

CS5322 Database Security

Chapter 1: DAC

- Access control
 - 确保所有对象的访问都是经过授权的
 - a scheme for mapping subjects to allowed actions
 - 核心三要素
 - 对象：需要保护的系统资源
 - 主体：请求访问资源的实体
 - 访问模式：所允许的操作
 - 访问控制的要求
 - 不可绕过：安全机制必须是强制性的，不能被轻易绕开；
 - 强制执行组织策略：访问控制需要能够实施和强制执行组织的安全规定；
 - 强制执行最小权限限制
 - 最小权限原则
 - 每一个主体应该只被授予完成其工作所必需的最小访问权限集合，访问权限应该在需要时被添加，在任务完成后被丢弃（或收回）；
 - 这可以限制因意外或错误操作所导致的损害，并且它能限制可能被攻破的特权程序的数量。
 - Access control类型
 - Mandatory access control强制访问控制：系统决定谁可以访问什么，用户不能修改它；
 - Discretionary access control自主访问控制：如果一个用户拥有某个访问权限，他/她可以将该权限授予另一个用户；
 - Role-based access control基于角色的访问控制：系统将访问权限授予角色，用户被分配到不同的角色；
- Discretionary Access Control
 - DAC策略基于以下内容，来管理主体对对象的访问：
 - 主体的身份标识
 - 对象的身份标识
 - 权限

- 自主性: 一个主体可以自行决定将访问权限授予其他主体;
- 一个用户可以将权限授予另一个用户，后者又可以将其授予其他人，这是通过 `GRANT` 来完成的;
- DAC操作
 - `GRANT` operation

代码块

```

1  GRANT PrivilegeList | ALL PRIVILEGES
2  ON Relation | View
3  TO UserList | PUBLIC
4  [WITH GRANT OPTION]

```

- `GRANT` operation example

代码块

```

1  Bob:
2      GRANT select, insert ON Employee TO Ann WITH GRANT OPTION;
3  Ann(从Bob那里)接收到了对Employee表的select和insert权限，她可以将这些权限
   授予其他人；
4  Ann:
5      GRANT select (Name), insert ON Employee TO Jim;
6  Jim(从Ann那里)接收到了Employee表中Name列的select权限和insert权限，但他不
   能将这些权限再授予其他人，因为无WITH GRANT OPTION;

```

- Checking `GRANT` operation

- 系统有一个授权目录 (authorization catalog)，用于跟踪每个用户可以授予哪些权限，每当一个用户执行 `GRANT` 操作时，系统会检查该权限是否确实可以被授予，如果不能，相应的权限将不会被授予；

- `REVOKE` operation

代码块

```

1  REVOKE PrivilegeList | ALL PRIVILEGES
2  ON Relation | View
3  FROM UserList | PUBLIC

```

- 通常来说，由于WITH GRANT OPTION的存在，撤销操作可能会变得很棘手；（例如，Bob 授权给 Ann，然后 Ann 又授权给 Jim，之后 Bob 从 Ann 那里撤销了权限，...）

- Recursive Revocation
 - 当一个权限被撤销时，所有由这个被撤销的权限直接或间接授予出去的权限，也必须被一并撤销；
 - 记录下每一次权限授予的时间戳，当一个权限被撤销时，使用这些时间戳来决定是否需要进行级联撤销；（这是因为可能一个用户权限由多个用户授予，看撤销操作的那步，他是否能继续授予其他权限）
 - 级联撤销的缺点：相当具有破坏性
 - 系统需要小心地撤销一个可能很长的权限列表，这会带来相当大的开销；
 - 撤销操作可能会使大量应用程序失效，从而中断依赖这些权限的服务；
- Non-Cascading Revocation
 - 当用户A撤销对用户B的一个授权时，B曾授予他人的相关权限会被重述，就好像它们是由用户A授予的一样；

- 视图/基于内容的授权

- 在关系型数据库管理系统 (RDBMS) 中，视图通常被用来支持基于内容的访问控制：
 - 定义一个包含主体S被允许访问的元组（或统计信息）的视图，将select/insert/update等权限授予该视图，而不是授予底层的表；
 - 对视图的查询，会通过视图组合被转换为对基表的查询

代码块

```

1  CREATE VIEW Vemp AS SELECT * FROM Employee WHERE Salary < 20000;
2  GRANT Select ON Vemp TO Ann;
3
4  Ann的查询: SELECT * FROM Vemp WHERE Job = 'Programmer';
5  用AND组合: SELECT * FROM Employee WHERE Salary < 20000 AND Job =
   'Programmer';

```

- Authorizations on Views

- 创建视图的用户被称为视图定义者
- 视图定义者在视图上获得的权限取决于：
 - 定义者在基表上拥有的权限；
 - 视图的语义，即它在基表关系上的定义；
- 视图定义者不会收到那些无法在视图上执行的操作所对应的权限，例如，`alter` (修改结构) 和 `index` (创建索引) 不适用于视图；

- DAC局限性

- 全局策略：DAC允许用户自行决定其数据的访问控制策略，而不管这些策略是否与全局策略相一致；
- 信息流：信息可以从一个对象被复制到另一个对象，原始数据的所有者可能无法控制其副本；

Chapter2: Oracle VPD and MAC

- Fine-grained access control
 - 为不同角色的不同查询都创建视图——视图会太多
 - 解决方案一：让应用程序来处理
 - 不在数据库里为每个学生创建视图，而是让应用程序（比如一个成绩查询网站）来检查当前登录用户的身份，然后由应用程序来生成带有特定条件的查询语句；
 - 问题：应用程序的代码可以看到Grades表中的所有数据，如果这个应用程序被黑客攻击或劫持，那么整张Grades表的数据就可能被泄露；
 - 更希望将访问控制的逻辑放在数据库内部，而不是依赖于外部的应用程序；
 - 解决方案二：参数化视图

代码块

```
1  CREATE VIEW StudentGrades AS SELECT * FROM Grades WHERE student =
$username
```

- 问题：应用程序需要为不同角色的用户编写不同的查询，对学生，程序需要查询 StudentGrades 视图，对讲师，程序需要查询原始的 Grades 表，如果有一种对授权透明的方法来避免这种复杂性，会更加方便；
- 解决方案三：视图替换
 - 不为用户创建视图，而是在“幕后”修改用户提交的查询，以此来实现访问控制；(Oracle VPD)

Oracle VPD

- Oracle VPD
 - 有时也被称为Oracle行级安全(Row-Level Security, RLS)或细粒度访问控制(Fine Grained Access Control, FGAC)；
 - Idea
 - 将安全策略与数据库对象（如表、视图）关联起来；
 - 透明地给 WHERE 子句添加谓词；
 - 这些谓词是由策略中用户定义的函数生成的
 - 函数可以访问会话参数来核验身份，例如当前登录的用户名；

- Policy function

- 例1: 每个人都只能看到成绩为 A+ 的那些行

代码块

```
1  CREATE FUNCTION check_grade(
2      v_schema IN VARCHAR2,      v_obj IN VARCHAR2 )
3  RETURN VARCHAR2 AS condition VARCHAR2(200);
4  BEGIN
5      condition := 'grade = ''A+'''';
6      RETURN condition;
7  END check_grade;
8
9  --v_schema 为数据库名CedricDB
10 --v_obj 为表格名Grades
11 --condition 为要添加的谓词
```

- 例2: 每个学生只能看到自己的成绩

代码块

```
1  CREATE OR REPLACE FUNCTION check_grade(
2      v_schema IN VARCHAR2,      v_obj IN VARCHAR2)
3  RETURN VARCHAR2 AS condition VARCHAR2(200);
4  BEGIN
5      condition := 'student = SYS_CONTEXT(''grade_app'',
6      ''stu_name'')';           RETURN condition;
7  END check_grade;
8
9  -- SYS_CONTEXT(''grade_app'', ''stu_name'')返回在grade_app上下文中的
10 stu_name参数
```

- SYS_CONTEXT

- 检索Oracle环境中的参数
- SYS_CONTEXT语法

代码块

```
1  SYS_CONTEXT(namespace, parameter, [length])
2
3  -- namespace是一个内置的或用户创建的上下文环境, Oracle提供了一个
   名为USERENV的内置命名空间, 可提供关于当前Oracle会话的信息
4  -- parameter是上下文环境中的一个属性, 必须事先使用
   DBMS_SESSION.set_context 过程进行设置
```

5 -- length是可选的，指定参数值的字节长度

- USERENV: built-in namespace部分参数示例
 - SESSION_USER: 登录时数据库用户的名称
 - SESSION_USERID: 登录时数据库用户的ID（标识符）
 - DB_NAME: 数据库的名称
 - ISDBA: 如果用户被认证为拥有 DBA 权限，则返回TRUE
- 例3: 每个学生只能看到她自己的成绩，DBA可以看到所有的成绩

代码块

```
1 CREATE OR REPLACE FUNCTION check_grade(
2     v_schema IN VARCHAR2,      v_obj IN VARCHAR2)
3     RETURN VARCHAR2 AS condition VARCHAR2(200);
4 BEGIN
5     IF ( SYS_CONTEXT('USERENV', 'ISDBA') ) THEN
6         RETURN ''; --无谓词，即无限制
7     ELSE
8         RETURN 'student = SYS_CONTEXT(''grade_app'',
9             ''stu_name'')';
10    END IF;
11 END check_grade;
```

代码块

```
1 BEGIN
2     DBMS_RLS.DROP_POLICY( -- 删除旧策略
3         object_schema => 'CedricDB',
4         object_name   => 'Grades',
5         policy_name   => 'check_grade_policy');
6
7     DBMS_RLS.ADD_POLICY (
8         object_schema => 'CedricDB',
9         object_name   => 'Grades',
10        policy_name   => 'check_grade_policy',
11        policy_function => 'check_grade',
12        update_check   => TRUE);
13        -- 扩大检查的范围，不仅检查SELECT，也检查UPDATE和DELETE，只能更新
14        -- 自己的成绩
15    END;
```

- Policy

- 把策略函数加到对应的表格上

代码块

```

1  BEGIN
2      DBMS_RLS.ADD_POLICY (
3          object_schema    => 'CedricDB',
4          object_name     => 'Grades',
5          policy_name    => 'check_grade_policy',
6          policy_function => 'check_grade');
7  END;

```

- Column-level VPD

- 例子：能看到自己的成绩和别人的姓名，可以复用例3的策略函数，但需要更改策略

代码块

```

1  BEGIN    DBMS_RLS.ADD_POLICY (
2      object_schema    => 'CedricDB',
3      object_name     => 'Grades',
4      policy_name    => 'check_grade_policy',
5      policy_function => 'check_grade',
6      sec_relevant_cols => 'grade', --告诉数据库：这条check_grade_policy
策略是一条懒人策略，请不要一上来就执行它，请你先看看用户到底要查询哪些列，只有当
用户请求的数据包含了grade这个敏感列时，你才需要激活我的策略函数，并把函数返回的
WHERE条件加上去；
7      sec_relevant_cols_opt  => dbms_rls.ALL_ROWS ); --保留所有行，但敏感列
grade里的值设为'NULL'
8  END;

```

- 如果一张表有多个策略，会通过 AND 语法强制执行
- VPD问题
 - 未考虑聚合函数带来的不一致性
 - 对表T应用了一个策略，那么该策略的函数F就不能访问表T，因为有潜在的递归风险

MAC

- 在MAC中，由一个系统级的全局策略来决定谁有权访问，单个用户不能更改这个策略；
- 多级安全（Multi-level security, MLS）是MAC的代表性模型
 - 与对象相关的三个关键概念：安全级别，分区和标签
 - 每个对象都有一个安全级别，以表明其敏感度

- 例如：非涉密 (unclassified)，内部 (confidential)，机密 (secret)，绝密 (top secret)；
- 每个对象可能属于某些分区（即，类别）
 - 例如：财务，制造，农业；
- 对象的安全级别和分区共同构成了该对象的标签；
- 例子：一个标签 (机密， {财务， 欧洲}) 意味着：该对象涉及欧洲的财务，其安全级别是机密；
- 每个主体也有一个标签
 - 例子：如果一个用户有一个标签 (绝密， {财务， 亚洲})，那么她拥有访问关于财务和/或亚洲的绝密文档的许可；
- 多级安全规则：一个主体S可以访问一个对象O，前提是S的许可不低于O的安全级别，并且S有权访问O所在的分区（即S的分区包含O的分区）；
 - 符号化表示： $L(O) \subseteq L(S)$
- BLP model：提出了一个MLS的形式化数学模型
 - 禁止向上读：主体S可以读取对象O，当且仅当 $L(O) \subseteq L(S)$
 - 禁止向下写：主体S可以在对象O写，当且仅当 $L(S) \subseteq L(O)$ (为了防止非法的信息流动)
- Applying MAC to databases
 - 给每个数据库对象和主体都附上一个标签，基于这些标签执行访问控制；
 - 可能的访问控制粒度
 - 为每张表贴一个标签
 - 为每个元组（行）贴一个标签
 - 为元组中的每个值一个标签
 - 一个关系 $R(A_1, A_2, \dots, A_d)$ 可以被扩展到一个多级安全关系 $R(A_1, C_1, A_2, C_2, \dots, A_d, C_d, TC)$
 - C_i 是一个属性，代表与 A_i 相关联的安全级别；
 - TC 代表一个元组所有属性中最高的那一个安全级别；
 - 高等级用户可以获取到低或同等级 A_i ，同一元组中不可获取 A_i 标记为 $NULL$
 - 但是这个 $NULL$ 不能为主键，如果等级不够获取主键，那这一行都不能获取；
- Polyinstantiation多实例化
 - 允许一个实体（比如'Dave'）在数据库中同时拥有多个、处于不同安全级别的实例；
 - 用“原始主键+所有安全等级”做新的主键，是最安全的

Chapter 3: Statistical Database

- 访问控制确保了直接对对象的访问是授权的，但不能防止对敏感信息的推断；
 - 使用statistical database减轻推断攻击
- Statistical database
 - Idea:
 - 不提供对详细记录的访问权限，只提供SUM, MEAN, MEDIAN, COUNT, MAX, 和 MIN等函数提供记录的统计信息；
 - 额外的措施来进行推断控制
 - Query auditing
 - 核心思想：跟踪用户对统计数据库的查询，以判断查询是否会泄露敏感信息
 - 两种方法
 - 在线审计：实时跟踪查询，并拒绝那些不安全的查询；（效率更低）
 - 离线审计：保留所有查询的日志，并运行离线测试来检查是否已经发出了不安全的查询；
 - Data perturbation
 - 核心思想：修改原始数据，基于修改后的数据来回答用户的查询，而不是原始数据；
 - Output perturbation
 - 核心思想：向每个查询的答案中注入噪音，以隐藏敏感信息
 - 例子：查询：“有多少学生得了A+？” —— 答案：[0, 5]
 - Query auditing
 - query set size control
 - query auditing的最简单形式
 - 核心思想：对每个查询的选择性（满足该查询条件的元组数量）强制施加要求
 - 不能低于K（最低选择性数）
 - query set overlap control
 - 要求：对任意两个查询，他们的重叠元组最多只能有r个
 - 检查所有“两两查询”的组合是不够的，如果我们检查所有“k查询”($k > 2$)的组合重叠呢？如果攻击者利用比重叠更复杂的查询关联，信息仍可能泄露；结论：查询集重叠控制并不能真正起作用。
 - More advanced auditing

- 记录下用户发出的所有查询，使用一个高级算法来判断这些查询是否能揭示任何特定的元组，但这只有在查询被限制为特定类型时才能做到；
- 针对 SUM 查询可以构建一个线性系统 $Q = WX$
 - X 为查询元素的向量
 - $w_{ij} = 1$ ，如果 q_i 查询中包含了第 j 个元组，否则 $w_{ij} = 0$
 - 如果这个线性系统可以唯一地确定某个值为 x_k 的解，那么意味着这些查询能够揭示 x_k （信息泄露）
 - x_i 具有唯一解的充要条件是：矩阵 W 的第 i 列向量 w_i ，不能被其他所有列向量 $\{w_1, w_2, \dots, w_n\}$ 线性表示；
- 非线性查询很难设计类似的系统
- Inference from denial of queries
 - 为了避免从拒绝访问中推断信息，应该确保审计算法是数据独立的；
 - 算法应该考虑所有可能的数据库表，只要存在一种可能的表，在那个表上查询会泄露信息，就拒绝该查询；
- Data perturbation
 - 泛化
 - idea：修改元组让它们更难区分
 - 例1：

代码块

```

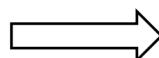
1 COUNT(*) WHERE Age = 20 AND Gender = 'F';
2 -- Query result: [0, 2]
3
4 SUM(Grade) WHERE Age = 20 AND Gender = 'M';
5 -- Query result: [0, 150]

```

- 例2：

| Name | Age | Gender | Program | Grade |
|-------|-----|--------|---------|-------|
| Alice | 20 | F | CS | 100 |
| Cathy | 21 | F | CS | 90 |
| Bob | 21 | M | IS | 80 |
| Dave | 20 | M | IS | 70 |

Original Table T



| Name | Age | Gender | Program | Grade |
|------|-------|--------|---------|-------|
| * | 20-21 | F | CS | 100 |
| * | 20-21 | F | CS | 90 |
| * | 20-21 | M | IS | 80 |
| * | 20-21 | M | IS | 70 |

Generalized Table T^*

- 泛化有效的原因：无论用户发出多少查询，它们能得到的最好结果就是 T^*
 - k-Anonymity

- 一个泛化表是k-anonymous的要求：如果对于表中的任意一个元组，在非敏感属性上，都至少有 $k-1$ 个其他元组与它无法区分；
- 例子：上面的 T^* 是2-anonymous
- 基本原理：如果存在 k 个无法区分的元组，那么攻击者将无法唯一地将一个个体链接到任何一个元组上；
- 问题：k-匿名确保了一个个体可以被链接到至少 k 个元组上，但它没有对这些元组的敏感属性做任何保证，当这些元组恰好拥有同质的敏感值时，k-匿名就失效了；
 - 改进的核心思想：将元组泛化成无法区分的组，并使得每个组都拥有一组多样化的敏感值，这引出了 l -diversity的概念；
- L-diversity
 - 一个泛化表是 l -diverse 的，当每个不可区分的元组都至少有 l 个良好表示的敏感值；
 - 如果在每个无法区分的组中，都有 l 个“被充分代表”的敏感值，那么攻击者将无法唯一地将一个个体链接到任何一个特定的敏感值上。
 - 例子：如果我们设定良好代表的成绩是两两之间相差5分，那么 T^* 是2-diverse的；
 - 一个好的算法会保留更多的信息；
 - 局限性：
 - “良好代表”是很难定义的；
 - 攻击者能够根据一条背景知识，从一个无法区分的组中排除掉一些敏感值；
 - 攻击者可能知道泛化算法是如何工作的，知识泄露来自于对泛化算法行为的推理；
- 尽管泛化有很多不足，但是在实践中仍然非常常用（一般更多使用k-anonymity），因为它非常容易理解，此外其他实践中的方法有：data swapping, synthetic data generation, random perturbation
- Kerckhoffs's principle: A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

Chapter 4: Oracle Label Security & Role-based Access Control

Oracle Label Security

- Oracle Label Security
 - 一种Oracle提供的多级安全版本
 - 粒度：每条记录/元组一个标签
 - 无多重实例化
 - 基于三个因素控制对数据的访问
 - 被请求访问的元组的标签

- 请求访问的用户会话的标签
 - 用户会话的策略权限（除了基本的标签等级外，是否被授予了特殊权限，比如“即使等级不够也能读取”的特权）
 - Data labels含有3个部分
 - 一个敏感度级别（有符号形式和数字形式，数字是越大级别越高）
 - 零个或多个隔离区/门类，即类别
 - 零个或多个有层级结构的分组
 - 隔离区和分组的区别
 - 隔离区是非层级的，而分组是层级的
 - **如果一个用户想要访问某条记录，必须拥有该记录所有隔离区的访问权限，但只需要拥有该记录所属分组中任意一个或其上级分组的访问权限即可**
 - User labels有多个组成部分
 - 最高级别，最低级别：用户被授权访问的最高和最低敏感度
 - 默认级别：用户连接数据库时默认使用的级别
 - 行级别：用户插入新记录时，该记录的默认级别
 - 读隔离区，写隔离区：用户被授权可以分别进行读取和写入的隔离区
 - 读分组，写分组：用户被授权可以分别进行读取和写入的分组
 - Access Control based on Labels
 - 一个用户会话可以读取一条记录，当且仅当所有以下条件都满足：
 - 用户的级别高于或等于记录的级别
 - 用户的读隔离区完全覆盖记录的所有隔离区
 - 用户的一个读分组出现在记录的分组中，或者是记录某个分组的上级
 - 一个用户会话可以写入一条记录，当且仅当所有以下条件都满足：（写的权限是被严格限制在一个“安全区间”内的，既不能越权写入更高级别的数据，也不能降级写入太低级别的数据，以确保信息在正确的密级范围内流动）
 - 记录的级别高于或等于用户的最低级别
 - 记录的级别低于或等于用户的会话级别
 - 用户的写隔离区完全覆盖记录的所有隔离区
 - 用户的一个写分组出现在记录的分组中，或者是记录某个分组的上级
- Basic Steps for Policy Creation and Enforcement in OLS
 - Create a policy

```

代码块 SA_SYSDBA.CREATE_POLICY(
2      policy_name => 'emp_ols_pol', -- 策略的名称
3      column_name => 'ols_col', -- 用于存储数据标签的列的名称
4      default_options => 'READ_CONTROL, WRITE_CONTROL'); -- 指定了策略应该
    在何时被强制执行, 可选的有READ_CONTROL, INSERT_CONTROL, UPDATE_CONTROL,
    DELETE_CONTROL, WRITE_CONTROL, ALL_CONTROL

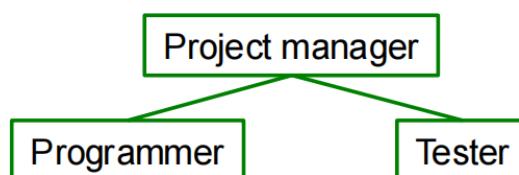
```

- Create the levels, compartments and groups
- Attach the policy to a schema or a table
- Attach labels to tuples, users, etc.
- Privileges in Oracle Label Security policies
 - 对特权用户，OLS可以提供特殊权限，允许它们绕过策略中的某些部分
 - 一组相关的特权，用于修改数据标签本身
 - WRITEUP：允许用户提升一条记录的级别，而不影响其隔离区或分组，用户可以将级别提升到其最高授权级别或以下的任何级别；
 - WRITEDOWN：允许用户降低一条记录的级别，而不影响其隔离区或分组，用户可以将级别降低到其最低授权级别或以上的任何级别；
 - WRITEACROSS：允许用户修改数据的隔离区和分组，而不改变其敏感度级别；

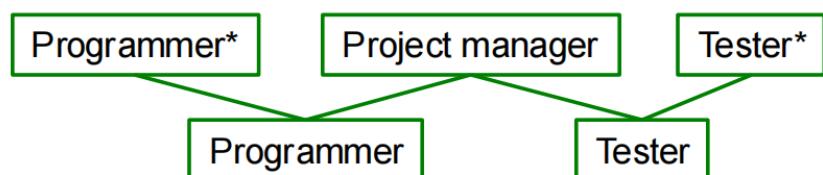
Role-based Access Control

- Motivation：在DAC中，每个用户都可能拥有对一组不同对象的访问权限，如果用户和对象的数量非常庞大，授权的总数量可能会变得极其巨大，对于一个动态变化的用户群体，大量的授权和撤销操作会变得难以管理；
- RBAC方法：
 - 基于角色进行分类，相同角色相同权限；
 - 在用户和对象之间增加一个“角色”层，将访问权限分配给角色，而不是直接分配给用户，再将角色分配给用户；
- RBAC方法的好处
 - 授权的数量得以减少，因为角色的数量通常远小于用户的数量；
 - 角色的访问权限相对稳定，也就是说，系统不需要频繁地更改角色的授权；
 - 处理动态的用户群体只需要给用户分配或重新分配角色，而无需改变角色本身的访问权限；
- 常见RBAC模型
 - $RBAC_0$
 - 一个基础的RBAC模型，它明确了 用户、 角色、 许可（对对象的访问权限） 和 会话（用户激活其部分或全部角色的一个时间段）之间的关系：

- 许可被定义为对对象的访问权限，许可由系统管理员设置；
- 修改用户、角色、许可、PA和UA关系的权利被称为管理权限，这些权利是给管理员的，而不是给任何角色的；
- 会话由用户设置，一个用户可以在一个会话中激活他所拥有的角色的任意子集，一个用户可以在一个会话中改变角色；（会话是“运行时”的概念，它是用户与激活角色之间的映射）
- 四种关系：PA, UA, SU, SR
 - PA: 指的是许可和角色之间的关系，多对多
 - UA: 指的是用户和角色之间的关系，多对多
 - SU: 会话和用户之间的关系，多对一（一个会话只能属于一个用户，而一个用户可以有多个同时进行的会话）
 - SR: 会话和角色之间的关系，多对多，一个会话的许可权是其对应的所有角色许可权的并集
- $RBAC_1$
 - 通过引入**角色层级**来改进 $RBAC_0$
 - 角色层级RH是定义在角色集合上的一个偏序关系（偏序关系：自反性、传递性、反对称性）
 - 授予低层级角色的权限会自动地被其在层级中的上级角色所拥有，反之则不然；
 - 例子：程序员和测试员的权限会自动授予项目经理



- 如果允许程序员和测试员将他们“正在进行中”的工作保存在他们的私有空间里，并且只让项目经理看到已完成的工作
 - 增加两个额外的角色——私有角色Programmer* 和 Tester*
 - Programmer*拥有一些未授予给Programmer角色的对私有对象的访问权限



- 私有角色之间也可以形成一个子层级

- $RBAC_2$
 - 通过引入约束来改进 $RBAC_0$
 - 约束可以应用于会话、 SU、 SR、 UA和PA用以实施高层次的组织策略
 - 约束例子
 - 互斥角色(SoD): 为避免串通舞弊, 任何用户不能被同时分配 “采购经理” 和 “审计员” 这两个角色;
 - Cardinality: 限制一个用户可以拥有的最大角色数量;
 - 角色依赖: 一个用户只有在被分配了角色B之后, 才能被分配角色A;
- $RBAC_3$
 - $RBAC_1$ 和 $RBAC_2$ 的一种结合, 即同时拥有角色层级和约束
 - 问题: 假如有一个角色约束, 一个用户不能在同一个项目中同时是程序员和测试员, 但是该项目的项目经理会同时拥有程序员和测试员的访问权限, 这是否违反了约束——系统应该明确指定, 在存在此SoD约束的情况下, 像这样的角色是否被允许;
- NIST
 - 和 $RBAC_3$ 类似, 但是明确定义了从许可到对象和操作的映射关系, 包括两种类型的职责分离: 静态SoD, 动态SoD;
 - 一个SSoD/DSoD约束包含两个部分, RS (一个角色集合) 和 n (一个大于等于2的自然数)
 - SSoD它要求任何用户都不能被分配RS集合中的n个或更多个角色
 - DSoD它要求任何用户都不能在同一个会话中激活RS集合中的n个或更多个角色
- RBAC in Oracle

代码块

```

1  CREATE ROLE BruceWayne IDENTIFIED BY IamBatman
2  -- 创建一个名为 BruceWayne 的角色, 想在会话中使用此角色的用户必须提供密码
   "IamBatman"。
3  GRANT insert, update on T TO BruceWayne
4  -- 允许 BruceWayne 角色对表T执行 insert 和 update 操作。
5  REVOKE insert on T FROM BruceWayne
6  -- 从 BruceWayne 角色中撤销对表T的 insert 权限。
7  GRANT BruceWayne to Cedric
8  -- 将 BruceWayne 角色授予用户 Cedric。
9  User Cedric: SET ROLE BruceWayne IDENTIFIED BY IamBatman
10 -- 用户 Cedric 在一个会话中激活 BruceWayne 作为他的角色。
11 REVOKE BruceWayne FROM Cedric
12 -- 从用户 Cedric 处撤销 BruceWayne 角色。
13 DROP ROLE BruceWayne

```

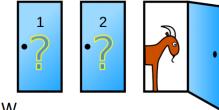
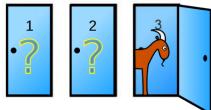
- 职责分离的实现：Not implemented with PL/SQL, Oracle Database Vault is needed

Chapter 5: Inference Analysis

- Data swapping
 - 核心思想：交换元组中的某些值，使其不再是“真实”的、可识别的记录；
- Synthetic data generation
 - 核心思想：从原始数据中构建一个统计模型，从这个统计模型中生成全新的、合成的数据；
- Random perturbation
 - 核心思想：为了保护隐私，每个用户在将她的元组交给服务器之前，都会向其中添加噪声（保护隐私，但同时允许服务器学习到有用的统计信息）；
 - Solution: randomized response
 - 每个用户有概率p给出他的真实答案，以概率1-p给出一个随机答案；
 - 受访者的真实答案没有被泄露，且经过扰动的答案仍然可以让服务器估计出调查结果；
 - 基本原理：知道由随机化响应引入的噪声的分布，因此，即使给定的是带有噪声的数据分布，也可以推断出原始的分布；
 - 把虚假的回答直接剔除，只考虑真实的回答；
 - Extension to more general questions
 - 例如：你的CS5322的成绩是多少？——解决方案：加zero-mean random noise to their answers；

Inference Analysis

- 评估数据扰动方法提供保护的程度；
- Bayesian inference
 - 通过比较先验信念（攻击者对H有的假设 $P(H)$ ）和后验信念（得到的关于H|D的信息 $P(H|D)$ ）来衡量D所提供的保护程度： $P(H|D) = \frac{P(HD)}{P(D)} = \frac{P(D|H)P(H)}{P(D)}$
 - 例子：



- Suppose you are on a game show, and you are given the choice of three doors
- Behind one door is a BMW; behind the others, goats.
- You pick a door, say No. 1
- The host, who knows what's behind the doors, opens another door, say No. 3, which has a goat.
- He then asks you, "Do you want to pick door No. 2?"
- Is it to your advantage to switch your choice?

- H: Door 2 has a BMW
- D: Host reveals Door 3 after you pick Door 1
- $\Pr[H] = 1/3$
- $\Pr[H | D] = \Pr[D | H] * \Pr[H] / \Pr[D]$
 $= \Pr[D | H] * 1/3 / \Pr[D]$
- $\Pr[D | H] = 1$
- $\Pr[D | H] = \Pr[D \wedge H] + \Pr[D \wedge (\text{Door 1 has a BMW})] + \Pr[D \wedge (\text{Door 3 has a BMW})]$
 $= \Pr[D | H] * \Pr[H] + \Pr[D | \text{Door 1 has a BMW}] * \Pr[\text{Door 1 has a BMW}]$
 $= 1 * 1/3 + 1/2 * 1/3$
 $= 1/2$
- So $\Pr[H | D] = 1 * 1/3 / (1/2) = 2/3$
- In other words, it is beneficial to switch to Door 2

- 这个 $P(D)$ 一般通过边缘化 $P(HD)$ 得到, $P(D|H)$ 大多时候是客观规律

解题关键：分清楚H和D——H为攻击者想知道的东西，D为泛化后的数据表

- Inference analysis: l- diverse example

| Name | Age | Gender | ZIP |
|-------|-----|--------|--------|
| Alice | 20 | F | 100000 |
| Bob | 20 | M | 100000 |
| Carl | 50 | M | 190000 |
| Dave | 50 | M | 190000 |

What the adversary knows

| Name | Age | Gender | ZIP | Disease |
|-------|-----|--------|--------|-----------|
| Alice | 20 | * | 100000 | Flu |
| Bob | 20 | * | 100000 | Dyspepsia |
| Carl | 50 | * | 190000 | Gastritis |
| Dave | 50 | * | 190000 | Flu |

Generalized Table D

| Name | Age | Gender | ZIP | Disease |
|-------|-----|--------|--------|-----------|
| Alice | 20 | F | 100000 | Flu |
| Bob | 20 | M | 100000 | Dyspepsia |
| Carl | 50 | M | 190000 | Gastritis |
| Dave | 50 | M | 190000 | Flu |

What the adversary knows

| Name | Age | Gender | ZIP | Disease |
|-------|-----|--------|--------|-----------|
| Alice | 20 | F | 100000 | Flu |
| Bob | 20 | M | 100000 | Dyspepsia |
| Carl | 50 | M | 190000 | Gastritis |
| Dave | 50 | M | 190000 | Flu |

Generalized Table D

| Name | Age | Gender | ZIP | Disease |
|-------|-----|--------|--------|-----------|
| Alice | 20 | * | 100000 | Flu |
| Bob | 20 | * | 100000 | Dyspepsia |
| Carl | 50 | * | 190000 | Gastritis |
| Dave | 50 | * | 190000 | Flu |

Generalized Table D

- Suppose that the adversary knows the Age, Gender, and ZIP of everyone, and has the following prior belief:
 - Alice has p_1 , Bob has p_2 , Carl has p_3 , Dave has p_4 probabilities to have flu, dyspepsia, and gastritis, respectively
 - The disease of everyone is independent
- What is the adversary's posterior belief that Alice has flu and Bob has dyspepsia?

- H: Alice has flu and Bob has dyspepsia
- D: The first two tuples in the generalized table are as shown
- $\Pr[H] = 1/3 * 1/3 = 1/9$
- $\Pr[H | D] = \frac{\Pr[D | H] * \Pr[H]}{\Pr[D]} = \frac{\Pr[D | H] * 1/9}{\Pr[D]}$
- $\Pr[D | H] = 100\%$, because when Alice has flu and Bob has dyspepsia, the first two tuples in the generalized table are always the same as what we observe
- $\Pr[H | D] = \frac{\Pr[D | H] * \Pr[H]}{\Pr[D]} = \frac{\Pr[D | H] * 1/9}{\Pr[D]} = \frac{1/9}{\Pr[D]}$

- $\Pr[H | D] = \frac{1/9}{\Pr[D]}$
- $\Pr[D] = \Pr[\text{Alice has flu and Bob has dyspepsia}] + \Pr[\text{Alice has dyspepsia and Bob has flu}]$
 $= 1/9 + 1/9 = 2/9$
- So $\Pr[H | D] = 1/2$
- In other words, after observing D, the adversary has 50% confidence that Alice has flu and Bob has dyspepsia

- 在实施l-diverse时必须考虑攻击者的背景知识

| Name | Age | Gender | ZIP |
|-------|-----|--------|--------|
| Alice | 20 | F | 100000 |
| Bob | 20 | M | 100000 |
| Carl | 50 | M | 190000 |
| Dave | 50 | M | 190000 |

What the adversary knows

| Name | Age | Gender | ZIP | Disease |
|------|-----|--------|-----------|---------|
| 20 | * | 100000 | Flu | |
| 20 | * | 100000 | Dyspepsia | |
| 50 | * | 190000 | Gastritis | |
| 50 | * | 190000 | Flu | |

Generalized Table D

- Suppose that the adversary knows the Age, Gender, and ZIP of everyone, and has the following prior belief:
 - Alice has p_1 , Bob has p_2 , Carl has p_3 , Dave has p_4 probabilities to have flu, dyspepsia, and gastritis, respectively
 - Bob has p_3 , p_4 , and $1 - p_3 - p_4$ probabilities to have flu, dyspepsia, and gastritis, respectively
 - The disease of everyone is independent
- What is the adversary's posterior belief that Alice has flu and Bob has dyspepsia?

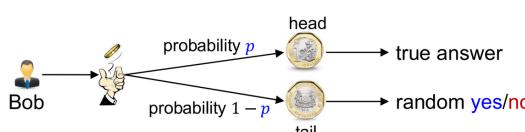
| Name | Age | Gender | ZIP |
|-------|-----|--------|--------|
| Alice | 20 | F | 100000 |
| Bob | 20 | M | 100000 |
| Carl | 50 | M | 190000 |
| Dave | 50 | M | 190000 |

What the adversary knows

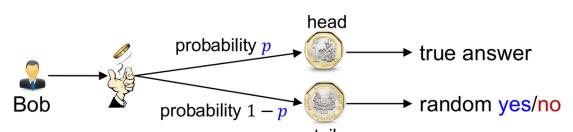
- The adversary's belief changes from $p_1 \cdot p_4$ to $\frac{p_1 \cdot p_4}{p_1 \cdot p_4 + p_2 \cdot p_3}$ after observing D
- How different can $p_1 \cdot p_4$ and $\frac{p_1 \cdot p_4}{p_1 \cdot p_4 + p_2 \cdot p_3}$ be?
- The difference can be arbitrarily large
 - Unless we make some assumptions about p_1, p_2, p_3, p_4
- This explains why we need to take into account the adversary's background knowledge when applying l-diversity

- Inference Analysis: randomized response

General case



- Assume that the retention probability is p
- H: The adversary's prior belief on Bob's answer
- D: Bob's perturbed answer
- How large and small can $\frac{\Pr[H|D]}{\Pr[H]}$ be?



- In general, we have

$$\frac{\Pr[H|D]}{\Pr[H]} = \frac{\Pr[D|H] \cdot \Pr[H]}{\Pr[D]} = \frac{\Pr[D|H]}{\Pr[D]} = \frac{\Pr[D|H]}{\Pr[D|H] + \Pr[D|\text{not } H]}$$

$$= \frac{\Pr[D|H]}{\Pr[D|H] \cdot \Pr[H] + \Pr[D|\text{not } H] \cdot \Pr[\text{not } H]} = \frac{\Pr[D|H]}{\Pr[D|H] \cdot \Pr[H] + \Pr[D|\text{not } H] \cdot (1 - \Pr[H])}$$
- There are only two possibilities for $\Pr[D | H]$ and $\Pr[D | (\text{not } H))$:
 - $\Pr[D | H] = p + \frac{1-p}{2}$ and $\Pr[D | (\text{not } H)] = \frac{1-p}{2}$, or
 - $\Pr[D | H] = \frac{1-p}{2}$ and $\Pr[D | (\text{not } H)] = p + \frac{1-p}{2}$

- In general, we have
$$\frac{\Pr[H|D]}{\Pr[H]} = \frac{\Pr[D|H] \cdot \Pr[H]/\Pr[D]}{\Pr[H]} = \frac{\Pr[D|H]}{\Pr[D]} = \frac{\Pr[D|H]}{\Pr[D \wedge H] + \Pr[D \wedge (\text{not } H)]}$$

$$= \frac{\Pr[D|H]}{\Pr[D|H] \cdot \Pr[H] + \Pr[D|(\text{not } H)] \cdot \Pr[\text{not } H]} = \frac{\Pr[D|H]}{\Pr[D|H] \cdot \Pr[H] + \Pr[D|(\text{not } H)] \cdot (1 - \Pr[H])}$$
- There are only two possibilities for $\Pr[D | H]$ and $\Pr[D | (\text{not } H)]$:
 - $\Pr[D | H] = p + \frac{1-p}{2}$ and $\Pr[D | (\text{not } H)] = \frac{1-p}{2}$, or
 - $\Pr[D | H] = \frac{1-p}{2}$ and $\Pr[D | (\text{not } H)] = p + \frac{1-p}{2}$
- Accordingly, $\frac{\Pr[H|D]}{\Pr[H]}$ only has two possibilities:
 - $\frac{\Pr[H|D]}{\Pr[H]} = \frac{p + \frac{1-p}{2}}{(p + \frac{1-p}{2}) \cdot \Pr[H] + \frac{1-p}{2} \cdot (1 - \Pr[H])} = \frac{1+p}{1-p+2p \cdot \Pr[H]} \leq \frac{1+p}{1-p}$
 - $\frac{\Pr[H|D]}{\Pr[H]} = \frac{\frac{1-p}{2}}{(\frac{1-p}{2}) \cdot \Pr[H] + (p + \frac{1-p}{2}) \cdot (1 - \Pr[H])} = \frac{1-p}{1+p-2p \cdot \Pr[H]} \geq \frac{1-p}{1+p}$
- 随机相应的推断分析: $\frac{1-p}{1+p} \leq \frac{\Pr[H|D]}{\Pr[H]} \leq \frac{1+p}{1-p}$, 也就是攻击者在攻击和推断后最多能得到 $\frac{1+p}{1-p}$ 的信息收益;

Chapter 6: Encrypted Database

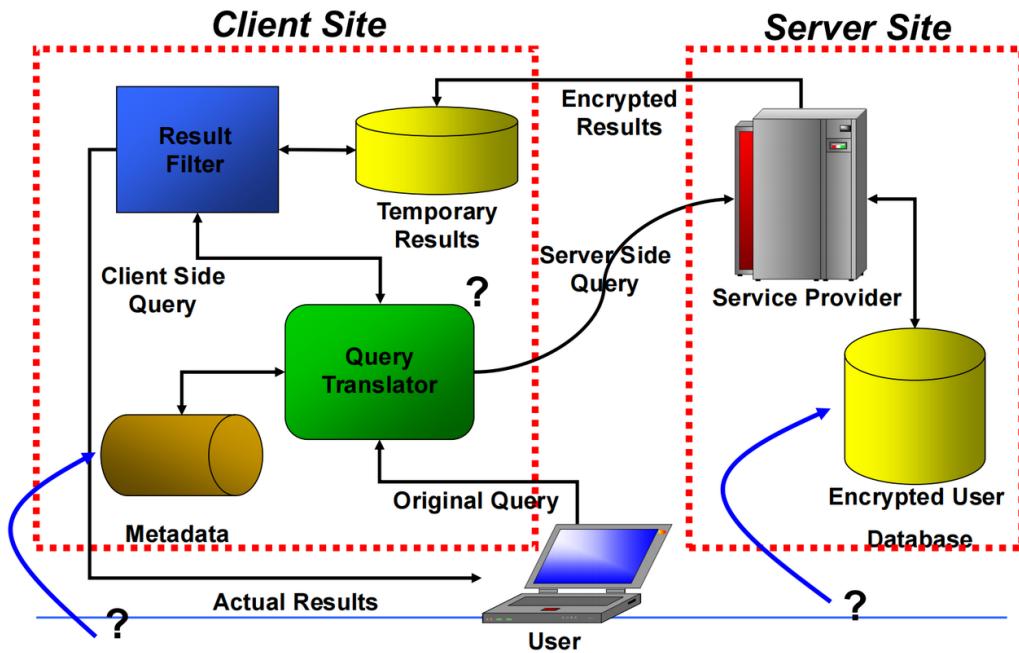
- Encrypted database
 - 核心思想: 让数据所有者在发给服务商数据库前进行加密;
 - 挑战: 如何在密文上进行计算, 从而在保护数据隐私的同时, 又不牺牲数据库的功能性;
 - 解决方案:
 - 同态加密Homomorphic encryption
 - 硬件辅助安全Hardware-assisted security
- Homomorphic encryption
 - 允许服务提供商对加密数据进行计算的加密方案;
 - 基本思想
 - 数据所有者将每个元组 t 加密成 $E(t)$, 并将所有加密后的元组传递给服务提供商;
 - 用户要求服务提供商对这些元组执行计算任务;
 - 服务提供商处理 $E(t_1), E(t_2), \dots$, 并得出一个加密的答案 $E(a)$, 然后将其发送给用户;
 - 用户解密 $E(a)$ 以获得答案 a ;
 - 关键问题
 - $E(a)$ 是从 $E(t_1), E(t_2), \dots$ 推导出来的, 整个过程不能让服务器知道加密密钥或 t_1, t_2, \dots 的明文;
 - 类型
 - 乘法同态: $E(m_1) \times E(m_2) = E(m_1 \times m_2)$
 - 加法同态: $E(m_1) \times E(m_2) = E(m_1 + m_2)$

- 全同态加密：上面两种都支持，仍然极其缓慢且占用大量空间，很难实用；
- Hardware-assisted security
 - 核心思想：在服务提供商的机器上安装一些安全处理器（必须值得信任），每当用户想要计算某些东西时，他直接与安全处理器对话；
 - 安全处理器：从机器获取加密数据，解密它们并计算结果，并将结果的加密版本发送回给用户；
 - Secure processor: Intel SGX
 - 是新一代Intel CPU中的一组CPU指令代码；
 - 允许用户在远程机器的内存中创建一个安全区域（称为enclave），飞地中的所有内容都被加密，并且机器所有者也无法看到，解密仅在数据被送到CPU进行计算之后才进行，计算过程也受到远程机器的屏蔽；（SGX的认证需要与英特尔进行通信）
 - 局限性：在某些情况下可用的计算资源有限，攻击是可能的；

HILM

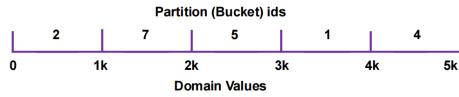
- HILM（在数据库-服务提供商模型中对加密数据执行SQL）
 - 基本思想
 - 将加密的元组分成组，并为每个组附加一些索引；服务提供商使用这些索引在组级别上处理加密的元组；
 - 在用户和服务提供商之间拆分查询处理任务：服务提供商返回与查询相关的一些元组组，用户解密这些元组组，并进一步处理它们以产生最终的查询结果；
 - 分组和索引：不是一行一行地处理数据，而是把数据分成粗粒度的“组”，然后给这些组贴上索引；
 - 粗粒度过滤：服务提供商看不到数据内容，但它能看懂这些索引，它根据用户的查询，利用索引进行一次粗略的、组级别的过滤；
 - 任务拆分：一个SQL查询被拆成两部分，服务提供商执行“粗过滤”部分，筛选出可能包含结果的数据组，然后把这些加密的组发回给用户，用户在本地将它们解密，然后执行“精细过滤”部分，得到最终准确的结果；
 - 示意图
 - 用户提交原始查询；
 - 客户端的查询转换器将其分解；
 - 一部分变成服务器端查询，发送给服务提供商；
 - 服务提供商在加密的用户数据库上执行查询，返回加密结果；
 - 客户端收到加密结果，解密成临时结果；
 - 客户端的结果过滤器根据客户端查询对临时结果进行处理，得到实际结果；

- 图中还有元数据，用于查询转换。

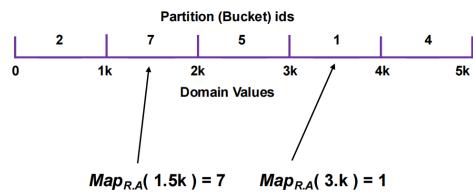


- 加密和索引数据

- Given an attribute A in table R to be indexed, first divide the domain of $R.A$ into a number of *buckets*
 - E.g., dividing the domain of Salary into $[0, 1k], [1k, 2k], [2k, 3k], \dots$,
- Then, assign an ID to each bucket



- Define a mapping function $map_{R,A}$ that maps any A value to its partition ID



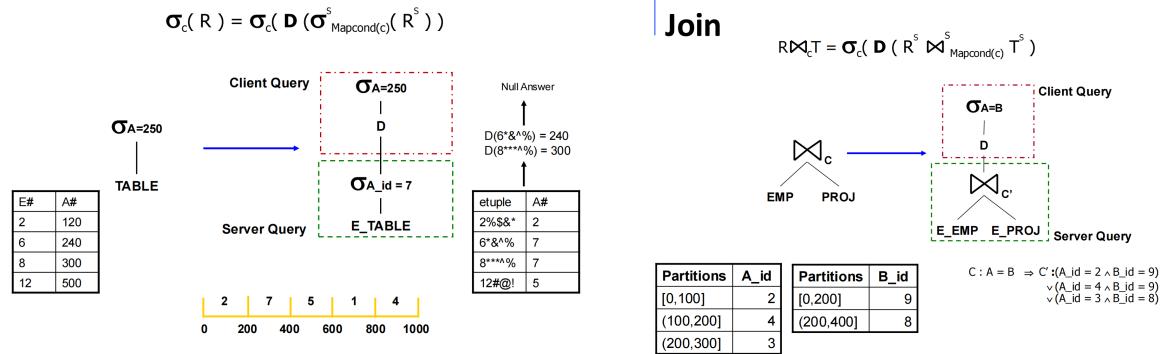
- 查询转换器

- 将用户的查询转换成一个将处理任务在用户和服务提供商之间拆分的查询执行计划

例子

```
1  SELECT name, pname FROM emp, proj WHERE emp.pid=proj.pid AND salary
> 100k
```

- 关键问题是，服务提供商无法看到 $emp.pid$, $proj.pid$ 和 $salary$ ，他只能看到桶ID，所以我们需要一种方法，将原始属性上的查询条件映射到桶ID上的条件；
- HILM 方法定义了一个映射函数 Map_{cond} 用于映射条件；
- SELECT和JOIN



- 现在知道了如何映射查询条件，将继续讨论如何生成一个查询执行计划（**最小化客户端完成的工作**）

- 基本思想
 - 从一个简单的查询计划开始，通过将操作下推到服务器站点来迭代地改进计划；
 - 这在某种程度上类似于传统查询生成中的启发式优化；
- 例如：上述查询中，最优方案是——服务器不仅利用 S_ID 筛选出高薪员工，服务器还利用 $\text{MapCond}(e.pid = p.pid)$ ，在服务器端直接对比桶ID，把 EMP 表和 PROJ 表给连接起来了，这个时候客户端只需要做最后的解密和精细过滤；

- Further optimization

- HILM 方法将每个元组作为一个整体来加密，因此，即使用户的查询只请求一个属性，HILM 也必须返回整个加密的元组——一种改进的方法：分别加密每个属性；
- 某些属性经常用于聚合，对于这些属性，我们可以使用部分同态加密；
- 局限性
 - 用户（客户端）需要为查询处理做大量的工作；
 - 查询翻译有时很复杂，尤其是对于 Joins；
 - 分桶方法没有任何正式的安全保证，目前尚不清楚服务提供商可以从属性的分区中推断出多少信息；

CryptDB

- CryptDB
 - 一种完全在服务器端处理查询，只要求用户解密结果，在必要时使用安全性较低的加密方案；
 - 仍然分为两个部分，但支持对加密数据的标准SQL查询，且完全在数据库服务器上处理查询，无需更改现有的DBMS；（不需要魔改数据库内核）
 - Client frontend (可信)：存储 schema (数据库结构)、主密钥，不执行任何查询；
 - Database server (不可信)：存储数据库并处理SQL查询，不被信任来保护数据隐私；
- 基本思想
 - 单独加密每个元组的每个值；

- 每个用户查询都直接翻译成一个对加密数据的服务器查询，无需使用复杂的翻译规则；
 - 不存在**单一高效的**加密方案能让服务提供商处理所有类型的查询，所以，使用**多种加密方案**，每种方案可以支持一种类型的查询，某些方案会比其他方案更安全，但为了查询处理，接受较低的安全保证；
 - 不同的加密方式用于不同的属性：
 - 支持等值搜索(e.g., `A=x`)，使用确定性加密；
 - 支持不等式/范围搜索(e.g., `A<=x`)，使用保序加密；
 - 支持关键字搜索(e.g., `A LIKE '%X%'`)，使用可搜索加密；
 - 如果无需支持搜索，使用随机加密(i.e., 两个相同的密文也不会相同)；
 - 不同的加密方案保密级别不同：随机>确定性>保序；（在所有能支持所需查询的方案中，选择最安全的）
 - 在一种加密上支持多种类型的搜索
 - 例如，为Rank生成两个加密版本，一个使用确定性加密（用于等值查询），另一个使用保序加密（用于范围查询）；
 - 但这会带来一个问题，但一个客户只进行这一列的等值查询时，服务商也知道顺序，超过了用户查询所需要揭示的——解决方案：将多种加密方案应用于一个洋葱中
- CryptDB: Onion
- 例如上述问题的解决方案为，将 `Rank` 列加密三次：
 - 使用保序加密方案加密 `Rank`，得到 `c1 = OPE(Rank)`；
 - 使用确定性加密方案加密 `c1`，得到 `c2 = DET(c1)`；
 - 使用随机加密方案加密 `c2`，得到 `c3 = RND(c2)`；
 - 示例：用户搜索 “`Rank = 2`”，向数据库提供解密 `RND` 层的密钥，之后，在 `DET` 层处理等值查询，换句话说，我们剥掉了洋葱的 `RND` 层，但是其他层 (`DET, OPE`) 仍然保留；
 - 总的来说，每个列最初都被加密成一个（或更多）洋葱，在用户开始发出查询后，一些层被剥离以启用查询处理，换句话说，为了查询处理，安全保证被降低了，然而，明文永远不会透露给服务提供商；
 - `Select SUM(Salary) FROM Employee WHERE Rank = 3`
 - `Rank = 3` 可以通过 `Rank` 列上的确定性加密来处理，处理 `SUM` 需要需要 `Salary` 列上的加法同态加密；
 - - `SELECT SUM(Salary) FROM Employee WHERE Salary > 30k`

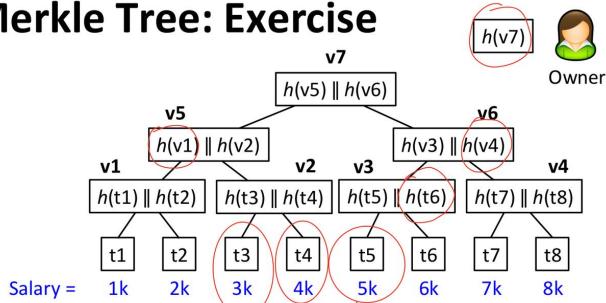
- 但不存在既能保证又加法同态的方案，这里需要对 `Salary` 建立两个洋葱：一个用来处理 `SUM(Salary)`（配合加法同态加密），另一个用来处理 `Salary` 上的等值和不等式搜索；
- CryptDB: Equi-Join
 - `SELECT * FROM Employee E, Department D WHERE E.Name = D.Head`
 - 等值连接目标：如果用户没有发出 `Employee.Name = Department.Head` 的连接查询，服务提供商不应该能够搞清楚哪个 `Employee.Name` 可以与哪个 `Department.Head` 连接：
 - CryptDB解决方案：
 - 使用不同的密钥加密这两列，得到 `DET_k1(Name)`, `DET_k2(Head)`
 - 使用不同的密钥为每一列生成一个标签，得到 `Tag_k3(Name)`, `Tag_k4(Head)` (但这两个标签的生成是同方法的，只是密钥不同，但当用户提出 `Employee.Name = Department.Head` 时，就把密钥改成相同，这样相同的值的标签就一样了)
 - $Tag_k(Name) = P^{k \times PRF(Name)}$, $Tag_{k_1}(Head) = P^{k_1 \times PRF(Head)}$
 - 其中， P 为公共参数， k 和 k_1 为两个不同密钥，PRF为伪随机函数
 - 将标签与加密值连接在一起，得到 `Tag_k3(Name) || DET_k1(Name)`, `Tag_k4(Head) || DET_k2(Head)` (因为不仅需要tag判断，还需要加密列读取数据)
- CryptDB局限性：无法处理跨列的复杂计算；

Chapter 7: Query Authentication

- Query Authentication
 - 假设不再追求保护数据保密性，因为加密数据库开销太大，目标变成：确保服务提供者正确地回答查询；
 - 服务提供者可能以下述方式错误地回答查询：（很多时候是因为性能原因导致的）
 - 返回查询结果中的部分元组（例如提前停止扫描）
 - 返回一些伪造的元组
 - 返回从数据库旧版本中得到的过期结果
 - 朴素解决方案
 - 当服务提供商回答查询时，返回查询结果+正确性证明——使用公钥密码学来实现
 - 数据拥有者有两把密钥
 - 私钥 sk : 自己保管
 - 公钥 pk : 公开给所有用户
 - 用 pk 加密的消息只能由 sk 解密，用 sk 加密的消息只能由 pk 解密，所以别人可以用 pk 验证消息是否真的来自数据拥有者；

- 让数据拥有者对数据库的每个元组进行**签名**，用私钥对元组加密，将加密版本和原始版本都交给服务提供者，当服务提供者返回查询结果时，同时返回对应的加密版本；
- 问题
 - 不能阻止云**丢弃元组**
 - 每个元组需要存两份：明文 + 签名
- Towards an improved solution
 - 分组加密
 - 让数据拥有者做以下事情：根据 Age 的不同取值，将元组分组，对每一组分别用私钥 sk 加密，将加密后的组发送给服务提供者；
- Merkle Tree
 - 密码学哈希函数（Cryptographic Hash Function）是一种将任意大小的数据映射为固定长度比特串的函数，具有以下性质：
 - 确定性（同样的输入一定产生同样的哈希值）
 - 计算高效（计算哈希非常快）
 - 除了尝试所有可能的消息之外，从其哈希值生成消息是不可行的；
 - 雪崩效应：输入微小变化 → 哈希值完全不同
 - 抗碰撞：无法找到两个不同消息具有相同哈希值
 - 通用算法
 - Consider a query on T.A in [x, y]
 - Among the tuples t with t.A < x, find the tuple tx whose A value is the largest
 - Identify the path from tx to the root of the Merkle tree
 - For every “left branch” on the path, collect the hash value of the branch
 - Among the tuples t with t.A > y, find the tuple ty whose A value is the smallest
 - Identify the path from ty to the root of the Merkle tree
 - For every “right branch” on the path, collect the hash value of the branch
 - Return tx, ty, and all tuples between them, and all hash values collected, as well as the encrypted Merkle root
- Example of Merkle Tree
 - | 每次查询结果必须给足信息，使用户自己能够断言：没有被隐藏或伪造的记录；

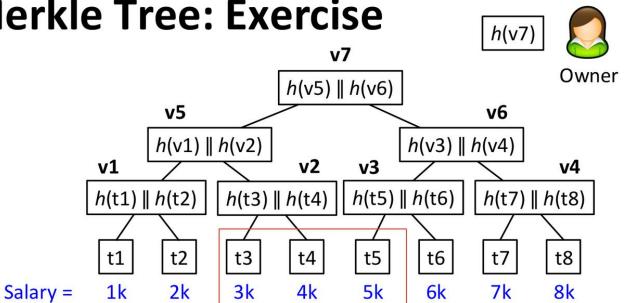
Merkle Tree: Exercise



- Consider a query on “Salary = 4k”
- What should the service provider return?

前用单验证数据单是否还有4k

Merkle Tree: Exercise



- Consider a query on “Salary in (3k, 4k]”
- What should the service provider return?

所以不要验证

• Extension to Multi-Dimensional Data

例如：SELECT * FROM Employee WHERE Age > 30 AND Salary > 10000

- 方法一：Build two Merkle trees on Age and Salary, respectively
- 方法二：Use multi-dimensional indices

• 如何理解Merkle树保证查询答案的完整性

- 任何一点数据被修改、伪造或删除，都会导致根哈希发生变化，而根哈希是被数据所有者签名的，服务端无法伪造；
- 返回的节点会完全包含整个查询区间，并且给出比区间最小值更小的左边界和比区间最大值的更大的右边界辅助验证，是否有值缺少；

Chapter 8: Watermarking Relational Database

• Watermarks

- 思路：在数据中引入虚假信息
 - 这些虚假信息应该难以被察觉（隐蔽性）；
 - 并且它不应过多地影响数据的可用性；
- 挑战
 - 用户可能不会泄露数据集 D 的完整副本（子集攻击），他可能选择只泄露一部分记录；
 - 用户可能会修改 D 中的记录，试图破坏水印（鲁棒性攻击）；

• AHK (A method for relational database)

- 核心思想：利用数据中数值的**最低有效位**；
- AHK基于以下假设：
 - 待加水印的表 T 拥有若干数值型属性 (A_1, A_2, \dots) 和一个主键 P ；
 - 泄露的数据中包含了主键 P 和部分数值属性；
 - 对于每个属性 A_i ，其**最低的 ξ 位**可以用来加水印；（改变少量低位不会显著影响数据可用性）

- 检测水印时不需要访问原始表 T (即盲检测) ;
- 基本思路
 - 对于每一个元组 t , 利用它的主键 $t.P$ 来决定
 - 这行要不要加水印?
 - 这行的哪一列要加水印?
 - 这一列的第几位 LSB 要加水印?
 - 但是不能让攻击者猜出哪些位被修改了;
- 解决方案: 使用密码学伪随机数生成器 (CPRNG) 来选择要加水印的行和位;
 - 一个确定性函数 G , 给定一个种子值, 生成一系列随机数, 没有种子, 就算不出下一个数;
 - 数据拥有者选一个密钥 K ;
 - 对于每一行 t , 种子 = 密钥 K 拼接上主键 $t.P$;
 - 然后用 G 生成的随机数来做决策;
- AHK watermarking algorithm
 - Input parameters:
 - G : a CPRNG
 - K : a secret key selected by the data owner
 - T : a table to be watermarked
 - d : the number of attributes in T
 - ξ : the number of least significant bits (LSB) suitable for watermarks
 - f : the fraction of tuples to be watermarked
 - For each tuple t in T
 - Seed G with $K \parallel t.P$
 - Get random numbers $r1, r2, r3, r4$ from G
 - If $(r1 \bmod 1/f) = 0$ <----- t has f probability to be watermarked
 - $i = r2 \bmod d$ <----- the i -th attribute $t.Ai$ will be marked
 - $j = r3 \bmod \xi$ <----- the j -th LSB of $t.Ai$ will be marked
 - if $(r4 \text{ is even})$ then set the j -th LSB of $t.Ai$ to 0
 - else set the j -th LSB of $t.Ai$ to 1
- 如果原始数据的这一位本来就是 0, 而随机数 $r4$ 让我们设为 0, 那么数据没有任何变化; (那怎么知道是否有加水印呢? ——对于“单个”比特, 根本无法区分; 但对于“整体”数据, 通过统计匹配率来判定)


```
If (match_count > total_count - t) then suspect data leak from User u
Else if (match_count < t) then suspect data leak and watermark manipulation from User u
```
- 可以处理属性被删除的情况

- 当算出该在第*i*列加水印时，先看看泄露数据S里有没有这一列，只有当这一列存在时，才把total_count加1，如果列都被删了，我们就当这行没加过水印，不去检测它；

- Attacking AHK watermarks

- 与另一个用户串通，获取两份加了不同水印的数据集，比较这两个数据集，找出哪些位是不同的；
 - 原理：每一份数据的水印位置都是随机选的，同一位在两份数据里都被加水印的概率很低，所以，一份数据里被改过的位，在另一份数据里很可能没被改，两者一对比就露馅了；
- 攻击的问题：我们只知道哪些位不同，但我们不知道这个差异是源于 Dataset 1 被改了，还是 Dataset 2 被改了，还是两个都被改了；
 - 解决方案一：盲猜；（把这些有差异的位随机翻转一半）
 - 两人合谋的情况：直接破坏掉不一样的水印
 - 解决方案二：多数/少数投票；（找第三个用户来合谋，现在只要发现有差异，我们就用多数/少数投票来推断原始值）——3人及以上的人数合谋的情况

- Guarding Against Collusion Attacks

- 让数据集之间的水印位互相关联，不再是完全独立的随机数；
 - 为了防止合谋，不能再每个人发一把完全不同的随机钥匙了，要设计一个精妙的矩阵；

| | |
|--|---------------|
| | D1+, D2+, D3+ |
| | D1-, D2+, D3+ |
| | D1-, D2-, D3+ |
| | D1-, D2-, D3- |

- Suppose that we are to give a dataset D to four users u_1, u_2, u_3 , and u_4
- We randomly divide D into three parts: D_1, D_2, D_3
- We choose a secret key K , and produce two watermarked versions of D_i
 - The first version D_{i+} is generated by the AHK algorithm
 - The second version D_{i-} is generated by flipping the watermarked bits in D_{i+}
- We then assign watermarked parts to users as shown above

| | |
|--|---------------|
| | D1+, D2+, D3+ |
| | D1-, D2+, D3+ |
| | D1-, D2-, D3+ |
| | D1-, D2-, D3- |

- Suppose that u_1 and u_3 collude
- They could compromise the watermarks in D_1 and D_2
- But the watermarks in D_3 still exists
- When the data is leaked, if we detect watermarks in D_3 (but not in the other parts),
- then we know that u_1 and u_3 must be among the culprits

| | |
|--|---------------|
| | D1+, D2+, D3+ |
| | D1-, D2+, D3+ |
| | D1-, D2-, D3+ |
| | D1-, D2-, D3- |

- Suppose that u_1, u_2 , and u_3 collude
- They could compromise the watermarks in D_1 and D_2
- But the watermarks in D_3 still exist
- When the data is leaked, if we find that the watermarks exist in D_3 (but not in the other parts)
- then we know that u_1 and u_3 be among the culprits
- Why can't we be sure about u_2 ?
 - Because u_1 and u_3 can already compromise D_1 and D_2

| | |
|--|---------------|
| | D1+, D2+, D3+ |
| | D1-, D2+, D3+ |
| | D1-, D2-, D3+ |
| | D1-, D2-, D3- |

- Suppose that u_1 and u_4 collude
- They could compromise the watermarks in D_1 and D_2 and D_3
- When the data is leaked, if we cannot detect any watermarks
- then we know that u_1 and u_4 must be among the culprits
 - Otherwise, the watermarks in D_1, D_2 , and D_3 cannot be compromised at the same time

| | | | |
|--|------------|------------|------------|
| | <i>D1+</i> | <i>D2+</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2+</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3-</i> |

- Suppose that *u1*, *u2*, and *u4* collude
- They could compromise the watermarks in *D1* and *D2* and *D3*
- When the data is leaked, if we cannot detect any watermarks
- then we know that *u1* and *u4* must be among the culprits
 - Otherwise, the watermarks in *D1*, *D2*, and *D3* cannot be compromised at the same time
- But we cannot be sure about *u2*

| | | | |
|--|------------|------------|------------|
| | <i>D1-</i> | <i>D2-</i> | <i>D3-</i> |
| | <i>D1-</i> | <i>D2+</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3+</i> |
| | <i>D1+</i> | <i>D2-</i> | <i>D3-</i> |

- This watermarking scheme is not good because
 - If *u1*, *u2*, and *u4* collude and use a majority vote, they can obtain a copy of the data that corresponds to $\{D1-, D2-, D3+\}$, which is the same as *u3*'s copy
 - By leaking this copy, *u1*, *u2*, *u4* can frame *u3* as the data leaker

| | | | | |
|--|------------|------------|------------|------------|
| | <i>D1+</i> | <i>D2-</i> | <i>D3-</i> | <i>D4-</i> |
| | <i>D1-</i> | <i>D2+</i> | <i>D3-</i> | <i>D4-</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3+</i> | <i>D4-</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3-</i> | <i>D4+</i> |

- This watermarking scheme is not OK, because
 - If any 3 users collude and follow the majority rule, they can leak $\{D1-, D2-, D3-, D4-\}$
 - When the data owner sees $\{D1-, D2-, D3-, D4-\}$, she would be unable to pinpoint any of the culprits, because every user can claim that it was the other 3 users who leaked the data

- 发现陷害攻击

- 我们需要寻找一组用户（合谋者），他们的组合能够覆盖目标用户（受害者）的所有特征；
-

| | | | |
|--|------------|------------|------------|
| | <i>D1+</i> | <i>D2+</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2+</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3+</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3-</i> |

- Suppose that *u1*, *u2*, and *u4* collude
- They could compromise the watermarks in *D1* and *D2* and *D3*
- But they could be more strategic
- Can they leak *D1-*, *D2-*, *D3+*, and pretend that the leak is caused by *u3*?
- No.
- Because they don't know which tuples are from *D2-* and *D3+*

| | | | | |
|--|------------|------------|------------|------------|
| | <i>D1+</i> | <i>D2-</i> | <i>D3+</i> | <i>D4-</i> |
| | <i>D1-</i> | <i>D2+</i> | <i>D3-</i> | <i>D4-</i> |
| | <i>D1-</i> | <i>D2-</i> | <i>D3+</i> | <i>D4+</i> |
| | <i>D1-</i> | <i>D2+</i> | <i>D3-</i> | <i>D4+</i> |

- This watermarking scheme is OK, because:
 - No 3-user-collusion can frame the remaining user, regardless of whether they follow the majority vote or the minority vote
 - If the users collude to destroy watermarks and they involve *u1*, then the watermarks in *D1* will be destroyed; in that case, *u1* will get caught, since he is the only user who has *D1+*
 - Every user collusion that does not involve *u1* would destroy a different set of watermarks, and hence, we can identify the colluding users by observing which watermarks are destroyed

(尽管上图中，*u4*可能会难以被判断是否为凶手，但是至少它是能识别出凶手的)

Paper

- Week 10 – EnclaveDB: A Secure Database using SGX
 - EnclaveDB 利用 Intel SGX 来保护数据在使用过程中依然具有机密性和完整性，能够防御包括恶意操作系统或虚拟机管理程序在内的强大攻击者。它将查询执行和事务管理放入 enclave 中，而将外部不可信存储通过加密与完整性验证方式保护。系统将 SQL 查询和事务逻辑编译为 enclave 内本地代码，从而减少 enclave 边界切换带来的性能开销。该架构在提供强安全保障的同时，仍保持接近传统内存数据库的性能水平。

- EnclaveDB leverages Intel SGX to ensure confidentiality and integrity of data-in-use, protecting against powerful adversaries such as malicious operating systems or hypervisors. It executes query processing and transaction management inside enclaves while securing untrusted storage through encryption and integrity verification. By compiling SQL queries and transaction logic into native enclave code, the system minimizes enclave boundary crossings and runtime overhead. As a result, EnclaveDB achieves strong security with performance comparable to traditional in-memory database engines.
- Week 11 – Operon: Encrypted Database for Ownership-Preserving Data Management
 - Operon 是一个面向协作环境的加密数据库系统，其目标是在多方数据共享场景下持续保持数据所有权保护。系统结合可信执行环境（TEE）与加密技术，使数据即便存储在共享的不可信服务器上，所有者仍完全掌控访问权限。Operon 强制执行细粒度的所有权访问策略，只有当查询方满足授权条件时，相关密文数据才会被解密并参与计算。系统支持在密文上进行高效查询与索引操作，在减少信息泄露的同时保证分析型工作负载可用性。通过将数据所有权与物理存储解耦，Operon 实现不依赖完全信任服务器的安全协作数据管理。
 - Operon is an encrypted database designed for collaborative environments to preserve data ownership in shared settings. It integrates TEE with cryptographic protection so that owners retain strict control over their data even when stored on shared untrusted servers. Operon enforces fine-grained ownership-based access rules, decrypting data only when the requester satisfies policy requirements. It enables efficient encrypted queries and indexing while limiting information leakage and supporting analytical workloads. By decoupling physical storage from data ownership, Operon enables secure collaborative data management without fully trusting the server.
- Week 12 – VeriDB: An SGX-based Verifiable Database
 - VeriDB 基于 Intel SGX 构建可验证数据库，专注于对高动态更新场景中的分析查询结果进行高性能验证。系统采用“双模式认证数据结构”，在事务更新与分析查询验证模式之间高效切换，以验证即时数据（Fresh Data）。所有验证逻辑均在 enclave 中执行，使客户端无需维护数据副本却能验证结果的正确性、完整性与新鲜度。VeriDB 为返回结果生成加密证明，确保云端无法对数据进行篡改、删除或伪造。
 - VeriDB is an SGX-based verifiable database focusing on verifying analytical query results over highly dynamic and frequently updated data. It introduces a dual-mode authenticated data structure that efficiently switches between update and verification modes to maintain freshness guarantees. All verification logic runs inside enclaves, enabling clients to verify result correctness, integrity, and freshness without storing local data copies. VeriDB produces cryptographic proofs to prevent untrusted cloud servers from tampering with or omitting records.

| 系统 | 安全目标（与考试答案一致） | 威胁模型（与考试答案一致） | 核心技术机制（仅来自答案内容，无额外扩充） |
|-----------|-------------------------|------------------------|--|
| EnclaveDB | 保护数据在使用中的机密性与完整性 | 恶意 OS / 恶意虚拟机管理程序等强攻击者 | <ul style="list-style-type: none"> 查询执行与事务管理在 SGX enclave 内 外部存储使用加密与完整性验证保护 SQL/事务逻辑编译为 enclave 原生代码以减少切换开销 性能接近传统内存数据库 |
| Operon | 多方共享环境中保持数据所有权保护 | 不可信服务器 + 未满足授权条件的访问方 | <ul style="list-style-type: none"> TEE + 加密结合的访问控制 细粒度所有权访问策略控制解密条件 密文上支持高效查询和索引 数据所有权与物理存储解耦，不依赖完全可信服务器 |
| VeriDB | 查询结果的正确性 + 完整性 + 新鲜度可验证 | 不可信云服务器伪造或遗漏数据 | <ul style="list-style-type: none"> 验证逻辑在 enclave 内执行 双模式认证数据结构（更新 & 验证切换） Fresh Data 验证能力 生成加密证明确保结果未篡改/遗漏 |