隐私计算

第四章 安全多方计算

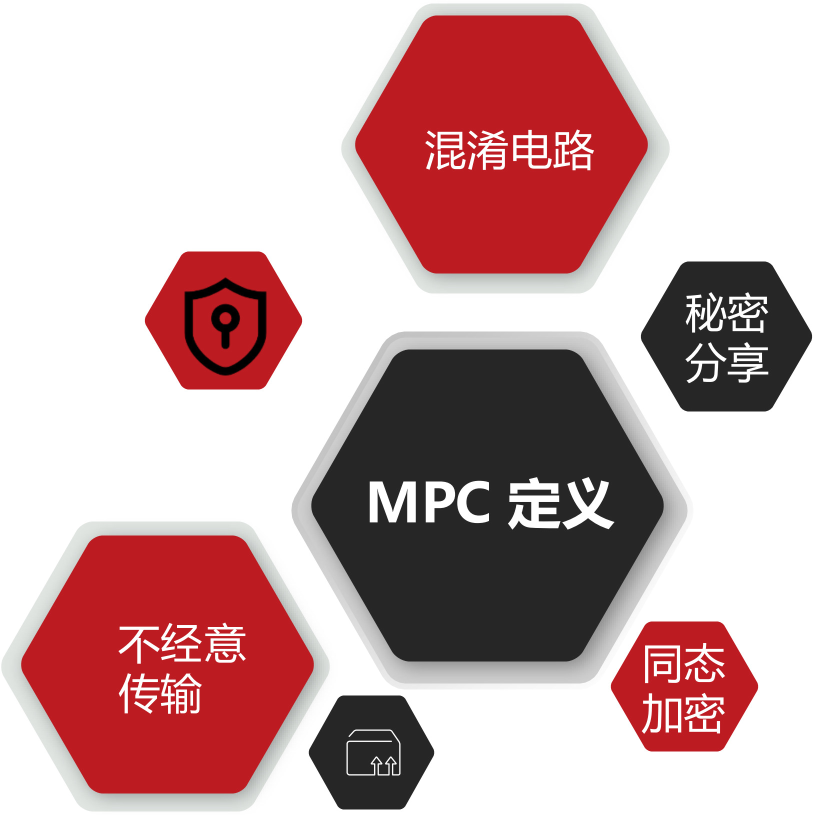
数据科学与工程学院

目录

/01

安全多方计算的发展及相关技术

安全多方计算（MPC）概述



安全多方计算（Secure Multi-Party Computation，MPC）可以允许多个参与方在保证各方输入安全性的前提下，共同执行某个预先设定好的函数

- MPC的概念是姚期智先生在 1982年提出
- 在 2015年之后被业界关注并尝试应用在实际场景中

广义的安全多方计算定义：包含任意参与方间**针对秘密持有数据的任意计算**

- 广义的定义几乎包含了整个密码学的协议内容，包括了传统加密、电子签名等

狭义的安全多方计算定义：强调通用性的**安全多方计算协议**

- 包含了**混淆电路**、**秘密分享**、**不经意传输**、**同态加密**等技术

MPC 分类方式

ECNU

威胁模型

半诚实模型：攻击者会监听其收到的所有信息，不会主动破坏协议

恶意模型：攻击者会主动拒绝执行协议、甚至破坏协议

参与方数量

- 两方MPC
- 三方MPC
- 任意多方MPC

根据实际计算场景以及性能考虑，选用不同的协议

通用/专有MPC

通用MPC可以实现任意计算

专有MPC只支持特定的某类运算（如隐私求交）

姚氏“百万富翁”问题

问题描述：

在 1982 年，姚期智先生提出了著名的“**百万富翁**”问题 ： 假设有两个百万富翁，不愿意直接告诉对方自己的财产具体是多少，但又想确定到底谁是更富有的人，怎么办？

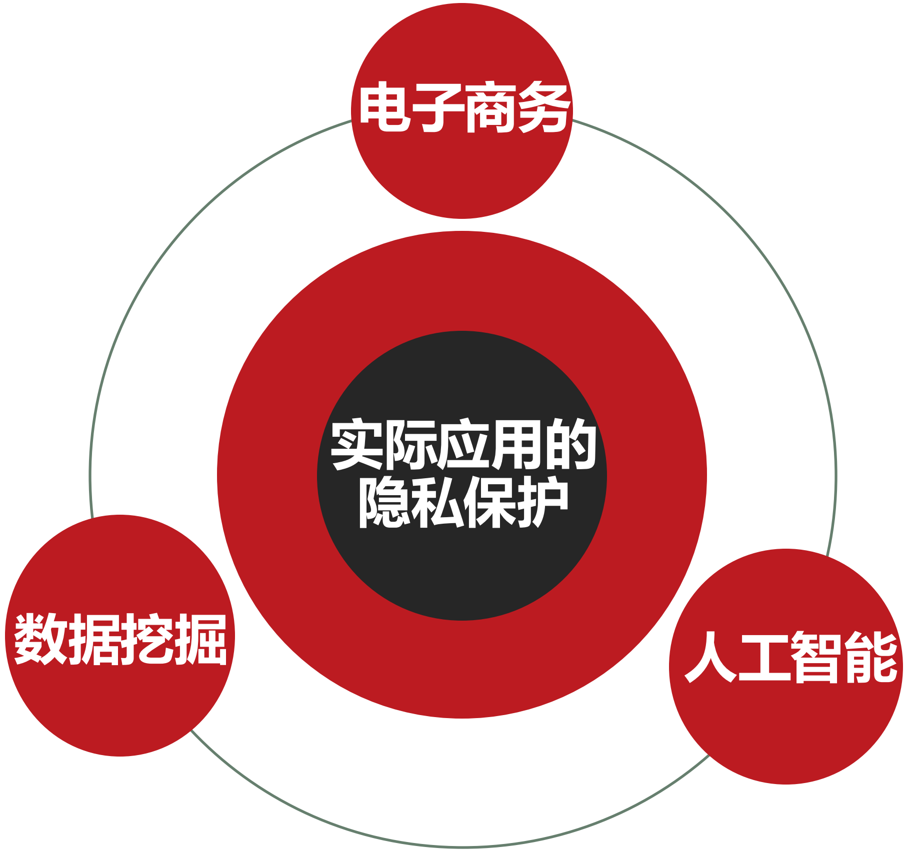
谁更有钱？

解决上述问题的第一个具体方案构造——**姚氏混淆电路**（ Garbled Circuit , GC ）**协议**，这是 MPC 协议的起源

混淆电路解决“百万富翁”问题

主要思想

- 两方场景下，其中一方(一般称为混淆方或者发送方)，将两方约定的计算函数 “编译”成电路的形式，将**电路对应的真值表加密打乱**——**混淆电路**生成
- 然后把得到的混淆电路发送给另一方解密(一般称为计算方或者接收方)，从而让计算方得到正确电路输出，而又不泄露各个参与方隐私输入



蓬勃发展：多种技术路径——秘密分享

ECNU

GMW协议

GMW协议可以用于计算算术和布尔电路。

- GMW协议将每个参与方的隐私输入在本地分成多个分片，并在参与方之间分享这些分片，此后再将这些分片作为输入，执行具体的函数电路运算。
- 单独的分片本身是无用的，每个分片都是用基于密码学的方式产生的——通常满足一次一密安全，因此每个分片本身都不能被用来反推出原始的秘密输入
- 只有当足够数量的来自同一个秘密值的分片被组合在一起时，原始秘密值才能被恢复。
- 在计算结束时，明文输出可以通过组合每个参与方的所有输出分片来得到。

蓬勃发展：多种技术路径——秘密分享

BGW协议

其基于 Shamir's secret sharing 实现

- BGW中每个秘密分片是按照t次多项式（一个事先定义的阈值）来进行构造的
- 只有组合超过t个分片，才可以解出多项式的常数项，继而恢复得到原始秘密值
- GMW 和 BGW 协议可以自然地推广到多方（参与方数量大于 2）场景

蓬勃发展：多种技术路径——秘密分享

ABY协议

- ABY 通过设计一种在算术共享、布尔共享和姚氏混淆电路协议之间进行高效切换的机制，解决了实际混合协议实现中的一个主要障碍
- 同时 ABY 还进行了一些重要优化，例如在模型生成阶段通过离线预计算来进行几乎所有的加密操作，以尽量减少模型在线计算的成本
- 混合的不同协议来进行某个部分的计算，采用合适的、最高效的协议去计算父问题下的某个子问题。

蓬勃发展：多种技术路径——不经意传输

ECNU

OT 简介

不经意传输（Oblivious Transfer，OT）协议是一类特殊的隐私计算协议

- 在 OT 协议中，接收方能够根据自身的选择，从发送方所提供的两个秘密中获取其中一个并进行接收
- OT 协议能够确保**发送方无法获知接收方的具体选择**，**并且接收方只能获取所选秘密**，无法获得未选中的秘密

OT 协议是安全多方计算中的一个基础协议或核心组件

蓬勃发展：多种技术路径——不经意传输

1995年之前，OT协议都是基于公钥密码学原语构建的，运行效率较低，实用性不强

OT 协议划分为在线阶段和离线阶段，提出 OT 扩展的概念；对少量由公钥密码学原语产生的基础OT进行扩展得到大量的 OT

2003年，Ishai 等人给出了第一个实用的 OT 扩展协议

近年，基于相关伪随机数发生器的OT 扩展协议可以以亚线性的通信开销产生大批量的 OT

蓬勃发展：多种技术路径——同态加密

ECNU

同态加密简介

同态加密（Homomorphic Encryption, HE）在 20 世纪 70 年代提出

定义：HE是将原始数据通过加密算法得到一系列密文，然后计算方在密文上直接进行相应的计算，密文计算和明文计算存在着某种对应关系；并且结算得到的是一个密态结果，需要私钥才能够解密获得正确的明文结果——和在原始明文上计算得到的结果相同

蓬勃发展：多种技术路径——同态加密

Gentry09方案是对于后续HE发展三个方面的引领作用

PHE 和 Gentry09 方案

- 早期的部分同态加密（PHE）是在传统的公钥加密方案上发展而来，其计算开销大和设计复杂性高等特性使其无法应对实际问题
- 现代HE方案则主要是在Craig Gentry的突破性工作Gentry09方案的基础上发展而来的

蓬勃发展：多种技术路径——同态加密

Gentry09方案为代表的第一代FHE方案在计算效率上还不足以解决许多实际问题，但在HE研究方向上取得了巨大的突破。

第二代FHE方案中出现针对常用整数算术和标量乘法的自举优化技术，如BGV和BFV。

第三代FHE方案开始于Gentry等人，设计了一种不同的噪声增长机制，具备更高的安全等级

CKKS 为代表的第四代FHE 方案中提出了关于多项式逼近和机器学习模型的针对性优化

- OpenFHE、Microsoft Seal、IBM Helib、隐语HEU等开源项目提供了常见的HE操作接口，并持续更新

/02

混 淆 电 路

混淆电路(GC)概述

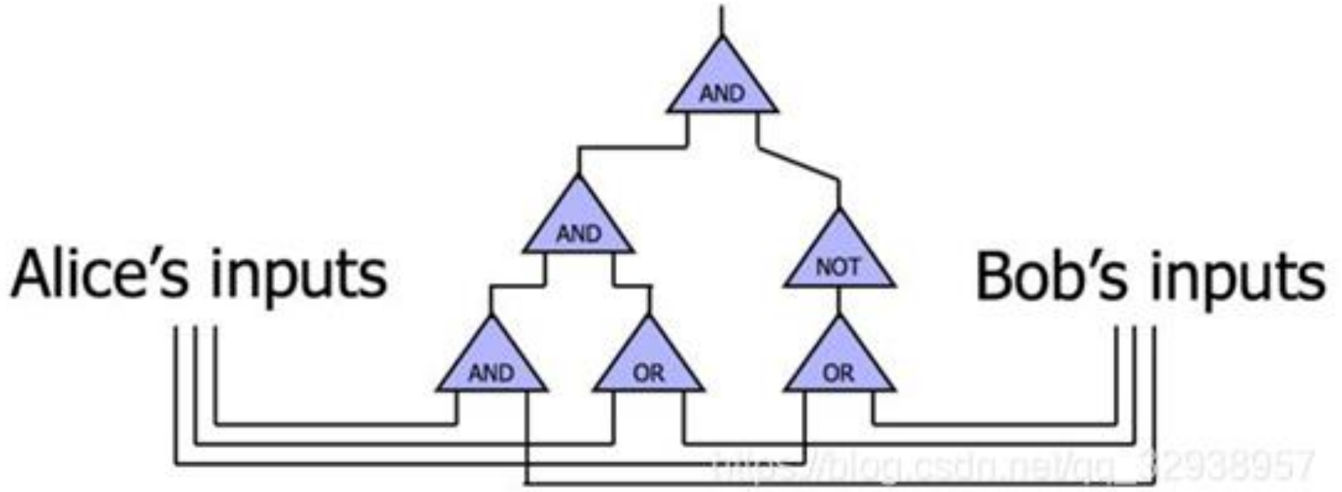
混淆电路

Garbled Circuit (GC) 又名混淆电路

混淆电路就是通过**加密**和**扰乱电路**的值来掩盖数据信息的

Private Function Evaluation

即某个参与方持有秘密数据 x，另一个参与方持有某个函数 $f(\cdot)$，GC 可以在不泄漏秘密数据以及函数的基础上进行隐私计算（这种计算范式和同态加密是一致的）



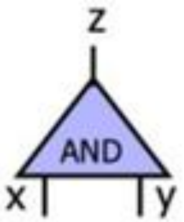
混淆电路(GC)基本思路

思路：函数即查找表

假设我们想要计算**函数XOR**，有两个输入数据$x, y\ \in \{0,1\}$，可以构建以下的查找表T：

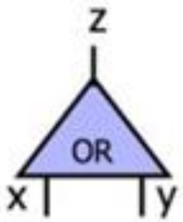
$$
T = \begin{bmatrix} 0, & 0, & 0 \oplus 0 = 0 \\ 0, & 1, & 0 \oplus 1 = 1 \\ 1, & 0, & 1 \oplus 0 = 1 \\ 1, & 1, & 1 \oplus 1 = 0 \end{bmatrix}
$$

假设有两个参与者Alice和Bob， Alice持有数据x，Bob持有y，如何把计算出的结果给到Bob?



Truth table:

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1



Truth table:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

混淆电路(GC)实现XOR函数——加密

加密+混淆过程

1、首先Alice根据x的取值范围 $\{0,1\}$ 来随机选取密钥$k_0^{(x)}, k_1^{(x)}$, 根据 y 的取值范围 $\{0,1\}$ 来随机选取密钥 $k_0^{(y)}, k_1^{(y)}$

2、通过对应的密钥加密T，得到T^*

3、重新打乱 T^* 后，Alice把打乱的 T^{**} 发送给Bob

$$T = \begin{bmatrix} 0, & 0, & 0 \oplus 0 = 0 \\ 0, & 1, & 0 \oplus 1 = 1 \\ 1, & 0, & 1 \oplus 0 = 1 \\ 1, & 1, & 1 \oplus 1 = 0 \end{bmatrix} \quad T^* = \begin{bmatrix} \mathrm{Enc}_{k_0^{(x)}, k_0^{(y)}}(0 \oplus 0 = 0) \\ \mathrm{Enc}_{k_0^{(x)}, k_1^{(y)}}(0 \oplus 1 = 1) \\ \mathrm{Enc}_{k_1^{(x)}, k_0^{(y)}}(1 \oplus 0 = 1) \\ \mathrm{Enc}_{k_1^{(x)}, k_1^{(y)}}(1 \oplus 1 = 0) \end{bmatrix} \quad T^{**} = \begin{bmatrix} \mathrm{Enc}_{k_1^{(x)}, k_0^{(y)}}(1 \oplus 0 = 1) \\ \mathrm{Enc}_{k_1^{(x)}, k_1^{(y)}}(1 \oplus 1 = 0) \\ \mathrm{Enc}_{k_0^{(x)}, k_0^{(y)}}(0 \oplus 0 = 0) \\ \mathrm{Enc}_{k_0^{(x)}, k_1^{(y)}}(0 \oplus 1 = 1) \end{bmatrix}$$

加密

打乱

混淆电路(GC)实现XOR函数——解密

ECNU

那么Bob如何解密出正确的结果呢?

1、Alice把输入 x 对应的密钥发送给Bob（假设 $x = 0, y = 1$，即发送的密钥为 $k_0^{(x)}$）

2、运行一个 2 选 1 的 OT 协议，Alice对 OT 协议输入 $k_0^{(y)}, k_1^{(y)}$，Bob对 OT 协议输入 y 得到 $k_1^{(y)}$（当 $y = 1$ 时）

3、Bob获取到了$k_0^{(x)}$以及 $k_1^{(y)}$，对 T^{**}中的第四条数据进行解密。

$$T^{**} = \begin{bmatrix} \mathsf{Enc}_{k_1^{(x)}, k_0^{(y)}}(1 \oplus 0 = 1) \\ \mathsf{Enc}_{k_1^{(x)}, k_1^{(y)}}(1 \oplus 1 = 0) \\ \mathsf{Enc}_{k_0^{(x)}, k_0^{(y)}}(0 \oplus 0 = 0) \\ \underline{\mathsf{Enc}_{k_0^{(x)}, k_1^{(y)}}(0 \oplus 1 = 1)} \end{bmatrix}$$

安全性：由于 OT 的定义决定了Bob只能获取到自己想要拿到的密钥，并不会得知其他密钥的具体值

混淆电路(GC)扩展应用和优化方法

扩展应用

1、实现布尔电路最小化单元：异、或、与

2、在此基础上支持任意的布尔运算

优化方法

Type	XOR Gate size	AND Gate size
传统 GC 方案	4λ	4λ
point-and-permute [9]	4λ	4λ
row-reduction [44]	3λ	3λ
free xor [45]	0	3λ
half gates [46]	0	2λ

GC优化: Point-and-permute

在上面例子中Bob需要遍历所有 T^{**} 中的内容尝试进行解密。那么有没有效率更高的办法?

可以把 x, y 密钥分成两部分（以 $k_0^{(x)}, k_1^{(y)}$为例），就能得到以下的结果:

$$k_0^{(x)} = \underbrace{01110110\ldots}_{\lambda\text{-bits}} \| \underbrace{1}_{|x|\text{-bits}} \qquad k_1^{(y)} = \underbrace{10101011\ldots}_{\lambda\text{-bits}} \| \underbrace{1}_{|y|\text{-bits}}$$

在此基础上，可以减少许多计算量

/03

秘密分享

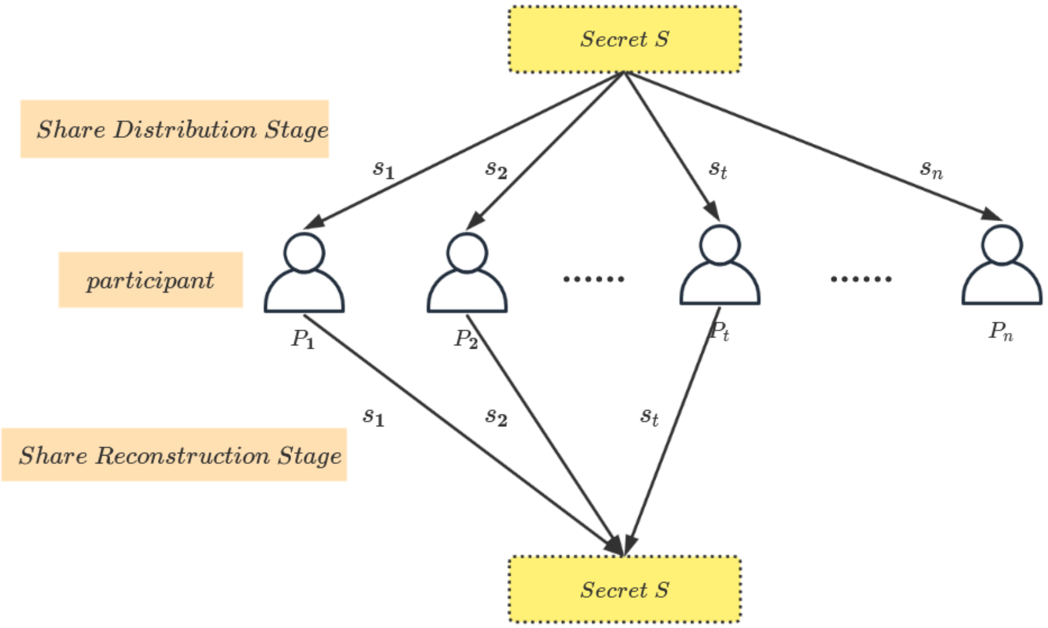
秘密分享（SS）概述

ECNU

SS 简介

秘密分享（SS）是一个非常基础的安全计算原语以及密码学工具

定义：通用的 (t, n) 阈值秘密分享协议会将某个秘密 s 拆分成 n 片，并由某个分发者分发给 n个不同参与方。其中在任意 ≥ t 个分片持有者决定重组秘密时，他们的分片可以被算法组合还原成秘密 s，而任意 t – 1 个参与者无法得到关于秘密 s 的任何信息。



(t, n) 阈值秘密分享

定义： 假设 D 是要分享的秘密的空间，D_1 是秘密分片的空间 。定义两个算法：$Shr : D \to D_1^n$ 以及 $Rec : D_1^k \longrightarrow D$，其中 $shr : D \to D_1^n$ 是将秘密分享为秘密分片，而 $Rec : D_1^k \longrightarrow D$ 表示了将秘密分片重构为秘密的算法。

这时，$(\boldsymbol{t}, \boldsymbol{n})$ −**阈值秘密分享**包含了一对算法 (Shr, Rec)，并且它们满足以下性质：

（1）正确性： 如果有$(s_1, s_2, \dots, s_n) = Shr(s)$. 那么：

$$\Pr\left[\forall k \geq t, Rec\left(s_{i_1}, \dots, s_{i_k}\right)\right] = 1$$

（2）完美安全性(Perfect Secrecy): 对于任意两个秘密$a, b \in D$以及任意可能的分片向量$\vec{v} = v_1, v_2, \dots, v_k$，其中$k < t$，则有：

$$\Pr[\vec{v} = Shr(a)|k] = \Pr[\vec{v} = Shr(b)|k]。$$

(n, n) 阈值秘密分享

平时看到的秘密分享算法多数指代 **(n, n) -阈值秘密分享**算法，即需要所有分片持有者同意重构分片才可进行秘密重构。

定义： 常见的(n, n)-阈值秘密分享算法 (Shr, Rec)遵循以下算法描述：

1、$Shr \rightarrow (s_1, s_2, \ldots, s_n)$：对于任意秘密$s$，随机在分片空间内选取$n-1$个均匀随机分片$s_i \leftarrow D_1$ (其中$1 \leq i \geq n$)，并定义$s_n = s - s_1 - \cdots s_{n-1} \in D_1$

2、$Rec(s_1, s_2, \ldots, s_n) \rightarrow s$：仅需要计算 $s \leftarrow s_1 + \cdots s_n \in D$即可

(n, n) 阈值秘密分享正确性与安全性证明

正确性证明： 根据定义，对于任意秘密s^*，我们可以计算$Shr(s^*) \rightarrow (s_1, s_2, \dots, s_n)$，那么对于正确性：

$$\mathrm{Rec}(s_1, s_2, \ldots, s_n) = s_1 + s_2 + \ldots + s_{n_1} + \underbrace{s_n}_{a - s_1 - s_2 - \ldots - s_{n-1}} \in D$$

$$= s^*$$

安全性证明： 同样，对于安全性来说，由于阈值为n，假设我们有任意某一秘密 s 的秘密分片$Shr(s^*) \rightarrow (s_1, s_2, \dots, s_n)$，那么对于任意秘密 $a, b \in D$，我们可以计算：

$$\Pr[(s_1, s_2, \ldots, s_n) = \mathrm{Shr}(a)] = \frac{1}{|D_1|^n}$$

$$\Pr[(s_1, s_2, \ldots, s_n) = \mathrm{Shr}(b)] = \frac{1}{|D_1|^n}$$

Shamir’s Secret Sharing 协议

定义： Shamir’s Secret Sharing 协议基于多项式的阈值秘密分享的实现，定义一个多项式 $p(x) = a_0 + a_1 x + \ldots + a_{t-1} x^{t-1}$其中 $a_i \in \mathbb{Z}_q$ 。

如果假定任意一方知道了 t 个多项式上的点（把这些点记为$(z_0, y_0), (z_1, y_1), \ldots, (z_{t-1}, y_{t-1})$），其中 $y_i = p(z_i)$，那么这一方可以构建如下的公式：

$$\begin{bmatrix} 1 & z_0 & \ldots & z_0^{t-1} \\ 1 & z_1 & \ldots & z_1^{t-1} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & z_{t-1} & \ldots & z_{t-1}^{t-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_{t-1} \end{bmatrix} \equiv \begin{bmatrix} y_0 \\ y_1 \\ \ldots \\ y_{t-1} \end{bmatrix} \bmod q.$$

即 $ZA \equiv Y \bmod q$。如果Z 的行列式不为零，可以求解 $A = Z^{-1}Y$，并根据拉格朗日插值来计算出多项式$p(x)$的所有系数。

Shamir’s Secret Sharing 的 Shr 和 Rec

1、$Shr(s) \rightarrow (s_1, s_2, \dots, s_n)$：现在要通过 Shamir’s Secret Sharing 来进行函数的实例化可以定义 $x = 0$（横轴坐标为零）的点为秘密值，即 $p(0) = a_0 = s$。接下来随机选取剩下的 $t-1$ 个系数即可，即 $a_1, \dots, a_{t-1} \leftarrow \mathbb{Z}_q$，这样就针对秘密 s 构建了一个完整的多项式 $p(x) = a_0 + a_1 x + \dots + a_{t-1} x^{t-1}$ 最终，让 Distributor（或者叫做 Dealer）来随机选取任意 n 个横坐标以及其相应的纵坐标发送给其他人。

2、$Rec(s_{i1}, \dots, s_{ik}) \rightarrow s$：有$k \geq t$，这时候如果假设的某个 Receiver 收到了 k 个不同的参数集合，那么就可以得到所有的**Lagrange coefficient**，进而计算出$p(x)$

Shamir’s Secret Sharing 示例

秘密分发

1. 假设秘密$S = 13$，确定$n = 5$，$t = 3$，选择模数$p = 17$

2. 生成t-1个小于或者等于p的随机数，确$a_1 = 10$，$a_2 = 2$， 同时，将S的赋值给$a_0 = 13$

3. 分别计算：

$$S_1 = (13 + 10 * 1 + 2 * 1^2)\ \bmod\ 17 = 8$$

$$S_2 = (13 + 10 * 2 + 2 * 2^2)\ \bmod\ 17 = 7$$

$$S_3 = (13 + 10 * 3 + 2 * 3^2)\ \bmod\ 17 = 10$$

$$S_4 = (13 + 10 * 4 + 2 * 4^2)\ \bmod\ 17 = 0$$

$$S_5 = (13 + 10 * 5 + 2 * 5^2)\ \bmod\ 17 = 11$$

4. 将（S_i，i）作为秘密分片分发给第 i 个人

Shamir’s Secret Sharing 示例

秘密恢复

1. 集齐任意$t = 3$个人的分片信息，加入第一个人（8，1），第二个人（7，2），第五个人（11，5）

2. 列出方程组：

$$S_1 = (a_0 + a_1 * 1 + a_2 * 1^2) \bmod 17 = 8$$

$$S_2 = (a_0 + a_1 * 2 + a_2 * 2^2) \bmod 17 = 7$$

$$S_5 = (a_0 + a_1 * 3 + a_2 * 5^2) \bmod 17 = 10$$

3. 求解方程组，得到$a_0 = 13$，$a_1 = 10$，$a_2 = 2$，则原本的秘密$S = a_0 = 13$

Shamir’s Secret Sharing 协议应用

分布式解密（使用秘密分享技术分享解密能力）

使用秘密分享技术分享解密能力

- 在理想场景中，假设存在某个用户使用非对称加密算法$(sk = x, pk = g^x)$，加密某条数据 m 并存储在一个可以被公开访问数据库中
- 假设该用户想把该密文的解密能力，交给有 n 个人的陪审团来决定，只有在不少于 t 个人决定解密时才可以完成解密操作

基于秘密分享的安全计算

安全多方计算协议：可以被类比为“电路”（Circuit），可以认为是由“门”（Gate）这一基础概念组合而成，每个门针对不同的输入数据进行安全计算，并将结果输出给下一个门进行计算。使用 [·] 符号来表示该数据是秘密分享在多方的。

- 图中Gate 协议真实的输入为：每个参与方持有**输入 1 数据分片** 以及**输入 2 数据分片**，
- 计算结束后结果为**输出数据分片**，并将结果发送给各个参与方。

基于秘密分享的安全计算——基础设定和加法

基础设定：通常基于秘密分享的MPC 是基于某个有限域$\mathbb{F}$或者某个循环群$\mathbb{G}$。后续展示的例子均为$\mathbb{G}$中的安全计算。为了更好地理解MPC，后续举例中所的计算均为**两方安全计算**以及**半诚实**的例子

加法：假设存在输入数据的秘密分片$[x]$以及$[y]$，加法门如下：

拆分计算过程：

P_0：计算$[x+y]_0 = [x]_0 + [y]_0 \in \mathbb{G}$

P_1：计算$[x+y]_1 = [x]_1 + [y]_1 \in \mathbb{G}$

验证正确性：$[x+y] = [x+y]_0 + [x+y]_1 = [x]_0 + [y]_0 + [x]_1 + [y]_1 = x + y \in \mathbb{G}$

基于秘密分享的安全计算——乘法

乘法（基于Beaver’s Triple）： 假设存在输入数据的秘密分片 $[x]$ 以及 $[y]$，乘法门如下图所示：

由于使用了$\mathbb{G}$，因此乘法操作没有逆元，所以使用Beaver提出的乘法三元组协助计算。

乘法三元组是与输入数据无关的一系列有相关关系的随机值，即$([a],[b],[c])$,其中$c = a \cdot b$，这些值分布在所有参与者之中。

基于秘密分享的安全计算——乘法

在 Beaver triple 生成结束后$([a],[b],[c])$,其中$c=a\cdot b$):

P_0: 持有$[a]_0,[b]_0,[c]_0 \in \mathbb{G}$ $\Leftarrow$ 不会得知 a,b,c 具体的值

P_1: 持有$[a]_1,[b]_1,[c]_1 \in \mathbb{G}$ $\Leftarrow$ 不会得知 a,b,c 具体的值

使用乘法三元组的乘法协议:

1、两个参与方分别计算$[x-a]_{\{0,1\}}$ 以及 $[y-b]_{\{0,1\}}$

2、共同重构出$[x-a]$以及$[y-b]$（公开值）

3、即可得到:

$$xy=(x-a+a)\cdot(y-b+b)=\underbrace{(x-a)(y-b)}_{\text{均为公开值}}+\underbrace{(x-a)b}_{(x-a)[b]_0+(x-a)[b]_1}+\underbrace{(y-b)a}_{(y-b)[a]_0+(y-b)[a]_1}+\underbrace{ab}_{[c]_0+[c]_1}$$

基于秘密分享的安全计算——乘法

接下来，各个参与方可以：

P_0：计算$[xy]_0 = (x-a)(y-b) + (x-a)[b]_0 + (y-b)[a]_0 + [c]_0 \in \mathbb{G}$

P_1：计算$[xy]_1 = (x-a)[b]_1 + (y-b)[a]_1 + [c]_1 \in \mathbb{G}$

在上述加法和乘法的基础上，我们就可以执行算数电路上的几乎任意计算。实际上，MPC 除了算数电路（Arithmetic Circuit）之外，也存在布尔电路，并且存在算数电路与布尔电路转换的协议

基于秘密分享的安全计算

问题：你可以想想上述协议是否可以用作浮点数值的安全计算吗？为什么？

思路：上述基于秘密分享的安全计算仅支持整数的安全计算。我们通常在计算机中使用整数来表示浮点数，例如我们想要使用128-bit大整数来表示浮点数12.8，那么我们会首先使用128-bit 中的16比特来表示小数部分，剩下部分表示整数，即$12.8 \rightarrow (12.8) \cdot 2^{16}$。所以，在计算浮点的$[xy]$时，我们要计算整数的$[xy] \cdot 2^{16} \cdot 2^{16}$，这里我们会发现结果会破坏浮点数的表示形式，因此需要删除最后 16 个 bit 以完成浮点数乘法的安全计算，这类协议被称作 Truncation。

/04

不经意传输

不经意传输概述

不经意传输（oblivious transfer）最初是在1981由Michael O.Rabin提出

- 在这种不经意传输中，发送者Alice发送一条消息给接收着Bob，而Bob以1/2的概率接收到信息
- 在结束后Alice并不知道Bob是否接收到了信息，而Bob能确信地知道自己是否收到了信息

2选1不经意传输（1 out 2 oblivious transfer），1985年被提出

- 在这种形式的不经意传输模型中，Alice每次发两条信息（m_1、m_2）给Bob。
- Bob提供一个输入，并根据输入获得输出信息，在协议结束后，Bob得到了自己想要的那条信息（m_1或者m_2），而Alice并不知道Bob最终得到的是哪条

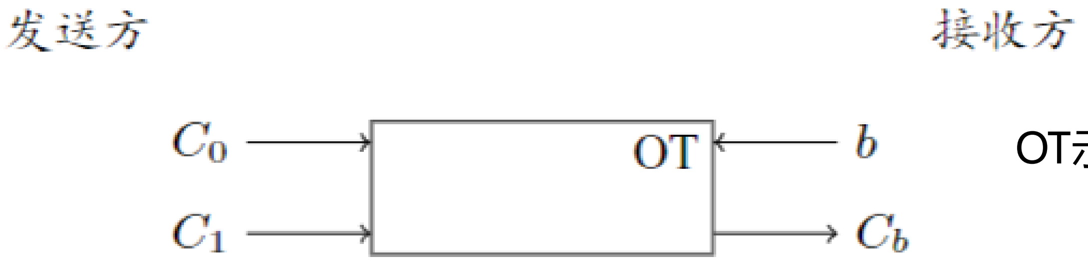
1986年，Brassard等人将2选1不经意传输拓展为k选1

不经意传输概述

OT 定义

不经意传输（Oblivious Transfer, OT）：若把加密/解密算法理解为发送方和接收方之间的一条“安全”的通信信道，发送方使用加密算法对消息 m 进行保护，接收方使用解密算法恢复消息 m

定义： OT 协议理解为发送方和接收方之间的一对通信信道 $(C_0\ C_1)$，其中发送方可以通过信道 C_0 和 C_1发送任意的消息，而接收方仅可以根据选择比特 b 接收信道 C_b 中的消息，同时发送方无法得知接收方选中的信道。



OT示例图

- OT 协议可以使接收方在不经意间获得发送方的某些信息，保护发送方和接收方的隐私。

基础 2-选-1 OT协议

ECNU

不经意传输

基于公钥密码学的基础2-选-1OT协议（Alice是发送方，Bob是接收方）

1. Alice生成公钥d、私钥e，选择两个随机数s_0, s_1以及公钥d发送给Bob
2. Bob按照接收数据序号 i 选择 s_i，并生成一个随机数 s，使用Alice的公钥 d 加密得到 $Enc(s)$

   计算 $s' = s_i + Enc(s)$ 发送给Alice
3. Alice计算 $s_0' = s' - s_0, s_1' = s' - s_1$，并使用私钥 e 解密 s_0', s_1' 得到 $Dec(s_0'), Dec(s_1')$；

   计算 $m_0' = m_0 \oplus Dec(s_0')$，$m_1' = m_1 \oplus Dec(s_1')$ 发送给Bob
4. Bob选择 m_i'，计算 $m_i' \oplus s$ 获取 m_i

OT 的应用——电子投票（承诺方案）

应用场景：电子投票（承诺方案）

问题描述：在投票场景中，所有参与方都需要预先决定投票的结果，然后再统一表决。但是如何保证参与方不会临时修改投票的结果呢?

解决方法：让发送方在不公开消息的同时向接收方承诺某个消息，在一定时间后，发送方公开所承诺的消息并证明此消息没被篡改

OT 的应用——等值测试

应用场景：等值测试

问题描述：著名的百万富翁问题是两个富豪想在不透露自己的财产的情况下，比较谁到底更加富有。假如我们考虑这个问题的弱化版本，如何比较两个富翁所具备的财产是否相等?

解决方法： 基于OT 协议，可以构造等值测试协议。

OT 的应用——等值测试

基于OT的等值测试协议：

1、发送方持有比特串：$x \in \{0,1\}^\ell$

接收方持有比特串： $\mathrm{y} \in \{0,1\}^\ell$；

2、接收方以比特串 y 作为选择比特与发送方建立 ℓ 个 ROT 分别：

$$\{(r_{i,0}, r_{i,1})\}_{i\in[\ell]} \text{ 和 } \{(y_i, r_i)\}_{i\in[\ell]}$$

3、发送方计算 $m_S = \sum_{i\in[\ell]} r_{i,x_i}$ 并发送，

接收方计算$m_R = \sum_{i\in[\ell]} r_i$ 并比较 m_S 与 m_R是否相等

/05

同态加密

同态加密（HE）概述

背景：数据安全处理关注如何在处理数据的同时不破坏其安全性质，一类典型的应用是云服务。

如何能够不解密就进行数据处理呢？->同态加密

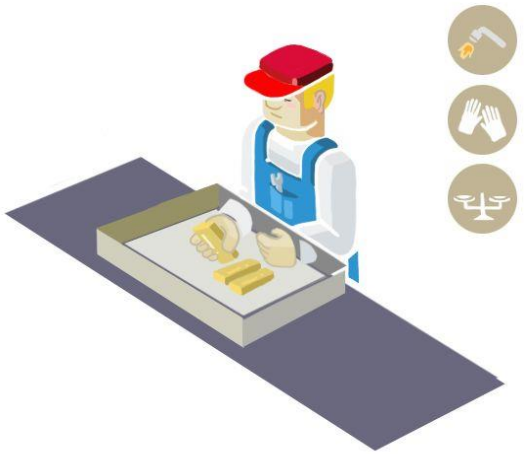
- 使用**同态加密（HR）**对数据进行加密后，他人可以不经解密，直接对加密数据进行处理，并且保证数据处理结果经解密后是正确的
- 换言之，对于同态加密来说，下面两个流程得到的结果是等价的
- **流程1**：数据加密 ->处理加密数据->得到加密后处理结果->解密处理结果->得到解密后处理结果
- **流程2**：数据加密->数据解密->处理数据->得到处理结果

- **同态加密**起源于私密同态 (Privacy Homomorphism)

同态加密（HE）概述

同态加密提供了一种对加密数据进行处理的功能

- 盒子：加密算法
- 盒子上的锁：用户密钥
- 将金块放在盒子里锁上：将数据用同态加密方案进行加密
- 加工：同态特性，在无法取得数据下直接对加密结果处理
- 开锁：对结果进行解密，直接得到处理后的结果



同态加密（HE）四个算法组件

同态加密是一种高级密码学原语，通常由4个算法构成：

1、密钥生成算法 KeyGen

2、加密算法 Enc

3、解密算法 Dec

4、运算算法 Eval

非对称加密HE 方案

1、加密

用公钥（Public Key, pk）对明文 m 进行加密得到密文 $c = Enc_{pk}(m)$

2、解密

利用私钥（Secret Key, sk）可以对密文 c 进行正确解密，得到原始明文 $m = Dec_{sk}(c)$

对称加密HE 方案

1、加密/解密

用 sk 进行加解密:

$$Dec_{sk}\left(Enc_{sk}(m)\right) = m$$

特殊的是，HE 方案的加密算法 Enc（即明文和密文之间）具备**同态性质**:

$$Enc(m_1 \odot m_2) = Enc(m_1) \odot Enc(m_2)$$

例如:

- 加法同态：E(a+b) = E(a) + E(b)
- 乘法同态：E(a×b) = E(a) × E(b)

同态加密应用

- HE 在许多实际场景中被广泛应用，包括匿名电子投票、安全外包计算、隐私保护机器学习等
- 还被用作其它密码学方案的基础构建工具，例如 MPC、零知识证明、函数加密等

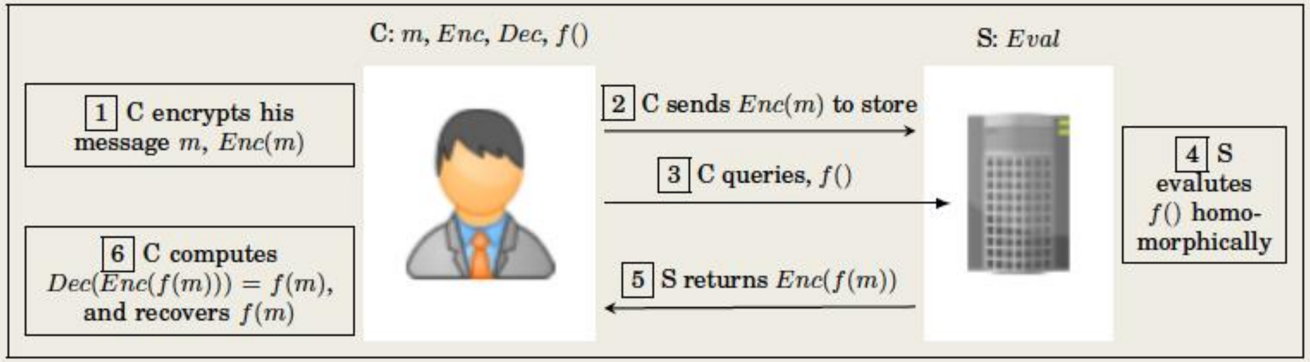


Fig. 1: A simple client-server HE scenario, where C is Client and S is Server

- 云计算: 使用同态加密，让云来对加密数据进行直接处理，并将处理结果返回
- 同态加密现在最需要解决的问题在于：**效率**
  - 加密数据的处理速度：密文操作更长时间
  - 精准度会变差：误差传递问题
  - 加密方案的数据存储量：存储空间问题

同态加密分类

- **部分同态加密（Partially HE,PHE）：** 仅支持一种同态运算（加法或乘法）的加密方案，又称为**半同态加密方案**
- **同态加密（Fully HE, FHE）：** 同时支持加法、乘法同态运算的加密方案
- **层级同态加密（Somewhat/Level HE）：** 支持一种同态运算的同时，还支持另一种次数受限的同态运算的加密方案，又称为**Somewhat**

部分同态加密

- 部分同态加密方案虽然仅能够支持一种同态计算，但也能够完成一些常见的安全计算，例如隐私数据的求和或求积。
- 一些著名PHE 方案已经在实际生活中被采用，包括 RSA ， Elgamal ， Paillier 等
- PHE 方案和普通的HE 方案相同，由4个算法组成，其运算算法 Eval 主要包含两类：

1、密文和密文的运算

2、密文和明文的运算

部分同态加密密文运算

密文和密文的运算

某些加法同态方案中，密文 $a = Enc(m_1)$ 和密文 $b = Enc(m_2)$ 相乘得到的结果密文：

$$c_{out} = a \cdot b = Enc(m_1 + m_2),$$

即 c_{out} 的底层明文是 a 和 b 对应的明文之和 $m_1 + m_2$

密文和明文的运算

某些加法同态方案中，密文 $a = Enc(m_1)$ 和明文 m_2 运算后得到的结果密文：

$$c_{out} = a^{m_2} = Enc(m_1 \cdot m_2),$$

即 c_{out} 的底层明文是 a 和 b 对应的明文之积 $m_1 \cdot m_2$。

部分同态加密——Elgamal 方案

Elgamal： 一个基于离散对数困难问题（Discrete Logarithm Problem, DLP）的新型**公钥加密算法**

- 原始的 Elgamal 算法，也被称为 Textbook/Plain Elgamal
- 是一种**乘法同态**的 PHE 方案
- 同态特性来自密文块 $m \cdot y^r$

Exponential Elgamal： 有时也被称为 Lifted Elgamal。是目前实际广泛应用的 Elgamal 变体。其在加密时，增加了对原始明文 m 进行了一个映射处理过程：

$$m' = g^m \, mod \; n \in \mathbb{G},$$

其中 m' 是映射后的明文，也是群$\mathbb{G}$中的一个元素。这个映射是一个典型的**单向函数**（one-way function）

Exponential Elgamal 定义

密钥生成算法 $KeyGen(1^\lambda) \rightarrow (sk, pk)$

1、使用群生成算法，生成一个阶为 n ，生成元为 g 的循环群 $\mathbb{G}$ ；

2、从整数群 $\mathbb{Z}_n^*$ 中随机选取一个元素 x ，计算：

$$y = g^x \bmod n \in \mathbb{G};$$

3、输出**私钥**：$sk = x$

公钥：$pk = (\mathbb{G}, n, g, y);$

Exponential Elgamal 定义

加密算法 $\mathrm{Enc}(m) \to c$

1、从整数群 $\mathbb{Z}_n^*$ 中随机选取一个元素 r；

2、计算得到密文：

$$c = (g^r \bmod n,\ g^m \cdot y^r \bmod n) = (c_1, c_2);$$

解密算法 $\mathrm{Dec}(c) \to m$

1、用私钥 $sk = x$，计算：

$$c_2/c_1^x = g^m \cdot y^r \cdot g^{r(-x)} = g^m \cdot (g^x)^r \cdot g^{-xr} = g^m;$$

2、使用离散对数求解算法从:

$$g^m \bmod n$$

求解明文 m；

Exponential Elgamal （加法）同态运算

同态运算步骤：

密文 $a = Enc(m_1)$ 和密文 $b = Enc(m_2)$

1、**底层明文相加，密文和密文的乘法运算** $Eval_1(a,b)$:

$$a \cdot b = Enc(m_1) \cdot Enc(m_2) = (g^{r_1}, g^{m_1} \cdot y^{r_1}) \cdot (g^{r_2}, g^{m_2} \cdot y^{r_2})$$

$$= (g^{r_1+r_2}, g^{m_1+m_2} \cdot y^{r_1+r_2}) \; mod \; n = Enc(m_1 + m_2 \; mod \; n)$$

2、**底层明文相加，密文和明文的运算**$Eval_2(a,m_2)$:

$$a \cdot g^{m_2} = (g^{r_1}, g^{m_1} y^{r_1} \cdot g^{m_2}) = (g^{r_1}, g^{m_1+m_2} y^{r_1}) = Enc(m_1 + m_2 \; mod \; n)$$

3、**底层明文相乘，密文和明文的运算**$Eval_3(a,m_2)$:

$$a^{m_2} = (g^{r_1}, g^{m_1} y^{r_1})^{m_2} = (g^{r_1 m_2}, g^{m_1 \cdot m_2} y^{r_1 m_2}) = Enc(m_1 \cdot m_2 \; mod \; n)$$

部分同态加密——Paillier 方案

Paillier 方案：1999 年，Paillier 提出了基于复合剩余问题的另一种PHE加密方案，是一种加法同态方案

- **复合剩余问题：**给定一个整数 $a \in \mathbb{Z}_{n^2}^*$，是否存在一个整数 $x \in \mathbb{Z}_{n^2}^*$，使得：
  $x^n \equiv a \bmod n^2$ 成立。

Paillier 方案定义

密钥生成算法 $KeyGen(1^\lambda) \to (sk, pk)$

1、调用模数生成算法$GenMod(1^\lambda)$，得到 n, p, q，

其中 $n = pq$

2、输出**私钥** $sk = (p-1)(q-1)$，**公钥** $pk = n$

加密算法 $\mathrm{Enc}(m) \to c$

1、从整数群 $\mathbb{Z}_n^*$ 中随机选取一个元素 r；

2、计算得到密文：

$c = (1+n)^m r^n mod\ n^2$；

解密算法 $\mathrm{Dec}(c) \to m$

1、使用私钥 sk 进行解密，计算的到明文:

$$\frac{\left(c^{sk}\ mod\ n^2\right) - 1}{\left((1+n)^{sk}\ mod\ n^2\right) - 1} = \frac{m \cdot sk \cdot n + 1 - 1}{sk \cdot n + 1 - 1} = \mathrm{m\ mod\ n}$$

Paillier 方案（加法）同态运算

同态运算步骤：

密文 $a = Enc(m_1)$ 和密文 $b = Enc(m_2)$

1、底层明文相加，密文和密文的乘法运算 $Eval_1(a, b)$:

$$a \cdot b = Enc(m_1) \cdot Enc(m_2) = (1+n)^{m_1} r_1^n . (1+n)^{m_2} r_2^n = (1+n)^{m_1+m_2} (r_1 r_2)^n \ mod\ n^2$$
$$= Enc(m_1 + m_2\ mod\ n)$$

2、底层明文相加，密文和明文的运算$Eval_2(a, m_2)$

$$a \cdot (1+n)^{m_2} = (1+n)^{m_1} r_1^n \cdot (1+n)^{m_2} = (1+n)^{m_1+m_2} r_1^n \ mod\ n^2 = Enc(m_1 + m_2\ mod\ n)$$

3、底层明文相乘，密文和明文的运算$Eval_3(a, m_2)$

$$a^{m_2} = (1+n)^{m_1 \cdot m_2} (r_1^{m_2})^n = Enc(m_1 \cdot m_2\ mod\ n)$$

全同态加密

全同态加密（FHE）： 从现代密码学伊始便备受追捧，被誉为“密码学圣杯”。FHE 能够在密文上执行任意的计算，以支持实现通用的安全计算。

FHE 突破： Gentry09 方案的提出， FHE 才逐渐从一个优美的概念发展成为了一个可用的工具。

- 截止2023 年，根据代表性方案的出现时间，大致可以将 FHE 划分为四代

四代全同态加密

第一代

代表性方案：理想格和近似最大公约数 FHE 方案

特点：受限于快速增长噪声，影响效率和安全性，几乎无法实际使用

第二代

代表性方案： Brakerski-Vaikuntanathan 和 Brakerski 的研究

特点：更好的噪声控制技术、降低了同态运算中的噪声增长速率。

第三代

代表性方案： Gentry 等人其提出的 GSW 方案

特点：通常比第二代的FHE 方案性能更低，但安全性更高——因为能够基于更弱的困难假设。

第四代

代表性方案： CKKS 及其后续方案

特点：支持浮点数的同态计算，非常适用于隐私保护机器学习。更强的应用能力，性能非常优异。

全同态加密发展

主要障碍：望而却步的计算、内存和通信消耗

虽然新方案和新的算法优化技术，让最新的 FHE 方案相较于首个FHE 方案 Gentry09 在性能上已经提升了至少1,000,000 倍，但是 FHE 的密文操依然比直接的明文操作慢至少 3 个数量级

新的优化技术

- 基于硬件加速，例如基于CPU 拓展指令集、GPU 、FPGA 或 ASIC 的 FHE 加速技术
- FHE 依赖的基础技术优化，例如NTT（Number Theoretic Transform）加速技术的研究，它是高性能的 FHE 方案实现中必不可缺的基础组件

工业界发展: OpenFHE 、Microsoft Seal 、IBMHElib、隐语等优秀开源项目发布、不断改进

THANKS