几分钟搞懂全同态加密FHE：运行模式与应用场景

原文标题：Fully Homomorphic Encryption: Introduction and Use-Cases

原文作者：Nicolas Gama 和 Sandra Guasch，SandBoxAQ

编译：Faust，极客web3

这篇博客是对全同态加密（FHE）的系统性介绍，但我们并不在此深入的探讨数学细节，而主要从基础的机制设计角度来解释这项技术，让读者初步理解FHE的基本运行逻辑，并介绍FHE的几个主要应用模式。



当提到“加密”时，人们最先联想到的应用场景通常是静态加密和传输中加密。前者将数据加密后存储在硬件设备中，如硬盘、移动设备，或是基于云端的服务器，在这种模式下，只有被授权的人能查看或改写解密后的明文内容。而传输中加密，目的是让通过互联网传输的数据只能被指定的人解读，即便数据通过公共路由器或通道来传输，中间人也无法私自解密出明文。

上述两种场景都依赖于加密算法，同时还会额外保证数据的完整性，即数据在传输过程中没有被中间人篡改，这被称为“认证加密”：一旦数据被加密后，数据传输过程中的参与者无法私自解密出明文（保密性），且任何中间人都不能随意篡改原始的密文（完整性/真实性）。

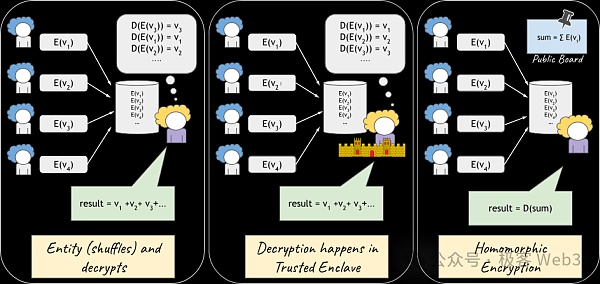
至于某些多方协作的场景，需要对密文进行一些复杂的处理，这属于隐私保护技术的范畴，全同态加密（FHE）就是其中之一。我们可以举一个线上投票的案例：假设在一次总统选举中，选民可以将自己的投票结果进行加密，然后提交给某个中间实体，该中间人将所有投票结果统一收集起来，算出每个总统候选人的得票数，最后只公布一个最终选举结果。

但不幸的是，当我们使用前面提到的“认证加密”方案时，负责统计投票结果的中间人要解密出所有人的投票数据明文，才能执行统计票数的任务，但这样就会让每个人的投票结果都暴露出来，无法保护隐私。理论上，我们可以对大家的投票数据进行混洗（某些电子投票协议确实会这么做），但与纸质选票不同，传统密码学机制在保证数据完整性（不被篡改）的同时，很难把加密后的选票数据与对应的选民身份分离开。

在线上投票方案中，我们可以在负责计票的中间人周围添加一道硬件隔离墙，比如TEE(可信执行环境)。这种技术使得恶意攻击者很难与计票程序展开互动，但硬件层面的漏洞可能让解密密文的密钥从TEE中泄露出来，而且与软件中包含的错误不同，硬件设计上的漏洞无法轻易修复。

综上，为了应对上述场景，我们可以引入全同态加密（FHE）技术。FHE是一种特殊形式的加密方案，允许在不解密密文的情况下直接对密文进行函数计算，获得该函数输出的加密后的结果，这样就可以保护隐私。

在FHE的场景下，函数 ? 的数学构造是公开的，因此输入密文 ? 输出结果?(?) 的处理流程是公开信息，可以在云端执行，而不会泄露任何隐私。这里要注意， ? 和 ?(?) 都是被加密过的密文，需要由密钥来解密，大多数时候两者对应的解密密钥是相同的。



上图展示了线上投票的三种加密/解密方案，其中E( ) 表示加密操作，而 D( ) 表示解密操作；

左侧图中，一个受信任的中间人在公布投票结果前，会对各个投票数据进行混淆和解密，我们必须假设该中间人不会泄露隐私，且投票统计结果是正确无误的；

在中间的图中，使用了TEE，TEE能保证数据完整性和隐私保护；

而在最右侧的图中，使用了同态加密技术：加密后的投票数据可以被公开加总求和，然后再解密得到结果，算出最终的投票数

FHE（全同态加密）是紧凑型加密方案，输出结果?(?) 的密文数据大小，以及解密该结果所需的工作量，仅取决于输入数据 ? 对应的原始明文，并不依赖于所采用的计算过程，这与那些非紧凑型的加密系统截然不同，后者往往简单地把 ? 与函数 ? 的源码连接起来，然后让接收者自己解密 ? 并输入到 ? 中来完成计算任务。

在现实中，FHE外包模式通常被视作TEE等安全执行环境的替代方案，FHE的安全性基于密码学算法，而不依赖于硬件等实体设备。因此，FHE完全不受被动侧信道攻击或云端服务器被攻击的影响。想象一下，当某人需要外包出去一些计算任务，但数据非常敏感，他可能不愿意使用搭建在AWS上的虚拟机（VM），因为这类基于云端的服务器背后往往存在更高级的控制人。他也可能对SGX或TEE这类东西犹豫不决，因为运行TEE的主机可以监控该计算任务在执行时产生的功耗或运行时间等，可能通过这些数据来推断出一些信息。

然而，如果使用FHE，将计算任务外包出去的人就可以安心——因为在FHE的系统中要破解私密信息，就必须破解其用到的密码学算法，但这在目前几乎无法做到。

但是，虽然密码学算法可以防止攻击者在不知道密钥的情况下破解 ? 对应的明文，但另一方面，通用延展性则允许攻击者对输出结果 ?(?) 进行修改，这相当于一种主动侧信道攻击：攻击者可以对执行加密算法的硬件进行针对性的攻击，影响输出结果。这听起来似乎很可怕，但在FHE的设计中，这类恶意攻击可以通过在计算流程中制造冗余来进行规避。

简要总结的话，FHE通常会用到几组密钥，包含以下几部分：

解密密钥（Decryption Key）：这是整个FHE加密系统中的主密钥，所有其他类型的密钥都可以根据主密钥来导出。解密密钥通常在用户本地生成，从不传输给外界，只有持有者本人能用它来解密FHE密文。这意味着密文即使在传输中被他人截获，也无法被解密，除非他们拥有解密密钥。

加密密钥（Encryption Key）：在FHE的公钥模式下，加密密钥是用来将明文转换为密文的密钥。当生成初始密文的人不是解密密钥/主密钥的持有者时，就会使用加密密钥来进行加密。该密钥通常中等大小，由一些随机的零加密组成。由于FHE支持仿射函数（affine functions），足以用于加密任何消息。

在公钥加密模式中，加密密钥通常是公开的，任何人都可以用它来加密数据，但只有解密密钥的持有者才能针对性的解密。

计算密钥（Evaluation Key）：计算密钥是用来对密文 ? 进行同态运算的专用密钥，使得FHE系统可以在不对密文 ? 进行解密的情况下，对密文进行同态运算。计算密钥可以像公钥一样被公开发布，即使他人获知了计算密钥，也无法破解出密文 ? ，而只能对密文进行同态运算得到一个输出结果。

此外，当使用计算密钥进行运算时，密文的结构保持不变，对密文 ? 进行同态运算得到的结果会被重新加密为新的密文，这确保了计算过程中的隐私性，即使计算过程是公开的也不会泄露机密数据。

在上述几种密钥的持有者中，解密密钥/主密钥持有者是最敏感的，他要确保整个同态操作的执行链条/流程有效无误，最终的密文是安全的，然后解密得到明文结果。如果在FHE的操作链条中引入恶意操作，解密密钥有可能会在解密时被泄露。但幸运的是，同态操作可以公开进行并被任何人所验证。

FHE的具体场景/模式

在本节中，我们将描述一些FHE中的常见场景/模式，并讨论每种模式的优缺点。

外包模式（Outsourcing Mode）



在此图中，左侧橙色的钥匙符号象征解密密钥（及其持有者Alice）。FHE密文通过与解密密钥相同颜色的锁来表示。贡献私密数据的一方用圆柱体表示：

在这里，只有Alice贡献了私密数据。而在Bob这边，同态计算函数和计算密钥是公开的，计算方案由绿色框表示，以确定性的方式进行，每个人都可以在本地重跑计算过程，并检测Bob给出的输出结果是否有错误

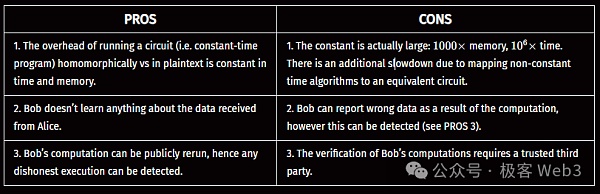
上述外包模式也是FHE的第一个历史性应用案例，旨在将普通的云计算转变为类似于SGX和TEE的私密计算，但上述FHE算法的安全性基于密码学算法，而非硬件层面的安全措施。在这种设置下，Alice拥有一些私密数据，但其计算能力有限，Bob拥有计算能力强大的云端服务器，而Bob不贡献任何额外的私密数据。

Alice可以把计算任务 ?() 的输入参数进行加密，得到密文 ? 并将其传送给Bob，Bob随后以同态加密的方式运算出?(?)的结果 ，并将加密后的结果送回给Alice。

鉴于当前的硬件设备性能，FHE外包模式的实际推广比较缓慢——如果要执行的计算任务 ?() 的复杂度是非线性的，对 ?() 进行FHE运算的耗时将达到原始计算的100万倍，内存开销大概会提高1000倍。目前很多组织正在开发FHE专用硬件以降低其计算成本，例如Darpa DPRIVE项目或CryptoLight。

目前，FHE外包模式实际主要用于PIR（私有信息检索）场景，其中公共服务器控制者Bob拥有一个大型的公共数据库，客户端Alice会发起读数据的请求，但又不想让Bob知道自己想查关于哪些人的数据。这类PIR场景受益于同态加密操作的线性和紧凑性，同时计算成本可以保持在合理范围内。

下面这个表格总结了FHE外包模式的优缺点。



Two Party Computing Mode（两方计算模式）

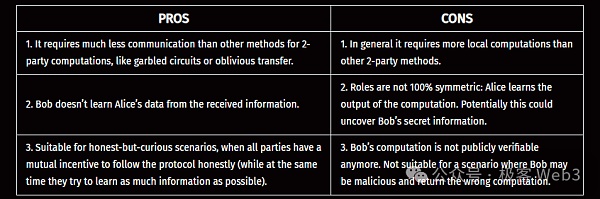


该图使用了与之前相同的颜色设置。这次，Bob在计算流程中贡献了一些私密数据。而Bob一方的计算流程不再是公开可验证的，用红色框表示，这种模式应仅限于通讯双方均为 honest-but-curious的场景：

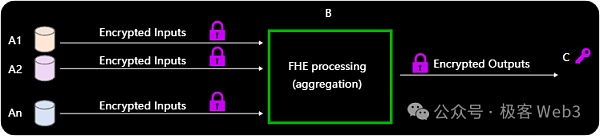
也就是说，在协议执行过程中，参与者（如Bob）会严格按照给定步骤进行计算，不会故意输出错误结果或破坏协议的执行。但Bob是“好奇”的，会尽可能从接触到的密文或其他中间数据中推断出敏感信息，但这并不会篡改协议的执行过程

在FHE两方计算的模式中，唯一的区别在于Bob会在计算过程中贡献一些私密数据，也就是把自己的私密数据加入到FHE计算过程中。在这种情况下，FHE是很好的两方计算解决方案，具有最小的通信复杂度，并可以通过其机制设计提供强有力的保证：Bob不会窥探到Alice的私密数据，而Alice也不会窥探到Bob的私密数据。

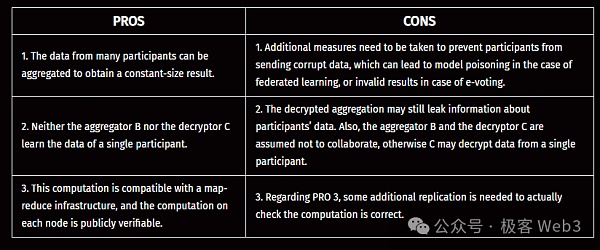
该场景的一个潜在应用案例是密码学中的“百万富翁问题”，假设Alice和Bob是两位百万富翁，他们想知道谁更富有，但又不想让对方知道自己具体有多少资产。该问题的解决方案通常用于现实的电子商务应用场景中。



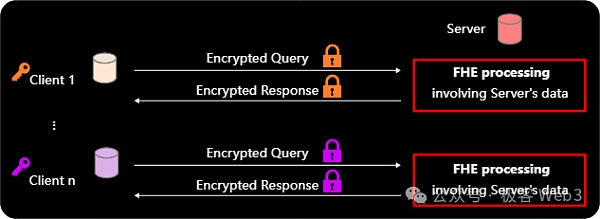
Aggregation Mode（聚合模式）



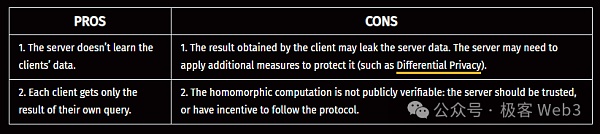
聚合模式是对外包模式的一种改进，将来自多个参与者的数据以紧凑（输出结果不会随着输入参数数量的增加而增长）且公开可验证的方式进行聚合。典型的用例包括联邦学习和线上投票系统。



Client-Server Mode（客户端-服务器模式）



这种模式是对前面提到的双方计算模式的改进，其中服务器会为多个拥有独立密钥的客户端提供FHE计算服务。FHE可以用于私有AI模型运算服务，例如，客户端有私密数据，服务器端有私有的AI模型或服务）。服务器拥有的私有AI模型是以明文形式存储在本地的，然后每个客户端都有自己的私密数据，希望放到AI模型中进行运算。最终每个客户端可以使用自己的密钥解析出自己所提交数据的运算结果。



关于同态加密的其他细节

同态加密如何确保外部计算结果是有效的？

在多方合作的场景中使用FHE是更容易的，因为每个参与者都有动机诚实地遵循协议规定。例如，FHE可以在位于两个不同国家但属于同一公司/组织内的两个法人实体之间进行加密计算并统计一些数据：诸如GDPR之类的法规允许你对外发布某些统计数据，但会禁止将所有个人数据集中存储在同一物理地点。

在这种情况下，使用FHE是可行的，所有参与者都有动机诚实地遵循协议规定。而在参与方并非彼此合作的场景中，确保计算任务已被正确执行的最简单方法，是引入冗余（类似于多签/共识）。例如，在前面提到的外包和聚合场景中，同态计算用到的函数公式是完全公开的，并且可以是确定性的，只要两个或更多独立实体得出完全相同的输出密文，那么整个计算过程就是正确的，结果也是可信的。冗余程度越高，最终结果的可信度就越高，但这需要在效率方面进行权衡。

此外，当承包计算任务的计算方通过对输入和输出密文进行数字签名来担保FHE结果有效时，每个人都可以重跑相同的FHE计算流程，并检查计算方给出的证明是否有效。任何计算方的欺骗行为都可以被检测出来并被惩罚，并可以与一个公开可验证的证书相关联，该证书会揭露欺骗行为和欺骗者——我们称这种模型为强隐蔽安全模型。

至于完全同态签名，是另一种验证计算正确性的方法，无需第三方验证者，但通常需要更多的软硬件资源来参与。

FHE如何确保最终接收者只解密最终结果而非中间变量？

最简单的方法是确保解密密钥持有者无法访问FHE计算流程中产生的中间密文。在双方计算场景或客户端-服务器场景中，Alice对输入结果进行加密，Bob对密文进行计算并将加密的输出结果传输回Alice，显然Alice只能解密最终结果，无法访问中间变量。

在基于云端服务器的场景，例如线上投票系统中，许多参与者会在AWS等公共的云端服务器上发送加密后的投票数据，这里会用到一种手段：解密密钥通常不会交给单个接收者，而是以秘密共享的方式分配给不同的人或机构（类似于MPC）。在这种情况下，只有通过执行多方计算并让持有解密密钥的成员之间在线通信，才能对特定密文进行解密。如果其中某些人拒绝配合其他人，则无法进行解密。这样就可以通过设置相应的阈值来提高系统的整体安全性。

同态加密的构建模块

同态加密有三种类型：部分同态加密（PHE）、分级同态加密（LHE）和完全同态加密（ FHE）。部分同态加密只允许我们对某些计算任务进行同态加密（例如求和、线性函数、双线性函数），而分级同态加密和完全同态加密可以支持任意的计算任务。

对于LHE来说，系统用到/产生的参数依赖于要执行的函数计算?()，并随着该函数计算复杂性的增加而增长，这反过来导致密文和密钥的大小也跟着增加，会消耗更多的存储和通讯资源。而FHE方案允许我们在给定的一组参数下（也就是给定的密钥和密文大小下），计算任何可以表示为二进制逻辑门电路的函数。也就是说，与LHE不同，即使需要计算的任务变得越来越复杂，FHE方案涉及的参数（以及密钥和密文）也不会变大。

因此，FHE是唯一一种能保证同态计算的内存消耗和运行耗时都与原始明文/任务成正比的模式。但是，FHE有一个技术上的问题：随着计算的持续进行，密文中包含的噪声（垃圾数据）会越来越多。为了避免噪声过多导致解密结果出错，FHE方案会定期执行一种代价较高的操作，称为自举（bootstrapping），它可以将噪声减少到可控水平。日后我们将对此进行更多的介绍和科普，大家敬请期待！

原文链接：https://cryptographycaffe.sandboxaq.com/posts/fhe-01/