|  |
| --- |
|  |
| 口令泄露查询密码协议系统构建 |
| [南开大学密码学实验案例] |
|  |
| **汪定 编著** |
|  |

|  |
| --- |
|  |

# 实验概述

作为应用最为广泛的身份认证方法，口令具有简单易用、成本低廉、容易更改等特性，在可预见的未来仍将是最主要的认证方法，因而口令安全在网络空间安全领域有着基础性的重要地位。但是，基于口令的身份认证系统需在服务器端维护一个用户口令数据库，这成为对攻击者极具价值的攻击目标。近年来，拖库、撞库等攻击手段导致大量口令数据泄露，势必给用户隐私及安全带来威胁。口令安全面临严峻挑战。

更为严重的是，很多网站往往难以及时发现这些口令泄露事件[1]，导致攻击者有足够多的时间利用这些泄露数据发动攻击，甚至将其发布在网上以获取收益。例如，2008 年发生的3.6亿 Myspace 数据泄露事件历经8年才被发现[2]；2014 年发生的 5 亿 Yahoo 口令泄露事件直到其中部分数据在2016年被出售到暗网上才被察觉[3]。此时再通知用户更新口令为时已晚。因此，及时向用户反馈口令泄露，降低泄露所带来的危害意义非凡。

在这个实验中，我们将**实现一个口令泄露查询系统**，便于用户随时可以查询自己的口令是否泄漏，及时更改密码进而保护自己的个人隐私以及财产安全。

# 目 录

[实验概述 2](#_Toc125884297)

[目 录 3](#_Toc125884298)

[第一章 基础知识 4](#_Toc125884299)

[1.1 当前研究 4](#_Toc125884300)

[1.2 论文阅读 5](#_Toc125884301)

[第二章 协议实现 6](#_Toc125884302)

[2.1 实验目的 6](#_Toc125884303)

[2.2 实验原理 7](#_Toc125884304)

[2.2.1 相关技术 7](#_Toc125884305)

[2.2.2 协议描述 9](#_Toc125884306)

[2.2.3 安全性分析 11](#_Toc125884307)

[2.3 实验内容和步骤 13](#_Toc125884308)

[2.4 实验报告和要求 15](#_Toc125884309)

[第三章 拓展部分 16](#_Toc125884310)

[参考文献 17](#_Toc125884311)

# 第一章 基础知识

课时：6课时。

## 当前研究

为了防御撞库攻击（Credential stuffing attacks），用于检查用户口令是否被泄露的第三方服务被不断提出，被称为C3（compromised credential checking）服务。例如，Troy Hunt和CloudFlare在2018年部署了HaveIBeenPWned（HIBP）[4]，之后，Firefox[5]、1Password[6]等网站也陆续部署了该服务。2019年谷歌在Chrome中推出了一个口令检查的扩展Password Checkup（GPC）[7],[8]，用来帮助Chrome用户检查他们的用户名和口令是否在谷歌收集的被破坏的凭据数据库中。同年，Li等人[9]提出了一个新协议用于检查泄漏的用户名-口令对 （IDB），该协议仅使用用户名划分桶。2021年，微软在Edge中引入了口令监控功能Microsoft Password Monitor[9]，据称这是第一个利用同态加密保护用户隐私的消费者应用程序。

然而，这些服务只检查确切的口令是否已经泄露，容易遭受最近提出的凭证调整攻击[11],[12],[13]，即攻击者会猜测用户泄露口令的变体。在现实中，用户口令重用现象较为普遍，用户经常会在不同的网站上选择相似的口令（例如，根据网站策略将旧口令中的部分小写字母替换成大写字母，或简单地在旧口令后面添加一个“1”）[11],[14],[15]。Pal等人[12]通过模拟估计，尽管使用了C3服务，但这种凭证调整攻击者可通过不到10次猜测内破坏10%的已经泄露口令的用户的其他帐户。

为解决这个问题，Pal等人[12]设计了“Might I Get Pwned”(MIGP) 。该模型基于他们首次提出的可感知相似度的C3服务，对于用户输入的口令，不仅查询该口令本身是否泄露，还检查该口令的相似变体是否存在泄露数据集中，以警告用户不要选择与已泄露口令相似（或相同）的口令，这增强了客户端泄露检测的及时性。然而，该方案采取在客户端生成额外变体的策略，恶意客户端可能会借此扩大猜测个数，从而加快突破服务器数据集。此外，服务器也缺乏相应的措施来感知客户端的恶意猜测行为。

## 论文阅读

自主阅读以下两篇文章（6课时），理解口令泄露查询协议基本概念、基本安全模型、常见攻击方法（online guessing attacks and breach extraction attacks），指出协议存在的各种安全漏洞，改进现有协议，或设计新的安全高效的协议。

1. Protecting accounts from credential stuffing with password breach alerting.

协议描述在论文的Algorithm 1~4，以下简称为GPC[8]。

1. Protocols for Checking Compromised Credentials.

协议描述在论文的Figure 5，以下简称为IDB[9]。

# 第二章 协议实现

课时：6课时。

## 2.1 实验目的

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；

数据报方式不限，可基于TCP或UDP；

程序界面不限，可使用命令行界面或图形化界面。

实验环境不限，可基于Linux/Windows等

## 2.2 实验原理

### 2.2.1 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[16]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。

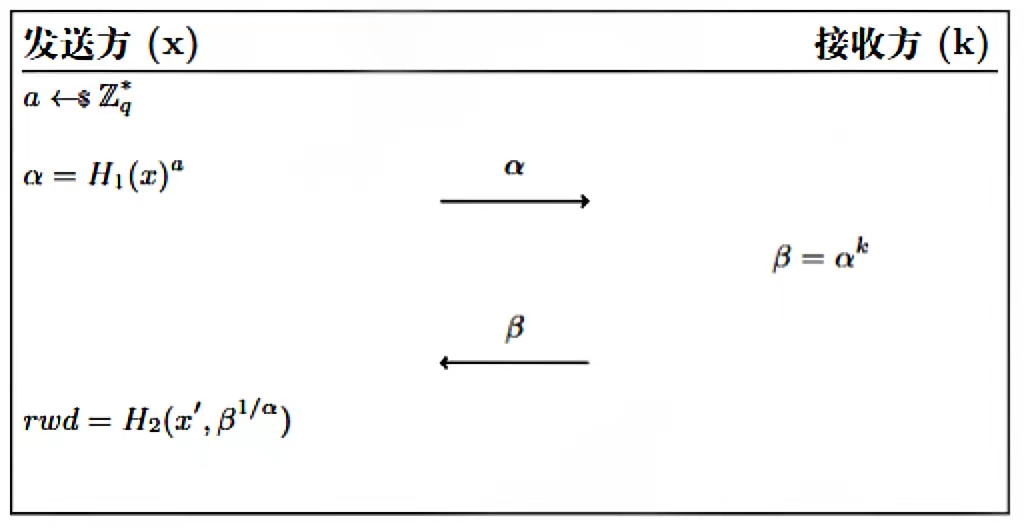


图1 基于2HashDH[16]的OPRF

**2) Argon2哈希函数**

Argon2[17]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。

### 2.2.2 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**

1. **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。
2. **诚实但好奇的[18]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**

本协议框架可以参考图2。

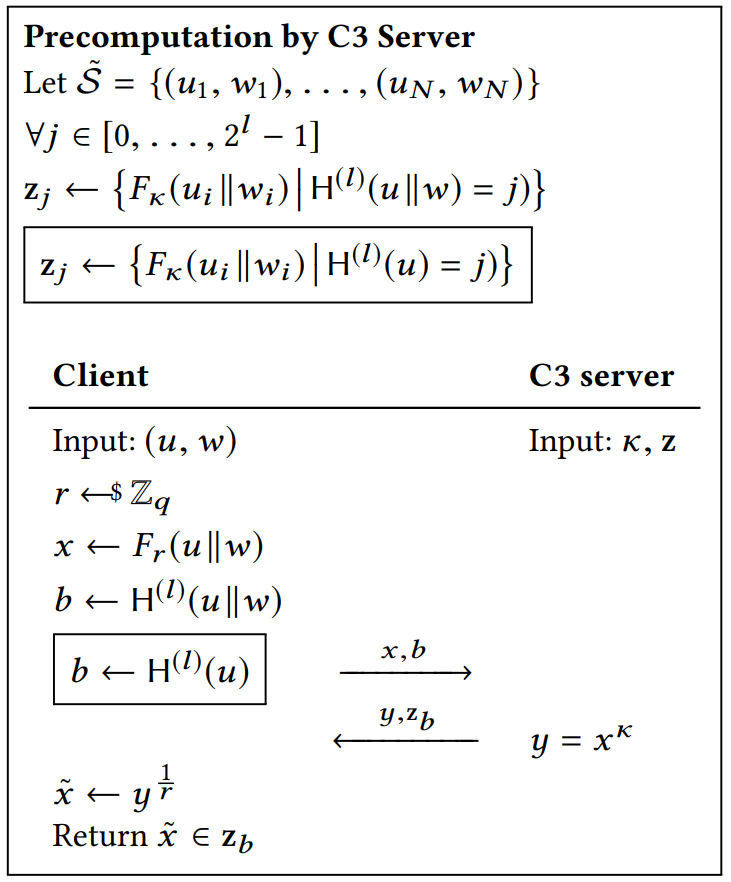


图2 GPC协议[8]

（1）创建数据库

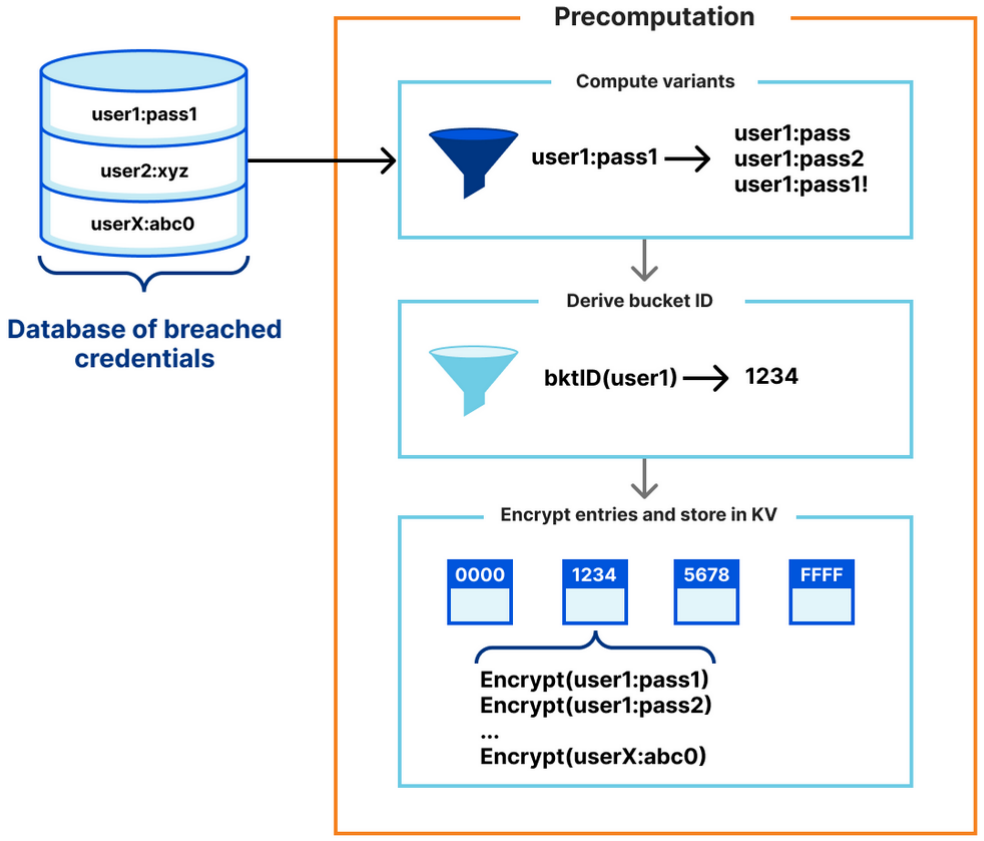


图3 数据库的与计算

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。

### 2.2.3 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。

1. 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。

1. 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。

1. 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[19]。

1. 对抗离线猜测攻击

服务器端数据库使用基于2HashDH[16]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。

## 2.3 实验内容和步骤

服务器预处理文件，最终成功运行。

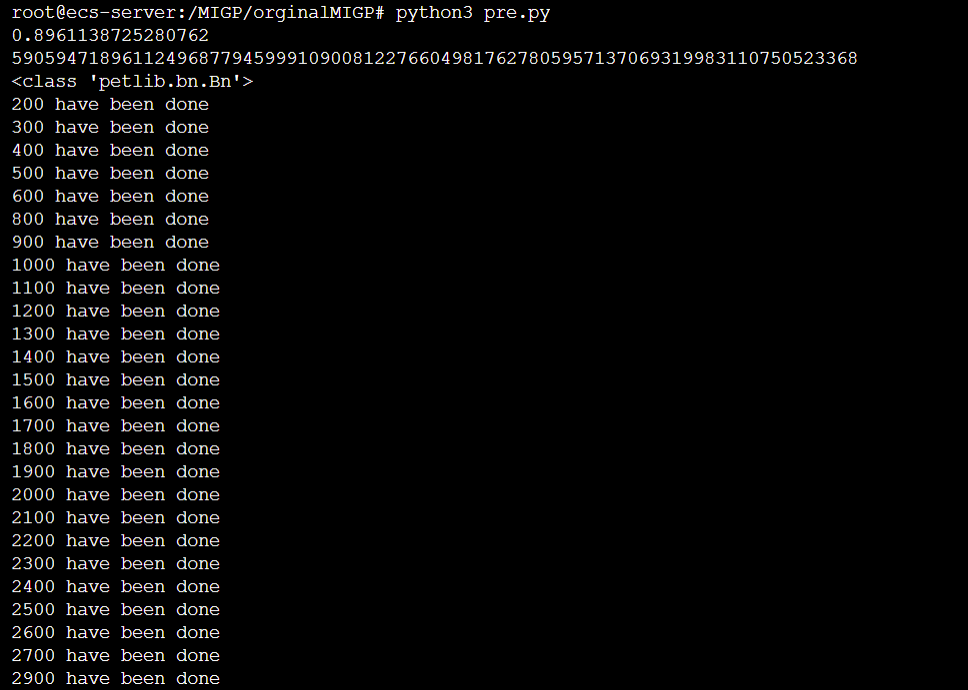


图4 服务器预处理运行结果

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。

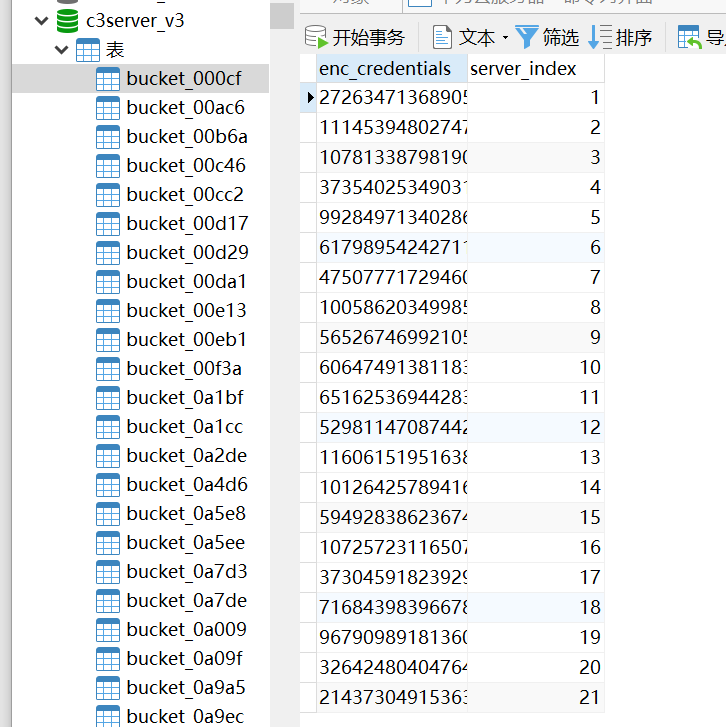


图5 服务器端的数据库

启动服务器，可正常运行。

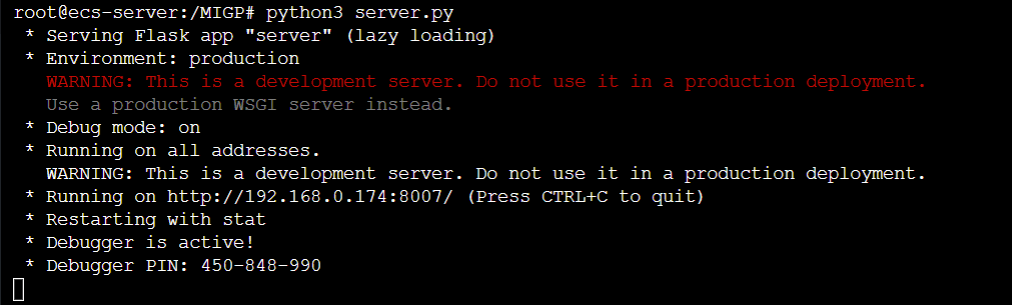


图6 服务器程序运行

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。

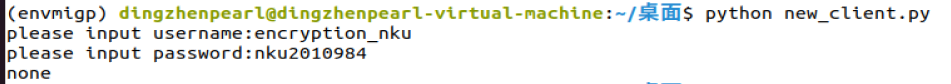


图7 客户端在口令未泄露情况下的运行示例

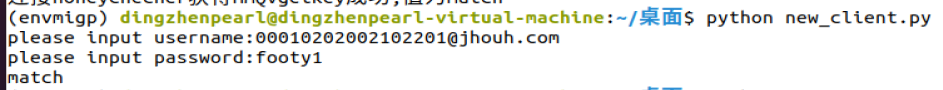


图8 客户端在真实口令泄露情况下的运行示例

## 2.4 实验报告和要求

实现协议并撰写实验报告（6课时），模板如下：

|  |
| --- |
| **密码学课程第 次实验报告**  **实验名称：**  学号： 姓名： 班级：   1. 实验目的 2. 实验内容说明   （概述本次实验要做什么）   1. 实验原理   （画图并简要说明）   1. 实验步骤   （分布介绍依次完成了哪些代码，从而实现了什么样的功能）   1. 实验结果分析   （给出运行结果截图，注意需要对实验结果进行分析，包括服务器存储开销，服务器端预处理运行时间，客户端查询完成时间，查询结果是否正确）   1. 总结感想   （说说本次实验的总结感想） |

# 第三章 拓展部分

解决以下任意一个问题，额外增加10~20分。

1. 支持相似口令泄露查询，参见MIGP协议[12]，需编程实现。
2. 当仅查询口令（不是用户名-口令对）是否泄露时，考虑通过重新规划桶来减少客户端的口令泄露，参见FSB协议[9]，需编程实现。
3. 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。
4. 如果服务器使用的密钥泄露了，服务器能及时发现吗？思考并设计新的协议来解决这一问题（可以考虑与honeywords[1],20][]的结合），无需编程。

# 参考文献

1. Ding Wang, Yunkai Zou, Qiying Dong, Yuanming Song, Xinyi Huang. How to attack and generate honeywords[C]//Proc. IEEE S&P 2022:489-506.
2. Charles Weir. Cracking the MySpace List–First Impressions[EB/OL],July 2016. <Http://reusablesec.blogspot.com/2016/07/cracking-myspace-list-first-impressions.html>
3. Cox J. Yahoo ’aware’ hacker is advertising 200 million supposed accounts on darkweb[EB/OL].2016. <https://www.vice.com/en/article/aeknw5/yahoo-supposed-data-breach-200-million-credentials-dark-web>.
4. Troy Hunt. 2018. Have I Been Pwned? [EB/OL] <https://haveibeenpwned.com/Passwords/.>
5. Testing Firefox Monitor, a New Security Tool. [EB/OL].<https://blog.mozilla.org/futurereleases/2018/06/25/testing-firefox-monitor-a-new-security-tool/.>
6. Finding pwned passwords with 1Password. [EB/OL]. <https://blog.1password.com/finding-pwned-passwords-with-1password/.>
7. Jennifer Pullman, Kurt Thomas, and Elie Bursztein. 2019.Protect your accounts from data breaches with Password Checkup. [EB/OL]. <https://security.googleblog.com/2019/02/protect-your-accounts-from-data.html.>
8. Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley,Luca Invernizzi, Borbala Benko,Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. Protecting accounts from credential stuffing with password breach alerting [C]//Proc. USENIX Security 2019:1556-1571.
9. Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, Thomas Ristenpart. Protocols for Checking Compromised Credentials [C]//Proc. ACM CCS 2019:1387-1403
10. Kristin Lauter, Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. 2021. Password Monitor: Safeguarding passwords in Microsoft Edge. [EB/OL]. <https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/.>
11. Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, XiaoFeng Wang. The tangled web of password reuse [C]//Proc. NDSS 2014: 23-26.
12. Bijeeta Pal, Tal Daniel, Rahul Chatterjee, Thomas Ristenpart. Beyond credential stuffing: Password similarity using neural networks[C]//Proc. IEEE S&P 2019: 417-434
13. Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, Xinyi Huang. Targeted online password guessing: An underestimated threat[C]//Proc. ACM CCS 2016: 1242–1254
14. Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, Alain Forget. Let’s go in for a closer look: Observing passwords in their natural habitat[C]//Proc. ACM CCS 2017: 295–310.
15. Yinqian Zhang, Fabian Monrose, Michael K. Reiter. The security of modern password expiration:an algorithmic framework and empirical analysis[C]//Proc. ACM CCS 2010: 176–186
16. Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model[C]//Proc. ASIACRYPT 2014: 233–253.
17. Jean-Philippe Aumasson. The password hash Argon2, winner of PHC [EB/OL]. <https://github.com/P-H-C/phc-winner-argon2>
18. Oded Goldreich, The Foundations of Cryptography-Volume 2: Basic Applications. Cambridge UniversityPress [EB/OL].[http://www.wisdom.weizmann.ac.il/%7Eoded/focvol2.html](http://www.wisdom.weizmann.ac.il/%7Eoded/focvol2.html%20)
19. 李飞著,信息安全理论与技术,西安电子科技大学出版社,2016.03,第140页
20. Ding Wang, Haibo Cheng, Ping Wang, Jeff Yan, Xinyi Huan. A security analysis of honeywords[C]//Proc. NDSS 2018:1–15.