# 场景：

某服务的密码是大小写数字感叹号星号组合的10字节长的一个东东（密码空间为64的10次方）。Md5后存放。Md5密码库被暴露了，现在需要根据MD5值破解出明文

# 方法一：字典攻击

常见的密码组合有64的10次方种，即2的60次方种，即100亿亿种，每个条目占用26字节（16+10），假设每个机器硬盘2.6T大小，则需要100万台机器存储。**空间复杂度太大，放弃**（实际情况要好很多，因为用户倾向与使用可以预测的弱密码而非完全随机的密码，使得不需要构建如此大的字典也能取得不错的攻击效果）

# 方法二：暴力破解

假设单机每秒完成1亿次md5计算和比对，1万台机器同时运算，每秒钟完成1万亿次计算，则完成100亿亿次计算，需要1千万秒，即需要125天才能破解一个密码。**时间复杂度太高，搞一个高价值密码还好，不适合大批量密码的破解，放弃**

实际情况要好很多，因为用户倾向与使用可以预测的弱密码而非完全随机的密码，使得不需要遍历如此大的密码空间也能取得不错的破解效果。

尤其是密码爆库后，如果存储是密码未加盐的一次或者多次md5，黑客可以对一个可能的密码值计算密文，然后用密文到存储库里面去查，如果查到，就知道了该用户的明文密码，这样的破解效率通常很高。

# 方法三：彩虹表

彩虹表是综合了字典攻击和暴力破解的方式，在空间和时间复杂度上取一个平衡。原理如下：

对于md5运算，我们记作H（），对于一个密码，有

对于所有的密码组合，有集合P（大写的P），被H映射到所有的md5值集合Q（大写的Q），记作

## STEP 1：构造R函数

我们构造函数R，实现集合Q到集合P的映射，即，要求R的运算高效、且尽量多的辐射两个集合里的元素。

注意不是要找到函数R满足，因为找出这样的R不可能。

举个例子来说明R可以怎么构造：

可以理解为将一个大小为2^128的集合Q映射到64^10大小的集合P，那么我们把集合Q中的每个元素看作是一个128位的整数，把集合P中的元素看作是一个10位的64进制的整数（Q集合比P集合大很多很多）。对任意的，我们将其对64^10取模，然后结果用64进制表示出来即可（有点像base64编码）。这个R函数对计算机来说是小菜一碟。

## STEP 2 选择合适的n

这个n，是用于折中空间和时间的关键参数，用于将字典攻击的字典缩小n倍，但也会导致每次查表带来的计算更大量。

我们假设n=100万。

## STEP 3 构造长度为n的链表

我们对各种做这样的链式计算：

并将映射关系保存起来，注意只保存上述链表的头尾，我们叫这个表为彩虹表，表里的条目记作。按照前面计算的密码空间大小，约合2.6T大小，即彩虹表的大小。

这个构造过程应该是很长时间的，假设单机每秒完成1亿次md5计算或R映射转换，100台机器同时运算，那么100亿亿大小的密码空间遍历完，**需要2500天，即8年。虽然久远，但也非不可能完成的任务**

实际工作中，彩虹表通常存放在一个一个等大小的文件里，每个文件由多个链表组成，每个链表只保存首尾的p，且按尾端的p排序，方便查找。相同的尾端的p认为是一个链表，去重。

## STEP 4 查彩虹表破解一个密码的过程

对于一个，希望能通过查彩虹表找出对应的，使得满足。

**第一步**：计算得到

**第二步：**从文件中快速查找彩虹表中满足以下条件的条目：

情况一：如果能够找到，那么就从相应的进行链式计算，计算到第n步：

这一步里的就是我们要找的，因为它满足

这样就完成了一个密码的破解。

情况二：如果找不到，我们做如下计算：

从文件中快速查找彩虹表中满足以下条件的条目：

如果能够找到，那么就从相应的进行链式计算，计算到第n-1步：

这时候的就是要找的密码明文，因为他满足：

至此，得到了密码明文。

如果找不到等与 的条目怎么办呢？ 举一反三，计算更长的得到，又查找。。。如此反复。

相比字典攻击，彩虹表的存储缩小到原来的100万分之一，但破解单个密码的运算量大了近亿倍（字典攻击查表法，时间复杂度是O(1) ）。

相比暴力破解，彩虹表的有一定的存储量，但破解单个密码的运算量降低了近亿倍。

## 实战中的问题

**问题一：链表交叉并合并的问题。**通常明文的空间会比密文的空间要小，否则摘要算法就选择不当。那么R函数在将密文映射到明文的时候，容易出现多个密文映射出同一个明文的情况，将导致链表交叉并合并，使得存储和构建链表的计算工作有效性降低。

例如下面的例子，p表示明文，c表示密文，有这样两个链，每个链都有3个环节：

因为和都映射到**，**使得这两个链表重合起来。在链表的位置约靠前，越倒霉，因为后面重合的越多。由于这样的在两个链表里的位置还不一样，通常这两个链表的尾端p不相同，也无法通过排序去重发现。就像下图示意的



实际运用中，可以采取多个R函数的方式， n的长度同链表的长度，每个环节的映射对应一个R函数。那即使出现了上面的这样的交叉点，那后面的环节是不重合的，即链表不会大量环节重合，只会发生相交。就像下图示意的



假设都在第三个环节出现了相同的p，这样还是会导致后面大量的重合，但是最后的尾端p也相同，在彩虹表文件内部的排序和去重处理过程中，会去掉其中一条链表。就像下图示意的：



问题二：**明文-密文对覆盖不全的问题。**对于比较大的明文空间（例如1-10位的由英文字母和数字构成的密码），要做到100%的覆盖和破解率，应该说不可能，通常在95%就比较完美了。构建好的R函数，尽量覆盖到足够多的明文，是提高覆盖率的一个方法。

另外就是通过冗余的方式，例如用不同的R函数（如果链表每个环节用一个不同的R函数，那就应该说是R函数群了，见问题一）构建多个彩虹表。

同一个彩虹表，里面一定有很多链表中间的节点是重合的，这些是无效的存储和计算。更别提采用不同R函数的多个彩虹表了。但是从另外一个方面看，这些是有意义的，它使得尽量多的覆盖到明文-密文对，提高破解率。

就像网站<http://project-rainbowcrack.com/index.htm里提供的rtgen>工具，他的参数table index和part index，就相当于对R函数的微调，构建出不同的彩虹表文件，提高覆盖率。

构建这样一个测试：

破解4位长度的由数字组成的密码。使用如下命令构造彩虹表：

./rtgen md5 numeric 4 4 0 100 100 0

理论上看， 100条链表，每条链表100个环，如果R函数足够完美，就能覆盖所有密码空间（10000个密码），但实际上这样构建出来的彩虹表，破解率非常低（密码1234的摘要都破解不了），需要调整table index或者part index，多产生几个彩虹表文件。

./rtgen md5 numeric 4 4 1 100 100 0

./rtgen md5 numeric 4 4 2 100 100 0

./rtgen md5 numeric 4 4 3 100 100 0

## 应对办法

**方法一：密码存储加盐。**

简单的说，就是每个用户有一个不同的值，加入到密码的摘要运算里。例如md5( md5(passwd)+salt)。这样，黑客无法用一个预先算好的彩虹表或者字典对所有用户进行破解，必须为每个用户构建一个彩虹表或者字典。

当然，密码存储加盐，对于暴力破解也是提高了门槛。试想一下，如果存储是未加盐的一次或者多次md5，黑客可以对一个可能的密码值计算密文，然后用密文到存储库里面去查，如果查到，就知道了该用户的明文密码，这样的破解效率通常很高。存储加盐后，黑客无法在未知用户ID和盐的情况下计算密文，也就无法到库里反查。

但是这个方法对于破解单个用户的密码，没有提升什么门槛。

**方法二：key stretching(或者叫key strengthening)，密钥加强。**

基本思路就是让每次鉴权的时间复杂度大到刚刚好，使得正常用户的体验能够接受，但是黑客暴力破解或者构建彩虹表的成本大幅度提高。

例如，后台密码存储是 A=md5( md5(….md5( md5(passwd)+salt)))…) (n次md5计算)，当下普通的计算机进行上述运算假设需要1秒的处理时间。

验证过程中，要求客户端提交A作为key加密（用户ID、B、时间戳等等），而B= md5( md5(….md5( md5(passwd)+salt)))…) (n-1次md5计算)。

服务端收到客户端提交后，尝试用A解密，如果成功，并校验用户ID、时间戳等等字段，并且校验md5(B)=A，这样保证密码存储被宝库后不能伪造登录。因为黑客只获得了A，没有B，且A不能推导B，所以不能伪造登录

服务端验证通过，视情况可能保持存储的A不变，也可能将A再做一次md5运算的到AA=md5(A)， 删除A，存储AA。这样的目的是为了对冲计算机硬件增强导致的黑客门槛降低。

如果服务端需要对现有的简单密码存储方案（例如存储的密码一次md5）进行全面升级，那么必须进行一次高密度的计算。假设有10亿用户，投入1000台服务器，每台服务器每秒钟可以计算一个加强后的存储。那么需要300小时，即13天。另外，服务器端需要为用户改密投入较多的机器，假设改密请求峰值每秒1000次，那么需要投入1000台机器用于改密的处理。如果发生突发事件，需要大规模的用户突击改密，这个时候建议降低计算复杂度，先把密码存起来再说，等请求高峰过后，对这些用户的复杂度进行提升。

密钥加强的标准算法有PBKDF2、bcrypt、scrypt。他们在常见的语言和算法库都有提供，例如openssl、perl等等。

对于黑客构建彩虹表或者暴力破解门槛的提升，假设原来进行一次验证只需要十万分之一秒，而现在需要1s，那么门槛提升了10万倍。更关键的是，这个库随着时间推移，能够对冲硬件能力提升给黑客带来的便利。

**方法三：密码变长** 让密码变长，也是非常有效的办法。可能有个误区，认为加一些@#!稀奇古怪甚至中文字符作为密码会让密码变得更安全，实际上，对于计算机来说，都是byte，差别不会特别大,对于避免人工猜会有些帮助。主要还是长度！

## 硬件技术

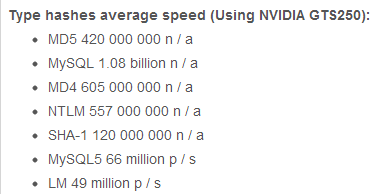
很明显，强大的硬件技术，有利于暴力破解和字典/彩虹表的构建。在搜索引擎里输入”FPGA MD5 crack”或者”GPU MD5 crack”可获得大量信息。

**FPGA** 现场可编程逻辑阵列(FPGA)是用于数字电路设计与逻辑验证的大规模集成电路芯片，常用于专用集成电路(ASIC)的原型设计与算法的开发验证。使用FPGA,可以迅速地将算法实现成数字电路并进行功能验证。MD5和sha-2在FPGA中已经早有实现，相当于将软件直接用硬件实现，一般速度会提升许多。利用FPGA构建多块摘要运算电路，插入到电脑的pci接口，每个FPGA电路相当于一个电脑。这样大大的提升了运算能力。

根据网上的资料：NSA@home is a fast FPGA-based SHA-1 and MD5 bruteforce cracker. It is capable of searching the full 8-character keyspace (from a 64-character set) in about a day in the current configuration for 800 hashes concurrently, using about 240W of power. This performance is equivalent to over 1500 Athlon FX-60 CPUs, which would take about 250kW.

从上面的数据可以算到：大概相当于一台pc电脑的功耗，每秒钟可进行30亿次的8位密码尝试。而我的台式机（2013年）每秒可进行大约400万次。确实近千倍的提升

**GPU** GPU的计算能力比CPU每个核大约快一个数量级，且核数众多，通常还能在主板上平行扩展多个GPU。同时nVIdia推出的CUDA编程框架，使得GPU编程学习掌握门槛大大降低。根据网上的资料显示一个32核的GPU的运算能力：



也就是说每秒可进行420M次密码md5计算，比我的台式机快100倍。针对8位每位64个字符的密码，大概10天即可完全遍历该密码空间。

我自己测试，sha256的计算过程相比md5，没有慢一倍。我的机器同样进行10M次摘要， MD5用了2.4s， sha256用了4.1s。

