浙江大学

本 科 生 毕 业 设 计
开题报告

学生姓名:	宋志平
学生学号:	3120101274
指导教师:	卜佳俊
年级与专业:	计算机科学与技术 12 级
所在学院:	计算机学院

一、题目：　　　　　个人密码的同步管理应用　　　　　　　　

二、指导教师对开题报告、外文翻译和中期报告的具体要求：

密码管理器（Paasword）是一种新型的互联网应用，得到了学术界和产业界的广泛关注，涉及计算机安全、密码学等多个方向的研究。

围绕市面上流行的密码管理器应用，跟踪和调研相关领域取得的最新技术和成果，完成 10～15 篇中英文文献的阅读，并在此基础上完成 3000 字以上的文献翻译。开题报告方面，要求 3000 字以上，要求阐明本领域所涉及到的基本理论、本人的研究内容、研究目标以及进度安排，并拟定研究方案。

此外，希望通过大量阅读文献，理解研究领域的普遍思路，并理解其中的数学理论，培养严谨、细致的研究作风，为之后的科研工作打下良好的基础。

指导教师（签名）__________

2016 年 **3** 月 **30** 日

毕业设计开题报告、外文翻译和中期报告考核

导师对开题报告、外文翻译和中期报告评语及成绩评定：

宋志平同学在其所进行的毕业设计“个人密码的同步管理应用”中，通过全面深入的项目调研，确定实施方案，并通过编程实现和大量实验，达到了导师所要求的个人密码的同步管理应用（毕设题目）的要求。毕业设计语句通顺，逻辑严密，条理清楚，有一定应用价值达到了本科生毕业论文的水平。

成绩比例	开题报告 占（20%）	外文翻译 占（10%）	中期报告 占（10%）
分 值			

导师签名___________

年　　月　　日

答辩小组对开题报告、外文翻译和中期报告评语及成绩评定：

成绩比例	开题报告 占（20%）	外文翻译 占（10%）	中期报告 占（10%）
分 值			

开题报告答辩小组负责人（签名）___________

年　　月　　日

-

目录

本科毕业设计开题报告

1. 项目背景

随着互联网的快速兴起，如何管理自己众多的账号、密码就成为了一个问题。一方面是忘记账号、密码给人带来的麻烦，找回账号、密码需要一定的时间。另一方面是安全性上的挑战，很多人的密码过于简单，并且多个账号共用一个密码，一旦被破解可能会带来连环损失。

在进入了互联网云时代之后，这个问题显得更为重要。与 PC 时代的单一终端不同，现在的人往往拥有多个互联网终端设备：PC、手机、平板等。因此，在不同的平台上，同步这些私人重要信息显得尤为重要。

另外，还有一些非个人账号密码类的重要信息需要保存，它们往往难以记忆。例如服务器和数据库的登录信息、重要人物的联系方式(手机、Email)、软件序列号等等。

结合当今流行的几款互联网产品和使用经验来看，需要机密保存的对象可以被简单地分为以下几种：

- 非重要的账号、密码。

他们多数为一些论坛、网络游戏的账号密码。在数量上，这一类型的账号、密码最多，用户忘记账号、密码可以说是常事。它们往往可以比较容易地被找回，一旦发生账号、密码被盗的情况，对于用户而言损失有限。

- 重要的账号的密码，例如 QQ、微信、Facebook 等。

除了社交账号之外，一些网站的高价值密码也可被归入其中，例如 GitHub等。这些账号对于个人用户而言价值颇高，一旦被盗，会对用户造成不小的损失。他们往往也是被盗次数最多的账号，例如在公共网络或是网吧等环境下登录 QQ，很容易产生盗号情况。

- 关系到个人财产的账号密码。

例如银行、证券账户的密码。这些账号的价值很高，一旦被盗往往会带来不可挽回的损失。而银行、证券账户的密码往往比较简单，多为 6 位数字。银行、证券账户的密码又难以被暴力破解，银行系统往往有输入次数限定。6 位的数字方便记忆，一般而言不会被暴力破解，通常而言不适合存储在云端。

- 服务器、数据库系统的登录信息。

服务器地址、端口、登录帐号、密码往往难以记忆，被盗也会带来一定的损失。现在，有一些保险箱应用提供了保存这些信息的选项。

- 重要的非密码类信息。

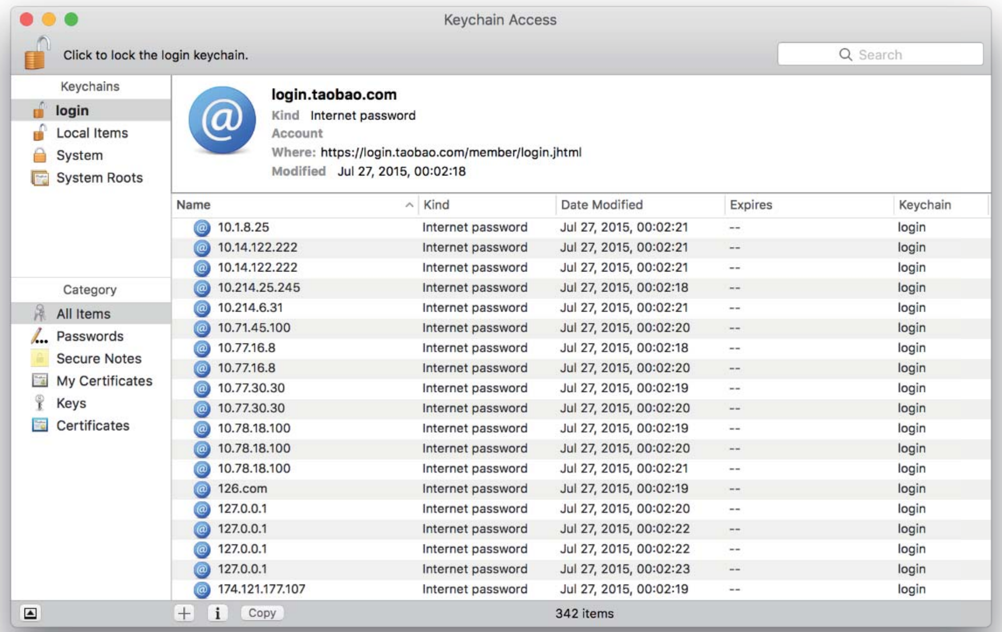
例如个人隐私的笔记、银行卡账号、重要人物的联系方式等。

下面介绍部分市面上的类似产品：

- Keychain Access

钥匙串是苹果公司 Mac OS 中的密码管理系统。它在 Mac OS 8.6 中被导入，并且包括在了所有后续的 Mac OS 版本中，包括 Mac OS X。一个钥匙串可以包含多种类型的数据：密码（包括网站，FTP 服务器，SSH 帐户，网络共享，无线网络，群组软件，加密磁盘映像等），私钥，电子证书和加密笔记等。

钥匙串访问（Keychain Access）是一个 Mac OS X 应用程序，它允许用户访问和配置钥匙串的内容（包括网站，FTP 服务器，SSH 帐户，网络共享，无线网络，群组软件，加密磁盘映像等内容的密码等），加锁或解锁钥匙串，显示系统存储的密码，管理根证书，密钥和加密笔记。



login.taobao.com
Kind Internet password
Account
Where: https://login.taobao.com/member/login.jhtml
Modified Jul 27, 2015, 00:02:18

Name	Kind	Date Modified	Expires	Keychain
10.1.8.25	Internet password	Jul 27, 2015, 00:02:21	--	login
10.14.122.222	Internet password	Jul 27, 2015, 00:02:21	--	login
10.14.122.222	Internet password	Jul 27, 2015, 00:02:21	--	login
10.214.25.245	Internet password	Jul 27, 2015, 00:02:18	--	login
10.214.6.31	Internet password	Jul 27, 2015, 00:02:21	--	login
10.71.45.100	Internet password	Jul 27, 2015, 00:02:20	--	login
10.77.16.8	Internet password	Jul 27, 2015, 00:02:18	--	login
10.77.16.8	Internet password	Jul 27, 2015, 00:02:20	--	login
10.77.30.30	Internet password	Jul 27, 2015, 00:02:19	--	login
10.77.30.30	Internet password	Jul 27, 2015, 00:02:20	--	login
10.78.18.100	Internet password	Jul 27, 2015, 00:02:19	--	login
10.78.18.100	Internet password	Jul 27, 2015, 00:02:20	--	login
10.78.18.100	Internet password	Jul 27, 2015, 00:02:21	--	login
126.com	Internet password	Jul 27, 2015, 00:02:19	--	login
127.0.0.1	Internet password	Jul 27, 2015, 00:02:20	--	login
127.0.0.1	Internet password	Jul 27, 2015, 00:02:22	--	login
127.0.0.1	Internet password	Jul 27, 2015, 00:02:22	--	login
127.0.0.1	Internet password	Jul 27, 2015, 00:02:23	--	login
174.121.177.107	Internet password	Jul 27, 2015, 00:02:19	--	login

图 1　苹果 Keychain Access

- 1Password

1Password 是一个由 AgileBits 公司开发的密码管理软件。它能用来存放各种不同的密码，除普通登录密码，信用卡、软件许可证等敏感信息也可一同存放在 PBKDF2 加密的虚拟保险箱里，由主密码保护。1Password 支持 Windows、Mac、iOS、Android 等主流操作系统，Mac / iOS 版本已经支持中文。

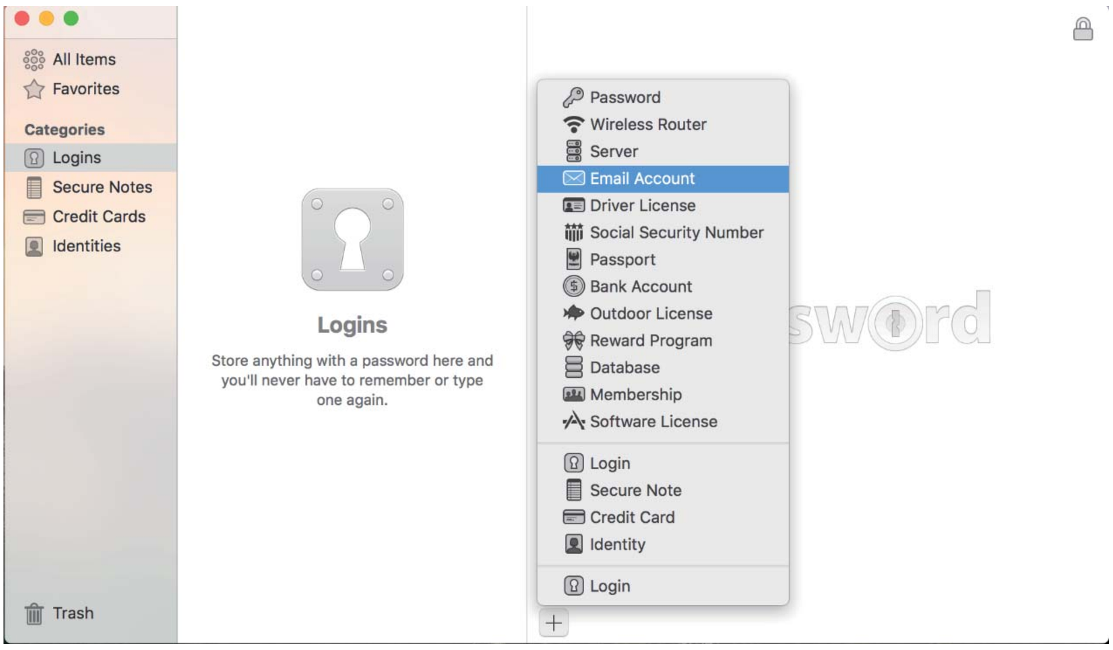


图 2　1Password 在 OS X 上的界面

1Password 可以在本地存放密码，不与任何远程服务器接触。也可以选择与 Dropbox（全平台），本地 Wi-Fi（仅 Mac 与 iOS 设备），或 iCloud（仅 Mac 与 iOS 设备）同步。只需记住一个主密码（Master Password），并用其来保护并管理所有密码，是 1Password 密码管理方案的主要理念。

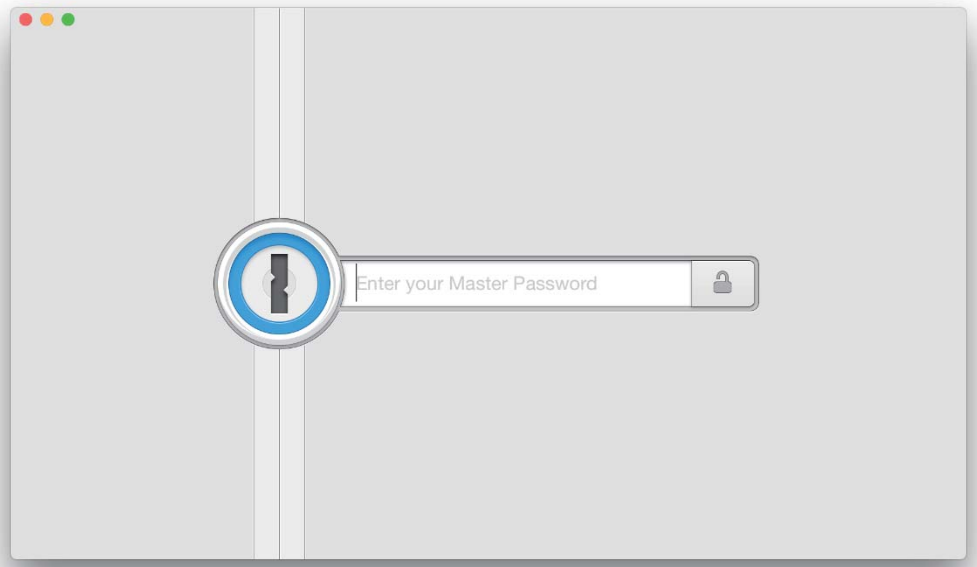


图 3　1Password 的主密码界面

2. 目标和任务

制作一个轻量化的个人密码的同步管理应用，它可以跨平台（Android/Web）同步用户的私密个人信息。本应用以一个主密码（Master Password）对用户密码进行管理，用户只需要记住 Master Password，就可以保护和管理所有密码。

在 Android 设备上，用户可以查看和管理自己的个人账号、密码，并且在 app 的登录过程中，可以选择是否将密码存入密码管理器中。在网页端，在保证安全的情况下，用户可以查看或是修改自己的个人私密信息。另外，制作一个运行于 Chrome 浏览器上的应用插件，方便用户管理密码。

密码管理器的核心是安全问题，本应用在安全问题上，应该要做到以下几个方面：

1. 使用安全的数据格式，用来保存应用数据。
2. 抵御传输过程中的网络攻击，保证数据难以被盗窃。
3. 保证数据库系统的安全，让一般的网络攻击者难以进入数据库获得数据。

考虑到有一部分人需要使用随机密码，因为这样可以降低通过字典被暴力破解的概率。个人密码的同步管理应用还需要一个密码生成器，用户可以通过指定密码的位数、数字的数量和特殊字符的数量来生成随机密码。

除此之外，本应用还应该做到跨平台的外观一致性，保证在不同的平台上，UI 的风格相似，带给用户出色的用户体验。

3. 可行性分析

- 技术可行性

从技术上来说，本应用主要有 4 个技术上的部分：加密算法、Android 和 Web 前端、服务器后端和 Chrome 插件扩展程序。 这 4 个部分上均有相对成熟的技术和解决方案，具体的关键技术考虑，放在第 4 节。

- 实用可行性

从实用性的角度看，对于大多数互联网用户而言，有管理密码的需求。

现在的互联网应用更强调轻量化，在 PC 端通过浏览器交互比通过客户端交互更为轻量化，今天市面上的大多数产品在 Windows/Mac 端仍然离不开客户端。此次毕业设计在应用的轻量化有一定的创新性。

- 时间可行性

对于本科的毕业设计而言，制作一个密码管理器在工作量、难易程度上适中。可以确保在 2 个多月的时间内做完，也有一定的技术难度。

4. 初步技术方案和关键技术考虑

本应用以一个主密码（Master Password）对用户密码进行管理，用户只需要记住 Master Password，就可以保护和管理所有密码。本应用大致分为 4 个部分：云服务器、Android 客户端、网页端、Chrome 插件，具体的结构如下所示：

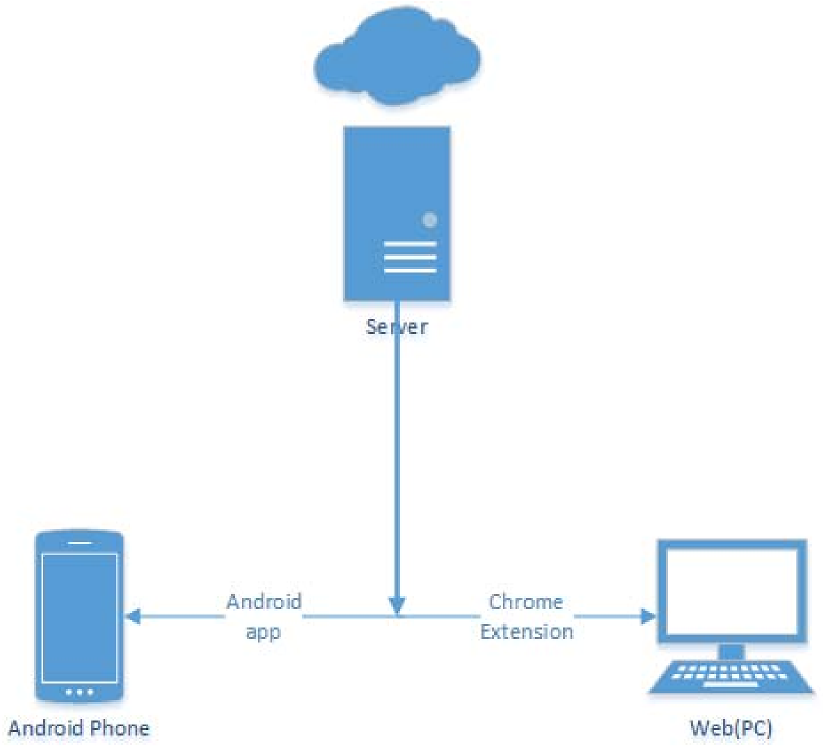


图 4　程序总体架构

将用户的私密数据分为以下几类：登录帐号，用于存放某网站/某应用的用户名和密码；银行卡，用于存放银行卡相关信息，但并不包含银行卡密码；身份名片；服务器，用于存放服务器的登录和服务提供商等信息；数据库，用于存放数据库的登录信息。具体的数据格式如下所示：

表 1 登录帐号（Logins）

列字段名	中文名	说明
website	网站名	用于标识在哪个网站
username	用户名	用户名
password	密码	用户名所对应的密码
labels	标签	标识它属于哪一类的登录帐号
notes	注释	与本帐号相关的一些注释、辅助信息

表 2 银行卡（Bank card）

列字段名	中文名	说明
type	类型	借记卡、信用卡或是其他类型
number	银行卡号	记录银行卡的卡号
bands	信用卡品牌	可多选，提供常见的信用卡品牌供选择，例如 Visa、MasterCard、UnionPay、JCB 等
valid_from	起始日	记录银行卡的有效起始日期
expiry_date	到期日	记录银行卡的到期日期
labels	标签	标识它属于哪一类的银行卡(例如工资卡、双币卡等)
notes	注释	与本银行卡相关的一些注释、辅助信息

表 3 身份名片（Indentity）

列字段名	中文名	说明
name	姓名	记录姓名
sex	性别	记录性别
birthdate	生日	记录生日
phone	电话	记录电话
email	电子邮箱	记录性别
company	公司	记录公司
labels	标签	标识此人属于哪一类的人(同学、同事、上司等)
notes	注释	与相关的一些注释、辅助信息

表 4 服务器(server)

列字段名	中文名	说明
address	地址	服务器的地址
username	用户名	登录用户名
password	密码	用户名对应的密码
port	端口号	记录端口号
labels	标签	标识服务器类型
notes	注释	与服务器相关的一些注释、辅助信息
provider	服务提供商	服务器提供商相关的信息

表 5　数据库（database）

列字段名	中文名	说明
type	数据库类型	标识数据库的类型，例如 Mysql、SQL Server、Oracle 等
username	用户名	登录时的数据库用户名
password	密码	用户名对应的密码
database	数据库名称	数据库名称
connection options	连接方式	使用何种方式连接数据库
labels	标签	标识它属于哪一类的数据库
notes	注释	与该数据库相关的一些注释、辅助信息

个人密码的同步管理应用还需要拥有主密码的控制器，例如应用在手机上被切出或是一段时间不用后，需要再次输入 Master Password。这样做可以减少密码或是关键信息被偷窥的可能，增强了程序的安全性。

在实现过程中，主要的难点有加密算法、Web 前端和加密通信、数据库系统这三个难点，具体的分析如下：

- 加密算法

考虑使用高级加密标准（Advanced Encryption Standard，AES）加密算法对用户的密码进行加密。AES 是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的 DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（National Institute of Standards and Technology，NIST）于 2001 年 11 月 26 日发布于 FIPS PUB 197，并在 2002 年 5 月 26 日成为有效的标准。2006 年，高级加密标准已然成为对称密钥加密中最流行的算法之一。

AES 在软件及硬件上都能快速地加解密，相对来说较易于实现，且只需要很少的内存。并且，AES 在运行 Android 系统的 ARM 上 已经被广泛应用。

- Web 前端和加密通信

考虑使用 Javascript 编写前端，并将前后端分离，使用加密通信的方式传输数据。

- 数据库系统

考虑使用互联网云服务器，在 Linux 服务器上搭建 MySql 数据库系统。

相对 NoSQL 式数据库，传统的关系型数据库更为稳定，对数据库的 ACID 要求更为严格，尤其更契合本应用对原子性和一致性的要求。对数据的一次增加/修改/删除应该被看作一次标准的 transcation，因此使用传统的 SQL 式数据库更为合适。

5. 预期工作结果

在基于云服务器的数据库平台上，制作出一个可用于多平台同步、轻量化的个人密码管理器。它要做到能通过一定量级的网络安全测试，抵御一定强度的网络攻击。此外在美观性上，本应用要做到多平台下的 UI 风格统一、页面和 app 布局合理，便于用户的使用。

6. 进度计划

- 在 4 月中旬完成服务器后端的设计，并有简单的前端页面。
- 在 5 月上旬完成 Android 端的设计，并有完成版的前端页面。
- 在 5 月下旬完成对 Chrome 插件扩展程序的设计，预留出时间进行 bug 的修改。

本科毕业设计外文翻译

原文：On the Security of Password Manager Database Formats

聚焦密码管理器数据库格式的安全性

摘要

密码管理器是一类重要的软件，用户用它去安全地保存高价值和敏感的信息，从网上银行密码、登录信息到护照和社保卡账号。令人惊讶的是，目前鲜有针对这些应用程序提供的安全性所做的学术研究。

本文带来了第一个严谨的分析，就这些流行的密码管理器的存储格式。我们定义了两个现实的安全模型，用于表示真实世界中敌人的能力。接下来，我们将会展示当我们的模型被敌人攻击时，是如何的不堪一击。我们的分析表明了绝大多数的密码管理器数据库格式，甚至会被那些水平低的攻击者攻破。

1. 引言

互联网上提供的服务数量不断增加，平均每个用户所需要记住的密码数量也相应地不断增加，以至于对于大多数用户而言，记住每一个账户的新高强度密码，成为了不可能。

用户通常有两种方式来解决这一个问题。一种常见的方式是在不用的网站上复用相同的密码。这种方式增加了潜在风险，如果密码被盗、被破解或是有一个服务向攻击者妥协，攻击者就可能在用户的所有在线服务上复用这些密码。另一种方式是使用“密码管理器”，对每一个不同的网站存储高强度的(随机的)密码。密码管理器是一种软件，它要求用户记住一个高强度的密码用来访问用户密码的数据库。记住一个高强度的密码，对于那些在不同的在线服务设置不同的密码为得到安全性的用户而言更为合适。

使用密码管理器还有其他的潜在好处。完整的 URL（或者是域名）往往和对应的密码存储在一起，用来自从填写登录表单。因此，那些依赖密码管理器的用户更难以成为网络钓鱼攻击的受害者，即使用户进入了一个与用户目标网站外观相仿的恶意网站，因为密码管理器不会自动登录，提供了一层额外的保护。

用于储存在密码数据库中的信息通常十分敏感，大多数的密码管理器保护其中的内容，禁止未授权的访问。数据库格式一般依赖于数据保护的加密码，通常是通过用户输入的主密码产生的加密/解密码。

这种保护旨在允许用户在未经信任的地方保存密码数据库。大多数密码管理器的

开发商建议将密码数据库存储在 USB 设备、云端或是移动设备上，旨在快速访问保存的密码。这些存储方式也会使潜在的攻击者掌握整个数据库。即使数据库密码存储在本地硬盘上，攻击者也可能通过其他方式获取到一份拷贝。

如果密码管理器的数据库格式是不可信的，那么再好的密码管理器的优点都会被掩盖，用户或许会变得更不安全并且更容易受到个人信息泄露带来的感染：隐私意识强的用户想让他们的浏览习惯变得私密化，因此会经常删除 cookie、历史和缓存记录。正相反，密码管理器代表了长期的存储场所，存储着密码的唯一拷贝，因此其中的内容通常永远不会被删除。如果一个密码管理器的数据库通过储存未被加密的 URL 的方式，泄漏了例如浏览习惯的信息，那么清除缓存和浏览记录并不能阻止攻击者获取敏感信息。在本论文中，我们将分析几个最流行的密码管理器的数据格式所带来的安全性。我们定义了两个不同的对手：一个被动的攻击者，他只会从密码数据中推断信息；一个主动的攻击者，他会修改原始数据的内容。我们强调，使用“行业标准方法”，例如 AES-CBC 来保证安全的数据库格式是不够的，即使 AES-CBC 的实现方式是正确的。我们并不打算提供一个详尽的列表，记录对于所有密码管理器的所有可能的攻击。更确切的说，我们将常见的密码管理器数据格式作为模型，并且给出的典型攻击的例子。

这篇论文的内容如下所示：第 2 节给出了本研究中，一个密码管理器的概述；第 3 节介绍了我们的系统和攻击模型；第 4 节分析了模型中不同的数据库格式。在第 5 节中，我们将讨论各种数据库格式的一般化问题，第 6 节是一些相关工作。我们的结论在第 7 节。

2. 密码管理器概述

密码管理器们在不同的方面表现各异，包括数据库格式、功能性、开源性、支持的平台和是否支持云存储。表 1 总结了我们认为的密码管理的主要特点。一些流行的密码管理器创造他们自己的数据库格式，被他们独家使用。尤其是对那些嵌入在浏览器中的密码管理器而言。

表 1 这张表详细展示了我们分析的密码管理器，包括了软件使用的数据库格式、存储选项和支持的平台。另外我们表示了是否开源和密码管理器是否被集成在浏览器中。

密码管理器	数据库格式	存储	开源	平台	浏览器集成
Google Chrome	Chrome	本地/云	是	Win/Mac/Linux	是
Mozilla Firefox	Firefox	本地/云	是	Win/Mac/Linux	是
Internet Explorer	MSIE	本地	否	Win	是

1Password	1Password	本地/云	否	Win/Mac	否
KeePass 1.x	KDB	本地	是	Win	否
KeePass 2.x	KDB/KDBX4	本地	是	Win/Mono	否
KeePassDroid	KDB/KDBX4	本地	是	Android	否
KyPass	KDB/KDBX4	本地	是	iOS	否
PassDrop	KDB/KDBX4	本地	否	iOS	否
PINs	PINs	本地	否	Win	否
Password Safe	PasswordSafe	本地	是	Win	否
Password Gorilla	PasswordSafe	本地	是	Win/Mac/Linux	否
Roboform	Roboform	本地/云	否	Win/Mac/Linux	是

我们把这些写进我们的分析里，因为这些密码管理器被广泛使用。一些独立的密码管理器共享相同的数据库格式，所以虽然每个密码管理器带给用户不用的体验，构成基础的存储格式却是相同的。

这篇论文的剩余部分将仅仅聚焦于数据库格式和他们提供的安全性，而不是每种密码管理器的实现。我们假设每种密码管理器正确地实现了各自说明的格式。因此，我们不考虑对于加密边信道攻击或者对于具体实现的攻击。相反，我们调查特定存储格式所提供的安全性。因为这个原因，我们的分析主要着眼于提供本地存储的密码管理器。我们将在后面的工作中分析云端的密码管理器。

我们调查了 9 种流行的密码管理器的数据库格式。3 种数据库格式是集成在浏览器中的密码管理器所用：Google Chrome，Mozilla Firefox 和微软 Internet Explorer；6 种格式被大量的独立密码管理器所用：1Password，KDB、KDBX4、PasswordSafe v3，PINs 和 RoboForm。

3. 对手和系统模型

我们考虑两种高效率的对手：Adv_r拥有密码数据库的访问权限，Adv_{rw}拥有读写权限。这两个对手的目标是获取尽可能多的信息，并且对于Adv_{rw}而言，还要制造出一个数据库。第一，它不是由用户所创建；第二，数据库一旦被打开，不会触发任何来自密码管理器的警告和错误信息。很显然，Adv_{rw}显然比Adv_r更为强大：任何可以由Adv_r发起的攻击都可以由Adv_{rw}产生。这两个对手都被允许收集多种数据库在不同时间点的快照，用来探测数据库内容的修改。

我们强调了我们的分析不依赖任何对于用户环境的修改，例如篡改密码管理器的源代码或是安装密码记录器。我们仅仅关注由密码管理器的数据格式带来的安全性，并让密码管理器运行在最高安全设置。我们假设用户选择了高强度、高熵的主密码并且所有底层加密算法都被正确执行。此外，我们假设没有额外的机制可以阻止文件篡改。这个允许我们可以比较数据库格式本身带来的安全性。

3.1 不可信存储

考虑哪些拥有加密数据库完全访问权限的对手，并且记录它的不同版本。这样的对手明显可以使用任何版本的记录来替换现在的数据库，只要主密码没有被更换。这大体上是一次对积基于云和本地的数据库格式攻击的重演。

我们如下定义的安全概念不会去记录这次攻击，我们也不会通过其他方式尝试去对待它。为了能够保护它免受攻击，一个密码管理器必须在可信的介质上保存一些本地状态（例如对于最近版本的加密数据库散列）。因此，当这次攻击的对象与存储于云上或是无人看管的 USB 设备中的密码数据库相关时，它并没有因为数据库格式而减少损失。我们将这种情况剔除考虑。

3.2 安全定义

我们定义四种算法，代表不同的密码管理器功能：**Setup**、**Create**、**Open** 和 **Valid**。这些算法的定义如下所示：

定义 1. *密码管理器 PM 包含以下高效的算法：***Setup**$(\cdot)$*是一种随机算法，给出安全参数* 1^k *，输出主密码 mp；***Create**$(\cdot,\cdot)$*是一种随机算法，输入 mp 和三元数组* $RS=\{(r_1,n_1,v_1),\cdots,(r_l,n_l,v_l)\}$ *(代表记录集)，输出数据库 DB；***Open**$(\cdot,\cdot)$*是一种确定性的算法，给出 mp 和数据库 DB，输出在 DB 中编码的记录集 RS 如果 RS 是一个存在的记录集，也就是说，存在一个数据集* DB'*，使得*$DB' \leftarrow Create(mp,RS)$*，并且互相正交；***Valid**$(\cdot,\cdot)$*是一种确定性的算法，输入一个主密码 mp 和一个数据库 DB，如果*$\mathrm{Open}(\mathrm{mp},\mathrm{DB}) \neq \perp$ *则返回 1.*

实际上，**Valid** 在密码管理器中，是在 **Open** 功能中实现：如果验证有效失败，则密码管理器会返回一个错误而不是数据库内容。

我们同时定义两种新的游戏，我们分别称呼为不能辨别的数据库游戏（IND-CDBA）和延展性的选择数据库游戏（MAL-CDBA）。前者获取现实中被动对手的能力，也就是说，这个对手拥有密码数据库的只读权限。后一种模型和主动对手，他拥有密码数据库的读写权限。

游戏 1（不能区别的数据库游戏 $\mathbf{IND-CDBA_{Adv_r,PM(\kappa)}}$.） *挑战者 Ch 通过运行 PM 与 Adv*$_r$ *交互，如下所示：*

- *Ch 运行* $mp \leftarrow Setup(1^k)$
- *Adv*$_r$ *输出两个记录集 RS*$_0$，*RS*$_1$.
- *Ch 均匀地随机选则 a 位 b 并且数据库* $DB_b \leftarrow Create(mp, RS_b)$ *返回到 Adv*$_r$.
- *Adv*$_r$ *最终输出一位 b'，当且仅当* $b = b'$ *时，这个游戏最终输出 1.*

我们说 Adv$_r$ 获得了 IND-CDBA 游戏的胜利，如果挑战者能让游戏输出 1.

定义 2（IND-CDBA 安全性）. *密码管理器* $PM = (Setup, Create, Valid, Open)$ *是 IND-CDBA 安全的，如果存在一个可忽略不计的函数 negl，对于任意可能的多项式级别时间复杂性的对手* $\mathrm{Adv_r}$ *而言，我们有* $\Pr\left[\mathrm{IND-CDBA_{Adv_r,PM(\kappa)}} = 1\right] \le 1/2 + \mathrm{negl}(\kappa)$.

对于大多数数据库格式而言，一个攻击者可以轻松地获得 IND-CDBA 游戏胜利通过提交两种大小不同的记录集。实际上，这意味着着下面的事实：数据库文件的大小通常大致正比于数据库中记录的数量，因此攻击者可以简单地从观察加密数据库大小来获取信息。虽然我们认为这是一次有效的进攻，但会在脆弱性分析之时忽略它。如果数据库格式仅仅是这种攻击的受害者的话，我们认为它是安全的。

附录 A 展现了 IND-CPA 和 IND-CDBA 之间的关系。特别地，这表明了 IND-CPA 安全性就意味着 IND-CDBA 安全性。

游戏 2 （$\mathbf{MAL-CDBA_{Adv_{rw},PM(\kappa)}}$.） *挑战者 Ch 通过运行 PM 与* Adv_{rw} *交互，如下所示:*

- *Ch 运行* $mp \leftarrow Setup(1^k)$
- Adv_{rw} *适应性地输出了 n 组记录集* RS_i *并且从 Ch 接收，对应的数据库* $\mathrm{DB_i} \leftarrow \mathrm{Create(mp, RS_i)}$
- Adv_{rw} *最终输出数据库* DB' *；游戏输出 1 当且仅当* $\mathrm{Valid}(DB') = 1$ *并且对于所有的* $i \le n$ *而言，* $DB' \neq DB_i$。

我们说 $\mathrm{Adv_{rw}}$ 获得了 MAL-CDBA 游戏的胜利，如果挑战者能让游戏输出 1.

定义 3 （MAL-CDBA 安全性） *密码管理器* $\mathrm{PM = (Setup, Create, Valid, Open)}$ *是 MAL-CDBA 安全的，如果如果存在一个可忽略不计的函数 negl，对于任意可能的多项式级别时间复杂性的对手* $\mathrm{Adv_{rw}}$ *而言，我们有* $\Pr\left[\mathrm{MAL-CDBA_{Adv_{rw},PM(\kappa)}} = 1\right] \le \mathrm{negl}(\kappa)$.

我们对于 MAL-CDBA 安全性的定义相当于概念“存在的密文不可伪造”。这个安全性概念表明了，IND-CPA 安全就意味着 IND-CCA 安全。

“密文的完整性”（也被称为 INT-CTXT 安全性）是一个相关的安全概念。特别地，MAL-CDBA 和 INT-CTXT 的主要区别是，一个 INT-CTXT 的对手也会可以 **Verify** 权限。

我们认为，MAL-CDBA 安全性（也包括 IND-CDBA 安全性）是一种合适的安全性概念，对于实际中的密码管理器数据库格式而言。考虑一种非 MAL-CDBA 安全的数据库格式，也就是说，Adv_{rw} 可以计算出他选择的数据集的编码，并且可以生成对应的有效输出 DB’。这种格式在如下的 4 步攻击面前，是脆弱的：

(1) Adv_{rw}用一个由 Adv_{rw}创建的新数据DB′，替换了 Alice 的密码数据库 DB。DB’中包含了 amazon.com 账号的登录证书。

(2) Adv_{rw}现在引诱 Alice 去访问 amazon.con，在此时密码管理器自动登录了由 Adv_{rw}创建的账号信息。

(3) Alice 购买了商品，在收银台时，Alice 被要求往她的账号中添加信用卡。因为 Alice 信任 amazon.com，她照做了。

(4) 现在， Adv_{rw}用DB′替换了 Alice 本来的密码数据库。

Adv_{rw}现在拥有了一个可以使用 Alice 信用购物的账号。对于 Alice 而言，很难察觉到此次攻击，她并没有从她的密码管理器或是 amazon.com 上收到任何警告信息，因为数据库形状完好并且登录信息对应已有账号。另外，当 Alice 和 amazon.com 通信之时，SSL/TLS 并不能起到任何作用。而且，在攻击者恢复了 Alice 的原始数据后，Alice 甚至不能找出恶意账号的用户名。

4. 数据库格式漏洞

现在展示我们的分析，其中包括许多正在被独立的和基于浏览器的密码管理器使用中的数据库格式。对于每一种格式而言，我们提供相关特征的简要说明并且分析它的安全性，基于在第 3 节中定义的安全模型。如果数据库格式允许不同的安全级别，我们分析最安全的配置。

4.1 Google Chrome

格式描述. Google Chrome 在用户配置文件目录下的 SQLite 数据库文件中存储用户名和密码。这个数据库既没有提供保密性，也不提供完整性。

Google Chrome 可以在 Google 的服务器上选择性地存储所有浏览器偏好(包括密码)，以允许不同设备间的同步。Chrome 的支持页面声称，密码以加密的形式存储在 Google 的服务器上。

安全性分析. 任何能够访问数据库文件的用户可以恢复其所有的内容，并做出任意的修改。因此，用户不可以信赖 Chrome 密码管理器的对于数据的安全性和完整性，并且应该实施它之外的安全措施。

4.2 Mozilla Firefox

格式描述. Mozilla Firefox 将登陆数据存储在 SQLite 数据库中。用户可以(选择性的)设定一个主密码用于加密数据库的内容。URL 永远以未被加密的形式存储，无论有没有主密码。

因为数据库是 Firefox 用户资料的一部分，它可以在不同设备间自动同步，通过 Firefox Sync、手动同步或是存储在 USB 设备上，这样就可以用于不同的计算机。

安全性分析. Firefox 不提供对于 Adv_r 的任何有效保护。为了赢得 IND-CDBA 游戏的胜利，Adv_r 创建两个大小相同的记录集 RS_0 和 RS_1，这两者在 URL 字段上至少有一处不同。加密的数据库DB_b可以被立刻识别因为 URL 们并没有被隐藏。在实践中，这意味着一个攻击者可以获得相当多的信息，例如用户有密码访问保护的网站，并且它可以影响用户安装基于用户信息的网络攻击。此外，给定两个同一数据库的不同版本，攻击者可以识别哪些记录已被修改，和这些记录对应的域名。

类似地，对于任意的非空数据库DB，一个主动的对手 Adv_rw 可以轻松赢得 MAL-CDBA 游戏，通过从DB中用不同的 URL 替换掉 1 个或多个 URL 来构造DB'。由于这些记录不是被整体保护的，Firefox 不能探测到这样的攻击。这可以被 Adv_rw 用来进行一次很有效的中间人攻击，通过用虚假的域名替换掉合法域名的方式。这样一来，密码管理器会向对手控制的网站自动提交敏感信息。这次攻击会变得更有效，如果 Adv_rw 可以修改 Firefox 的收藏夹数据库，该数据库就被存放在密码数据库旁边。

4.3 Microsoft Internet Explorer

格式描述. Internet Explorer 将用户名和密码存储在注册表中。每条记录存储为独立的注册表项，并且使用该系统的登录证书进行加密。当用户在地址 url 处填入密码表，Internet Explorer 会计算$h = \text{SHA-1}(url)$，并且加密用户名和密码$c = E_k(\text{metadata} \,||\, \text{username} \,||\, \text{0x00} \,||\, \text{password} \,||\, \text{0x00})$，其中 metadata 包含附加信息例如加密元素的大小。

加密的执行通过 CryptProtectData 系统调用，它使用在 CBC 模式下的 Triple-DEC 和基于散列的 MAC。k 从(1)一个随机盐(存储在密文中)，(2) url 和(3)当前用户的 Windows 登录证书中衍生出来。最后，Internet Explorer 产生一个新的注册表项，键为 h、值为 c。

Internet Explorer 密码管理器的安全性取决于用户帐户密码的强度。因此，没有密码的账号不能对密码数据库提供任何保护。

安全性分析. 对于 Adv_r 而言，Internet Explorer 并不是安全的。类似地，为了赢得 IND-CDBA 游戏的胜利，Adv_r 创建两个大小相同的记录集 RS_0 和 RS_1 ，这两者在 URL 字段上至少有一处不同。

附带网址 url 的描述记录 rec 在 RS_0 中而不是在 RS_1 中。Adv_r 可以迅速识别出哪些记录集对应挑战 DB_b，通过计算$h = \text{SHA-1}(url)$并判断 h 是否在DB_b中。

实际上，一个被动的攻击者可以使用 Internet Explorer 的密码数据库，去判断用户是否访问了某个特定的网页、输入了他的用户名/密码，甚至用户是否删除了他的访问记录和缓存。

假设 CryptProtectData 使用了安全的 MAC，一个主动的攻击者就不能变更密码条目。然而，Adv_{rw} 通过删除对应的注册表条目，可以删除密码记录，这样 Adv_{rw} 就可以轻松地赢下 MAL-CDBA 游戏。

4.4 1Password

格式描述. 1Password 使用多个文件存储它的数据库。每个文件中包含数据库条目，以 JSON 格式存储。数据库条目在一个名为“content.js”的文件中列出。

1Password 允许用户对于每条记录选择不同的“安全级别”。1Password 允许用户对于每条记录选择不同的“安全级别”。最低的安全级别对应未加密的条目，而高级别意味着一些敏感的项，例如用户名和密码，会被用户主密码派生的密钥加密。无论安全级别是什么，一部分项例如条目的标题，永远不会被加密。我们分析最高安全级别提供的安全性。加密方案使用 AES-128 的 CBC 模式。

无论是记录还是索引文件，都没有对于完整性的保护。因此仅当 JSON 解释器不能处理数据库时，数据库的错误才能被检测到。

安全性分析. 1Password 的数据库格式十分容易受一些弱点的影响，这些弱点给了对手 Adv_r 和 Adv_{rw} 一些不可忽视的优势，在赢得 IND-CDBA 游戏和 MAL-CDBA 游戏上。

Adv_r 可以用如下的方式，百分百的概率获得 IND-CDBA 游戏的胜利：Adv_r 创建两个相同大小的数据集 RS_0 和 RS_1，并且有两条记录 r_0 和 r_1 分别来自数据集 RS_0 和 RS_1，它们至少在以下字段中有一项不同：title, location, locationKey, createdAt, updatedAt 或是 typeName。这些字段分别对应：记录的标题，记录的 URL，浏览器插件自动填充完成的 URL，创建的时间，最后修改的时间和记录的类型(例如：网页表单、私人笔记、信用卡信息)。因为这些记录并没有被加密，Adv_r 可以简单地查明比特 b，通过测试哪条记录属于 DB_b。在实际中，这意味着一个可以访问 1Password 数据库的对手就可以阅读这些字段，并且从用户的浏览习惯中推测敏感信息。

Adv_{rw} 可以用如下的方式，百分百的概率获得 MAL-CDBA 游戏的胜利：Adv_{rw} 线则一个任意的记录集RS并且收到对应的数据DB。然后，Adv_{rw} 可以(1)改变上面列出的任何字段，和 / 或(2)移除任何条目，通过删除其对应的数据库文件并对应地修改“content.js”索引文件。在一般情况下，只要数据库仍然由一组正确的 JSON 字符串组成，1Password 不会显示任何警告。实际上，这意味着任何攻击者可以实施钓鱼攻击，通过将一个合法的 URL 替换为对手控制的网站。

此外，如果 Adv_{rw} 输出至少两个数据集，说$RS \neq RS'$并且接受到 MAL-CDBA 游戏中两个对应的数据看DB和DB'，它能从DB和DB'的记录中构造出DB''。除此之外，这个可以让一个攻击者，用旧版本的记录替换数据库中的个人记录。

7. 结论

密码管理器是一种用于安全地存储敏感信息的重要软件。本文做出了第一个严谨的分析，对于流行的密码管理器所使用的存储格式。

我们定义了两种现实中的安全模型，旨在表现现实世界中攻击的能力。一种用于被动攻击者，一种用于主动攻击者。我们分析了在我们的安全模型中，流行的密码管理器所使用的数据库格式，对于每个易受攻击的格式，我们提供了有条理的论据说明它们为什么容易被破坏。我们还展示了理论上的何种弱点，会对应何种现实中的攻击。此外，当数据库格式被证明是安全的时候，我们提供了有条理的证明。

不幸的是，大多数的格式会被一些弱小的对手轻易攻破。出于这个原因，用户应该谨慎考虑，某种特定的数据库格式在云端、USB 设备、在与他人共用的计算机上存储数据是否合适。

最后，我们的工作表明，构造一种提供安全性、易用性、低计算和存储开销、使用标准加密工具的格式是完全可能的。

On the Security of Password Manager Database Formats

Paolo Gasti and Kasper B. Rasmussen

Computer Science Department
University of California, Irvine
{pgasti,kbrasmus}@ics.uci.edu

Abstract. Password managers are critical pieces of software relied upon by users to securely store valuable and sensitive information, from online banking passwords and login credentials to passport- and social security numbers. Surprisingly, there has been very little academic research on the security these applications provide.

This paper presents the first rigorous analysis of storage formats used by popular password managers. We define two realistic security models, designed to represent the capabilities of real-world adversaries. We then show how specific vulnerabilities in our models allow an adversary to implement practical attacks. Our analysis shows that most password manager database formats are broken even against weak adversaries.

1 Introduction

As the number of services offered on the Internet continues to increase, the number of passwords an average user is required to remember increases correspondingly, to the point where it is no longer feasible for most people to remember a new, strong password, for every account.

Users typically solve this problem in one of two ways. A common solution is to reuse the same password on many different websites [1]. This approach increases the potential damage if a password is stolen, cracked, or if a service that has access to it is compromised, since the attacker will be able to reuse it on all online services that share the password. Another approach is to use a "password manager" to store strong (random) passwords for each site. A password manager is a piece of software that requires a user to remember a single strong master password, used to decrypt the password manager's database. Remembering a single master password is much more feasible for users, who still get the security benefits of using a different password for each online service.

Using a password manager has other potential benefits. Full URLs (or at least domain names) of are typically stored alongside the corresponding passwords, and used to fill login form automatically. As such, users who rely on password managers are less susceptible to typo-squatting and phishing attacks [2,3]: even if a user is directed to a malicious website that is designed to look identical to the website the user expects, the password manager will not log in automatically, providing an extra layer of protection.

Due to the sensitivity of the information typically stored in password databases, most password managers protect their content from unauthorized access. Database formats typically rely on encryption for data protection, where the encryption/decryption key is generated from a master password entered by the user.

This protection is also often designed to allow users to store the password manager database on untrusted storage. Several producers of password managers suggest storing password databases on USB sticks [4–6], in the cloud [7,8] or on mobile devices [9–11], to allow convenient access to stored passwords. These storage options however, can also enable potential attackers to get hold of the database. Even when a password database is stored on a local hard drive, it may be possible for an attacker to obtain a copy through other means.

If the password manager database format is insecure, then all the advantages of a good password manager are wasted and the user may actually be less secure and more susceptible to, e.g., leakage of private information: privacy-conscious users may want to keep their browsing habits private and therefore delete cookies, history and cache often. On the contrary, password managers represent long-term storage facilities, storing (ideally) the only copy of passwords, and therefore their content is typically never deleted. If a password manager database leaks information about browsing habits, e.g., by storing URL's unencrypted, then clearing the cache and browsing history does not prevent an attacker from learning sensitive information.

In this paper we analyze the security provided by the database formats of some of the most poplar password managers in use at the moment. We define two different adversaries: a passive attacker that only tries to infer information from a password database, and an active attacker that modifies the content or meta-data. We highlight that using "industry standard practices", such as AES-CBC, is not enough to obtain a secure database format, even assuming the implementation of AES-CBC is correct. Note that we do not attempt to provide an exhaustive list of all possible attacks on all password managers. Rather, we model the security provided by common password manager database formats and provide examples of practical attacks.

The rest of this paper is organized as follows: Section 2 provides a brief overview of password managers used in our study; Section 3 introduces our system- and attacker models, while Section 4 analyzes the various database formats in such models. In Section 5 we discuss various general issues regarding database formats, and Section 6 covers related work. We conclude in Section 7.

2 Overview of Password Managers

Password managers differ in many aspects, including database format, functionality, availability of source code, supported platforms and access to cloud storage. Table 1 summarizes the main features of the password managers we considered. Some popular password managers invent their own database format, used exclusively by them. This is especially true for the password managers embedded

Table 1. This table shows the password managers that where analysed in detail, along with the database format used by the software, the storage options available and the platforms supported. In addition we indicate whether the source code is available and whether the password manager is integrated with a browser.

Password Manager	Database Format	Storage	Open Source	Platform	Browser Integration
Google Chrome [12]	Chrome	local/cloud	✓	Win/Mac/Linux	✓
Mozilla Firefox [13]	Firefox	local/cloud	✓	Win/Mac/Linux	✓
Internet Explorer [14]	MSIE	local	×	Win	✓
1Password [9]	1Password	local/cloud	×	Win/Mac	✓
KeePass 1.x [15]	KDB	local	✓	Win	×
KeePass 2.x [15]	KDB/KDBX4	local	✓	Win/Mono	×
KeePassDroid [11]	KDB/KDBX4	local	✓	Android	×
KyPass [10]	KDB/KDBX4	local	✓	iOS	×
PassDrop [16]	KDB/KDBX4	local	×	iOS	×
PINs [17]	PINs	local	×	Win	×
Password Safe [18]	PasswordSafe	local	✓	Win	×
Password Gorilla [19]	PasswordSafe	local	✓	Win/Mac/Linux	×
Roboform [20]	Roboform	local/cloud	×	Win/Mac/Linux	✓

in major browsers. We include these in our analysis because these password managers are widely used [21]. Several stand-alone password managers share the same database format, so even though each password manager provide a different experience to the user, the underlying storage format is the same.

In the rest of this paper focus solely on database formats and the security they provide, rather than on each password manager implementation. We assume that the password managers themselves correctly implement what the format specifies. As such, we do not consider, e.g., side channel attacks on the cryptographic primitives, or other attacks against the implementation. Rather we investigate the best possible security achievable given a specific storage format. For this reason our analysis focuses primarily on password managers that provide local storage, at least as an option. We leave the analysis of "cloud-only" password managers to future work.

We investigate nine popular password database formats. Three database formats used by in-browser password managers: Google Chrome, Mozilla Firefox and Microsoft Internet Explorer; and six formats used by a large number of stand-alone password managers: 1Password, KDB, KDBX4, PasswordSafe v3, PINs and RoboForm (refer to Table 1.)

3 Adversary and System Model

We consider two efficient adversaries: $\mathsf{Adv_r}$ who has read access to the password database, and $\mathsf{Adv_{rw}}$ who has read-write access. The goal of both adversaries is to extract as much information as possible and, for $\mathsf{Adv}_{\mathsf{rw}}$, to produce a database that (1) was not created by the user and (2) once opened, will not trigger any warning or error message from the password manager. Clearly, $\mathsf{Adv}_{\mathsf{rw}}$ is strictly stronger than $\mathsf{Adv}_{\mathsf{r}}$: any attack that can be performed by $\mathsf{Adv}_{\mathsf{r}}$ is also available to $\mathsf{Adv}_{\mathsf{rw}}$. Both adversaries are allowed to gather multiple snapshots of the database at different points in time, in order to detect modifications in the database content.

We emphasize that our analysis does not rely on any modification of the user environment, e.g., tampering with the password manager code or installing a key logger. We focus solely on the security provided by the password manager databases, when the password manager software is operated in the "most secure" setting provided. We assume that users choose a strong, high-entropy, master password and that all underlying cryptographic algorithms (e.g., encryption, MAC, etc.) are properly implemented. Additionally, we assume that no additional mechanisms are in place to prevent file tampering. This allows us to compare the security offered by the database formats themselves.

3.1 Untrusted Storage

Consider an adversary who has full access to an encrypted password database, and is able to record different versions of it. Such an adversary can clearly use any of the recorded versions to replace the current database, as long as the master password did not change. This is essentially a replay attack that applies to both cloud-based- and local database formats.

The security notions we define below do not capture this attack, nor do we attempt to address it in any other way. In order to protect against it, a password manager must maintain some local state (e.g., a hash of the latest version of the encrypted database) on a trusted medium. As such, while this attack is clearly relevant when a password database is stored on the cloud or on an unattended USB drive, it cannot be mitigated by the database format alone. Therefore we exclude it from our analysis.

3.2 Security Definitions

We model password managers by defining four algorithms that represent various functionalities: Setup, Create, Open and Valid. These algorithms are defined as follows:

Definition 1. *A password manager $\mathcal{PM}$ consists of the following efficient algorithms:* $\mathsf{Setup}(\cdot)$ *a probabilistic algorithm that, given a security parameter 1^{κ}, outputs a master password mp;* $\mathsf{Create}(\cdot,\cdot)$ *a probabilistic algorithm that, on input mp and a set of triples $RS = \{(r_1, n_1, v_1), \ldots, (r_\ell, n_\ell, v_\ell)\}$ (which represents a record-set), outputs a database DB;* $\mathsf{Open}(\cdot,\cdot)$ *a deterministic algorithm that, given mp and a database DB, outputs the record-set RS encoded in DB if RS is a valid record-set, i.e, there exist DB' such that $DB' \leftarrow \mathsf{Create}(mp, RS)$, and $\perp$ otherwise; and* $\mathsf{Valid}(\cdot,\cdot)$ *a deterministic algorithm that takes as input a master password mp and a database DB and returns 1 if* $\mathsf{Open}(mp, DB) \neq \perp$.

In practice, Valid is implemented by password managers within the Open functionality: if validation fails, the password manager returns an error rather than the database content.

We also define two new games, which we call *indistinguishability of databases* game ($\mathsf{IND\text{-}CDBA}$) and *malleability of chosen database* game ($\mathsf{MAL\text{-}CDBA}$). The former captures the capabilities of a realistic passive adversary, i.e., an adversary that has read-only access to a password database. The latter models and active adversary, which is allowed both read and write access to a password database.

Game 1 (Indistinguishability of databases game $\mathsf{IND\text{-}CDBA}_{\mathsf{Adv_r},\mathcal{PM}}(\kappa)$). *A challenger* Ch *running* $\mathcal{PM}$ *interacts with* $\mathsf{Adv_r}$ *in as follows:*

- Ch *runs* $mp \leftarrow \mathsf{Setup}(1^\kappa)$.
- $\mathsf{Adv_r}$ *outputs two record-sets* RS_0, RS_1
- Ch *selects a bit* b *uniformly at random and the database* $DB_b \leftarrow \mathsf{Create}(mp, RS_b)$ *is returned to* $\mathsf{Adv_r}$.
- $\mathsf{Adv_r}$ *eventually outputs bit* b'; *the game outputs 1 iff* $b = b'$.

We say that $\mathsf{Adv_r}$ *wins* the $\mathsf{IND\text{-}CDBA}$ game if it can cause it to output 1.

Definition 2 ($\mathsf{IND\text{-}CDBA}$ security). *A password manager* $\mathcal{PM} = (\mathsf{Setup}, \mathsf{Create}, \mathsf{Valid}, \mathsf{Open})$ *is* $\mathsf{IND\text{-}CDBA}$*-secure if there exists a negligible function* negl *such that, for any probabilistic polynomial time adversary* $\mathsf{Adv_r}$, *we have that* $\Pr[\mathsf{IND\text{-}CDBA}_{\mathsf{Adv_r},\mathcal{PM}}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa)$.

For most database formats an attacker can trivially win the $\mathsf{IND\text{-}CDBA}$ game by submitting two record-sets of different sizes. In practice, this corresponds to the fact that the size of the database file is often roughly proportional to the number of records in the database and therefore an adversary may be able to infer information by simply observing the size of an encrypted database. While we do consider this a valid attack, we ignore it in the vulnerability analysis. Database formats that are *only* vulnerable to this attack will be considered secure.

Appendix A shows the relationship between $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CDBA}$. In particular, it shows that $\mathsf{IND\text{-}CPA}$-security implies $\mathsf{IND\text{-}CDBA}$-security.

Game 2 (Malleability of chosen database game $\mathsf{MAL\text{-}CDBA}_{\mathsf{Adv_{rw}},\mathcal{PM}}(\kappa)$). *A challenger* Ch *running* $\mathcal{PM}$ *interacts with* $\mathsf{Adv_{rw}}$ *in the following way:*

- Ch *runs* $mp \leftarrow \mathsf{Setup}(1^\kappa)$.
- $\mathsf{Adv_{rw}}$ *adaptively outputs* n *record-sets* RS_i *and receives, from* Ch, *the corresponding databases* $DB_i \leftarrow \mathsf{Create}(mp, RS_i)$.
- $\mathsf{Adv_{rw}}$ *eventually outputs* DB'; *the game outputs 1 iff* $\mathsf{Valid}(DB') = 1$ *and* $DB' \neq DB_i$ *for* $i \leq n$.

We say that $\mathsf{Adv_{rw}}$ *wins* the $\mathsf{MAL\text{-}CDBA}$ game if it can cause it to output 1.

Definition 3 ($\mathsf{MAL\text{-}CDBA}$ security). *A password manager* $\mathcal{PM} = (\mathsf{Setup}, \mathsf{Create}, \mathsf{Valid}, \mathsf{Open})$ *is* $\mathsf{MAL\text{-}CDBA}$*-secure if there exists a negligible function* negl *such that, for any probabilistic polynomial time adversary* $\mathsf{Adv_{rw}}$, *we have that* $\Pr[\mathsf{MAL\text{-}CDBA}_{\mathsf{Adv_{rw}},\mathcal{PM}}(\kappa) = 1] \leq \mathsf{negl}(\kappa)$.

Our definition of $\mathsf{MAL\text{-}CDBA}$ security is equivalent to the notion of "existential unforgeability" of ciphertexts, introduced in [22]. As shown in the same paper, this security notion along with $\mathsf{IND\text{-}CPA}$ security implies $\mathsf{IND\text{-}CCA}$ security.

"Integrity of ciphertexts" [23] (also known as $\mathsf{INT\text{-}CTXT}$ security) is a related security notion. In particular, the main difference between $\mathsf{MAL\text{-}CDBA}$ and $\mathsf{INT\text{-}CTXT}$ is that an adversary for $\mathsf{INT\text{-}CTXT}$ is also given access to the $\mathsf{Verify}(mp, \cdot)$ oracle.

We argue that $\mathsf{MAL\text{-}CDBA}$ security (together with $\mathsf{IND\text{-}CDBA}$ security) is an appropriate security notion for a password manager database format in practice. Consider a database format that is not $\mathsf{MAL\text{-}CDBA}$-secure, i.e., where $\mathsf{Adv}_{\mathsf{rw}}$ can compute the encryption of a record-set of its choice, and produce the corresponding valid output DB'. This format would be vulnerable to the following four-step attack:

> (1) $\mathsf{Adv}_{\mathsf{rw}}$ replaces Alice's password database DB with a new database DB' containing the login credentials for an `amazon.com` account created by $\mathsf{Adv}_{\mathsf{rw}}$. (2) $\mathsf{Adv}_{\mathsf{rw}}$ now induces Alice to go to `amazon.com`, at which point the password manager automatically logs into the account created by $\mathsf{Adv}_{\mathsf{rw}}$. (3) Alice buys something; during checkout, Alice is requested to add her credit card to the account; since she trusts `amazon.com`, she complies. (4) $\mathsf{Adv}_{\mathsf{rw}}$ now replaces DB' with Alice's original password database.

$\mathsf{Adv}_{\mathsf{rw}}$ is now in possession of an account which can be used to purchase goods on Alice's behalf. It is very hard for Alice to detect this attack; she does not receive any warning from her password manager or from `amazon.com`, since the database is well formed and the login information corresponds to an existing account. Additionally, SSL/TLS does not help since Alice is communicating with `amazon.com`. Furthermore, Alice may not even be able to find out which username was used in the maliciously crafted account after the adversary restores her original password database.

4 Database Format Vulnerabilities

We now present our analysis, which includes several database formats currently in use by stand-alone and browser-based password managers. For each format, we provide a short description of the relevant features and analyze its security with respect to the security model defined in Section 3. If the format allows for different levels of security, we analyze the most secure configuration.

4.1 Google Chrome

Format Description. Google Chrome stores usernames and passwords in an SQLite database file in the user profile directory. This database provides neither secrecy nor integrity.

Google Chrome can optionally store all browser preferences (including passwords) on Google's servers to allow synchronization between different devices.

Chrome's support pages claim that passwords are stored in encrypted form on Google's servers [24].

Security Analysis. Any user with access to the database file can recover all its content and make arbitrary modifications. As such, users cannot rely on Chrome's password manager for integrity or secrecy of their data, and should implement additional security layers around it.

4.2 Mozilla Firefox

Format Description. Mozilla Firefox stores login data in an SQLite database. Users can specify an (optional) master password that is used to encrypt the database content. URLs are always stored unencrypted regardless of whether a master password is used or not.

Since the database is part of Firefox' user profile, it can be automatically synchronized across multiple devices, either through Firefox Sync [25], manually (e.g., using rsync [26]), or stored on a USB stick and used on different computers.

Security Analysis. Firefox does not provide an effective protection against $\mathsf{Adv_r}$. In order to win in the $\mathsf{IND\text{-}CDBA}$ game, $\mathsf{Adv_r}$ creates two same-size record-sets RS_0, RS_1 which differ in at least one URL field. The encrypted database DB_b can be immediately identified since URLs are not concealed. In practice that means that an attacker can learn a considerable amount of information, such as the websites in which the user has password-protected access, and it can mount effecting phishing attacks based on user information. Moreover, given two different versions of the same database, the attacker can identify which entries have been modified and their corresponding domain name.

Similarly, given any non-empty database DB an active adversary $\mathsf{Adv_{rw}}$ can trivially win the $\mathsf{MAL\text{-}CDBA}$ game by building DB' from DB replacing one or more URLs with a different valid URL. Since the entries are not integrity protected, Firefox cannot detect such an attack. This can be used to mount a very effective man-in-the-middle attack by replacing legitimate domain names with fraudulent ones controlled by $\mathsf{Adv_{rw}}$. In this way, the password manager will automatically submit sensitive information to an adversary-controlled website. The attack is even more effective if $\mathsf{Adv_{rw}}$ can also modify Firefox' bookmark database, which is stored in the profile alongside the password database.

4.3 Microsoft Internet Explorer

Format Description. Internet Explorer stores usernames and passwords in the registry. Each record is stored as a separate registry entry and encrypted using the system login credentials. When a user fills-in a password form at address url, Internet Explorer computes $h = \text{SHA-1}(url)$ (where and url uses the unicode character set) and encrypts username and password as $c = E_k(\texttt{metadata} \parallel \texttt{username} \parallel \texttt{0x00} \parallel \texttt{password} \parallel \texttt{0x00})$, where `metadata` contains additional information such as the size of encrypted elements.

The encryption is performed using the `CryptProtectData` [27] system call, which uses Triple-DES in CBC mode and a hash-based MAC. k is derived from (1) a random salt (also stored in the ciphertext), (2) url and (3) the Windows login credential for the current user. Finally, Internet Explorer creates a new registry entry with key h and value c.

The security of Internet Explorer's password manager depends on the strength of the user account password. As such, accounts with no password provide no protection of the password database.

Security Analysis. Internet explorer is not secure against $\mathsf{Adv_r}$. Similarly to Firefox, $\mathsf{Adv_r}$ wins the $\mathsf{IND\text{-}CDBA}$ game by building two same-size record-sets RS_0, RS_1 which differ in at least one URL.

Say record rec with URL url is in RS_0 but not in RS_1. $\mathsf{Adv_r}$ can immediately recognize which record-set corresponds to the challenge DB_b by computing $h =$ SHA-1(url) and verifying whether h is in DB_b.

In practice, a passive adversary can use Internet Explorer's password database to determine whether a user has visited a particular web page and entered his username/password, even if the user deletes his browsing history and cache.

Assuming that `CryptProtectData` uses a secure MAC, an active adversary cannot alter password entries. However, $\mathsf{Adv_{rw}}$ can delete password entries by removing the corresponding registry entry, and as such $\mathsf{Adv_{rw}}$ can easily win the $\mathsf{MAL\text{-}CDBA}$ game.

4.4 1Password

Format Description. 1Password stores its database in multiple files. Each file contains a database entry, stored in JavaScript Object Notation (JSON). Entries are listed in an index file called "content.js".

1Password allows users to select a different "security level" for each record [28]. The lowest security level corresponds to unencrypted entries, while the highest level means that sensitive fields, such as username and password, are encrypted with a key derived from the user's master password. Regardless of the security level, some fields, e.g., the title of an entry, are never encrypted. We analyze the protection offered by the highest security level.

The encryption scheme used is AES-128 in CBC mode. Neither the records nor the index file are integrity protected. As a result, database corruption is only detected when the JSON parser fails to process the database.

Security Analysis. 1Password's database format is affected by vulnerabilities that give adversaries a non-negligible advantage in both the $\mathsf{IND\text{-}CDBA}$ and $\mathsf{MAL\text{-}CDBA}$ games.

$\mathsf{Adv_r}$ can win $\mathsf{IND\text{-}CDBA}$ with probability 1 as follows: $\mathsf{Adv_r}$ builds two same-size record-sets RS_0, RS_1 such that there exist two records r_0, r_1 from RS_0 and RS_1 respectively, which differ in at least one of the following fields: `title`, `location`, `locationKey`, `createdAt`, `updatedAt` or `typeName`. These fields correspond to: the title of the record, the record URL, the URL used by the browser plugin to perform auto-complete, the time of creation and last update and the type of record (e.g., web form, protected note, credit card information). Since these fields are never encrypted, $\mathsf{Adv_r}$ can trivially determine bit b by testing which record belongs to DB_b. In practice this means that an adversary with access to a 1Password database can read these fields and thus gather sensitive information about the user's browsing habits.

$\mathsf{Adv_{rw}}$ can win the $\mathsf{MAL\text{-}CDBA}$ game with probability 1 as follows. $\mathsf{Adv_{rw}}$ selects an arbitrary record-set RS and receives the corresponding database DB. Then, $\mathsf{Adv_{rw}}$ can (1) alter any of the fields listed above, and/or (2) remove any entry by deleting the corresponding database file and altering the "content.js" index file correspondingly. In general, as long as the database is still composed of a set of correct JSON strings, 1Password will not show any warning. In practice, this means that an adversary can mount phishing attacks by replacing a legitimate URL with one pointing to an adversary-controlled website.

Additionally, if $\mathsf{Adv_{rw}}$ outputs at least two record-sets, say $RS \neq RS'$ and receives the corresponding databases DB, DB' in the $\mathsf{MAL\text{-}CDBA}$ game, it can construct DB'' selecting records from both DB and DB'. This allows an adversary, among other things, to replace individual records in a database with older versions.

4.5 KDB (aka KeePass 1.x)

Format Description. The KDB database is composed of a single file, divided in two sections: an unencrypted header (*hdr*) and an encrypted body (*bdy*). *bdy* stores the encryption of the various database entries. *hdr* contains, among other things, the number of groups and entries in the database and the hash of *bdy* before encryption [15]. This hash is computed every time the database is modified, and is used to check integrity. After decryption, the password manager verifies that the computed plaintext hashes to the same value stored in *hdr*. If this check fails, the application reports that either the database is corrupted or the master password entered by the user is incorrect.

Security Analysis. Given a database DB, the hash stored (unencrypted) in *hdr* is computed deterministically from the record-set RS encoded in DB. This allows an adversary $\mathsf{Adv_r}$ to win the $\mathsf{IND\text{-}CDBA}$ game with probability 1 as follows. $\mathsf{Adv_r}$ selects two same-size record-sets $RS_0 \neq RS_1$ and computes their hash $h_i = H(RS_i)$. Once it receives a challenge database DB_b, $\mathsf{Adv_r}$ checks whether the header of DB_b contains h_0 or h_1 and outputs its choice for b' accordingly.

In practice, given two databases, this allows $\mathsf{Adv_r}$ to determine whether their content is identical even if their corresponding ciphertexts are different. Also, assuming that the record-set encrypted in a database has lower entropy than the database master password, $\mathsf{Adv_r}$ can recover the content of the record-set by simply making a guess and comparing it against the hash value in *hdr*. In other words, the complexity of breaking the database is a function of $\min(\eta_{mp}, \eta_{RS})$ – where η_{mp} is the entropy of the master password and η_{RS} is the entropy of the record-set – rather than just a function of the master password.

hdr is not authenticated and, as such, is susceptible to malicious modifications. This can be used by $\mathsf{Adv_{rw}}$ to win the $\mathsf{MAL\text{-}CDBA}$ game with probability 1 by selecting a challenge record-set RS which contains one or more entries. When $\mathsf{Adv}_{\mathsf{rw}}$ receives DB he changes the value corresponding to the number of entries (stored in hdr) to a smaller number. Since bdy is not altered, the hash verification does not fail. However, the record-set has been altered since the number of entries shown in the password manager is now the one chosen by $\mathsf{Adv}_{\mathsf{rw}}$.

We verified that the latest version of KeePassX (0.4.3) is susceptible to this attack. Moreover, if the victim makes any change in the modified database, KeePassX stores only the entries displayed. This can lead to silent (undetected) corruption of the database.

4.6 KDBX4 (aka KeePass 2.x)

Format Description. The KDBX4 database format is composed of a single password-protected file, divided in two sections: an unencrypted header (hdr) and a main encrypted body (bdy). hdr contains several fields, including `mseed` and `tseed` (used to compute the encryption/decryption key from the user-provided password), `IV`, `pskey` and `ssbytes`, used for secrecy and integrity protection as detailed below.

bdy contains the database records encoded as a single XML string, optionally compressed using the gzip algorithm [29] before encryption. bdy is encrypted using AES-256 in CBC mode, although Twofish is also available. The first 32 bytes of bdy contains the encryption of the `ssbytes` field in order to efficiently verify whether the provided master password is correct. The next 32 bytes of the body contain the hash of the (possibly gzip-compressed) XML string representing the various entries. This hash is used to detect modifications in the database.

In addition, all passwords in the XML string are XOR-ed with a pseudo-random string, computed using Salsa20 [30]. Every time the database is saved, a random 256-bit key k is generated and stored unencrypted in the `pskey` field; each password pwd_i is then encoded as $s_i = pwd_i \oplus \text{Salsa20}(k, IV)$ using a fixed value IV. Each pwd_i uses a different portion of the keystream generated by Salsa20. Passwords are recovered as $pwd_i = s_i \oplus \text{Salsa20}(k, IV)$.

Security Analysis. KDBX4 fixes some of the weaknesses of KDB. hdr does not store the (unauthenticated) number of entries, therefore an adversary cannot alter this value to remove content from the password database. Also, the hash of the unencrypted record-set is now stored in encrypted form. This prevents an adversary from verifying its guesses on the database content and from determining whether two encrypted databases carry the same content. More generally, $\mathsf{Adv}_{\mathsf{r}}$ cannot mount any successful attack on a KDBX4 database except with negligible probability. Due to lack of space, we omit the proof of IND-CDBA security for KDBX4, which is available in the extended version of this paper [31].

Unfortunately, this format introduces new vulnerabilities. Similarly to KDB, the main problem of this format is the lack of authentication of hdr. As such, is it susceptible to modifications. In particular, $\mathsf{Adv}_{\mathsf{rw}}$ can win the MAL-CDBA game with probability 1 as follows. $\mathsf{Adv}_{\mathsf{rw}}$ outputs a challenge record-set RS. Then, after receiving the corresponding database DB, it replaces the value stored in the `pskey` field of *hdr* with an arbitrary 256-bit string and outputs that as DB'. This modification is not detectable by the password manager, i.e., $\mathsf{Valid}(DB') = 1$, since the integrity check on the records is performed *before* the XOR with the output of Salsa20. However, a different `pskey` value will cause all passwords to appear as pseudo-random data after the decoding process.

It is impossible to recover from this attack unless it is detected immediately, i.e., before the user applies any modification to the record-set. The only way to recover the database content is to restore the original `pskey` value. However, this value is replaced with a fresh one, and all passwords are "re-scrambled" accordingly, each time the database is modified and saved. For this reason if a user alters, and then saves, a corrupted database, all passwords previously affected by the attack are lost forever.

This attack highlights a remarkable design flaw. Even an accidental bit-flip in the `pskey` field, e.g., due to a transmission error, cannot be detected, and leads to complete corruption of the database. Such corruption is unlikely to be immediately detected by users, who may subsequently add new entries. Over time, the database will be composed of both correct and corrupted entries, making it difficult to reconstruct the damaged records from a backup.

As an extension to the previous attack, $\mathsf{Adv_{rw}}$ can alter `pskey` in such a way that an arbitrary (small) number of bits of the first password(s) in the database are not altered. To do that $\mathsf{Adv_{rw}}$ computes a value k' such that the first n bits of Salsa20(k, IV) are equal to the first n bits of Salsa20(k', IV). Then $\mathsf{Adv_{rw}}$ stores k' in `pskey`. k' can be computed in exponential time in n, and therefore is practical only when n is small. As a proof of concept, we developed an application that implements such attack. The application is available upon request.

Finally, $\mathsf{Adv_{rw}}$ can also win the $\mathsf{MAL\text{-}CDBA}$ game as follows. Given an arbitrary database DB, $\mathsf{Adv_{rw}}$ flips a bit in the first 16 bytes of `ssbytes`, and then flips the corresponding bit in the `IV` field of *hdr* to create DB'. The password manager cannot detect the change, i.e., $\mathsf{Valid}(DB') = 1$, since flipping a bit in `IV` causes the corresponding bit in the first block of plaintext to be flipped as well (using CBC-mode), and no additional side effect. Since the first block of plaintext corresponds to the first 16 bytes of `ssbytes`, the modification produces a new correct database. This allows $\mathsf{Adv_{rw}}$, given a database DB, to produce up to $2^{128} - 1$ different databases $DB'_1, \ldots, DB'_{2^{128}-1}$ containing the same record-set as DB.

4.7 PINs

Format Description. The PINs database is stored in a single file, and encrypted using AES in CBC mode. Records are encrypted separately and stored one record per line, using hexadecimal representation written as ASCII text.

The first line of each database defines the version of the software used to create the database, while the second line contains the encryption of the string "`#TEST VERIFY`" followed by a variable number of up to fifty random bytes. This is used to verify that the user-provided master password is correct. After deriving the database encryption/decryption key from the user's input, PINs decrypts the second line and determines whether the result corresponds to the expected string.

Security Analysis. Each line containing user data is encrypted with AES in CBC mode, which is known to be $\mathsf{IND\text{-}CPA}$-secure [32]. As shown in Appendix A, $\mathsf{IND\text{-}CPA}$ security implies $\mathsf{IND\text{-}CDBA}$ security. Therefore $\mathsf{Adv_r}$ cannot extract any information from an encrypted database, besides the number of records and their approximate length.

However, PINs' database file does not provide any kind of data integrity. As such, an adversary can exploit the malleability of the CBC mode of operation to modify the content of the database. Since each line is encrypted separately, changes in one record do not affect other records. $\mathsf{Adv_{rw}}$ can exploit this property to win the $\mathsf{MAL\text{-}CDBA}$ game with probability 1. After receiving a challenge database DB corresponding to an arbitrary record-set RS, $\mathsf{Adv_{rw}}$ flips one bit in any of the records to obtain a new database DB' which is considered correct by PINs, i.e., $\mathsf{Valid}(DB') = 1$. $\mathsf{Adv_{rw}}$ can also remove arbitrary entries, or replace them with versions collected from different challenge databases.

4.8 PasswordSafe v3

Description. The PasswordSafe v3 database is composed of a single file containing all entries [33]. The file can be logically divided into two parts: a header (*hdr*), an encrypted body (*bdy*). *hdr* includes (among other fields) an `IV` and a pair of 256-bit keys, `K` and `L`, which are used to encrypt *bdy* and to provide authentication using HMAC respectively. `K` and `L` are encrypted using Twofish [34] in ECB mode, under a key derived from a user-provided master password. *bdy* contains the various database entries, and terminates with an HMAC computed over all fields (before encryption) from *hdr* to the last entry of *bdy*, only excluding the database version number.

Security Analysis. PasswordSafe v3 is $\mathsf{IND\text{-}CDBA}$-secure *and* $\mathsf{MAL\text{-}CDBA}$-secure. (Due to lack of space we omit a formal proof of this statement. The proof is available in the extended version of the paper [31].) As such, neither $\mathsf{Adv_r}$ nor $\mathsf{Adv_{rw}}$ can win their respective games with non-negligible probability over 1/2.

However, we identified a design flaw that, although irrelevant in our security model, should be considered when adopting this format. The PasswordSafe v3 database format stores both the encryption key and the MAC key used to secure the database content in the file header. In this way, if the master password is changed, the database does not need to be re-encrypted. This technique is usually adopted by encrypted file systems (e.g., [35]) to avoid having to re-encrypt all the data if the master password is changed. However, we believe that this choice may not be appropriate for a password database file. In particular, every time the database is modified, `IV` is changed and therefore the whole database is re-encrypted. For this reason, the reuse of the same values for `K` and `L` does not imply any savings.

Additionally, this specification detail opens the door to an attack. Assume that an adversary is able to obtain the master password for an encrypted database. Using the master password, the adversary would also be able to retrieve (and store) `K` and `L`. Subsequently, even if the user changes her master password, the adversary can still decrypt and/or modify any new version of the database. The only way to recover from a compromise of the master password is to completely discard the database and create a new one, i.e., changing the master password serves no purpose. It should be noted that some implementations that use the PasswordSafe v3 format are not vulnerable to this attack (e.g., Password Safe [18]), since they choose a new random `K` and `L` every time the database is saved. This makes such implementations less efficient than they could be, but secure.

4.9 Roboform

Format Description. Roboform stores its password database in several files. Each file contains a header, which encodes two URLs: `goto`, which is used as a bookmark by Roboform's browser plugin, and `match`, which is used by Roboform's plugin to determine which username/password record should be used on each web form.

The rest of the record is composed of a short header and an encrypted payload. Roboform allows users to choose between AES, Blowfish, DES, Triple-DES and RC6 for payload encryption.

Security Analysis. Roboform's password format is vulnerable to attacks from both $\mathsf{Adv_r}$ and $\mathsf{Adv_{rw}}$ in our security model.

Adversary $\mathsf{Adv_r}$ can win the $\mathsf{IND\text{-}CDBA}$ game with probability 1 by constructing two same-size record-sets RS_0 and RS_1 which differ in at least one of the URLs in their records. Since neither the `goto` nor the `match` fields are encrypted, $\mathsf{Adv_r}$ can always identify which record-set corresponds to challenge DB_b. As a proof of concept, we wrote a small script that decodes the `goto` and `match` URLs. The script is available upon request.

In practice, this allows $\mathsf{Adv_r}$ to gather recover a list of web site visited by the user even if web cache and history have been deleted.

Similarly, $\mathsf{Adv_{rw}}$ can win the $\mathsf{MAL\text{-}CDBA}$ game with probability 1, since neither of the URLs stored in Roboform's database are integrity protected. $\mathsf{Adv_{rw}}$ requests a database corresponding to an arbitrary recordset RS, and after receiving the corresponding database DB, creates DB' by altering one or both URLs. The lack of integrity protection means that $\mathsf{Valid}(DB') = 1$.

In practice, an adversary can use this vulnerability to mount a phishing attack by altering URLs and redirecting users to a malicious website designed to capture login credentials.

5 Discussion

Table 2 summarizes the result of our security analysis. Almost all the formats are vulnerable to attack either in the $\mathsf{IND\text{-}CDBA}$ or $\mathsf{MAL\text{-}CDBA}$ security model, or

Table 2. Vulnerabilities overview. This table shows, for each format, whether it is secure (✓) or broken (×) in the two security games IND-CDBA and MAL-CDBA, defined in Section 3. [1]PasswordSafe v3 is secure in our model but with an interesting design flaw (see Section 4.8).

	Read-Only Attacker (IND-CDBA)	Read-Write Attacker (MAL-CDBA)
Google Chrome	×	×
Mozilla Firefox	×	×
Microsoft Internet Explorer	×	×
1Password	×	×
KDB (aka KeePass 1.x)	×	×
KDBX4 (aka KeePass 2.x)	✓	×
PINs	✓	×
PasswordSafe v3	✓[1]	✓[1]
Roboform	×	×

both. What does that mean for the use of these formats in practice? The answer depends on the security provided by the storage mechanism that hosts the password database. We divide the database formats into three classes: *Class I*: those that can be used on an insecure storage medium. According to our analysis, the only format in this class is PasswordSafe v3; *Class II*: those that can be used if the underlying storage mechanism provides integrity and data authenticity. This class contains KDBX4 and PINs; and *Class III*: those that can be used securely only if the underlying storage provides integrity, authenticity and secrecy. This class contains the remaining formats.

Class I password managers can be used safely without any special considerations, except for one caveat with PasswordSafe v3 described in Section 4.8. To safely use Class II password managers in practice, users should make sure never to rely on any information in the database that could have been changed by a malicious adversary. For example, if privacy is not a concern and the password database is kept on, e.g., a read-only smart card, KDBX4 and PINs *can* be used to securely store passwords.

There is nothing inherently wrong with storing passwords in a Class III password manager, e.g., an unencrypted text file, as long as the user is made aware that the format provides no secrecy, integrity or authenticity. In fact, if the user is taking additional steps to store an unencrypted password database on a secure medium (e.g., an encrypted file system) this may be a perfectly safe approach. As an example, Google Chrome stores passwords in a database format that is not designed to provide security. To use the Google Chrome password manager in practice, users should completely prevent access to the database from any unauthorized party (e.g., other users of the same machine).

It seems fair to require that a password manager that asks users to authenticate themselves with a password, at least provides secrecy and data authenticity. This is currently only achieved by a single password database format, namely

PasswordSafe v3. As a general rule, a password manager should be explicit about the security offered by the underlying database format.

6 Related Work

Although the concept of a password manager is well known and used by people all over the world, there is very little scientific literature on the subject.

In 2003 Luo and Henry proposed a method for protecting multiple accounts [36]. Their solution requires a user to remember only one password, called a common password, to access any of a number of accounts. The authors propose a Web based implementation with a password calculator written in JavaScript.

In an attempt to solve the same problem, Blasko published an IBM Research Report in 2005 [37] proposing a Wristwatch-Computer Based Password-Vault. Blasko describes the design and implementation of a wearable computer with wireless connectivity, processing, input, and display capabilities, that is meant to store a users passwords for different services.

A year later, Gaw and Felten published a study of Password Management Strategies for Online Accounts [21]. The authors studied how many passwords 49 undergraduates had, and how often they reused these passwords. At that time about 38% of the people participating in the study used password managers. More than two thirds of those used online, web based password managers. With the inclusion of password managers in popular browsers, that number is presumably significantly higher today.

In 2009 Englert and Shah published a paper on the Design and Implementation of a secure Online Password Vault [38]. This works describes an architecture where encryption and decryption is done locally on the user's machine but storage done online.

Bonneau and Preibusch reported results of, what they claim is, the first large-scale empirical analysis of password implementations deployed on the Internet [39]. This study included 150 websites which offer free user accounts for a variety of purposes, including the most popular destinations on the web and a random sample of e-commerce, news, and communication websites. This work does not deal directly with password managers but the findings support the claim that many online services use poor practices when dealing with user credentials. This serves to highlight the need for password managers and consequently, the need for secure password manager database formats.

In [40], Belenko and Sklyarov analyze the security of several password manager applications running on iOS and BlackBerry smartphones. Their analysis focuses on a passive adversary, who is able to access a password database *at rest*. The goal of the adversary is to determine the database master password, and therefore access the protected data. The authors show that most password managers either force the user to protect the database using a short (four digit) PIN, or do not use expensive key derivation functions to compute the database encryption/decryption key from the master password. This allows an adversary to perform password recovery attacks relatively short time for low-entropy passwords.

7 Conclusion

Password managers are critical pieces of software used to securely store sensitive information. This paper presents the first rigorous analysis of the storage formats used by popular password managers.

We defined two realistic security models, designed to represent the capabilities of real-world attacks. One for passive and one for active attackers. We analyzed popular password manager database formats in our security models; for each vulnerable format, we provided a formal argument for why it is broken. We also showed what the theoretical vulnerability means in terms of practical attacks. Additionally, when a database format was found to be secure, we provided a formal proof.

Unfortunately, most formats turned out to be broken even against very weak adversaries. For this reason, users should carefully consider whether a particular database format is acceptable for storing data in the cloud, on a USB drive or on a machine shared with other users.

Finally, our works shows that it is indeed possible to construct a format that provides security, usability and low computation and storage overhead, using standard cryptographic tools.

References

1. Trusteer: Reused Login Credentials, `http://www.trusteer.com/sites/default/files/cross-logins-advisory.pdf`
2. Herzberg, A.: Why Johnny can't Surf (Safely)? Attacks and Defenses for Web Users. Computers & Security 28(1-2) (2009)
3. Dhamija, R., Tygar, J., Hearst, M.: Why Phishing Works. In: SIGCHI Conference on Human Factors in Computing Systems. ACM, New York (2006)
4. RomanLab Co. Ltd.: USB password manager: When your password database is right where you need it, `http://www.anypassword.com/password-database-in-usb-password-manager.html`
5. Siber Systems, Inc.: Roboform2go for USB drives, `http://www.roboform.com/platforms/usb`
6. Portable Apps: Keepass password safe portable, `http://portableapps.com/apps/utilities/keepass_portable`
7. 1Password: Automatic Syncing Using Dropbox, `http://help.agilebits.com/1Password3/cloud_syncing_with_dropbox.html`
8. KeePassDroid: Dropbox and KeePassDroid, `http://blog.keepassdroid.com/2010/06/dropbox-and-keepassdroid.html`
9. AgileBits, Inc.: 1password, `https://agilebits.com/onepassword`
10. Vanhove, M.: Kypass, `http://itunes.apple.com/us/app/kypass/id425680960?mt=8`
11. Pellin, B.: Keepassdroid, `http://www.keepassdroid.com`
12. Google: Get a fast, free web browser, `https://www.google.com/chrome/`
13. Mozilla: Firefox, `http://www.mozilla.org/`
14. Microsoft: Internet Explorer 9, `http://windows.microsoft.com/en-us/internet-explorer/products/ie/home`
15. KeePass – A Free and Open-source Password Manager, `http://keepass.info/`

16. Muiznieks, R.: Passdrop, http://itunes.apple.com/us/app/passdrop/id431185109?mt=8
17. PINs, Secure Passwords Manager, http://www.mirekw.com/winfreeware/pins.html
18. Password Safe – Simple & Secure Password Management, http://passwordsafe.sourceforge.net/
19. Pilhofer, F.: Password Gorilla, http://www.fpx.de/fp/Software/Gorilla/
20. Siber Systems, Inc.: RoboForm, http://www.roboform.com/
21. Gaw, S., Felten, E.: Password Management Strategies for Online Accounts. In: SOUPS 2006. ACM Press, Pittsburgh (2006)
22. Katz, J., Yung, M.: Unforgeable Encryption and Chosen Ciphertext Secure Modes of Operation. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 284–299. Springer, Heidelberg (2001)
23. Bellare, M., Namprempre, C.: Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. J. Cryptology 21(4) (2008)
24. Google: Protect your synced data, http://support.google.com/chrome/bin/answer.py?hl=en&answer=1181035
25. Mozilla: Firefox Sync for Mobile, http://www.mozilla.org/en-US/mobile/sync/
26. Frazier, M.: Sync Firefox from the Command Line, http://www.linuxjournal.com/content/sync-firefox-command-line
27. Microsoft Dev Center: CryptProtectData function, http://msdn.microsoft.com/en-us/library/windows/desktop/aa380261(v=vs.85).aspx
28. AgileBits, Inc.: 1password agile keychain design, http://help.agilebits.com/1Password3/agile_keychain_design.html
29. GNU zip: The GZIP homepage, http://www.gzip.org/
30. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs. LNCS, vol. 4986, pp. 84–97. Springer, Heidelberg (2008)
31. Gasti, P., Rasmussen, K.: On The Security of Password Manager Database Formats. Technical report, UCI (2012), Available from Cryptology ePrint Archive, http://eprint.iacr.org
32. Damgaard, I., Nielsen, J.: Expanding Pseudorandom Functions; or: From Known-Plaintext Security to Chosen-Plaintext Security. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 449–464. Springer, Heidelberg (2002)
33. Password Safe V3 Database Format, http://passwordsafe.svn.sourceforge.net/viewvc/passwordsafe/trunk/pwsafe/pwsafe/docs/
34. Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C.: Twofish: A 128-Bit Block Cipher. Current 21(1) (1998)
35. Ferguson, N.: AES-CBC + Elephant diffuser A Disk Encryption Algorithm for Windows Vista. Technical report, Microsoft Research (2006)
36. Luo, H., Henry, P.: A Common Password Method for Protection of Multiple Accounts. In: International Symposium on Personal, Indoor and Mobile Radio Communication (2003)
37. Blasko, G., Narayanaswami, C., Raghunath, M.: A Wristwatch-Computer Based Password-Vault. Technical report, IBM Research Division (2005)
38. Englert, B., Shah, P.: On the Design and Implementation of a secure Online Password Vault. In: ICHIT 2009. ACM Press (2009)
39. Bonneau, J., Preibusch, S.: The Password Thicket: Technical and Market Failures in Human Authentication on the Web. Information Security 8(1) (2010)

40. Belenko, A., Sklyarov, D.: "Secure Password Managers" and "Military-Grade Encryption" on Smartphones: Oh, Really? Technical report, Elcomsoft Co. Ltd. (2012), http://www.elcomsoft.com/WP/BH-EU-2012-WP.pdf
41. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman & Hall/CRC (2008)

A Relationship between IND-CPA and IND-CDBA

In this section we shed light on the relationship between our notion of IND-CDBA-security and the standard IND-CPA-security. Let $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ be a $\mathsf{IND\text{-}CPA}_{\mathsf{Adv_r},\Pi}(\kappa)$-secure encryption scheme. We recall the standard definition of IND-CPA security [41]:

Game 3 **($\mathsf{IND\text{-}CPA}_{\mathcal{A},\Pi}(\kappa)$).** *Indistinguishability of chosen plaintext attack. A challenger* Ch *running* Π *interacts with* $\mathcal{A}$ *as follows:*

- Ch *runs* $mp \leftarrow \mathsf{Setup}(1^\kappa)$.
- $\mathcal{A}$ *is given oracle access to* $\mathsf{Enc}_{mp}(\cdot)$
- *Eventually* $\mathcal{A}$ *outputs two same size messages* RS_0, RS_1
- Ch *selects a bit* b *uniformly at random and the ciphertext* $DB_b \leftarrow \mathsf{Enc}(mp, RS_b)$ *is returned to* $\mathcal{A}$.
- $\mathcal{A}$ *eventually outputs bit* b'*; the game outputs 1 iff* $b = b'$.

Definition 4 **(IND-CPA security).** *An encryption scheme* $\Pi = (\mathsf{Setup}_\Pi, \mathsf{Enc}_\Pi, \mathsf{Dec}_\Pi)$ *has indistinguishable encryptions under chosen plaintext attack if there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any efficient adversary* $\mathcal{A}$, $\Pr[\mathsf{IND\text{-}CPA}_{\mathcal{A},\Pi}(\kappa) = 1] \leq 1/2 + \mathsf{negl}(\kappa)$.

It is easy to see that $\mathsf{IND\text{-}CPA}_{\mathcal{A},\Pi}(\kappa)$ security implies $\mathsf{IND\text{-}CDBA}_{\mathsf{Adv_r},\mathcal{PM}}(\kappa)$ security. Let $\mathcal{PM} = (\mathsf{Setup}, \mathsf{Create}, \mathsf{Open}, \mathsf{Valid})$ where $\mathsf{Setup} = \mathsf{Setup}_\Pi$, $\mathsf{Create} = \mathsf{Enc}_\Pi$, $\mathsf{Open} = \mathsf{Dec}_\Pi$ and Valid is defined as in Section 3.

Assume $\mathsf{Adv_r}$ is an adversary that has a non-negligible advantage in the IND-CDBA game. We show how to build a simulator SIM that uses $\mathsf{Adv_r}$ to win the IND-CPA game. SIM lets $\mathsf{Adv_r}$ choose RS_0 and RS_1, and forwards these to Ch. Ch returns DB_b which is forwarded to $\mathsf{Adv_r}$. Eventually $\mathsf{Adv_r}$ outputs its choice for b', and SIM uses it to answer the challenger. Since $(\mathsf{Setup}, \mathsf{Create}, \mathsf{Open})$ is defined as $(\mathsf{Setup}_\Pi, \mathsf{Enc}_\Pi, \mathsf{Dec}_\Pi)$, Sim's advantage is identical to $\mathsf{Adv_r}$'s.