快速字典攻击密码使用时空权衡

纳拉亚南和什马提科夫
德克萨斯大学奥斯汀分校
{arvindn, shmat}@cs.utexas.edu

摘要

人类令人难忘的密码是计算机安全的主要支柱。为了减少密码对暴力字典攻击的脆弱性，许多组织强制执行复杂的密码创建规则，并要求密码包括数字和特殊字符。我们证明，只要密码仍然是人类令人难忘的，即使潜在密码的空间很大，它们也很容易受到“智能字典”的攻击。

我们的第一个见解是，容易记住的密码中的字母分布很可能与用户母语中的字母分布相似。使用自然语言处理中的标准马尔可夫建模技术，这可以用来大大减少要搜索的密码空间的大小。我们的第二个贡献是一个精确枚举剩余密码空间的算法。ffi这允许应用时空贸易技术，限制内存访问到一个相对较小的“部分字典”大小的表，并支持一个非常快的字典攻击。

我们在一个真实世界的用户密码哈希数据库上评估了我们的方法。我们的算法使用a 2 10成功地恢复了67.6%的密码$\times^9$查找空间这比奥克斯林的“彩虹”攻击要高得多，后者是目前已知的搜索大密钥空间的最快的技术。这些结果对人类令人难忘的字符序列密码作为一种身份验证机制的可行性提出了质疑。

类别和主题描述符：K。[安全与保护]：身份验证。4.6【安全与保护】：身份验证；
E.3.数据加密：分码

通用条款：安全

关键词：密码，字典攻击，时空交易，密码分析，马尔可夫模型

CCS ‘05,2005年11月7-11日，美国弗吉尼亚州亚历山大市。

1.介绍

人们常说，人类总是安全链中最薄弱的一环。社会工程在渗透系统方面比对行攻击更成功。类似地，破解密码很少是通过破解所使用的密码来完成的，更多的情况是通过利用人类生成的环境来完成的。

猜测人类生成的密码的问题已经被研究了很久。Morris和汤普森[27]在他们1979年发表的关于UNIX密码安全的论文中描述了现代读者非常熟悉的暴力和字典攻击。前者是基于短弦更容易记住的观察，后者是基于有意义的单词比随机字符序列更令人难忘。

从那时以后，字典攻击的技术水平就没有取得多大进展。目前的密码破解者，如JohnRipper[29]，使用基本相同的两种技术，再加上一些规则，将每个字典单词转换成一组相关的单词（比如补充一个数字）。ffiffi通过使用欧克斯林的“彩虹”技术[28,1]，可以使暴力攻击变得更加有趣，该技术使用预先计算来加快破解单个密码的过程。一种常见的辩护是使用密码创建规则，称为组合规则，它要求密码从某些常规语言中提取密码，并要求密码包含数字和非字母数字字符。其目标是增加搜索空间的大小，使朴素的字典攻击不可行。

我们证明了快速、毁灭性的“智能字典攻击”甚至可以出现在不易受到暴力或直接字典攻击的大密码空间上。我们的见解是，密码，即使长且带有额外的字符，仍然必须是令人难忘的。人类的局限性意味着密码的熵相当小：一个NIST文档[7]估计，用户生成的8个字符的密码有18到30位的随机性，即使英语字典单词被删除，组合规则被强制执行。在非常一般的意义上，我们的攻击通过快速枚举候选密码，这些密码可以产生少量的随机性。例如，在32个字符时，这个字符串远远超出了使用当前硬件进行暴力攻击的范围。然而，从直观上看，这个字符串不是非常随机的，应该很容易猜测。

在形式上，这可以通过说Kol-

这个字符串的莫戈罗夫复杂度[23]很低。科尔莫戈罗夫的复杂性。k.a.字符串被定义为输出它的最短图灵机的长度。在任何合理的编码方案中，指令序列重复打印16次，都应该有大约30位的复杂性，这符合我们的直觉，即生成的字符串很容易猜测。

一个明显的方法是尝试枚举所有k复杂度小于某个阈值的字符串，并将结果用作字典。这有两个问题。首先，k-复杂度是不可计算的。第二，人类对随机性的感知与计算中的随机性有所不同。ffi认知科学家格里思和特南鲍姆•[15,16]对主观随机性进行了研究，他们认为我们需要一种不同的方法来衡量一个字符串的感知信息内容。我们需要的是对人类密码生成过程进行严格的多学科研究，以及对结果密码空间进行非常精确的枚举的算法。ffi这种算法的存在将是快速字典攻击脆弱性的一个有力指标。

我们先朝两个方向迈出了第一步。我们假设，与用户母语中的单词的语音相似性是可记忆性的一个主要贡献者。我们使用基于马尔可夫模型的“马尔可夫过滤器”来捕获这一特性，这是自然语言处理[32]中的一种标准技术。我们的另一个观察结果是，人类在密码中使用非字母字符的方式可以通过inite自动机进行建模，这可以被认为是对开膛手约翰和其他密码破解工具所使用的“规则”的一种概括。

ffiffi其次，我们提出了一种枚举满足马尔可夫长度的所有字符串的算法，以及一种枚举确定性自动机接受的给定长度字符串的算法。更准确地说，我们给出了一些算法，给定一个索引i，生成i^{th}满足这些属性的字符串（无需搜索整个空间！）。接下来，我们将这两种算法结合成一个混合算法，从而产生i^{th}同时满足马尔可夫过滤语言和规则语言的字符串。最后，我们展示了如何使用这个算法在这个搜索空间上实现一个时空贸易。

最著名的时空攻击技术。它使用了一种被称为“彩虹表”的特殊数据结构。我们的混合攻击在相同大小的搜索空间上具有与彩虹攻击相同的预计算时间、存储需求和平均密码分析时间。然而，与香草彩虹攻击在整个混合长度的密钥空间(e。g.，″所有长度为8”的字母数字字符串)，我们的攻击具有明显更高的覆盖率，因为我们的搜索空间是明智地选择的，并且只由具有低主观随机性的“令人难忘的”字符串组成。

当然，字典攻击的覆盖范围只能通过将其应用于“真实世界”的用户密码来衡量。在人工生成的密码上获得的结果并不是很令人信服，因为不能保证生成算法产生的密码与用户自己选择的密码相似。我们在一个由密码软件提供给我们的150个真实用户密码的数据库上评估了我们的技术。我们的混合攻击达到了67.6%的覆盖率

(i．e.，超过三分之二的密码被成功恢复)，使用了210大小的搜索空间$\times^{9}$.详细的结果见第5节。我们使用的过滤器可以在附录A中找到。相比之下，奥克斯林的攻击的覆盖率仅为27.5%[1].

我们相信，随着一个更大的搜索空间(约3 10个$\times^{12}$，这就是密码破解的方法。com实现使用)，以及更全面的正则表达式，覆盖率最高可达90%。一个更严格和更大的关于彩虹裂纹[33]实现的实验正在进行中。

抵御字典的攻击。使用密码散列可以击败基于预先计算的网元字典攻击，从而挫败我们的混合攻击。

ffi使用一个有害的密码可以使攻击者降低一个常数因素，这实际上是在UNIX隐窝（）实现中完成的。然而，由于客户端可能运行的平台范围大，这种技术只能产生有限的收益。例如，某些浏览器中的Javascript实现非常慢。

费尔德迈耶和Karn [10]调查了提高密码安全性的方法，并得出结论，唯一能够进行显著长期改进的技术是让用户增加他们生成的密码的熵。正如我们所展示的，实现这一点比之前想象的要困难得多。

在上述调查之后，还有大量关于密码认证的密码协议和从人类令人难忘的密码[3,24,2,6,19,14,13]生成会话密钥的工作。ffi这些协议的目的是击败对参与者共享低熵秘密的协议的在线字典攻击。

密码验证密钥交换（PAKE）协议的一个缺点是，它们通常依赖于不现实的假设，如多个不合作的服务器或双方将密码以明文存储（一个例外是[6]的PAK-X协议）。在实践中，将客户端密码存储在服务器上是非常危险的，但即使对于“可证明是安全的”的PAKE协议，安全证明也隐式地假定服务器不会被破坏。

此外，我们的攻击在有限的意义上甚至适用于PAKE协议协议，因为我们的马尔可夫过滤器也使在线字典攻击更快。因此，我们的攻击引发了我们的问题，即人类生成自己的字符序列密码是否有意义。随着时间的推移，情况只会变得更糟，因为硬件功率呈指数级增长，而人类的信息处理能力保持不变的[25]。考虑到记忆性和高主观随机性之间存在一个基本的缺陷，我们的工作可能意味着密码作为一种身份验证机制的可行性。

组织论文。在第2节中，我们解释了基于马尔可夫模型和确定性初始自动机的过滤。在第3节中，我们讨论了一般的时空贸易，特别是欧克斯林的彩虹攻击，并表明它工作在任何搜索空间，只要有一个生态算法来计算它的i ffith元素在秒内

[1] 我们使用了由小写字母和数字组成的字符集来进行奥赫斯林的攻击。简单的技巧可以提高覆盖范围，例如使用不同的字符集运行多个时间空间交易，以及使用一些小长度的暴力。

第四部分，我们提出了满足马尔可夫过滤器的字符串、DFA接受的字符串和满足这两个条件的字符串的搜索空间枚举算法。第5节包含了我们的实验结果。结论见第6节。

2. 过滤

在本文的其余部分中，我们将交替使用单词的密钥和密码，密文和密码哈希，以及密钥空间和字典。这样做的原因是，强力密码分析的标准技术，如第3节中描述的技术，通常指密钥和密码文，即使应用于密码和它们的哈希值。

.12马尔可夫滤波

由人类生成的字母密码，即使不是字典单词，也不太可能均匀地分布在字母序列的空间中。事实上，如果被要求随机选择一系列字符，一个说英语的用户很可能会生成一个序列，其中每个字符与它在英语文本中出现的频率大致相等分布。对我们的密码数据库的分析显示，有大量的字母密码，它们既不是字典单词，也不是随机序列(我们使用了开放墙。com字典，其中包含约400万字的[30])。

马尔可夫模型常用于自然语言处理中，是语音识别系统[32]的核心。一个马尔可夫模型确定了一个符号序列上的概率分布。换句话说，它允许对具有某些属性的字符序列进行采样。事实上，马尔可夫模型以前已经在密码环境中使用过。在这项工作之后，我们学习了“可扩展多语言密码生成器”软件，它是由Jon卡拉斯在1991年编写的，并使用这种技术为用户生成密码。（我们还被告知，20世纪80年代末，洛斯阿拉莫斯国家实验室也使用了一种类似的方法来生成“随机但可发音”的密码。）因此，马尔可夫建模在猜测用户生成的密码方面非常有效也就不足为奇了。

在零阶马尔可夫模型中，每个字符都是根据潜在的概率分布生成的，并且独立于之前生成的字符。在hrst阶马尔可夫模型中，每个二向图（有序对）字符都被分配一个概率，每个字符都是通过查看前一个字符生成的。在数学上，在零阶模型中，

$$P(\alpha) = \Pi_{X \in \alpha} v(x)$$

而在一阶模型中，

$$P(x1x2...xn) = v(x1\Pi_{i=1}^{n-1}) \; v(xi+1|xi)$$

其中，P（.）是字符序列上的马尔可夫概率分布，xi是单个字符，v函数是英文文本中单个字母和字母的频率。

当然，一个字典不是一个概率分布，而是一个集合。因此，为了创建一个马尔可夫字典，我们通过应用一个阈值θ将概率离散为两个层次。这将了零阶字典

$$Dv,\theta = \{\alpha : \Pi_{X \in \alpha} v(x) \quad \theta\} \geq$$

和错误的字典

$$Dv,\theta = \{x1x2...xn: v(x1\Pi_{i=1}^{n-1}) \; v(xi+1|xi) \; \theta\} \geq$$

零阶模型产生的单词看起来不太自然，但它已经可以通过不考虑绝大多数字符序列，大幅减少可信的密码空间的大小。考虑8个字符的序列。如果选择了θ，则使字典的大小为$^{1}{}_{7}$的关键空间(我。e.,86%的序列被忽略)，那么根据模型生成的序列有90%的概率属于字典。换句话说，15%的密码空间包含90%的可信密码。曲线上其他有趣的点是：一个字典，其中包含$^{1}{}_{-11}$的密钥空间有80%的覆盖率，和一个字典与$^{1}{}_{-40}$的密钥空间有50%的覆盖率，i。e.，只有2.5%的密钥空间需要考虑覆盖所有可能的密码！一阶模型可以做得更好。结果如图1所示。

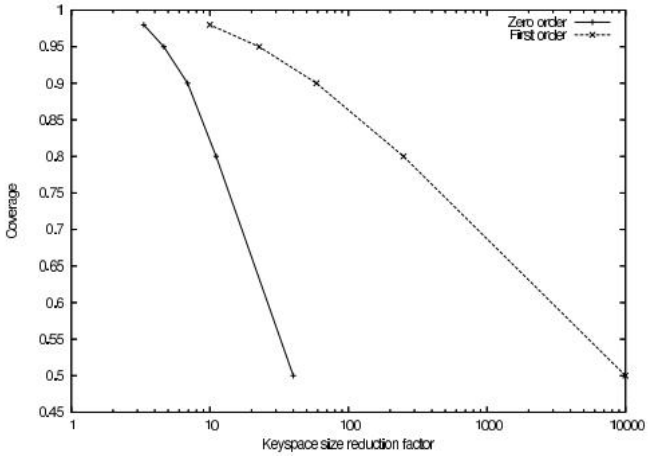

图1：覆盖率压缩图

零阶模型不被一阶模型所排除。零阶模型更适用于某些常用的密码生成策略，例如由句子中每个单词的首字母组成的首字母缩写词。

不用说，通过马尔可夫过滤在密钥空间压缩中使用的字母频率的分布是特定于语言的，而本文中使用的分布只适用于英语用户选择的密码（据推测，对于其他基于字母的语言也可以找到类似的分布）。如果用户的母语不知道，则有两种方法可以推广该技术。首先，可以组合两种或多种语言的密钥空间（第4.6节）。其次，我们有可能想出一种相当适合多种语言的发行版(e。g.，所有日耳曼语或所有罗曼语)。我们还没有尝试过这样做，也不知道它能有多好。

2.2使用有限自动机进行过滤

仅由字母序列组成的搜索空间不太可能很好地覆盖可信的密码空间。人类通常会在密码中混合大写和小写字符，而系统强制执行的密码创建规则通常要求他们输入一些数字，有时还会添加特殊字符。然而，即使有了这些广告-

但是，结果密码的分布远不是随机的。下面是一些常见模式的例子（这个列表绝不是有害的）：

- o在字母数字密码中，所有数字都可能在末尾。
- o字母序列的第一个字符比其他字符更有可能被大写。
- o虽然主要由小写字符和少数大写字符组成的字母序列很常见，但相反的情况则不正确。

确定性初始自动机是表达这类性质的理想方法。首先，我们指定一组常见的正则表达式（“所有小写”、“一个大写后所有所有小写”、“大写字符后所有数字”等等）。我们将字典作为与马尔可夫过滤器匹配的序列集，并且也被至少一个对应于正则表达式的初始自动机所接受。因此Dv，θ，〈Mi〉= {α :Πx∈α≥v(x) θ，和∃i：Mi接受α}。我们的完整字母表由26个小写字符和26个大写字符、10个数字和5个特殊字符（空格、连字符、下划线、句号和逗号）组成。我们选择这些密码有些武断，因为我们觉得它们是虚拟神经密码中最常见的密码。如果不知道应用程序的实际字符集，就不可能考虑所有特殊字符。

这给了我们一个67个字符的字母表。8个字母序列的相关密钥空间为10^{15}，使得暴力搜索不可行。虽然可以为这67个字符的符号集编写正则表达式，但得到的算法不是很好。ffi因此，我们放弃了一点表达性，并将字符集分为四类（小写、大写、数字和特殊字符，我们将分别表示为a、a、n和s），并考虑自动机的输入字母表仅由这四个符号组成。我们所使用的正则表达式列在附录A中。

3.时间空间权衡

最基本的预计算技术是计算和存储按哈希排序的密钥空间中所有密码的哈希，这样密码分析几乎是瞬时的。但是，这需要使用相等于密钥空间大小的存储空间。Hellman [17]展示了如何在密码分析密码的一般背景下，以牺牲攻击时间为代价来减少存储需求。这一传统的工作原理如下。

给定一个混合明文P，在键空间K上的映射f为f (k) = R (E_k(P))，其中E是加密函数，R是一个将密文映射到密钥的缩减函数。通过f的迭代应用程序，我们创建了一个键的“链”。关键的观察是，通过只存储链的hrst和最后一个元素，我们可以确定给定密文对应的键是否在时间O (t)中属于该链（也ind键），其中t是链的长度。

创建一个链的工作原理如下。给定一个起点k0，我们计算k1= f(k0),k2= f(k1)， ...,kt= f(kt-1).钥匙k0和kt被储存起来，而其余的则被扔掉。当给定一个密文C来进行密码分析时，我们通过计算k = R (C)和k来恢复密钥$_i$= f^{i-1}(k)用于i = 1, 2, ...观察，如果键k属于链，则键i。e., k = k_j对于一些，是i，然后是f^{t-i}(k) = k_t.因此，在f最多t应用之后，我们可以确定链是否包含k。➡进一步应用f suce来计算kffi i-1从k0.因为ki-1满足了C = E的特性k_{i-1}(P)，这是我们正在寻找的关键。

这并不确定，因为不同的链可能会合并。因此，我们需要为每个表使用多个带有不同约简函数的多个表，每个表中有许多链。该算法的存储需求和密码分析时间均为O(|K|$^{2/3}$)，其中K是密钥空间。

Rivest [9，p.100]提出了一种改进方法，通过减少密码分析过程中的内存访问次数，大大加快了算法的实际性能。这是使用“区分点”来完成的，这些点是具有一些特殊属性的键（例如前10位为0）。要求链的端点被区分点的优点是，在密码分析期间，只有当一个键具有区分属性时，才必须在内存中查找它。进一步的论文[11,21,5,22,34]提出了区分点算法的优化和/或更好的分析。

Oechslin [28]进行了重大改进，使用“彩虹链”而不是区分点。彩虹链对链中的每个点使用一个不同的约简函数。在[28]中显示，彩虹链与具有相同存储需求的不同点获得相同的覆盖范围，但密码分析时间明显更快。

实验结果令人印象深刻。彩虹攻击[1]的在线实现将长度为8的密码转换在字符集[a-z0-9](密钥空间大小为2.810×^{12}).它的成功概率为0.996，平摊的密码分析时间小于10分钟，使用预先计算的表，大小约为48 GB。

3.1使用索引的一般时空权衡

查找属性

我们观察到，在雷和Oechslin算法中，约简函数可以表示为从密文空间到{0,1，的映射。..|K| 1}，由{0,1，中的映射组成。..|K| 1}到K。约简函数，以及算法的任何其他部分，都不会对密钥空间的大小做出任何假设。在Rivest的攻击中，作为一个区分点的属性可以从键空间索引而不是键本身来计算。在彩虹攻击的情况下，从密文空间到密钥空间索引的映射通过彩虹表的选择进行参数化，但从密钥空间索引到键的映射与Rivest的攻击相同。因此，我们可以对第2节中描述的可信密码的压缩字典实现攻击。

然而，这并不是一件小事。ffi压缩字典必须具有存在一个生态枚举算法的属性，该算法以索引i作为输入并输出ith字典的元素。在下一节中，我们将介绍零阶马尔可夫字典、一阶马尔可夫字典、确定性初始自动机（DFA）、具有一些可索引超空间的任意密钥空间，以及混合马尔可夫/DFA字典的索引算法。

4.索引算法

.14个零阶马尔可夫字典

我们使用的字典是零阶马尔可夫整数的一个稍微修改过的版本，其中我们只考虑混合长度的字符串。这是因为我们想对不同的长度使用不同的阈值。第4.5节中的混合算法将演示如何将多个字典组合成一个字典。修改后的字典是D v , θ , ℓ= {α： |α| =ℓ和Π X∈ α v (x) θ }≥

本节中算法的关键是概率分布的离散化。为了做到这一点，我们首先用加法而不是乘法形式重写过滤：D v , θ , ℓ= {α： |α| =ℓ和 P X∈ α ≥μ(x) λ}，其中μ(x)=log v (x)和λ = log θ 。

接下来，我们将μ的值离散到最接近的μ的倍数0对于一些合适的μ0.如果我们对μ使用一个更大的值0，我们降低了对记忆的需求，但也失去了准确性。在我们的实验中，我们选择了μ0这样就可以看到大约有1000个“关卡”（见下文）。

接下来，我们讨论“部分字典”D v , θ , ℓ, θ /, ℓ/如下所示。设α是任何字符串/和Π X∈ α v (x) = θ /.然后 D v , θ , ℓ, θ /, ℓ/ = {β : α β ∈ D v , θ , ℓ}.

注意D v , θ , ℓ, θ /, ℓ/是精心设计的，因为Π X∈ α β v (x) = Π X∈ α v (x) Π X∈ β v (x) = θ / Π X∈ β v (x).因此，我们选择哪个α并不重要。直观地说，对于任何字符串前缀，部分字典包含所有可能的字符序列的列表，这些字符序列可以附加到这个前缀中，从而生成的完整字符串满足马尔可夫属性。

我们现在提出了一个递归算法来计算一个部分字典的大小

```
| D v、阈值、总长度、电平、电流长度_|
```

请注意，该算法在预计算阶段只执行一次且（而不是对每个键执行一次），因此，它的敏感性不会影响密码分析时间。ffi这里的mu指的是上述μ函数的离散化版本。

```
partial_size1（当前长度，级别）
{
    如果级别>=阈值：返回0，如果总长度=当前长度：返
    回1

    总和=0
    表示字母表中的每个字符
        和=和+partial_size1(当前长度+1,
            +级mu（char）
    回报金额
}
```

D的计算v , θ , ℓ, . , ℓ/取决于D v , θ , ℓ, . , ℓ/+1.因此，部分大小被计算并存储在一个大小为ℓ乘以级别数的二维数组中，计算按ℓ的递减顺序进行/.

我们并不特别关心这个算法的精确性，因为它只在预计算过程中执行。ffi但是，请注意，运行时间（用于计算所有部分大小）与总长度、字符数和级别数的乘积是线性的。

下面是另一种递归算法，它将一个索引输入到密钥空间中，并返回相应的密钥（该算法在密码分析阶段执行）：

```
get_key1（当前长度、索引、级别）
{
    如果总长度=当前长度：返回“”

    总和=0
    表示字母表中的每个字符
        new_level =级别+mu（字符）
        //从预先计算的数组中查找
        大小= partial_size1[
            当前长度+1][new_level]
        如果总和+大小为>索引
                                //  ‘| ’是指字符串连接
            返回字符| get_key1(
                电流长度+1,
                索引和，new_level)
        和=和+大小

    //控件无法到达这里
    打印“索引大于密钥空间大小”；退出
}
```

get_key算法使用latict_size来确定第一个字符（这导致在预先计算的部分大小表中查找一个值），然后重复相对于第一个字符重新计算的索引，并根据第一个字符的频率调整阈值。

为了索引到整个密钥空间，我们调用当前长度为=0和级别为=0的get_key1。

我们注意到在该算法中使用的思想与著名的维特比算法从语音处理[12]的相似性。

4.2一阶马尔可夫字典

在零阶模型的情况下，我们定义了长度限制字典及其部分版本。然而，在读取了部分字符串之后，我们现在需要跟踪最后一个字符，因为这次我们使用的是二重图频率。

```
partial_size2（当前长度，prev_char，级别）
{
    如果级别>=阈值：返回0，如果总长度=当前长度：返
    回1

    总和=0
    表示字母表中的每个字符
        如果电流长度=0
            new_level =亩（char）
        其他的
                new_level =级别+ mu（prev_char，char）
        和=和+partial_size2(当前长度+1,
            char，新级别）
}

get_key2（当前长度、索引、prev_char、级别）
{
    如果总长度=当前长度：返回“”

    总和=0
    字母表中的字符
        如果电流长度=0
            new_level =亩（char）
        其他的
```

```
            new_level =级别+ mu（prev_char，char）
        大小=partial_size2(当前长度+1，
            char，新级别）
        如果总和+大小为>索引
            返回字符| get_key2(
                电流长度+1，
                索引和，字符，new_level)
        和=和+大小

    //控件无法到达这里
    打印“索引大于密钥空间大小”；退出
}
```

.34.确定性有限自动机

该算法类似于零阶马尔可夫字典的算法，除了我们有状态和状态转换，而不是水平和字符频率。get_key3算法与get_key1非常相似，因此被省略了。

```
partial_size3（当前长度，状态）
{
    如果当前长度=总长度
        如果状态是接受状态，请返回1
        其他：返回0

    总和=0
    字母表中的字符
        new_state =转换（字符串，状态）
        如果new_state不是空的话
            和=和+ partial_size3(
                当前长度+1，new_state)
    回报金额
}
```

.44任何关键空间

我们现在描述了一个关于任何密钥空间K的索引算法，只要有一个关于某些超空间K的索引算法，它就可以工作/)K和一个测试程序，给出了α∈K/，决定是否α∈K。例如，我们可以简单地索引到给定长度的（单片）字符序列，并测试一个字符序列是否满足一个马尔可夫过滤器，因此，我们可以使用这个算法索引到马尔可夫字典。缺点是预算涉及枚举K/ 通过它的索引算法，如果K是稀疏的，这可能是非常昂贵的/.例如，考虑具有一阶马尔可夫整数的10个字符的密码序列是相当合理的。这将密键空间压缩10倍5，但是要使用下面的算法来实现这一点，将需要在大于10的密钥空间上进行迭代14.此外，索引本身并不是很有趣。另一方面，它提供了一个很好的起点，因为进一步的关键空间优化。

给定一个参数t，该算法除以空间K/ 在大小为t的箱子中，预先计算每个箱子中K的成员数量并存储它们。当它得到一个索引时，它会迅速找出它属于哪个箱子，并遍历K中的所有键/ 并测试每个箱子的成员资格。

让|K/ | = mt.

```
计算bins(t)
{
    count=0
    对于i=，0到m-1
        对于j = i*t到i*t+（t-1）
            如果K‘的第j个键属于K
                计数=计数+1
        本[i]=计数
}
```

对于每个i，这将计算前i个箱子的累积计数。

```
get_key（指数）
{
    i=二进制搜索（bin[]，索引）
    //，即bin[i] <索引<= bin[i+1]

    count=0
    对于j = i*t到i*t+（t-1）
        键=是K‘的第j个键
            如果键属于K
                count++
            如果计数=索引- bin[i]
                返回电键
}
```

该算法需要O(|K/ |)预计算时间、O（）存储和O（t + log）索引时间。$\frac{|\kappa'||\kappa'|}{t}$观察
该算法与第n素数[4]的算法非常相似。

.54个混合马尔可夫字典/DFA字典

设A为字符集，并考虑组合字典Dv，θ，ℓ_1,M,ℓ_2= ｛α　：|α|　=ℓ,$P_{x\in\alpha,x\in A}$= ℓ/,$\Pi_{x\in\alpha,x\in A}\geq v(x)\theta$，M接受α｝。

如前所述，我们的inite自动机在符号集{A，a，n，s}上工作。所有小写字符都用a表示，所有大写字符都用a表示，所有数字都用n表示，所有特殊字符都用自动机输入中的s表示。

```
get_key5（索引）
{
    计数1=partial_size1（0,0）
    计数2=partial_size2（0，初始状态）

    索引1=索引/count2//商被截断
    索引2=指数-索引1*计数2

    键1=get_key1（0，索引1,0）
    // wlog，我们假设键1由
    //小写字符
    键2=get_key2（0，索引2，初始值_state）

    钥　= ""
    匙　= 1
    pos 键2中的字符：
    为　如果字符是“a”
    了　　将密钥1附加到密钥
            pos = pos+1
        如果字符是“A”
            向键追加大写字母（键1）
            pos = pos+1
        如果字符既不是“a”也不是“a”
            将字符附加到键

    返回电键
}
```

本质上，这个算法会查看get_key2输出中期望使用字母表字符的位置，并替换get_key1返回的字符串。将自动机与一阶马尔可夫滤波器相结合的工作原理类似。

.64多个关键空间

最后，我们展示了如何将多个不相交的密钥空间组合成单个空间。

```
get_key6(K1、K2、Kn、索引）...
{
    总和=0
    对于i = 1到n
        如果总和+大小（Ki）>索引
            返回get_key (Ki，索引和）
        和=和+大小（Ki）

    //，这是无法达到的
    打印“索引大于键空间大小之和”
}
```

.74可能的优化

在本节中，我们将描述一些尚未实现的优化。混合攻击的主要标准是索引应该比哈希算法花费更少的时间。因此，我们希望混合算法使用大约50个一100表查找，并且表必须进入缓存。自动机算法非常快，因为它的输入字母非常小；当有大量键空间时，get_key6可能会很慢，但它可以通过预先计算累积和和适当键空间的二值搜索来加速。因此，我们主要关心的是马尔可夫式的移民组织。我们可以通过重新排序字符来加快get_key1的速度，以便首先出现更频繁的角色，将平均迭代次数减少到6次（如果字符按字母顺序排序，则从13个减少）。这将8个字符字符串的表查找减少到50个以下。同样的策略也适用于get_key2。

5.实验

我们的第一个实验是测量奥克斯林的彩虹攻击的覆盖范围。我们的混合攻击。我们使用了一个数据库，其中包含142个真实的用户密码，并由密码软件提供。我们认为这是一个有代表性的消息来源。我们使用6个字符的字母数字序列（仅小写字符）进行彩虹攻击，密钥空间大小为36^6= 2.17 10^9.为了建模常见的密码模式，我们创建了一组大约70个正则表达式，它们列在附录a中。

下表比较了由彩虹攻击恢复的密码数量和。我们的攻击。

类别	计数彩虹混合动力车		
长度最多为5个，	63	29	63
长度为6个	21	10	17
长度7	18	0	10
长度为8、A或A、其他的	9 31	0 0	6 0
合计	142 96(67.6%)	39(27.5%)	
仅长度6≥	79 33(41.8%)	10(12.7%)	

时空贸易的概率性质的本质已经被忽略了，因为概率可以是通过增加表的大小而任意增加（并且对两种攻击的依赖性是相同的）。

这个实验验证了我们的基本假设，但还需要进一步的实验。由于数据库汇编密码的方式，我们数据库中的密码可能不能代表典型的用户密码。在创建正则表达式时，我们可以访问这些密码，尽管我们确实非常小心地编写了这些表达式，而没有使用关于数据库中密码的特殊知识。一个更好的实验将涉及到一个只包含密码哈希值的数据库，e。g.，从一个具有庞大、多样化的用户基础的系统中获得的/等/密码文件的内容。我们目前正在计划这样的实验，这还将涉及预先计算的值更大的存储，使搜索更大的密钥空间。

6.结论

针对密码的攻击有多种多样，其中基于字典的攻击只是一个子类。最简单的部署方法是社会工程攻击，如假冒、贿赂、网络钓鱼和登录欺骗。其他直接利用人类漏洞的攻击还包括肩部保险和垃圾箱潜水。基于密码的身份验证系统似乎特别容易受到协议弱点的影响，这些弱点可以通过击键记录、“谷歌黑客”、窃听和基于时间和声学发射的侧信道攻击来利用。在基于字典的攻击中，值得一提的是，美国特勤局最近报告了[20]成功定制词典，从分析硬盘中收集的特定信息，包括他们的文件、电子邮件、网络浏览器缓存和访问网站的内容。

抵御记忆和在线词典对人类记忆密码的攻击是一项艰巨的任务。可能的技术包括图形密码[18,36,8]，反向图灵测试[31,35]，以及用生物识别信息[26]强化密码。然而，这些技术需要对身份验证基础设施进行重大更改。未来研究的一个有趣的问题是，人类难忘的密码是否来自非文本空间(e。g.，面孔或几何图像)很容易受到像我们这样的攻击，基于筛选出不太可能的候选人和其余的非常生态的枚举。只有在其中一种被提议的方法被广泛采用，并且真实用户选择了这样的密码之后，才有可能调查这个问题。

致谢我们非常感谢密码软件公司的德米特里·苏明为我们的实验提供了密码材料。

7.参考文献

[1] MD5在线破解使用彩虹表。
http://www.passcracking.com/.

[2] M.贝拉尔，D。poincheval和P。罗格威。经过验证的密钥交换安全抵御字典攻击。在程序中。欧洲墓穴100年，LNCS第1807卷，第139-155页。施普林格，2000年。

[3] S.贝尔洛夫和M。梅里特加密的密钥交换：基于密码的协议，安全地免受字典攻击。在程序中。IEEE安全和隐私

研讨会，第72-84页。IEEE计算机学会，1992年。

[4] A.布克。第n个素数算法。
http://primes.utm.edu/nthprime/algorithm.php, 2005.

[5] J.Borst，B.普雷内尔和J。范德瓦尔。关于穷尽键搜索和表预算之间的时间记忆传统。在程序中。第19届比荷卢联盟信息论研讨会，第111-118页，1998年。

[6] V.博伊科，P。麦肯齐和S。帕特尔可证明安全的密码认证密钥交换使用死亡地狱人。在程序中。欧洲地下室的00年，LNCS的第1807卷，第156-171页。施普林格，2000年。

[7] W.毛刺，D。Dodson和W。波尔克。电子认证指南。NIST特别出版物800-63,2004年。

[8] D.戴维斯，F。蒙罗斯和M。赖特。关于用户选择在图形密码方案。在程序中。13日USENIX
安全研讨会，第151-164页。美国，2004年。[9] D.拒绝。密码学和数据安全。
艾迪生-韦斯利，1982年。

[10] D.C.费尔德迈尔和P。R.卡恩。UNIX密码安全——十年后。在程序中。加密货币，89年，LNCS的第435卷，第44-63页。施普林格，1989年。

[11] A.菲亚特和M。纳尔。反转函数的严格的时间/空间传统。在程序中。STOC ‘91，第534-541页。ACM，1991年。

[12] G.D.福尼维特比算法。《IEEE的诉讼程序》，第61(3)页：268-278页，1973年。

[13] C.高级职员，P。麦肯齐和Z。拉姆赞。使用隐藏的平滑子组进行密码身份验证的密钥交换。在这些程序中，2005年。

[14] O.戈德里奇和Y。林德尔。使用人类随机密码生成会话密钥。在程序中。加密货币‘01，LNCS的第2139卷，第408-432页。施普林格，2001年。

[15] T.L.格里斯和J。B.特南鲍姆概率、算法的复杂性和主观的随机性。《2003年，认知科学学会第25届年会论文集》。

[16] T.L.格里斯和J。B.特南鲍姆从算法到主观的随机性。在神经信息处理系统的研究进展16,2004。

[17] M.赫尔曼一种密码分析的时间记忆技术。《IEEE《信息论学报》，26：401-406,1980。

[18] I.杰明，A。梅耶尔，F。蒙罗斯，M。回转器，和A.鲁宾。图形化密码的设计与分析。在程序中。第8届USENIX安全研讨会，第135-150页。美国，1999年。

[19] J.Katz，R。奥斯特罗夫斯基和M。扬有见之明经过密码身份验证的密钥交换，使用
令人难忘的密码。在程序中。欧洲墓穴‘01，LNCS第2045卷，第475-494页。施普林格，2001年。

[20] B.科尔布斯。DNA基因的关键。《华盛顿邮报》，2005年3月28日。
http://www.washingtonpost.com/wp-dyn/articles/A6098-2005Mar28.html.

[21] K.库苏达和T。松本优化的密码分析的时间记忆交易及其在DES、FEAL-32和跳跳中的应用。《IEICE的基本面交易》，E79-A(1)：35-48年，1996年。

[22] K.库苏达和T。松本实现更高
在不增加内存大小的情况下，密码分析的时间记忆交易的成功概率。TIEICE： IEICE通信/电子/信息和系统学报，1999年。

[23] R.李和P。维丹尼。科尔莫戈罗夫简介
复杂性及其应用。施普林格，1997年。

[24] S.笨蛋。开放密钥交换：如何在不加密公钥的情况下击败字典攻击。在程序中。安全协议研讨会，LNCS的第1361卷。施普林格，1997年。

[25] G.A.厂主神奇的数字7，加负2：对我们处理信息的能力有一些限制。《心理评论》，第63：81-97页，1956年。

[26] F.蒙罗斯，M。Reiter和S。韦泽尔。基于击键动态的密码强化。国际信息安全杂志，1(2)：69-93,2002。

[27] R.莫里斯和K。胸腺素密码安全：一个案例历史记录。《ACM通讯》，第22卷，第2期。11，第594-597页，1979年。

[28] P.奥克斯林做一个更快的密码分析
时间记忆的交换。在程序中。加密货币‘03，LNCS第2729卷，第617-630页。施普林格，2003年。

[29]外墙项目。开膛手约翰的密码破解者。
http://www.openwall.com/john/, 2005.

[30]外墙项目。节目列表集合。
http://www.openwall.com/wordlists/, 2005.

[31] B.平卡斯和T。磨沙机保护密码免受字典攻击。在程序中。第九届ACM计算机和通信安全会议（中国化学会），第161-170页。ACM，2002年。

[32] L.R.拉宾纳。一个关于隐马尔可夫模型的教程
以及在语音识别中的选定应用。
《IEEE诉讼程序》，77(2)：257-286,1989年。

[33]朱双雷。项目雨水裂缝。
http://www.antsight.com/zsl/rainbowcrack/, 2005.[34] F.标准，G。RouvroyJ。J.Quisquater和
J.莱格特的变体一种使用区分点的时间/记忆传统：新的分析和FPGA结果。在程序中。CHES 2002，LNCS的第2523卷，第593-609页。施普林格，2002年。

[35] S.斯图布利宾和P。van Oorschot。使用登录历史记录和人在循环中。在程序中。金融密码学，LNCS的第3110卷，第39-53页。施普林格，2004年。

[36] J.索普和P。van Oorschot。图形字典和令人难忘的图形密码空间。在程序中。第13届USENIX安全研讨会，第135-150页。美国，2004年。

附录

A.针对普通密码模式的正则表达式

为了简化正则表达式和索引算法，我们分别指定了长度限制。每个正则表达式都有输入字母表{A、a、n、s}，分别表示大写、小写、数字和特殊字符。对于每个正则表达式，我们指定4个数字，它们表示相应的符号类型允许在任何可接受的字符串中出现的次数。我们还指定是否使用马尔可夫标记，如果是，它是否应该是零或一阶。最后，我们指定阈值（回想一下，使用马尔可夫过滤，我们将只考虑马尔可夫权重高于阈值的字符串）。阈值是通过说明键空间应该属于字典的部分来指定的。表中的最后一列是键空间的大小。它不是密钥空间指定的一部分。组合的密钥空间的大小是最后一列中条目的总和。

确定每个正则表达式的阈值有些主观，但我们遵循了这些一般规则：任何密钥空间的大小都不能超过10^8；字典的大小应该是密钥空间的10%（基于马尔可夫过滤），除非字符数为4个或更少；在后一种情况下，它应该是密钥空间的30%。

不显示5个字符的正则表达式，因为它们对密钥空间大小的贡献太小。只要至少有6个字母字符，所有它们都是连续的，我们就使用一阶马尔可夫模型。

雷格斯	A	a	n	s	分数	马科夫	尺寸
a*	0	8	0	0	0.000478	1	10^8
a*	0	7	0	0	0.0124	1	10^8
a*	0	6	0	0	0.1	1	$.09\times310^7$
A*	8	0	0	0	0.000478	1	10^8
A*	7	0	0	0	0.0124	1	10^8
A*	6	0	0	0	0.1	1	$.09\times310^7$
Aa*	1	6	0	0	0.00178	1	10^8
Aa*	1	5	0	0	0.0540	1	10^8
n*	0	0	8	0	1	–	10^8
n*	0	0	7	0	1	–	10^7
n*	0	0	6	0	1	–	10^6
a*n*	0	6	1	0	0.0324	1	10^8
a*n*	0	5	1	0	0.1	0	$1.1810\times^7$
a*n*	0	4	2	0	0.3	0	$1.3710\times^7$
a*n*	0	3	3	0	1	–	$1.7510\times^7$
a*n*	0	2	4	0	1	–	$.76\times610^6$
a*n*	0	1	5	0	1	–	$.6\times210^6$
a*n*	0	1	6	0	1	–	$.6\times210^7$
A*n*	6	0	1	0	0.0324	1	10^8
A*n*	5	0	1	0	0.1	0	$.18\times110^7$
A*n*	4	0	2	0	0.3	0	$.37\times110^7$
A*n*	3	0	3	0	1	–	$1.7510\times^7$
A*n*	2	0	4	0	1	–	$6.7610\times^6$
A*n*	1	0	5	0	1	–	$.6\times210^6$
A*n*	1	0	6	0	1	–	$.6\times210^7$
a*A	1	6	0	0	0.0124	1	10^8
a*A	1	5	0	0	0.1	1	$3.0910\times^7$
Aa*	1	6	0	0	0.0124	1	10^8
Aa*	1	5	0	0	0.1	1	$3.0910\times^7$
Aa*n	1	5	1	0	0.0324	1	10^8
Aa*n	1	4	1	0	0.1	0	$.18\times110^7$
An*	5	0	1	0	0.1	0	$.13\times710^7$
An*	4	0	2	0	0.1	0	$6.8510\times^7$
An*	3	0	3	0	0.284	0	10^8
An*	2	0	4	0	0.3	0	$3.0310\times^7$
[an]*	1	0	5	0	1	0	$.56\times110^7$
[an]*	0	5	1	0	0.1	0	$.13\times710^7$
[an]*	0	4	2	0	0.1	0	$6.8510\times^7$
[an]*	0	3	3	0	0.284	0	10^8
[an]*	0	2	4	0	0.3	0	$3.0310\times^7$
...	0	1	5	0	1	0	$.56\times110^7$
...	6	0	0	1	0.0108	0	10^8
...	5	0	0	1	0.1	0	$.97\times210^7$
...	4	0	0	2	0.1	0	$.7\times110^7$
[as]*	3	0	0	3	1	–	$.38\times410^7$
[as]*	2	0	0	4	1	–	$6.3410\times^6$
[as]*	1	0	0	5	1	–	$.88\times410^5$
[as]*	0	6	0	1	0.0108	0	10^8
[as]*	0	5	0	1	0.1	0	$.97\times210^7$
[as]*	0	4	0	2	0.1	0	$.7\times110^7$

	0	3	0	3	1	–	$.38\times410^{7}$
	0	2	0	4	1	–	$6.3410\times^{6}$
	0	1	0	5	1	–	$.88\times410^{5}$