，这证明了一些非常重要的东西。

具体而言，假设我（验证者）在观察到诚实协议的执行之后有一些策略“提取”有关 Google 着色的有用信息。那么当我被时间机器愚弄的时候，我的策略应该也同样适

用。从我的角度来看，协议运行在统计上是相同的。我无法分辨。

因此，如果我所能提取的信息量在“真实实验”和“时间机器实验”中是相同的，但是 Google 放入“时间机器”实验的信息量恰好为零 - 那意味着即使在现实世界的协议下 Google 也没有泄露任何有用的信息。（我的理解就是：加入时间机器实验之后，Google 没有给出我们任何解决方案，但是在我看来，这个着色方案一样是可信的，那么在现实世界，同理。）

上述实验主要用来辅助理解，什么是零知识

摆脱帽子和时间机器

当然，我们不能在庞大的网络拓扑图中使用帽子和时间机器什么的来解决问题。

把事情联系在一起，我们首先需要把我们的协议带入数字世界。这就要求我们构建一个“帽子”的数字等价物：既隐藏数字价值，又同时“约束”（或“承诺”）制造者的东西，并且在验证之后她不能改变主意。

幸运的是，我们有这个应用程序的完美工具。这就是所谓的数字承诺计划。承诺方案允许一方在保密的情况下“承诺”给定的信息，然后“打开”由此产生的承诺，揭示内部的内容。它们可以由各种成分构成，包括（强）密码散列函数。

给定一个承诺方案，我们现在拥有了我们所需要的所有要素来以电子方式运行零知识协议。证明者首先将其顶点颜色编码为一组数字消息（例如数字0,1,2），然后对每个数字消息产生数字承诺。这些承诺被发送到验证器。当验证者在边缘挑战时，证明者只显示对应于两个顶点的承诺的开放值。

所以我们设法消除了帽子。但是，我们如何证明这个协议是零知识？

幸运的是，现在我们处于数字世界，我们不再需要一台真正的时间机器来证明这个协议的事情。一个关键的技巧就是在我们的设置中指定协议不会在两个人之间运行，而是在两个不同的计算机程序之间运行（或者是更为正式的概率图灵机）。

我们现在可以证明的是下面的定理：如果你能够想出一个在参与协议运行之后提取有用信息的计算机程序（对于验证者），那么就有可能使用“时间机器”在那个程序中，为了使得从证书没有提供任何信息开始的协议的“假”运行中提取相同数量的有用信息。

而且由于我们现在正在谈论电脑程序，显而易见，“时间倒退”很容易。事实上，我们一直在“倒退”电脑程序。例如，我们使用有快照功能的虚拟机软件就可以解决。

即使你没有花哨的虚拟机软件，任何计算机程序都可以被“回退”到一个较早的状态，只需从头开始重新编程，并为它提供完全相同的输入。只要包括所有随机数在内的输入是固定的，程序将始终遵循相同的执行路径。因此，您可以通过从一开始就运行程序来“回退”一个程序，并在达到某个期望的点时“分叉”执行。

最终我们得到的是下面的定理。如果有一个 Verifier 计算机程序通过与一些 Prover 交互地运行这个协议成功地提取信息，那么我们可以简单地使用该程序的 “回退” 技巧来提交一个随机的解决方案，然后通过倒退它的执行来 “欺骗” Verifier 无法正确回答其挑战。上面给出的逻辑是相同的：如果验证者在运行实际协议后成功地提取信息，那么它应该能够从模拟的基于 “回退” 的协议中提取相同数量的信息。但由于没有信息进入模拟协议，所以没有信息可以提取。因此，验证者可以提取的信息必须始终为零。

OK，这是什么意思呢?

现在，让我们回顾一下。我们知道，协议是完整的，基于我们上面的分析。在任何情况下，都不存在时间机器这个玩意儿。

同时，协议也是零知识。为了证明这一点，我们证明了任何成功提取信息的 Verifier 程序也必须能够从使用的协议运行中提取信息，并且首先没有信息可用。这导致了一个明显的矛盾，并告诉我们这个协议在任何情况下都不能泄漏信息。

这一切都有一个重要的好处。由于任何人“伪造”协议都是微不足道的，即使在Google向我证明他们有解决方案之后，我也不能重新还原协议的记录，以向任何其他人证明任何事情（比如法官） 。这是因为法官不能保证视频是真实录制的，而且我也不会像谷歌使用时间机器那样简单地进行编辑。这意味着协议转录本身不包含任何信息。协议只有在我自己参与的情况下才有意义，我可以确定它是实时发生的。

总结

可以使用散列函数来构建一个简单的承诺示例。要提交值“x”，只需生成一些（适当长的）随机数字，我们将其称为“salt”，并输出承诺C = Hash（salt || x）。要打开承

诺，你只需要显示“x”和“salt”。任何人都可以通过重新计算散列来检查原始承诺是否有效。这在一些关于函数本身的假设下是安全的。

原文链接：https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/

本文由 `Copernicus团队 冉小龙` 翻译，转载无需授权。