**密码学课程第4次实验报告**

**实验名称：**

学号： 2011428 姓名： 王天行 班级： 密码科学与技术

# 实验目的

（概述本次实验要做什么）

通过实际编程（GPC[8], IDB[9]任选其一即可）了解口令泄露查询协议的交互过程，掌握口令泄露查询协议的基本设计和分析方法。

编写简单的客户端-服务器，具体要求如下：

1. 实现最基本的口令泄露查询，用户输入用户名-口令对，可以了解到她的输入凭证泄露、未泄露。
2. 保证系统的安全性，能防范恶意客户端的蛮力搜索，又能预防诚实且好奇服务器试图获取用户输入的口令信息。
3. 保护协议主体的隐私，在交互过程中用户不会泄露她所查询的用户名和口令的明文，服务器也不会向用户泄露额外的数据。
4. 支撑多用户；
5. 服务器显示日志，记录与客户端交互过程；
6. 编程语言不限，可以使用Python/C++/Go/Java等；
7. 数据报方式不限，可基于TCP或UDP；
8. 程序界面不限，可使用命令行界面或图形化界面。
9. 实验环境不限，可基于Linux/Windows等

# 实验内容说明

1. 服务器预处理文件，最终成功运行。
2. 服务器端设置一个以c3server\_v3命名的数据库。它存储了加密后的用户名-口令对。加密凭据值依据用户名的哈希值前缀进行划分，具有同一哈希值用户名前缀的加密凭据划分在同一数据桶中。
3. 启动服务器，可正常运行。
4. 随后在客户端运行用于本地查询的python文件，输入想要查找的用户名-口令对，检查是否泄露。

# 实验原理

## 1.相关技术

### 1) 不经意伪随机函数

不经意伪随机函数（OPRF）包括发送方和接收方。其中，发送方输入秘密值，接收方掌握OPRF密钥，OPRF结合两者进行一系列运算得到输出。经过交互，发送方能够在不知道的情况下恢复出OPRF输出，接收方学习不到任何关于的知识，也不知道OPRF输出的值。具体而言，OPRF有多种实现方法。以2HashDH方案为例，如图1所示，其本质上是Chaum型的盲化Diffie-Hellman计算，在OM-gapDH假设下是安全的。

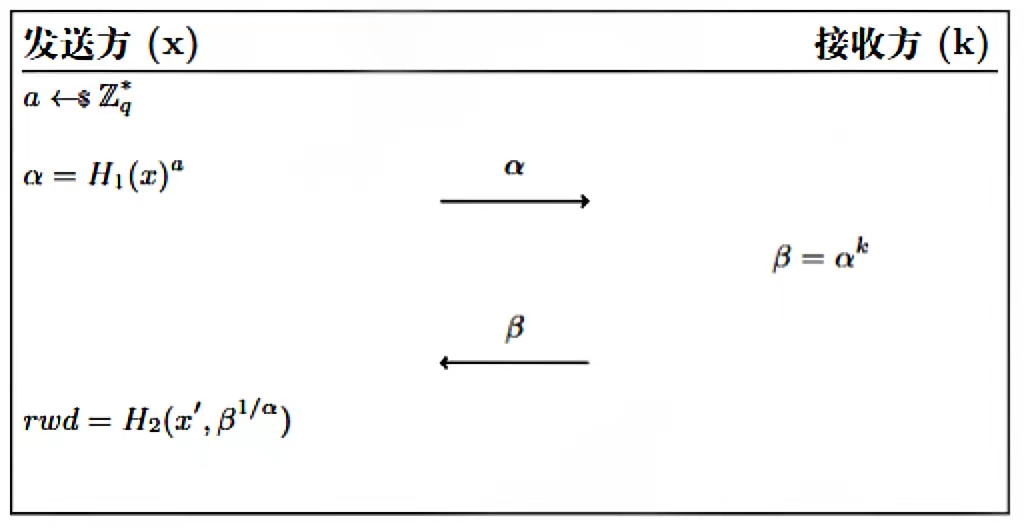


图1 基于2HashDH[16]的OPRF

### Argon2哈希函数

Argon2是 2015 年 7 月密码哈希竞赛的获胜者，这是一种有意占用资源（CPU、内存等）的单向哈希函数。在 Argon2 中，我们可以配置盐的长度、生成的哈希长度、迭代次数、内存成本和 CPU 成本，以控制哈希密码所需的资源。

Argon2有两类输入参数：必选参数和可选参数。必选参数包括要加密的消息P和随机数 S，分别代表password和salt。P的长度是0到字节，S的长度是8到字节。可选参数包括：并行程度、Tag长度、内存大小、迭代器的个数、版本号、安全值 、附加数据、以及Argon2的类型，0代表Argon2d，1代表Argon2i，2代表Argon2id。

Argon2d速度更快，并且使用数据依赖的内存访问方式，这使得它对GPU破解攻击有很强的抵抗力，适合没有side-channel timing attacks威胁的应用（例如加密货币）。

Argon2i则使用数据无关的内存访问，这对于密码哈希和基于密码的密钥推导算法来说是首选，其特点是速度较慢，因为它在内存上运行了更多的处理逻辑，以防止 tradeoff attacks 。

Argon2id是Argon2i和Argon2d的混合体，采用数据依赖型和数据独立型内存访问相结合的方式，从而可以同时抵御side-channel timing attacks和GPU破解攻击的能力。

## 2. 协议描述

### 1) 系统架构

系统由客户端和服务器组成。服务器维护一个加密数据库，加盐存储与已泄露的用户名及口令相关的记录，根据用户名的哈希前缀划分出若干个加密数据桶；客户端通过提供用户名及口令来查询口令是否泄露。

当客户端输入要查询的用户名及口令，服务器首先根据用户名的哈希前缀判断是否存在对应的数据桶，如不存在则表明未泄露，查询结束；否则，将相应的数据桶返回给客户端。客户端根据服务器的响应来判断此次查询的口令是否泄露。

### 2）安全模型

恶意客户端。恶意客户端可能希望通过查询获取另一个用户的口令。它可能知道目标的用户名，并有能力查询服务器。值得注意的是，服务器所使用的泄露数据集实际上已经被公开了，我们仍应该保护这些数据的私密性。

诚实但好奇的服务器。诚实即为它们不会伪造数据，将按照设定的协议执行，好奇则是指它们对用户的查询内容有一定程度的好奇，试图利用客户端的查询推测出更多的秘密信息，比如用户的口令。

### 3) 协议框架

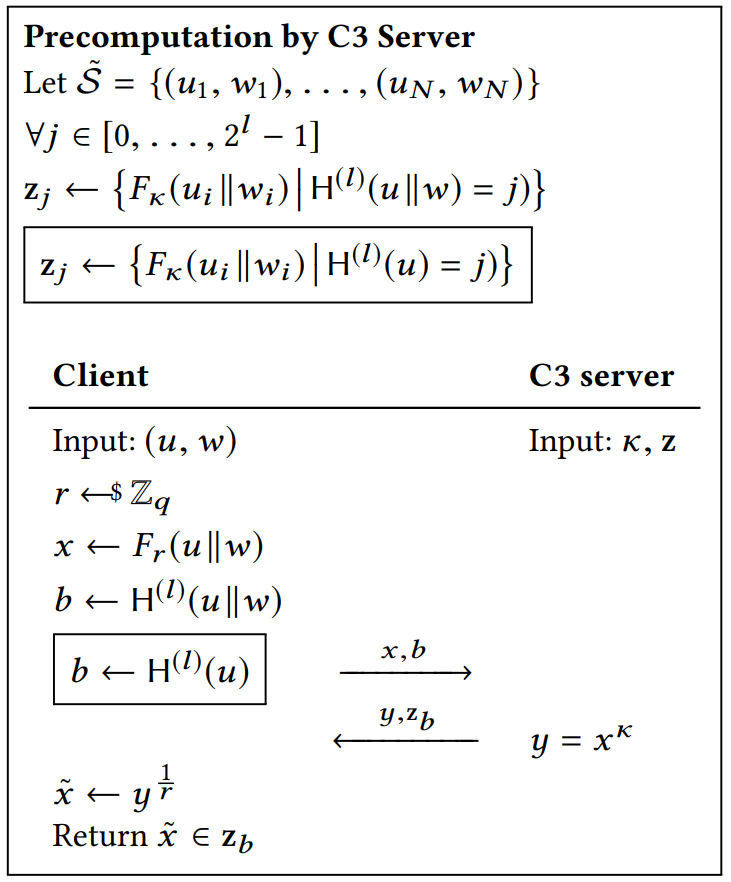


图2 GPC协议

本协议框架可以参考上图。

### （1）创建数据库

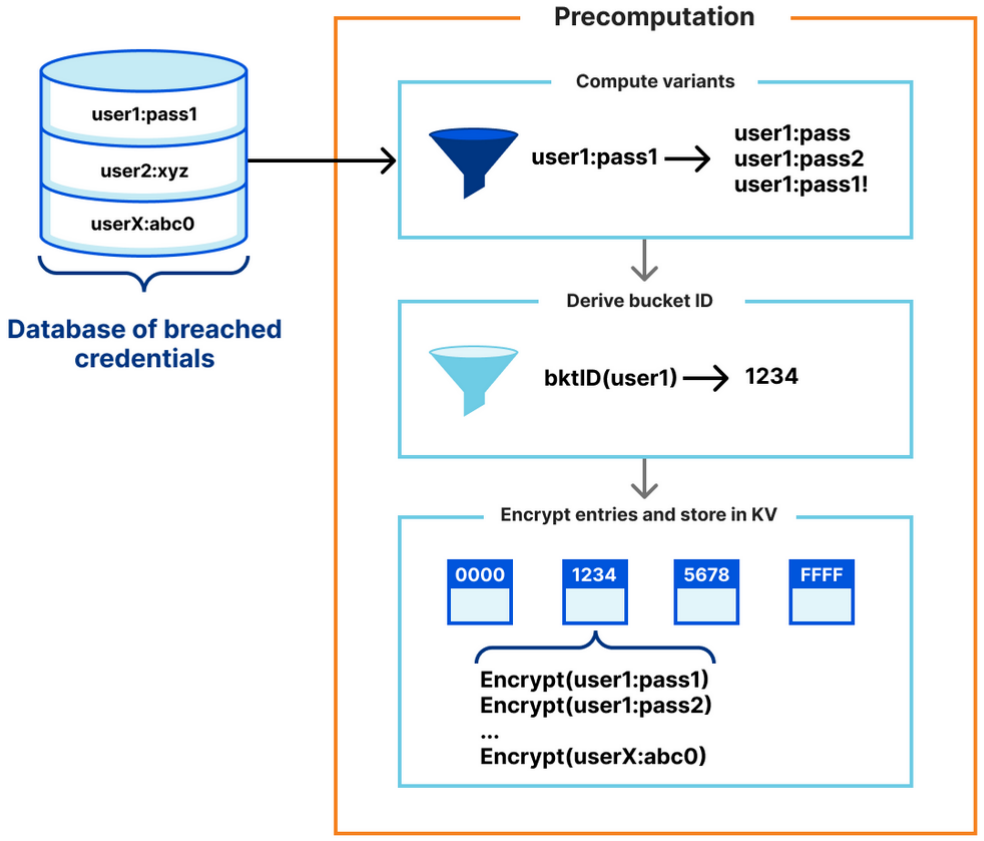


图3 数据库的与计算

数据库预处理流程可参考图3，要提供口令泄露查询的服务，服务器就必须事先构建一个包含所有已知泄露凭证的数据库。先对泄露凭证集进行数据预处理，再对数据进行加密运算得到，这里是一个伪随机函数（PRF）。接着将凭证加密后所得数据的前缀作为数据桶的桶标识符，将加密后的凭证数据存储到标识符为的数据桶中。

### （2）客户端向服务器发送查询请求

数据库准备完毕后，用户通过与客户端执行私有集合交集（PSI），以实现信息的交互。用户先在客户端输入他的用户名和口令，客户端对输入凭证计算，同时计算该凭证预估所在的数据桶的桶标识符。并将两个数据发送给服务器。

### （3）服务器向客户端发出响应

服务器用密钥对来自客户端的加密凭证进行幂运算，得到，并根据传入的桶标识符找到对应的数据桶，将和数据桶发送给客户端。

### （4）客户端进行本地检验

客户端根据传入的计算出。随后客户端检验是否在服务器传来的数据桶中。若在数据桶中，则告知用户口令已泄露。否则，则告知用户口令没有泄露。

## 3.安全性分析

### （1）数据安全性

在本系统中，为了防止敌手可能发起的离线字典攻击，服务器所维护的数据集与协议执行过程中所传输的数据均进行了加密，即使攻击者截获了客户端与服务器或者攻击服务器获取数据集，都无法获得明文信息，保证了数据的安全性。

### （2）客户端隐私

查询者匿名性：服务器仅能从用户的查询请求中知道用户名的哈希前缀，无法在足够大的数据集中锁定用户的身份。我们可以通过控制所使用哈希前缀的长度来平衡查询者匿名性与服务器端存储数据桶的大小这两个因素。

查询内容隐私：服务器无法有效地获取所查询用户名或口令的明文。OPRF的语义安全性和哈希函数的单向性保证了这一点。

查询结果隐私：服务器无法知道用户的凭据是否与其加密数据库中的条目匹配。这是通过OPRF的语义安全性来实现的，服务器无法获知OPRF输出的值，也就无法知道所查询的用户凭证是否与桶中元素相对应。

### （3）服务器隐私

除了服务器数据库中是否存在以所输入用户名的哈希前缀为索引的桶，以及查询口令的散列值是否与桶中元素匹配之外，用户无法了解有关服务器数据库的任何信息。这是通过OPRF的语义安全性来实现的，由于用户不知道服务器端用于加密的私钥，即使得到桶中的数据，也只能看到一些随机数，无法解密出含义。

### （4）对抗拒绝服务攻击（DoS）

攻击者可能不停地向服务器发送查询请求，耗费服务器的带宽或缓冲区，导致服务器不能正常提供泄露查询服务。

### （5）对抗离线猜测攻击

服务器端数据库使用基于2HashDH的OPRF加密存储，没有服务器的私钥，攻击者无法有效地恢复出明文。

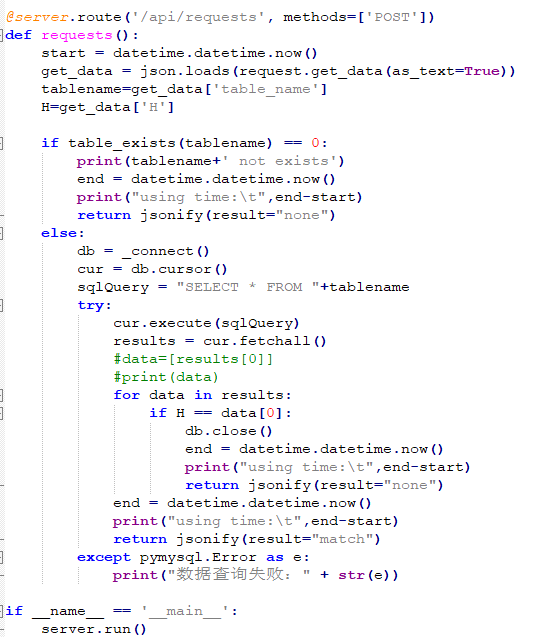
# 实验步骤

## pre.py处理数据



主要思想：所用的初始数据存储在test数据库的info数据表中，从该表中读取数据，调用solve函数来处理数据，计算哈希值并进行盲化处理，然后根据哈希前缀进行数据桶的划分并存储数据。

## server.py服务器端查询工作



主要思想：使用flask框架将其运行在http://localhost:5000/api/requests上，对于客户端发送的数据进行查询，若哈希前缀对应的表不存在，则直接返回none（没有泄露），若对应的表存在，则搜寻表中存储的数据，从而返回是否泄露。

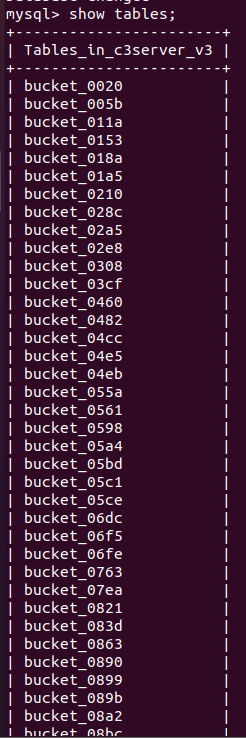
## new\_client.py客户端查询



主要思想：客户运行客户端并输入想要查询的用户名和口令，和预处理数据相似地进行数据处理。

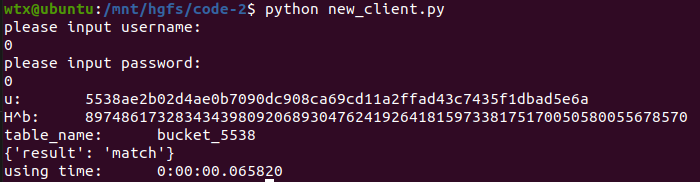
# 实验结果分析

## 1.处理数据后数据库的样子

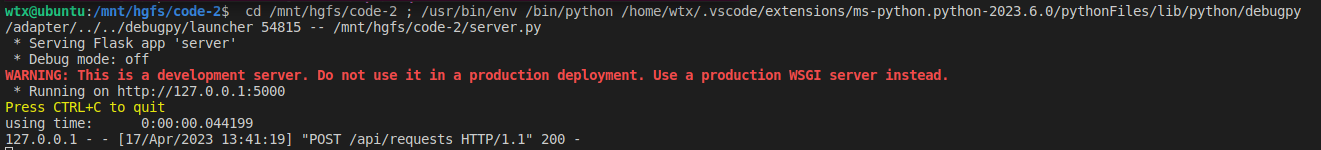


## 2.被泄露的情况下：

### Client

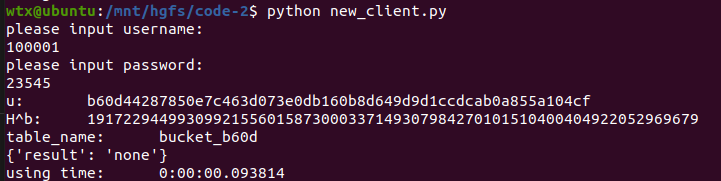


### Server



## 3.没有被泄露的情况下：

### Client



### Server



# 总结感想

Q.如果一个恶意的服务器想要返回错误的查询结果，客户端能察觉吗？思考并设计新的协议来解决这一问题，无需编程。

A.客户端由于没有对于返回信息进行验证处理，所以无法察觉。

新的协议：服务器对是否泄露的结果进行签名，并将签名后的结果发送给客户端，客户端在确认服务器身份后选择接收该消息。