



Rainbow Tables & RainbowCrack Introduction

Rainbow tables reduce the difficulty in brute force cracking a single password by creating a large pre-generated data set of hashes from nearly every possible password. Rainbow Tables and RainbowCrack come from the work and subsequent paper by Philippe Oechslin.¹ The method, known as the Faster Time-Memory Trade-Off Technique, is based on research by Martin Hellman & Ronald Rivest done in the early 1980's on the performance trade-offs between processing time and the memory needed for cryptanalysis. In his paper published in 2003, Oechslin refined the techniques and showed that the attack could reduce the time to attack 99.9% of Microsoft's LAN Manager passwords (alpha characters only) to 13.6 seconds from 101 seconds. Further algorithm refinements also reduced the number of false positives produced by the system.

The main benefit of Rainbow Tables is that while the actual creation of the rainbow tables takes **much** more time than cracking a single hash, after they are generated you can use the tables over and over again. Additionally, once you have generated the Rainbow Tables, RainbowCrack is faster than brute force attacks and needs less memory than full dictionary attacks.

Rainbow Tables are popular with a particularly weak password algorithm known as Microsoft LM hash. LM stands for LAN Manager, this password algorithm was used in earlier days of Windows and still lives on only for compatibility reasons. By default Windows XP or even Windows Server 2003 keeps the LM hash of your passwords in addition to a more secure hash (NTLM or NTLMv2). This allows for the benefit of backwards compatibility with older operating systems on your network but unfortunately makes the job of password cracking easier if you can obtain the LM hashes instead of the NTLM hashes.

Microsoft's LAN Manager algorithm and its weaknesses

LM is weak for several reasons. First, it's not case-sensitive (it converts everything to uppercase), which significantly reduces password search space. This means that even if my password is "PaSsWoRd" LM converts it to PASSWORD.

Consider the number of possible seven character passwords made exclusively from letters, i.e., no numbers, symbols, etc. For a case-insensitive algorithm (like LAN Manager) this means there's a total of 26 possible values for each character position in the password. Here's how a case-insensitive algorithm's search space compares to one for a case-sensitive algorithm:

Number of possible characters	Number of possible seven character passwords
26	26^7 (8,031,810,176 or about 8.031^9)
52	52^7 (1,028,071,702,528 or about 1.028^{13})

So if you have the same CPU, the same memory, and all other factors being equal; the doubling of possible values in the character space will result in, not the doubling of passwords to crack, but an increase a couple orders of magnitude in the number of passwords to crack.

So if we assume that LM can use every character on a standard US keyboard it has a character set of 69 possible characters.

¹ http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Oech03



Count	Class
26	all letters (upper and lower combined)
10	digits
10	shifted symbols from digit keys
22	remaining symbols and their shifted counterparts
1	space
<hr/>	
69	total

Case-sensitive algorithms (e.g., the password hashing algorithm on most Unix variants, and Microsoft's NTLM and NTLMv2) give us an additional 26 characters per position, or a character set of 95 possibilities.

To calculate the total number of possible passwords for a given algorithm, you add the total number of passwords for each valid password length (or at least each valid password length for the password space you're searching). LM's maximum effective password length is seven characters for a total search space of 7,555,858,447,479 possible passwords (about 7.556^{12}).

# characters in password	Number of possible passwords
1	69 (69^1)
2	4,761 (69^2)
3	328,509 (69^3)
4	22,667,121 (69^4)
5	1,564,031,349 (69^5)
6	107,918,163,081 (69^6)
7	7,446,353,252,589 (69^7)
<hr/>	
7,555,858,447,479 total	

A case-sensitive algorithm would give us an additional 26 characters, for a total of 95. With the larger character set (but sticking with the seven character password limit for the moment), the total search space is 70,576,641,626,495 possibilities (about 7.058^{13}).

# characters in password	Number of possible passwords
1	95 (95^1)
2	9,025 (95^2)



3	857,375 (95^3)
4	81,450,625 (95^4)
5	7,737,809,375 (95^5)
6	7,350,918,906,25 (95^6)
7	69,833,729,609,375 (95^7)

70,576,641,626,495 total

So, an algorithm that's case-sensitive has a password search space about nine times larger than Microsoft LAN Manager's

A second problem with LM is that it's effectively limited to only seven characters. LM can accept passwords up to 14 characters in length and treats any values in the eighth through fourteenth character position as a second, completely independent password. (Characters beyond the 14th are ignored.) For example, given a password of nine characters, LM hashes password's first seven characters and stores the result, then hashes the remaining two characters and stores them as well.

When attacking LM, an attacker can immediately tell which accounts have seven characters or less because they will all share the same value for the second half of the password. While it's true that passwords of 8-14 characters effectively require examination of two separate hash values, this is a relatively cheap operation and not one which has great impact on speed because the password is so short.

Does adding an eighth character *really* make that much of a difference? The short answer: absolutely! If LM could actually hash eight character passwords (as opposed to the effective seven character limit imposed by its design), its total search space would be noticeably larger, as shown here:

# characters in password	Number of possible passwords
1	69 (69^1)
2	4,761 (69^2)
3	328,509 (69^3)
4	22,667,121 (69^4)
5	1,564,031,349 (69^5)
6	107,918,163,081 (69^6)
7	7,446,353,252,589 (69^7)
8	513,798,374,428,641 (69^8)

521,354,232,876,120 total
(or about 5.214^{14})



5.214^{14} really *is* larger than 7.556^{12} . The additional character increases total search space by about a factor of about 69, a notable difference. Case sensitivity becomes even more important when you use algorithms capable of handling passwords with more than seven characters.

# characters in password	Number of possible passwords
1	95 (95^1)
2	9,025 (95^2)
3	857,375 (95^3)
4	81,450,625 (95^4)
5	7,737,809,375 (95^5)
6	7,350,918,906,25 (95^6)
7	69,833,729,609,375 (95^7)
8	6,634,204,312,890,625 (95^8)
<hr/>	
	6,704,038,042,500,000 total (or about 6.704^{13})

Modern password hashing algorithms (to include NTLM and NTLMv2) can use more than eight characters, so the search space involved rapidly becomes—to put it mildly—**VERY LARGE**. The maximum length for Windows passwords depends on the OS version. Windows 2000, Windows XP, and Windows Server 2003 support passwords up to 128 characters, but older versions of Windows (98, ME, NT) only support passwords up to 14 characters

One of the characteristics of strong authentication algorithms is that they're computationally expensive; meaning that trying to iterate through them a large number of times (like when someone's grinding through a dictionary) is supposed to take a while. LM's third weakness is that it's computationally cheap, at least compared with other password hashing algorithms. Here's some password calculation rates for several algorithms as implemented in John the Ripper (a well known password "cracker") on a low-end Athlon64 and a 3.2GHz Xeon. Note that this instance of John has been patched to be able to handle "NT MD4" (NTLM):

Hashing algorithm	"crypts/sec"	
	Athlon64 2800+	3.2GHz Xeon
DES, one salt	569,313	319,960
FreeBSD MD5	4,079	8,950
OpenBSD Blowfish	292	448
NT MD4 (NTLM)	1,101,000	991,817
NT LM DES (LANMAN)	5,180,000	4,524,000



DES (the old Unix `crypt()` function) is only a tenth as fast as LM (remembering that this is one area where speed is **BAD**). OpenBSD's Blowfish-based hashing algorithm gets the tortoise prize; its computational expense makes it a very unappetizing target for brute-force attacks. What these numbers mean is that I get 4,524,000 guesses a second against a LAN Manager password and I only get 319,960 guess a second against a DES password; a huge difference.

Finally, LM uses no salt. A salt is essentially a random value tossed into the computational mix when generating a password hash. The idea behind using a salt is it randomly changes the output of the hash function, making it much more difficult to simply compare hashes to determine if passwords are identical. If two users happen to pick the same password, an algorithm that uses salts will make it harder for an attacker to detect that because the hashes would be different.

Now that we know something about algorithms, their speeds, etc., we can figure out real-world attack speeds.

We've already determined the total number of possible passwords with LM is something like 7.556^{12} . We also know that a 3.2 GHz Xeon can go through these possibilities at a rate of around 4,524⁶/second. A rough estimate puts the time needed to chew through all the possibilities (using the 3.2GHz Xeon benchmarked above) in about 1.7⁶ seconds. A possibly more accurate estimate is:

$$(7,555,858,447,479 \text{ passwords}) / (4,524,000 \text{ passwords/sec}) = 1670203 \text{ seconds}$$

That's around 19 days to work through all possible passwords given LM's available search space. However, since the odds are that an attacker will guess your password within the first half of all possibilities (assuming a completely random search, even distribution, etc.), the half-life (about eight days) is a better number to use for planning a password's usable lifetime.

Here's how LM stacks up against the old Unix `crypt()` function and NTLM, using the timings and password spaces shown above (and limiting the other algorithms to a maximum of eight characters, even if they might be able to use more):

Algorithm	Size of password space	Half-life length in seconds
LANMAN	7.556^{12}	1.459^6 (~8 days)
NTLM	6.704^{15}	3.045^9 (~95 years)
<code>crypt()</code>	6.704^{15}	5.888^9 (~185 years)
*FreeBSD MD5	6.704^{15}	7.491^{11} (>11,000 years)

Most users change their passwords often enough that `crypt()` even though it's definitely showing its age, still provides quite a bit of protection against brute-force attacks (185 years for an 8 character password). It's also pretty clear that a smaller password search space (8 or less characters) plays a **BIG** role in successful attacks (8-9 days).

So these are realistic numbers for tools like John the Ripper and L0phtcrack (LC4/LC5) lets see how RainbowCrack stacks up against brute force and dictionary attack tools like John the Ripper and L0phtcrack.



Using Rainbow tables isn't password guessing; it's a database lookup. Like we already mentioned, the idea is that an attacker only has to compute hashes once (or buy a copy of them pre-computed), rather than attacking passwords through computational power.

For example:

LM configuration #0

charset	[ABCDEFGHIJKLMNOPQRSTUVWXYZ]
keyspace	8353082582
table size	610 MB
success probability	0.9990

Has a success probability of 99.90% and only takes up 610 MB.

LM configuration #1

charset	[ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]
keyspace	80603140212
table size	3 GB
success probability	0.9904

Has a success probability of 99.04% and takes up 3 GB.

LM configuration #5

charset	[ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#\$%^&*()-_+=]
keyspace	915358891407 ($2^{39.7}$)
table size	24 GB
success probability	0.99909

Has a success probability of 99.1% and takes up 24 GB. This is starting to get large but 1) not THAT large with as cheap as hard drive space is and 2) with the character set involved. Don't forget this will work on passwords up to 14 characters as well. What starts to make a difference is how long it takes to compute these tables.

LM configuration #6

charset	[ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#\$%^&*()-_+=~`[]{} ;:'"<>.,?/]
keyspace	7555858447479 ($2^{42.8}$)
table size	64 GB
success probability	0.999

Has a success probability of 99.9% and takes up 64 GB. This character set includes all possible characters on a standard keyboard (not including alt+xxx characters). So this table set is likely to crack any windows password up to 14 characters in minutes. This is great but on one computer it will take about 2 years to generate these tables (Faster Time-Memory Trade-Off Technique).



You can see demos of some of these configurations in action at the Project RainbowCrack website²

Using Rainbow Tables & RainbowCrack

Example 1:

First download RainbowCrack for your platform from www.antsight.com/zsl/rainbowcrack/.

We will use our LM alpha (configuration 0) rainbow tables.

** You will need to either create them or unzip³ them and they will look something like:

```
128,000,000 bytes  lm_alpha#1-7_0_2100x8000000_all.rt
128,000,000 bytes  lm_alpha#1-7_1_2100x8000000_all.rt
128,000,000 bytes  lm_alpha#1-7_2_2100x8000000_all.rt
128,000,000 bytes  lm_alpha#1-7_3_2100x8000000_all.rt
128,000,000 bytes  lm_alpha#1-7_4_2100x8000000_all.rt
```

If everything goes well, backup all files (recommended especially if you just made them and didn't download them) and then get ready to sort them.

To speed up the search of our rainbow table, we should sort the rainbow table with "rtsort.exe" in advance. In fact "rcrack.exe" only accepts sorted rainbow tables.

We sort the rainbow tables by using the following command:

Use these commands:

```
rtsort lm_alpha#1-7_0_2100x8000000_all.rt
rtsort lm_alpha#1-7_1_2100x8000000_all.rt
rtsort lm_alpha#1-7_2_2100x8000000_all.rt
rtsort lm_alpha#1-7_3_2100x8000000_all.rt
rtsort lm_alpha#1-7_4_2100x8000000_all.rt
```

Each command will take several minutes to complete. The "rtsort.exe" utility will sort the file and write back to the original file.

Notice: If free memory size is smaller than the file size, we can't load the file into memory at a time. In which case extra free disk space as large as the file to be sorted is required to apply an external sort.

Once rtsort has completed you are ready to use rcrack against some hashes.

To see available options just type "rcrack"

```
C:\rainbowcrack-1.2-win\rainbowcrack-1.2-win>rcrack
RainbowCrack 1.2 - Making a Faster Cryptanalytic Time-Memory Trade-Off
by Zhu Shuanglei <shuanglei@hotmail.com>
http://www.antsight.com/zsl/rainbowcrack/
```

```
usage: rcrack rainbow_table_pathname -h hash
       rcrack rainbow_table_pathname -l hash_list_file
       rcrack rainbow_table_pathname -f pwddump_file
rainbow_table_pathname: pathname of the rainbow table(s), wildchar(*, ?) supported
```

² Project RainbowCrack website www.antsight.com/zsl/rainbowcrack/

³ Free Rainbow tables for download via torrent <http://rainbowtables.shmoo.com/>



```
-h hash:                use raw hash as input
-l hash_list_file:      use hash list file as input, each hash in a line
-f pwddump_file:        use pwddump file as input, this will handle LAN Manager hashes only

example: rcrack *.rt -h 5d41402abc4b2a76b9719d911017c592
         rcrack *.rt -l hash.txt
         rcrack *.rt -f hash.txt
```

Launch the program by issuing the command:

```
rcrack c:\rainbowcrack\*.rt -l hashlist.txt
```

You should replace "c:\rainbowcrack\" with where you placed your sorted rainbow tables.

To crack some hashed windows passwords, the syntax is similar:

```
rcrack c:\rainbowcrack\*.rt -f pwddumpfile.txt
rcrack c:\rainbowcrack\*.rt -l justhashlist.txt
rcrack c:\rainbowcrack\*.rt -h 213D466DB5B288F0F82E44EC0938F4F4
```

Where pwddumpfile.txt is the results of using a hash dumping utility like pwddump2, pwddump3, samdump, etc to dump the LAN Manager's passwords.

If your password consists of only letters only, rcrack should be able to crack it with a success rate of 99.9%.

Let's try it against the following hash file in pwddump format (so use the -f option):

```
testuser1:"":0F20048EFC645D0A179B4D5D6690BDF3:1120ACB74670C7DD46F1D3F5038A5CE8:::
remote:"":E52CAC67419A9A224A3B108F3FA6CB6D:8846F7EAE8FB117AD06BDD830B7586C:::
joeuser:"":E52CAC67419A9A224A3B108F3FA6CB6D:8846F7EAE8FB117AD06BDD830B7586C:::
averageguy:"":299CCF964D9A359BAAD3B435B51404EE:A5C07214487C87B584E8877DE72DCA0B:::
harderpass:"":B75838F7A57EE67993E28745B8BF4BA6:EC50F8A8149C93EF45AECB8AF96658E6:::
demouser:"":261A6631FE44BA4993E28745B8BF4BA6:371D5760453C1B000BCC016F8E23A83C:::
randy:"":98B5AFEB67293D6AAD3B435B51404EE:A9F34664151F6360757B31644F37E025:::
Asmith:"":E165F0192EF85EBBAAD3B435B51404EE:E4EBE0E7EF708DC9FD240135D3D43D89:::
Bsmith:"":136A8418CF76C4F7AAD3B435B51404EE:3431E75AD08DCA56EB53AEAAB9926589:::
Csmith:"":BB26C063532826AA531C3383FDDBF2A:A2746ED4129985C0251D2B968C4889FE:::
Dsmith:"":A8EED815A197BD87AAD3B435B51404EE:F09A31889C35B8C9746B8F31FC3A868F:::
Esmith:"":5A9DB9F8BB5DF0CBAAD3B435B51404EE:5FCC20A69EC76AD91214102B4D7DE24E:::
Fsmith:"":213D466DB5B288F0F82E44EC0938F4F4:FAF10460760FA3F1ED804C7C724CB3D4:::
Gsmith:"":385A83A746BFA8F2AAD3B435B51404EE:1CC1B3958B564125D307BA8D9D60DF69:::
Hsmith:"":78BCCAEE08C90E29AAD3B435B51404EE:972E8E7D5568F70AC896B2C76E1395DC:::
Jsmith:"":59E2DB85E9D49595B75E0C8D76954A50:147D125645D463C33D72309525E9B0BC:::
Ksmith:"":59E2DB85E9D49595B75E0C8D76954A50:147D125645D463C33D72309525E9B0BC:::
Lsmith:"":13D855FC4841C7B1AAD3B435B51404EE:3DCEBC92C0ED8F52B1D759DD35CF3F0F:::
Msmith:"":D71808BF36F81510ADEE49688244F15A:45E8DA896575E2F5455B037FCC5AA51A:::
Nsmith:"":9C92FA4960AC2536AAD3B435B51404EE:C318744C4291EA46BC65082636CC9509:::
Osmith:"":1153C3961EE58C3BAAD3B435B51404EE:672532E8C0C490BD47254DAED1CDCB36:::
Psmith:"":4A01C0E45FCA767AAAD3B435B51404EE:39981702716E054CBE6840A3CFD60327:::
Qsmith:"":6842A19CC4C509E0AAD3B435B51404EE:9FDA95FD6FCEE9C2C998CB8010F61F16:::
Rsmith:"":BC472F3BF9A0A5F63832C92FC614B7D1:D2A80A79980CFA21CB58B7CB129E2CAD:::
Ssmith:"":09755C01D2789BD8AAD3B435B51404EE:62F740C2EA31E10B54DB64CE12E867A6:::
Tsmith:"":13D855FC4841C7B1AAD3B435B51404EE:3DCEBC92C0ED8F52B1D759DD35CF3F0F:::
Usmith:"":9E2204E2058AC9E9417EAF50CFAC29C3:476541DEC5CB507A795FC1E989C9D36F:::
Vsmith:"":7F9CD2D7C93421D3F9DE51FBDA2F725:16FAABB24B95B82EFC50B074B7324517:::
Wsmith:"":AC814111DF804A7482EFD6B2A69511D6:15B194EB8D8F27761E32F76B001553A0:::
Xsmith:"":AAD3B435B51404EEAAD3B435B51404EE:2321504F2FA9437FBBA66EA1623407D3:::
```




```
Ysmith:"":D5662E6B23655BF74EC0DA4207C2DE66:75344B75B5A96614FE179C0188A9634A:::
Zsmith:"":9224FC255C58C50E42B35806901777E7:0C105C9F4326C3AC100C2A5B7A04AD38:::
```

The Answers so you can check your work.

testuser1	testuser1	(2)	Ksmith	ABCdef123	(2)
remote	password	(2)	Lsmith	ABCdef	(1)
joeuser	password	(2)	Msmith	FOOTBALL!@#	(2)
averageguy	average	(1)	Nsmith	SOCCER	(1)
harderpass	rootwars	(2)	Osmith	CROKET	(1)
demouser	demopass	(2)	Psmith	COW123	(1)
randy	randy	(1)	Qsmith	HOWNOW	(1)
Asmith	ABCD	(1)	Rsmith	BROWNCOW	(2)
Bsmith	ef456	(1)	Ssmith	gHaNdI	(1)
Csmith	ABC789!@#12	(2)	Tsmith	ABCdef	(1)
Dsmith	3!@#	(1)	Usmith	RTdotnet	(2)
Esmith	456!@#	(1)	Vsmith	!pa55word!	(2)
Fsmith	ABCdef!@#	(2)	Wsmith	EASYoneISNTit	(2)
Gsmith	gHgHgH	(1)	Xsmith	C@NTcR8ckm3CanU?	(X)no LM
Hsmith	ABC123	(1)	Ysmith	LSOISDABEST	(2)
Jsmith	ABCdef123^	(2)	Zsmith	RAINBOWTABLEZ	(2)

**32 users and 47 LM hashes 48 Total hashes. Xsmith will only be saved as NTLM because it's greater than 14 characters.

You should see something similar to the following:

```

C:\Documents and Settings\NoOne\Desktop\CLI\rainbowcrack-1.2-win\rainbowcrack-1.2-win>rcrack d:\torrents\lm_alpha*.rt -f hashes4rainbowlab.txt
lm_alpha#1-7_0_2100x8000000_all.rt:
128000000 bytes read, disk access time: 6.01 s
verifying the file...
searching for 41 hashes...
plaintext of e52cac67419a9a22 is PASSWOR
plaintext of 4a3b108f3fa6cb6d is D
plaintext of b75838f7a57ee679 is ROOTWAR
plaintext of 93e28745b8bf4ba6 is S
plaintext of 261a6631fe44ba49 is DEMOPAS
plaintext of 98b5afeb67293d6a is RANDY
plaintext of e165f0192ef85ebb is ABCD
plaintext of 385a83a746bfa8f2 is GHGHH
plaintext of 13d855fc4841c7b1 is ABCDEF
plaintext of d71808bf36f81510 is FOOTBAL
plaintext of 9c92fa4960ac2536 is SOCCER
plaintext of 1153c3961ee58c3b is CROKET
plaintext of 6842a19cc4c509e0 is HOWNOW
plaintext of bc472f3bf9a0a5f6 is BROWNCOW
plaintext of 09755c01d2789bd8 is GHANDI
plaintext of 417eaf50cfac29c3 is T
  
```

Figure 1.1: Rcrack at work with an lm_alpha rainbow table



```

C:\> Command Prompt

statistics
-----
plaintext found:      26 of 41 (63.41%)
total disk access time: 62.51 s
total cryptanalysis time: 742.77 s
total chain walk step: 203410183
total false alarm:    195135
total chain walk step due to false alarm: 142852030

result
-----
testuser1    TESTUSE<notfound>  hex:54455354555345<notfound>
remote       password  hex:70617373776f7264
joeuser      password  hex:70617373776f7264
averageguy   average  hex:61766572616765
harderpass   rootwars  hex:726f6f7477617273
demouser     demopass  hex:64656d6f70617373
randy        randy     hex:72616e6479
Asmith       ABCd     hex:41424364
Bsmith       <notfound> hex:<notfound>
Csmith       <notfound><notfound> hex:<notfound><notfound>
Dsmith       <notfound> hex:<notfound>
Esmith       <notfound> hex:<notfound>
Fsmith       <notfound><notfound> hex:<notfound><notfound>
Gsmith       gHgHgH  hex:674867486748

```

Figure 1.2: The results of our cracking attempt. 26 of our 41 hashes found in about 12 minutes. Also notice that the hash for the password “password” is the same because there is no salting with the LAN Manager hashing algorithm.

```

statistics
-----
plaintext found:      26 of 41 (63.41%)
total disk access time: 62.51 s
total cryptanalysis time: 742.77s
total chain walk step: 203410183
total false alarm:    195135
total chain walk step due to false alarm: 142852030

result
-----
testuser1    TESTUSE<notfound>  hex:54455354555345<notfound>
remote       password  hex:70617373776f7264
joeuser      password  hex:70617373776f7264
averageguy   average  hex:61766572616765
harderpass   rootwars  hex:726f6f7477617273
demouser     demopass  hex:64656d6f70617373
randy        randy     hex:72616e6479
Asmith       ABCd     hex:41424364
Bsmith       <notfound> hex:<notfound>
Csmith       <notfound><notfound> hex:<notfound><notfound>
Dsmith       <notfound> hex:<notfound>
Esmith       <notfound> hex:<notfound>
Fsmith       <notfound><notfound> hex:<notfound><notfound>
Gsmith       gHgHgH  hex:674867486748
Hsmith       <notfound> hex:<notfound>
Jsmith       <notfound><notfound> hex:<notfound><notfound>
Ksmith       <notfound><notfound> hex:<notfound><notfound>
Lsmith       ABCdef  hex:414243646566
Msmith       FOOTBAL<notfound> hex:464f4f5442414c<notfound>
Nsmith       SOCCER  hex:534f43434552
Osmith       CROKET  hex:43524f4b4554
Psmith       <notfound> hex:<notfound>

```



```

Qsmith      HOWNOW  hex:484f574e4f57
Rsmith      BROWNCOW hex:42524f574e434f57
Ssmith      gHaNdI  hex:6748614e6449
Tsmith      ABCdef  hex:414243646566
Usmith      RTdotnet hex:5254646f746e6574
Vsmith      <notfound><notfound> hex:<notfound><notfound>
Wsmith      EASYoneISNTit hex:454153596f6e6549534e546974
Xsmith      hex:
Ysmith      LSOISDABEST hex:4c534f4953444142455354
Zsmith      RAINBOWTABLEZ hex:5241494e424f575441424c455a

```

Example 2:

We are going to build our own tables using Configuration #1

**Note if you built your configuration #0 tables using rtgen use winrtgen (see exercise 4)

configuration #1	
hash algorithm	lm
charset	alpha-numeric(ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789)
plaintext length range	1 - 7
key space	$36^1 + 36^2 + 36^3 + 36^4 + 36^5 + 36^6 + 36^7 = 80603140212$
t	2400
m	40000000
l	5
disk usage	$m * 16 * l = 3200000000 \text{ B} = 3 \text{ GB}$
success rate	0.9904
mean cryptanalysis time	7.6276 s
mean cryptanalysis time on a low memory system (free memory size much smaller than 610MB)	13.3075 s
max cryptanalysis time	40.6780 s

Table pre-computation commands:

```

rtgen lm alpha-numeric 1 7 0 2400 40000000 all
rtgen lm alpha-numeric 1 7 1 2400 40000000 all
rtgen lm alpha-numeric 1 7 2 2400 40000000 all
rtgen lm alpha-numeric 1 7 3 2400 40000000 all
rtgen lm alpha-numeric 1 7 4 2400 40000000 all

```

On a 666 Mhz machine the table pre-computation time is about 15 days 17 hours, my P4 3.2 GHz with 1GB of RAM I created a table a day; so about 5 days.



```

C:\WINDOWS\system32\cmd.exe - rtgen lm alpha-numeric 1 7 1 2400 40000000 all

F:\rainbowcrack-1.2-win\rainbowcrack-1.2-win>rtgen lm alpha-numeric 1 7 1 2400 4
0000000 all
hash routine: lm
hash length: 8
plain charset: ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
plain charset in hex: 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 5
4 55 56 57 58 59 5a 30 31 32 33 34 35 36 37 38 39
plain length range: 1 - 7
plain charset name: alpha-numeric
plain space total: 80603140212
rainbow table index: 1
reduce offset: 65536

generating...
100000 of 40000000 rainbow chains generated (3 m 4 s)
200000 of 40000000 rainbow chains generated (3 m 3 s)
300000 of 40000000 rainbow chains generated (3 m 3 s)
400000 of 40000000 rainbow chains generated (3 m 3 s)
500000 of 40000000 rainbow chains generated (3 m 3 s)
600000 of 40000000 rainbow chains generated (3 m 3 s)
700000 of 40000000 rainbow chains generated (3 m 31 s)
800000 of 40000000 rainbow chains generated (4 m 37 s)
900000 of 40000000 rainbow chains generated (4 m 36 s)
1000000 of 40000000 rainbow chains generated (3 m 45 s)

```

Figure 2.1: Creating our LM alpha-numeric rainbow tables.

Now run that table against the same hash file, don't forget to sort them first. You should crack most, if not all, of the alpha-numeric passwords, as opposed to alpha passwords only from configuration #0.

```

Command Prompt (2)

statistics
-----
plaintext found:      32 of 41 (78.05%)
total disk access time: 233.84 s
total cryptanalysis time: 233.05 s
total chain walk step: 211003249
total false alarm:    104620
total chain walk step due to false alarm: 91734872

result
-----
testuser1      testuser1  hex:746573747573657231
remote         password  hex:70617373776f7264
joeuser        password  hex:70617373776f7264
averageguy     average  hex:61766572616765
harderpass     rootwars  hex:726f667477617273
demouser       demopass  hex:64656d6670617373
randy          randy     hex:72616e6479
Asmith         ABCd     hex:41424364
Bsmith         ef456    hex:6566343536
csmith         <notfound> <notfound> hex:<notfound><notfound>
Dsmith         <notfound> hex:<notfound>
Esmith         <notfound> hex:<notfound>
Fsmith         <notfound><notfound> hex:<notfound><notfound>
Gsmith         gHgHgH   hex:674867486748
Hsmith         ABC123   hex:414243313233

```

Figure 2.2: The results of our attempts. 32 of 41 passwords were found. Note that I ran this on my 3.2 GHz machine because I created the tables on it and didn't want copy 3GB of rainbow tables to the slow computer.

```

statistics
-----
plaintext found:      32 of 41 (78.05%)
total disk access time: 233.84 s
total cryptanalysis time: 233.05 s
total chain walk step: 211003249
total false alarm:    104620

```



total chain walk step due to false alarm: 91734872

result

```
-----
testuser1      testuser1  hex:746573747573657231
remote         password  hex:70617373776f7264
joeuser        password  hex:70617373776f7264
averageguy      average   hex:61766572616765
harderpass      rootwars  hex:726f6f7477617273
demouser        demopass  hex:64656d6f70617373
randy           randy     hex:72616e6479
Asmith          ABCd      hex:41424364
Bsmith          ef456     hex:6566343536
csmith          <notfound><notfound> hex:<notfound><notfound>
Dsmith          <notfound> hex:<notfound>
Esmith          <notfound> hex:<notfound>
Fsmith          <notfound><notfound> hex:<notfound><notfound>
Gsmith          gHgHgH   hex:674867486748
Hsmith          ABC123    hex:414243313233
Jsmith          ABCdef123 hex:414243646566313233
Ksmith          ABCdef123 hex:414243646566313233
Lsmith          ABCdef    hex:414243646566
Msmith          FOOTBAL<notfound> hex:464f4f5442414c<notfound>
Nsmith          SOCCER    hex:534f43434552
Osmith          CROKET    hex:43524f4b4554
Psmith          COW123    hex:434f57313233
Qsmith          HOWNOW    hex:484f574e4f57
Rsmith          BROWNCOW  hex:42524f574e434f57
Ssmith          gHaNdI    hex:6748614e6449
Tsmith          ABCdef    hex:414243646566
Usmith          RTdotnet  hex:5254646f746e6574
Vsmith          <notfound><notfound> hex:<notfound><notfound>
Wsmith          EASYoneISNTit hex:454153596f6e6549534e546974
Xsmith          hex:
Ysmith          LSOISDABEST hex:4c534f4953444142455354
Zsmith          RAINBOWTABLEZ hex:5241494e424f575441424c455a
```

Example 3:

Compare the results of the same hash file with Cain in Brute force mode, John the Ripper, and LC4.

Cain, in brute-force mode with an alpha-numeric character set, says it will take about 10 hours.

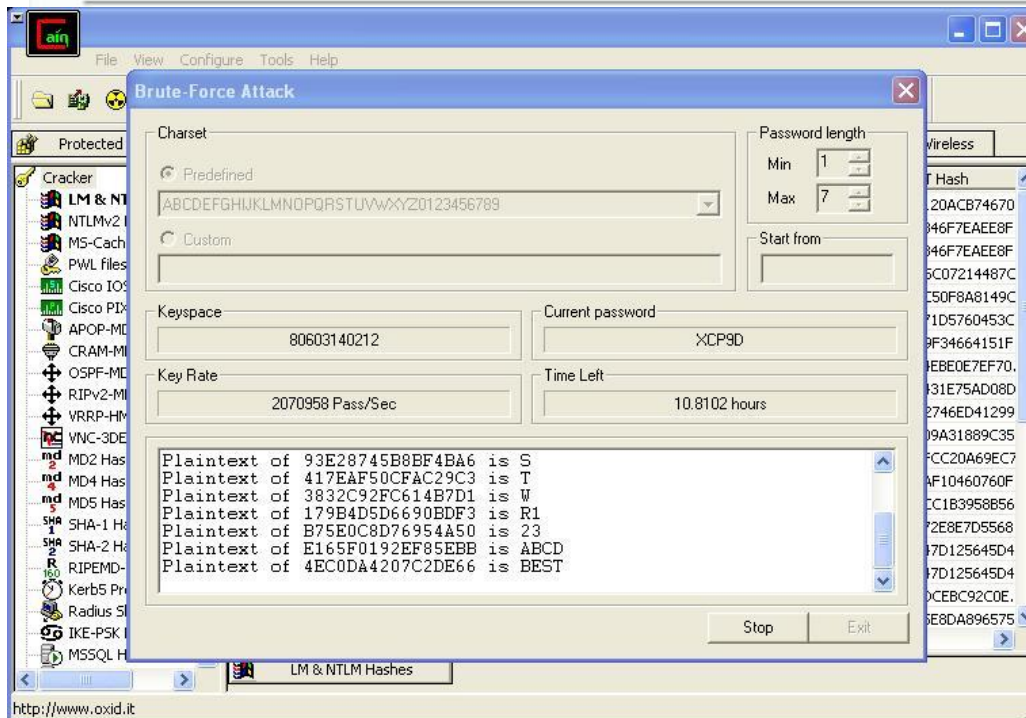


Figure 3.1: Cain in brute-force mode.

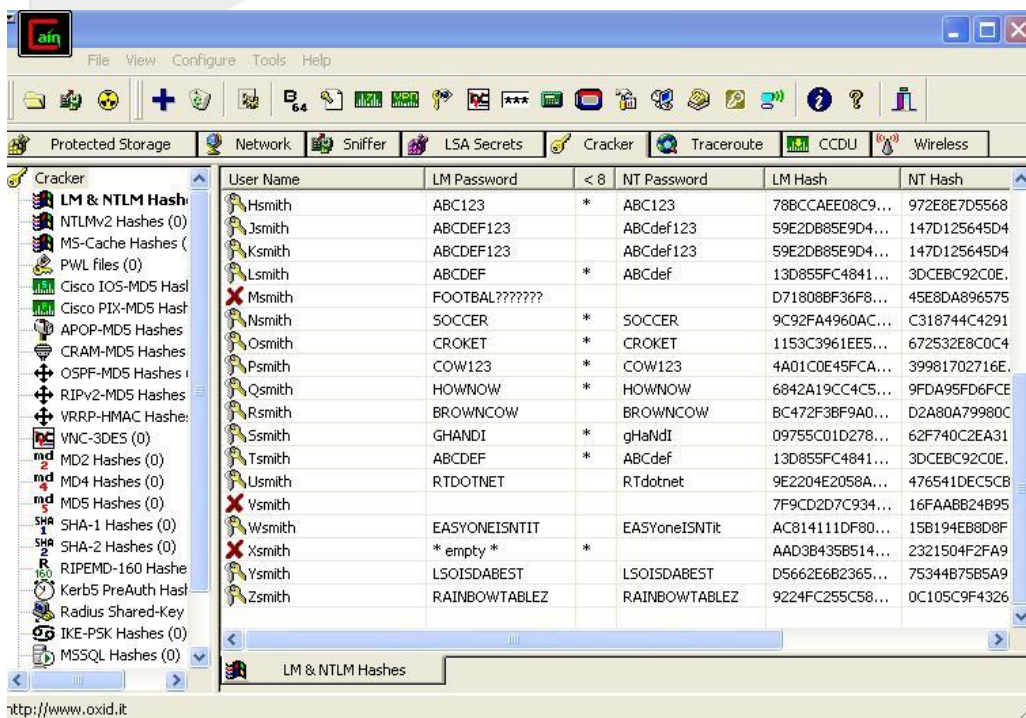


Figure 3.2: After 9+ hours it cracked 27 of the 41 hashes



John the Ripper, in default mode, was able to quickly (about 3 minutes) crack 32 of the 48 hashes.

```

C:\> Command Prompt - john hashes4rainbowlab.txt
Loaded 48 passwords with no different salts (NT LM DES [24/32 4K])
RANDY          (randy)
AVERAGE       (averageguy)
TESTUSE       (testuser1:1)
R1            (testuser1:2)
ABC123        (Hsmith)
PASSWORD      (remote:1)
PASSWORD      (joeuser:1)
SOCCER        (Nsmith)
FOOTBAL       (Msmith:1)
ABCDEF        (Lsmith)
ABCDEF        (Ismith)
RAINBOW       (Zsmith:1)
ABCD          (Asmith)
ABCDEF1       (Jsmith:1)
ABCDEF1       (Ksmith:1)
ABCDEF1       (Psmith:1)
D             (remote:2)
D             (joeuser:2)
T             (Usmith:2)
S             (harderpass:2)
S             (demouser:2)
W             (Rsmith:2)
              (Xsmith)
BEST          (Vsmith:2)
23            (Jsmith:2)
23            (Ksmith:2)
BROWNC0       (Rsmith:1)
Q#            (Psmith:2)
HOWNOW        (Qsmith)
CROKET        (Osmith)
GHANDI        (Ssmith)
RD!           (Usmith:2)
guesses: 32   time: 0:00:02:38 (3) c/s: 8803701   trying: BH9IRFU - SKON0IN
EF456         (Bsmith)

```

Figure 3.3: JTR at work.

After 24 hours we had 45 of the 48 hashes.

```

C:\> Command Prompt
RD!           (Usmith:2)
guesses: 32   time: 0:00:02:38 (3) c/s: 8803701   trying: BH9IRFU - SKON0IN
EF456         (Bsmith)
COW123        (Psmith)
ISNTIT        (Usmith:2)
TABLEZ        (Zsmith:2)
EASYONE       (Wsmith:1)
guesses: 37   time: 0:00:21:41 (3) c/s: 7499555   trying: EFLGRBA - EUCNU93
Q#12          (Csmith:2)
L!Q#          (Msmith:2)
3!Q#          (Dsmith)
GHCHGH        (Gsmith)
DEMOPAS       (demouser:1)
ROOTWAR       (harderpass:1)
RIDOTNE       (Usmith:1)
guesses: 44   time: 0:05:17:11 (3) c/s: 3851222   trying: AGWUHOA - N06718Y
guesses: 44   time: 0:19:45:10 (3) c/s: 2740978   trying: BNF9IZL - M6A50BH
LSOISDA       (Vsmith:1)
guesses: 45   time: 0:20:03:11 (3) c/s: 2734211   trying: GCE88ZA - FDWXTUF
guesses: 45   time: 0:23:13:08 (3) c/s: 2397177   trying: L2OLY!0 - JXXHFZK
guesses: 45   time: 0:23:19:56 (3) c/s: 2392985   trying: 3ETI3ZP - 4K00-ZP
guesses: 45   time: 0:23:36:10 (3) c/s: 2383601   trying: 3C7R1GG - 3BNBJ6E
guesses: 45   time: 1:00:00:18 (3) c/s: 2371816   trying: M2XSSCA - CH1TRPR
guesses: 45   time: 1:00:00:21 (3) c/s: 2371770   trying: M62NAPI - C1ZNT0N
Session aborted

C:\Documents and Settings\NoOne\Desktop\CLI\john-16\run>

```

Figure 3.4: JTR after 24 hours of cracking



The results of our efforts! All but 3 of the hashes were cracked in 24 hours by John. The “Xsmith” account with 15 characters was not cracked. With enough time we should have been able to find the passwords for “csmith” and “Vsmith”

Note that this really wasn’t a fair assessment since john will try characters not in our rainbow tables. If you want a really fair assessment, you should modify john’s ini file. But I don’t plan on doing it that. The point of the tables is the speed. But honestly, for this password file, John did really well.

```

C:\Documents and Settings\NoOne\Desktop\CLI\john-16\run>john --show hashes4rainbow
wlab.txt
testuser1:TESTUSER1:1120ACB74670C7DD46F1D3F5038A5CE8:::
remote:PASSWORD:8846F7EAE8FB117AD06BDD830B7586C:::
joeuser:PASSWORD:8846F7EAE8FB117AD06BDD830B7586C:::
averageguy:AVERAGE:A5C07214487C87B584E8877DE72DCA0B:::
harderpass:ROOTWARS:EC50F8A8149C93EF45AECB8AF96658E6:::
demouser:DEMOPASS:371D5760453C1B000BCC016F8E23A83C:::
randy:RANDY:A9F34664151F6360757B31644F37E025:::
Asmith:ABCD:E4EBE0E7EF708DC9FD240135D3D43D89:::
Bsmith:EF456:3431E75AD08DCA56EB53AEAA89926589:::
csmith:??????Q#12:A2746ED4129985C0251D2B968C4889FE:::
Dsmith:3!Q#:F09A31889C35B8C9746B8F31FC3A868F:::
Fsmith:ABCDEF!Q#:FAF10460760FA3F1ED804C7C724CB3D4:::
Gsmith:GHGHH:1CC1B3958B564125D307BA8D9D60DF69:::
Hsmith:ABC123:972E8E7D5568F70AC896B2C76E1395DC:::
Jsmith:ABCDEF123:147D125645D463C33D72309525E9B0BC:::
Ksmith:ABCDEF123:147D125645D463C33D72309525E9B0BC:::
Lsmith:ABCDEF:3DCEBC92C0ED8F52B1D759DD35CF3F0F:::
Msmith:FOOTBALL!Q#:45E8DA896575E2F5455B037FCC5AA51A:::
Nsmith:SOCCER:C318744C4291EA46BC65082636CC9509:::
Osmith:CROKET:672532E8C0C490BD47254DAED1CDCB36:::
Psmith:COW123:39981702716E054CBE6840A3CFD60327:::
Qsmith:HOWNOW:9FDA95FD6FCEE9C2C998CB8010F61F16:::
Rsmith:BROWNCOW:D2A80A79980CFA21CB58B7CB129E2CAD:::
Ssmith:GHANDI:62F740C2EA31E10B54DB64CE12E867A6:::
Tsmith:ABCDEF:3DCEBC92C0ED8F52B1D759DD35CF3F0F:::
Usmith:RTDOTNET:476541DEC5CB507A795FC1E989C9D36F:::
Vsmith:??????RD!:16FAABB24B95B82EFC50B074B7324517:::
Wsmith:EASYSOFT:15B194EB8D8F27761E32F76B001553A0:::
Xsmith:2321504F2FA9437FBB66EA1623407D3:::
Ysmith:LSOISDABEST:75344B75B5A96614FE179C0188A9634A:::
Zsmith:RAINBOWTABLEZ:0C105C9F4326C3AC100C2A5B7A04AD38:::

45 passwords cracked, 3 left

```

Figure 3.5: the results 45 passwords cracked

Let’s see how LC4 fairs against our password file, I did turn off the dictionary and hybrid modes on LC4 and selected alphanumeric as our characters in the session options, so this should be a pretty fair “time to crack the same hashes” test.

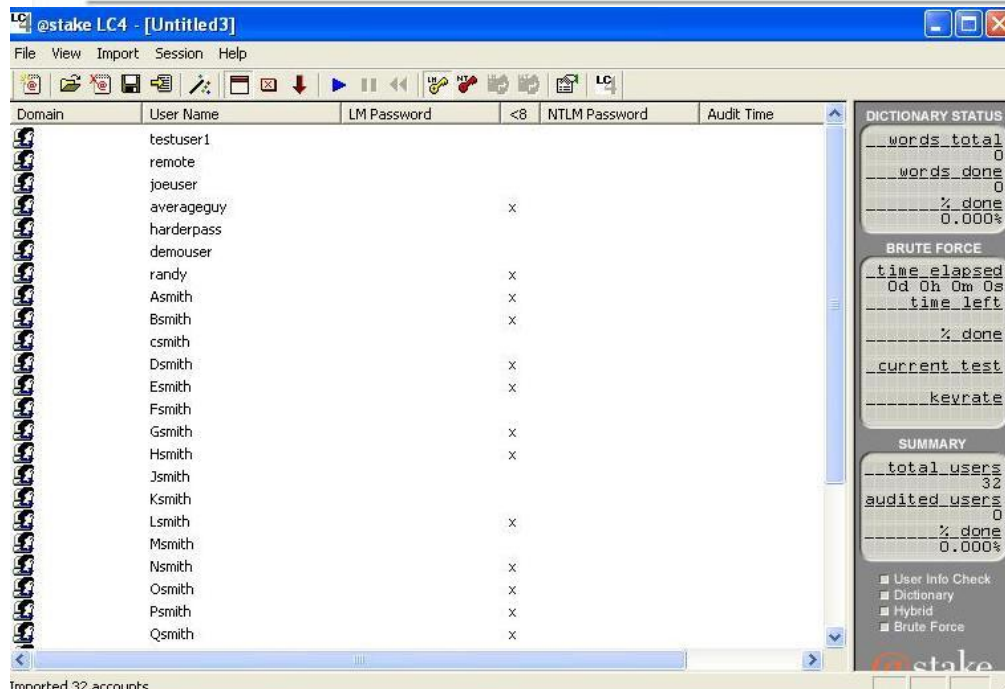


Figure 3.6: Loading the password file of 32 users into LC4.

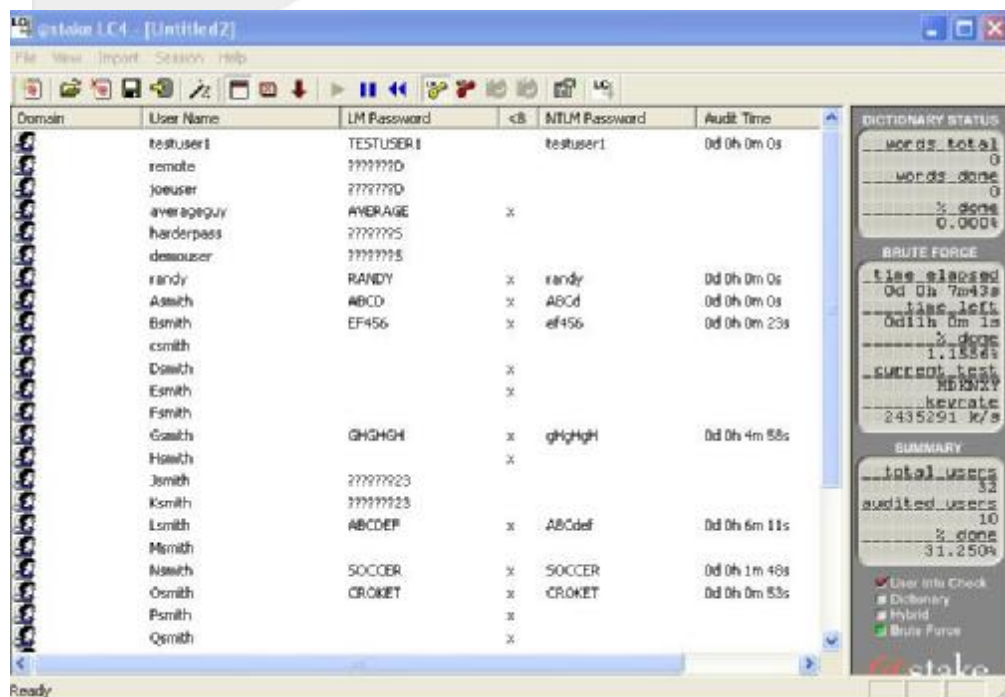


Figure 3.7: LC4 estimated about 11 hours to bruteforce crack the passwords using an alphanumeric character set.

In 11 hours we were able to crack 26 out of the 32 user accounts but the Xsmith account was not cracked because we did not attempt an NTLM attack.



Using Cain and Abel's Wintngen to create your Rainbow Tables. Wintngen supports Rainbow Tables for the following hashing/encryption algorithms: LM, FastLM, NTLM, CiscoPIX, MD2, MD4, MD5, SHA-1, SHA-2 (256), SHA-2 (384), SHA-2 (512), MySQL (323), MySQL (SHA1) and RIPEMD160.





Rainbow Table properties

Hash: lm Min Len: 1 Max Len: 7 Index: 0 Chain Len: 2400 Chain Count: 40000000 N° of tables: 1

Charset: alpha-numeric Edit
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Table properties:
Key space: 80603140212 keys
Disk space: 610.35 MB
Success probability: 0.607082 (60.71%)

Benchmark:
Hash speed: 717360 hash/sec
Step speed: 490484 step/sec
Table precomputation time: 2.26534 days
Total precomputation time: 2.26534 days
Max cryptanalysis time: 5.87175 seconds

Benchmark OK Cancel

Figure 4.2: Select LM, 1 to 7 for Min/Max Length, Chain Length 2400 and Chain Count 40,000,000.

As you see with one table we get about 60% success rate and it will take about 2 days to create the table on a P3 1GHz machine. Feel free to manipulate Chain Length (remember that it will increase success rate but increase computation time) to whatever you can handle for table pre-computation time. I will leave it at 2400 for now. But 60% isn't that great, for a 99.06% success rate you will need to create 5 tables (3 GB of space) and it will take about 12 days to create the tables. For a 99.63% success rate you will need to create 6 tables (3.57 GB of space) and it will take about 14 days to generate the tables. I will go with 5 tables for a success rate of 99.06%.

Rainbow Table properties

Hash: lm Min Len: 1 Max Len: 7 Index: 0 Chain Len: 2400 Chain Count: 40000000 N° of tables: 5

Charset: alpha-numeric Edit
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Table properties:
Key space: 80603140212 keys
Disk space: 2.98 GB (610.35 MB each table)
Success probability: 0.990635 (99.06%)

Benchmark:
Hash speed: 689464 hash/sec
Step speed: 481046 step/sec
Table precomputation time: 2.30978 days
Total precomputation time: 11.5489 days
Max cryptanalysis time: 29.9348 seconds

Benchmark OK Cancel

Figure 4.3: Creating 5 rainbow tables with a success rate of 99.06% using about 3 GB of space.



Here is a handy reference table:

There are some typical configurations (for LM hash type, length from 1 to 7) you can use, for example:

	#1	#2	#3	#4
Charset	alpha	alpha-numeric	alpha-num-sym14	all
Chain length	2,100	2,400	12,000	20,000
Chain count	8,000,000	40,000,000	40,000,000	100,000,000
Tables	5	7	13	20
Success rate	99.9%	99.9%	99.9%	99.6%
Total space	640 Mb	4,480 Mb	8,320 Mb	32,000 Mb
Max gen. time	18h 35m	6d 5h	67d 18h	369d
Max analysis time	8 s	16 s	15 m	53 m

Example 5:

Using Cain and Abel to crack passwords using Rainbow Tables

Step 1: Download⁴ and install Cain.

Step 2: Click on the “Cracker” tab. Select what type of passwords you want to crack. In this case LM & NTLM Hashes. Then right click and select “add to list.” Navigate to where you have your text file of hashes, select it and then select next.

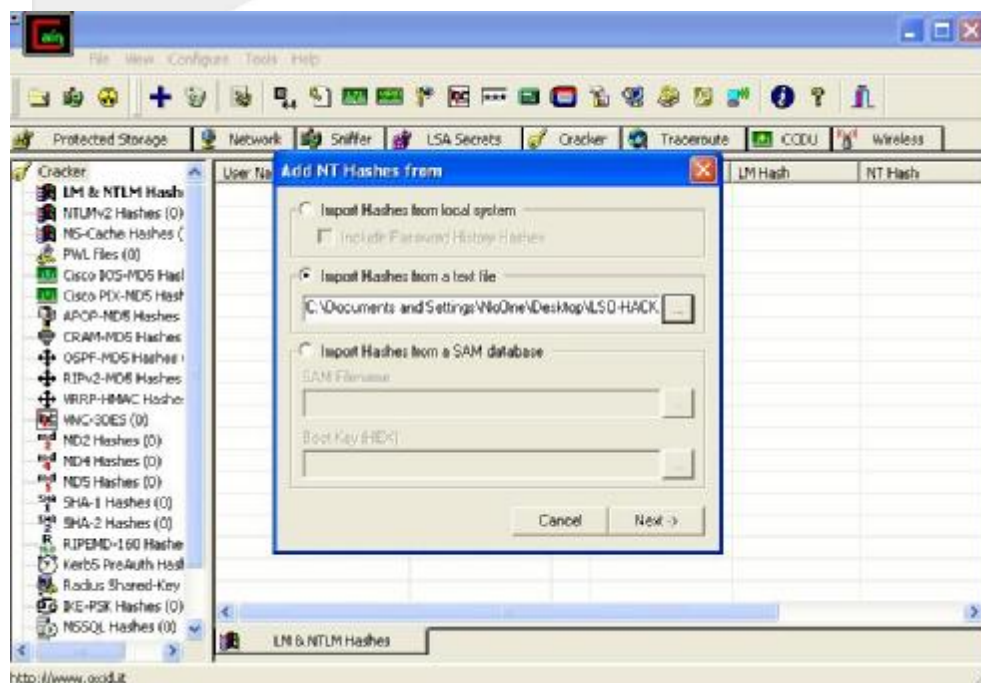


Figure 5.1: Loading hashes from file

⁴ Download Cain and Abel from: <http://www.oxid.it/>

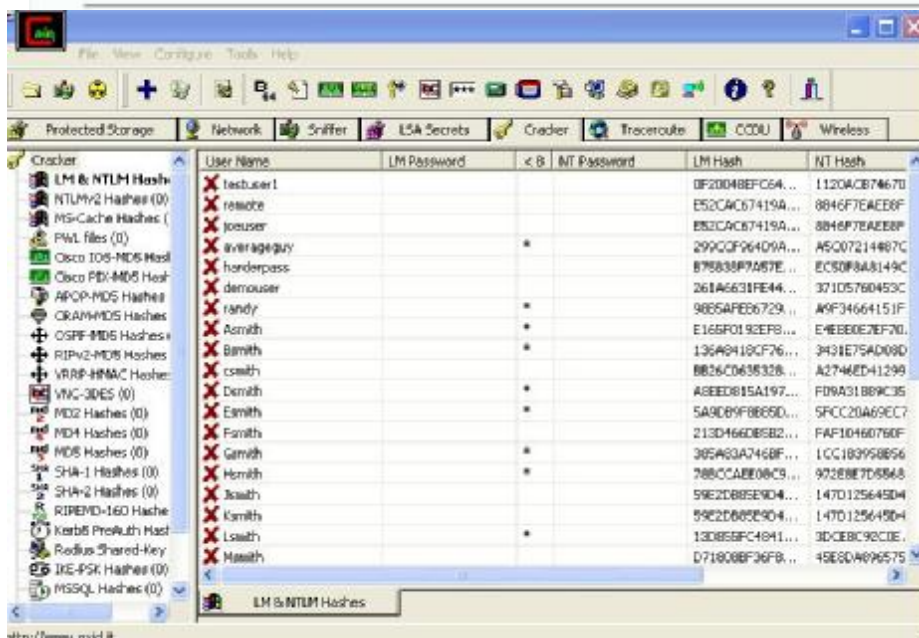


Figure 5.2: Hashes loaded into Cain, ready to be cracked.

Step 3: Right click and select “select all” then right click again and select cryptanalysis attack and “LM Hashes via RainbowTables”

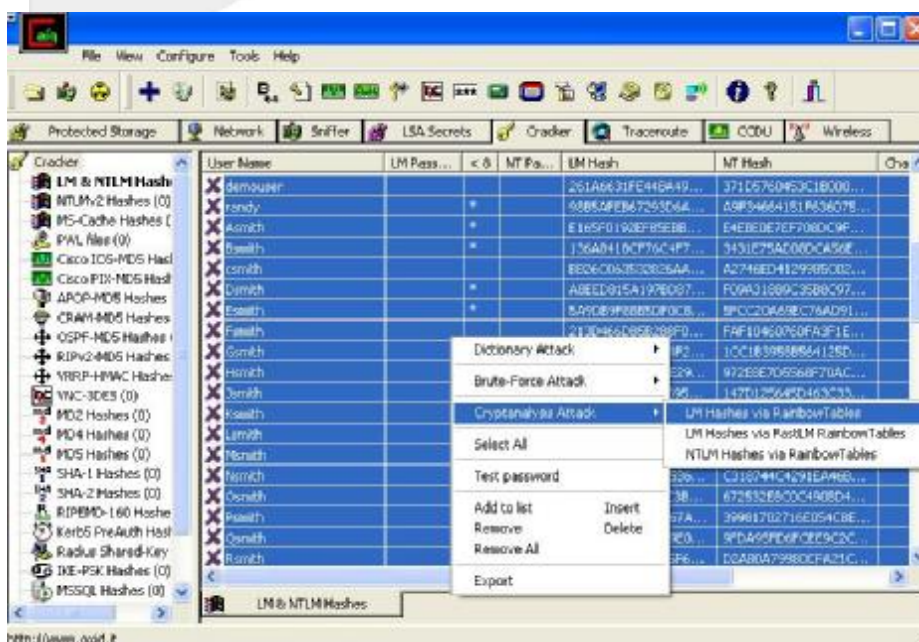


Figure 5.3: Selecting a cryptanalysis attack via RainbowTables.

Step 4: Click on Add Table. Then navigate to where you have your rainbow tables, highlight them all and select Open.

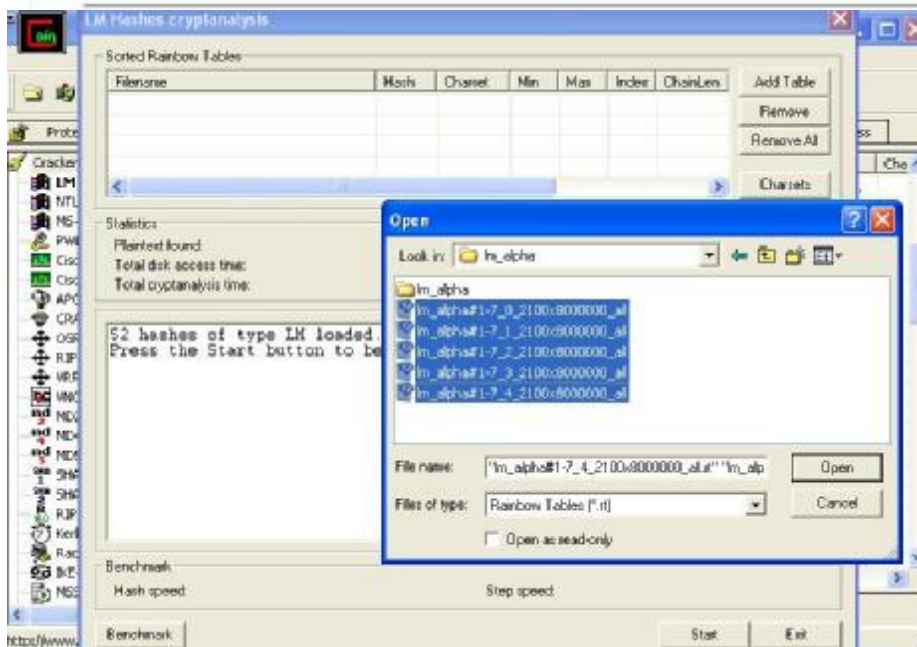


Figure 5.4: Adding your rainbow tables to use for cracking.

Step 5: Click on “Start” and Cain will start to work through the rainbow tables.

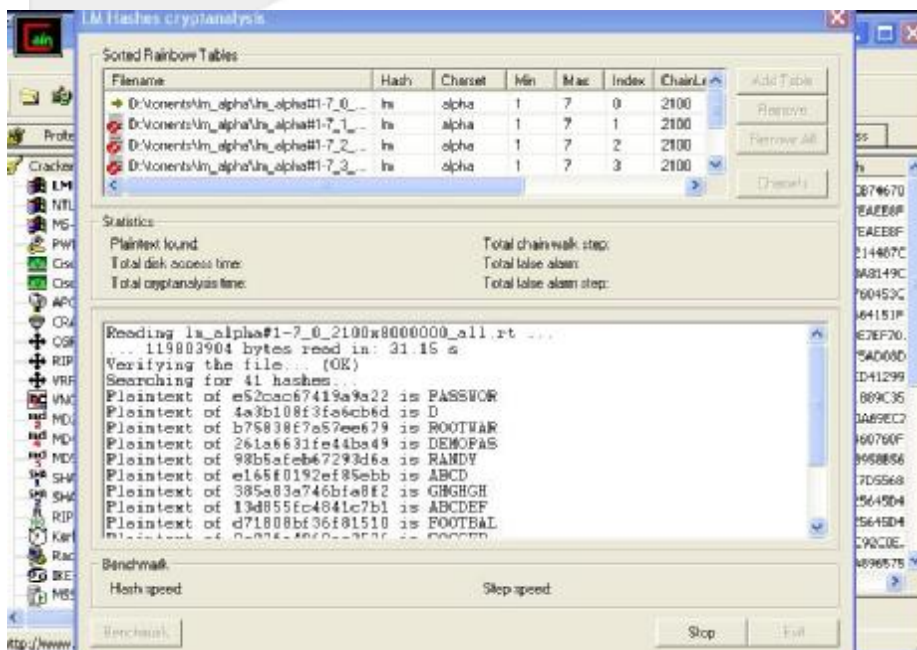


Figure 5.5: Cain working through the Rainbow Tables cracking passwords.

Step 6: When its all done click Exit and it will show you the cracked passwords.

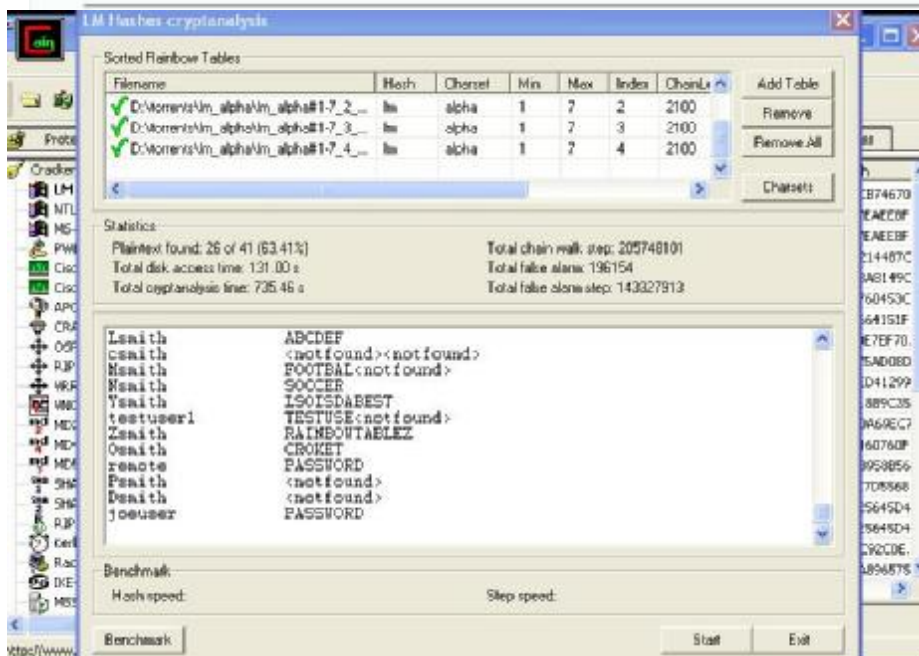


Figure 5.6: Cain finishes running through the Rainbow Tables.

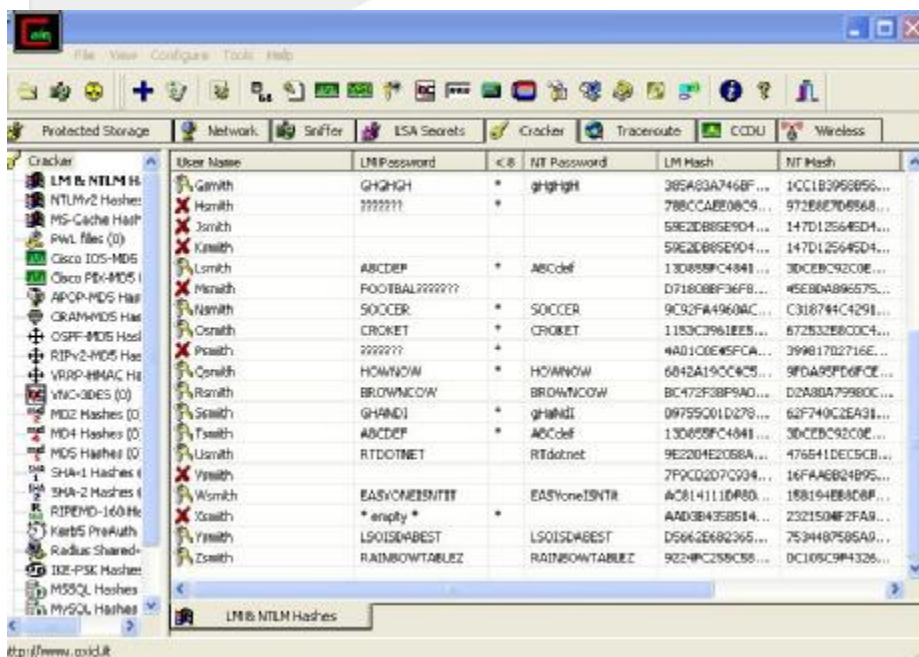


Figure 5.7: Our cracked passwords in Cain. Notice that Cain also found the NTLM password based on the LM password.



I am still confused what does “X, Y, or Z” mean?

Here are some things that may not be immediately clear when dealing with rainbow tables:

1- What does “t”, “m”, and “l” mean or stand for?

To answer this, let’s analyze an rtgen command:

```
rtgen lm alpha 1 7 0 2100 8000000 all
```

rtgen obviously means the program to run. “**lm**” means we want to generate LAN Manager tables. “**alpha**” mean we want to use the characters listed in our charset.txt file for alpha:

```
alpha = [ABCDEFGHJKLMNOPQRSTUVWXYZ]
```

“**1**” and “**7**” are our plaintext ranges. So we want passwords from “A” to “ZZZZZZ.” If we had put plaintext length range “4-6”, “AAAA” and “ZZZZZZ” would be among the key space; but “AAA” would not because it has a length 3. Remember that, for LAN Manager, passwords they are broken up into 7 character chunks, so there would be no need to do a plaintext range of 1 to 8. The “**0**” is our table number or rainbow **table count**, if you look at the rtgen commands to generate configuration #0 we create five tables 0 to 4. This is so we can split up tables between computers making the rainbow tables and to increase our success rate. “**2100**” is our rainbow **chain length**. Chain length increases the success rate per table but does not increase table size. It computes more hashes per chain but also takes longer to create and search the table. A common “upper” value for chain length is 4000-5000. “**8000000**” is our rainbow **chain count** of each rainbow table. Chain count is simply how many chains you want per table. Increasing this value produces larger files with higher success rates, but the overall computation time isn’t affected. You can adjust the chain count so your rainbow tables are conveniently sized (like for a CD or DVD). The “**all**” is our file title suffix or what we want appended to the end of our table’s file name, it can be anything you want.

2- What do the different chain lengths and chain counts mean?

Chain Length increases the success rate per table. It computes more hashes per chain but also takes longer to create and search the table. A common “upper” value for chain length is 4000-5000 but it can be whatever you want. Chain count is simply how many chains you want per table. Increasing this value produces larger files with higher success rates, but the overall computation time isn’t affected. You can adjust the chain count so your rainbow tables are conveniently sized (like for a CD or DVD) or to increase the success rate.

3- Why can’t I create just one rainbow table?

You can! But to get a high enough success rate that table will be too large to search in a reasonable amount of time. That is why we normally create several. Now we could, by adjusting chain length and chain count, create a giant rainbow table but we will have to sort it, which will take a long time and then search it, which will take an even longer time; thus reducing the whole point of rainbow tables. It would be a more efficient use of space to create many rainbow tables so you can sort and search them faster.

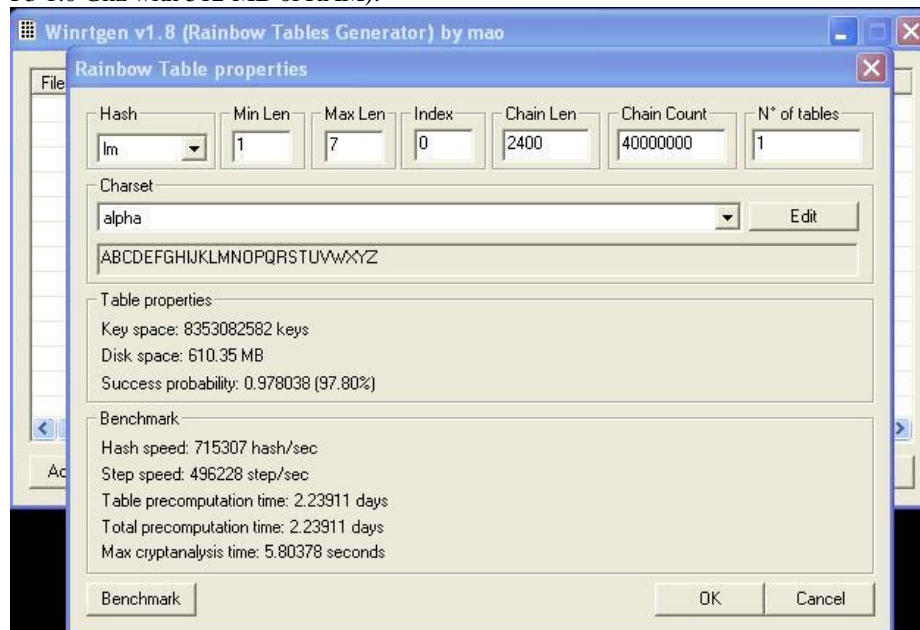
4-I am still confused!

Then 1) go read the paper: http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Oech03 and 2) check out the next section for some examples with Winrtgen which allows you to see (graphically) how changing values changes success rates, table size, and table generation time.

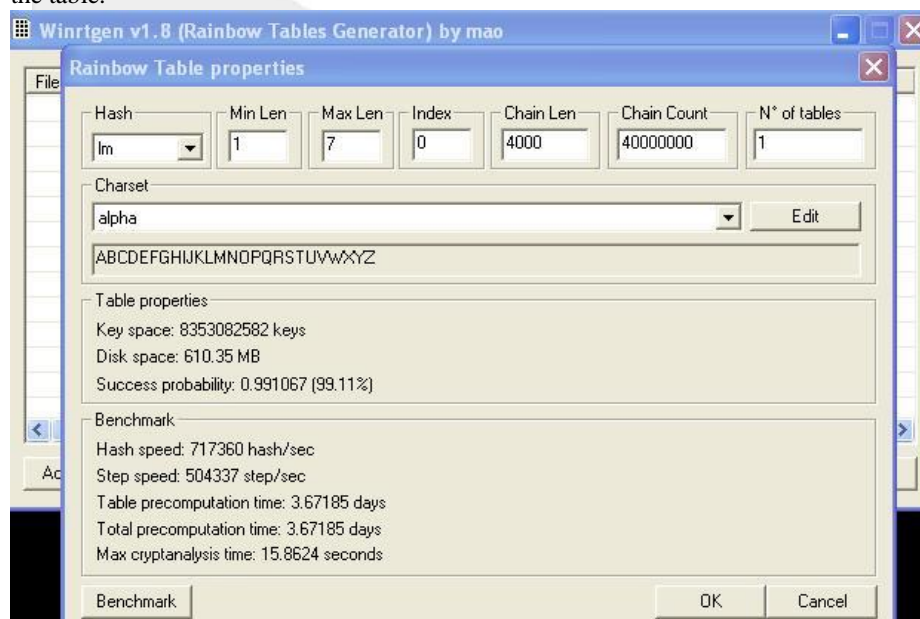


Using Winrtgen to see how chain length, chain count and number of tables effects success rate and computation time

We can see in this example that we get a 97.80% success rate with one LM ALPHA rainbow table with a Chain Length of 2400 and a Chain Count of 40,000,000. It will take 2.23 days to generate the table on the computer (a P3 1.0 Ghz with 512 MB of RAM).

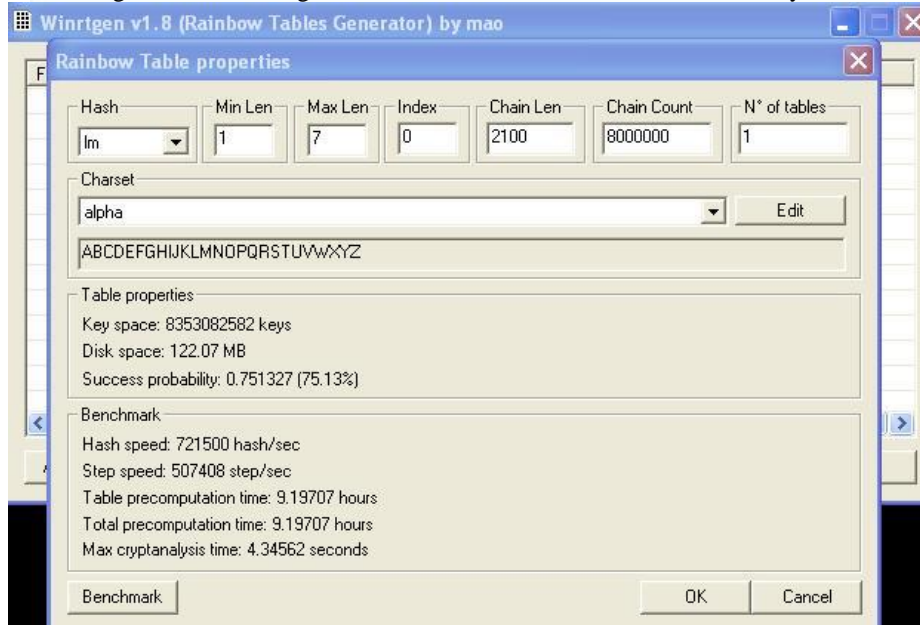


Increasing the Chain Length to 4000 increases our success rate to 99.11% but it now takes 3.67 days to generate the table.

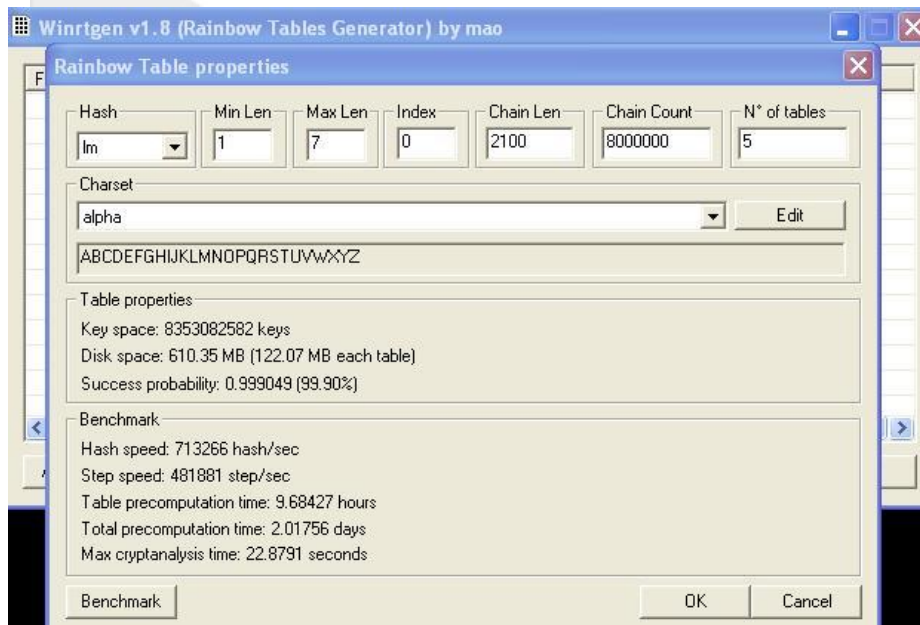




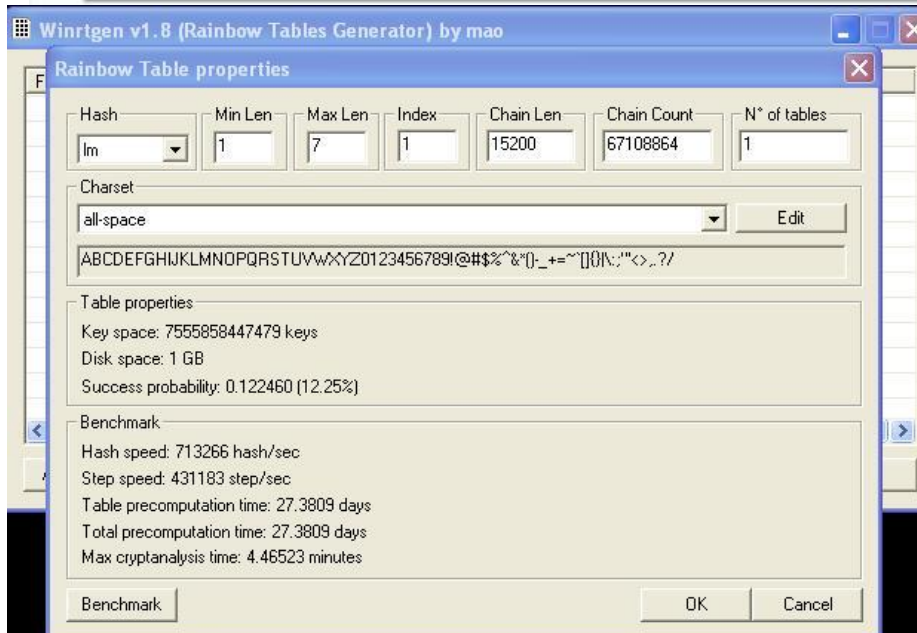
LM Configuration #0 Configuration with 1 table. 75% success rate but only takes 9 hours to generate the table.



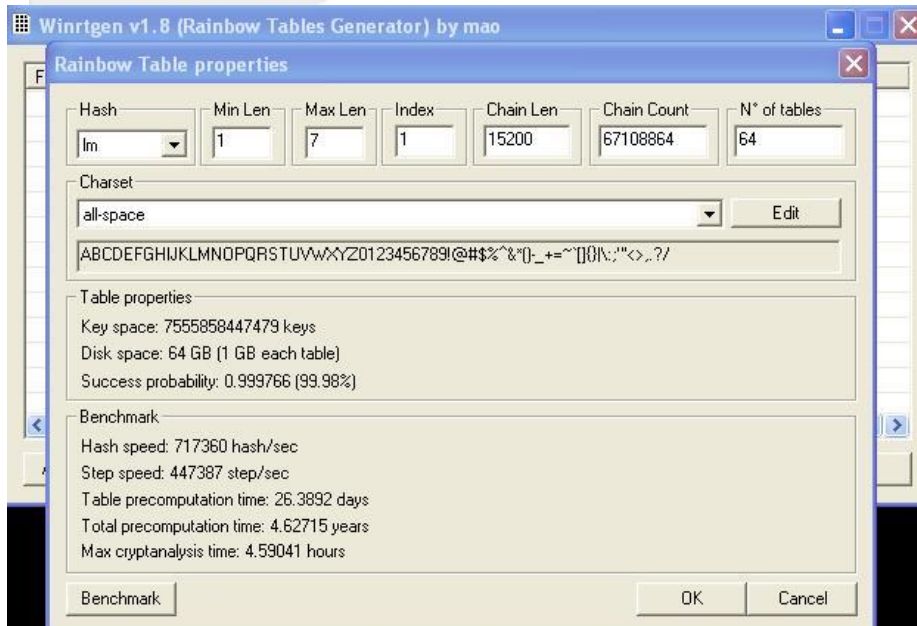
LM Configuration #0 with 5 tables (the recommended configuration). Notice that for roughly the same amount of time and space as our first example with a Chain Length of 2400 and a Chain Count of 40,000,000 and a success rate of 97.80% we can get 99.90% with this Rainbow Table configuration. Another thing to note that we don't see is sort time and how much longer it takes to sort one big table versus several smaller tables.



Let's see how long it takes to create tables to find "all" possible password combinations—minus "ALT-XXX commands." For a 1GB table it will take 23 days with a 12.25% success rate.



And it will take 4.6 years (on a P3 1GHz machine) to generate enough tables to reach 99.98%!!!



Protecting yourself against RainbowCrack attacks and other password attacks

- Limiting physical access
- Continue to force the use of special characters
- Keep up with updates

- Pass phrases
- Use NTLM or NTLMv2

What if my Windows password is longer than 14 characters or the LAN Manager hash is not stored?

If your systems do not require the LAN Manager (LM) hash (for example if you are running an Active Directory (AD) domain), or if your password is >14 characters long, the LM hash will be stored as the blank LM hash, even though the clear text password itself is not blank. Basically none of the cracking tools will see a LM hash.

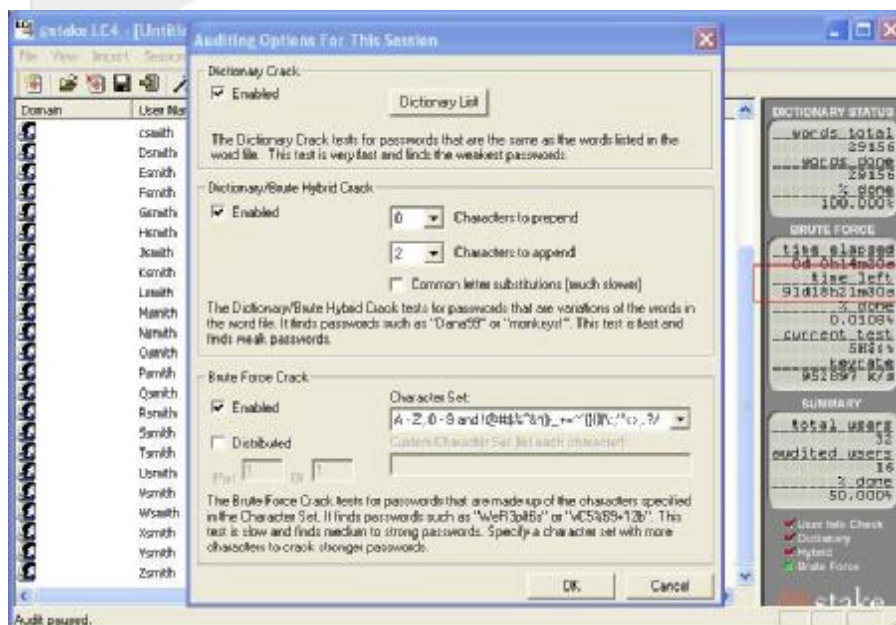
If this is the case, you will need to audit your password hashes against the NTLM character set.

Limiting physical access

It's a well known fact that if someone has physical access to a machine then it's not secure. They can walk off with it, take the hard drive, turn it off, etc. One common attack if you have physical access to a machine is to use a bootable Linux distro to simply boot into Linux and grab the SAM file off the windows partition. You can then crack it at your leisure. IronGeek wrote a good tutorial on this method and even has a video you can watch. You can get it here: <http://www.irongeek.com/i.php?page=security/localsamcrack2>

Continue to force the use of special characters

Even though rainbow tables can rip thru a LM password with any type of special character it still takes a large amount of time (1-2 years) to generate them, this will deter most people or force them to use an online hash cracking service⁵. It also greatly increases the time for brute force attempts. In LC4 we go from 9-11 **hours** to brute force alpha-numeric password to 91 **days** to brute force passwords with the possibility of all special characters (minus ALT-XXX passwords).



Brute-force cracking time is greatly increased by using special characters in your passwords. From 9 hours with just alpha-numeric to 91 days with all characters.

⁵ <http://www.rainbowcrack-online.com/> or <http://www.plain-text.info/>



Keep up with updates

Keep up with your security patches. While you can't protect against zero day exploits you can protect against exploits that have patches! All of the password dumping tools must have *administrative* level privileges to dump the hashes. You can keep the majority of the bad guys out by patching your machines promptly against public exploits. This will help keep you protected from that system/administrative level exploit that was just released to the public.

Pass phrases

Using pass phrases is the easiest and simplest way to protect you network from password cracking. If your password policy makes use of pass phrases that are greater than 14 characters AND use special characters you can protect yourself from all but the determined attackers. If your network is Windows 2000 and above you have a maximum length of 127 characters on your password/pass phrase; so sky's the limit. A pass phrase like "This is my Stupid Pass Phrase!" is long enough to be stored as NTLM or NTLMv2 (because it is longer than 14 characters), has Uppercase, Lowercase, Spaces, and Special Characters, and is easy to remember. This is a much more secure password than even "@w3cjd\$Beu=mDr". If you can get your users to do some character substitution on their pass phrases even better!

The use of strong passwords within an environment needs to be mandated for users. Using the stronger NTLMv2 hashing scheme won't prevent a successful dictionary attack. The use of strong passwords can be enforced on Windows NT through the use of the passfilt.dll. This is described in Microsoft Knowledgebase Article 161990⁶. The use of strong passwords in Windows 2000, XP and 2003 can be enforced by settings in the Group Policy, which is described in Microsoft Knowledgebase Article 225230⁷.

Use NTLM or NTLMv2

Instead of storing your user account password in clear-text, Windows generates and stores user account passwords by using two different password representations, generally known as "hashes." When you set or change the password for a user account to a password that contains fewer than 15 characters, Windows generates both a LAN Manager hash (LM hash) and a Windows NT hash (NT hash) of the password. These hashes are stored in the local Security Accounts Manager (SAM) database or in Active Directory.

The LM hash is relatively weak compared to the NT hash, and it is therefore prone to fast brute force attack. Therefore, you may want to prevent Windows from storing an LM hash of your password

Windows 2000-based servers and Windows Server 2003-based servers can authenticate users who connect from computers that are running all earlier versions of Windows. However, versions of Windows earlier than Windows 2000 do not use Kerberos for authentication. For backward compatibility, Windows 2000 and Windows Server 2003 support LAN Manager (LM) authentication, Windows NT (NTLM) authentication, and NTLM version 2 (NTLMv2) authentications. The NTLM, NTLMv2, and Kerberos all use the NT hash, also known as the Unicode hash. The LM authentication protocol uses the LM hash. The use of LAN Manager hashes on the network can be disabled on Windows NT, 2000, 2003 & XP through registry edits or through the Local Security Policy. The instructions to do so can be found at in Microsoft Knowledgebase Article 147706⁸. The storage of LAN Manager hashes also needs to be disabled; this can be done for Windows 2000, XP and 2003 again via registry edits or the Local Security Policy. The instructions to do so can be found at in Microsoft Knowledgebase Article 299656⁹.

Method 1: Implement the NoLMHash Policy by Using Group Policy

⁶ <http://support.microsoft.com/default.aspx?scid=kb:en-us:161990>

⁷ <http://support.microsoft.com/default.aspx?scid=kb:en-us:225230>

⁸ <http://support.microsoft.com/default.aspx?scid=kb:en-us:147706>

⁹ <http://support.microsoft.com/default.aspx?scid=KB:EN-US;q299656&>



To disable the storage of LM hashes of a user's passwords in the local computer's SAM database by using Local Group Policy (Windows XP or Windows Server 2003) or in a Windows Server 2003 Active Directory environment by using Group Policy in Active Directory (Windows Server 2003), follow these steps:

1. In Group Policy, expand **Computer Configuration**, expand **Windows Settings**, expand **Security Settings**, expand **Local Policies**, and then click **Security Options**.
2. In the list of available policies, double-click **Network security: Do not store LAN Manager hash value on next password change**.
3. Click **Enabled**, and then click **OK**.

Method 2: Implement the NoLMHash Policy by Editing the Registry

Windows 2000 SP2 and Later

To add this key by using Registry Editor, follow these steps:

1. Start Registry Editor (Regedt32.exe).
2. Locate and then click the following key:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa
3. On the **Edit** menu, click **Add Key**, type **NoLMHash**, and then press ENTER.
4. Quit Registry Editor.
5. Restart the computer, and then change your password to make the setting active.

Notes

- This registry key change must be made on all Windows 2000 domain controllers to disable the storage of LM hashes of users' passwords in a Windows 2000 Active Directory environment.
- This registry key prevents new LM hashes from being created on Windows 2000-based computers, but it does not clear the history of previous LM hashes that are stored. Existing LM hashes that are stored will be removed as you change passwords.

Windows XP and Windows Server 2003

1. Click **Start**, click **Run**, type **regedit**, and then click **OK**.
2. Locate and then click the following key in the registry:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa
3. On the **Edit** menu, point to **New**, and then click **DWORD Value**.
4. Type **NoLMHash**, and then press ENTER.
5. On the **Edit** menu, click **Modify**.
6. Type **1**, and then click **OK**.
7. Restart your computer, and then change your password.

Notes

- This registry change must be made on all Windows Server 2003 domain controllers to disable the storage of LM hashes of users' passwords in a Windows 2003 Active Directory environment. If you are a domain administrator, you can use Active Directory Users and Computers Microsoft Management Console (MMC) to deploy this policy to all domain controllers or all computers on the domain as described in Method 1 (Implement the NoLMHash Policy by Using Group Policy).
- This DWORD value prevents new LM hashes from being created on Windows XP-based computers and Windows Server 2003-based computers. The history of all previous LM hashes is cleared when you complete these steps.



Windows NT

Control of NTLM security is through the following registry key:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\control\LSA

Name: LMCompatibilityLevel

Type: REG_DWORD

Value: 5 : DC refuses LM and NTLM responses (accepts only NTLMv2)

Value: 4 : DC refuses LM responses

Value: 3 : Send NTLMv2 response only

Value: 2 : Send NTLM response only

Value: 1 : Use NTLMv2 session security if negotiated

Value: 0 : default - Send LM response and NTLM response; never use NTLMv2 session security

More information on the values:

Level 0 - Send LM and NTLM response; never use NTLM 2 session security. Clients use LM and NTLM authentication, and never use NTLM 2 session security; domain controllers accept LM, NTLM, and NTLM 2 authentication.

Level 1 - Use NTLM 2 session security if negotiated. Clients use LM and NTLM authentication, and use NTLM 2 session security if the server supports it; domain controllers accept LM, NTLM, and NTLM 2 authentication.

Level 2 - Send NTLM response only. Clients use only NTLM authentication, and use NTLM 2 session security if the server supports it; domain controllers accept LM, NTLM, and NTLM 2 authentication.

Level 3 - Send NTLM 2 response only. Clients use NTLM 2 authentication, and use NTLM 2 session security if the server supports it; domain controllers accept LM, NTLM, and NTLM 2 authentication.

Level 4 - Domain controllers refuse LM responses. Clients use NTLM authentication, and use NTLM 2 session security if the server supports it; domain controllers refuse LM authentication (that is, they accept NTLM and NTLM 2).

Level 5 - Domain controllers refuse LM and NTLM responses (accept only NTLM 2). Clients use NTLM 2 authentication, use NTLM 2 session security if the server supports it; domain controllers refuse NTLM and LM authentication (they accept only NTLM 2).

Method 3: Use a Password That Is at Least 15 Characters Long

The simplest way to prevent Windows from storing an LM hash of your password is to use a password that is at least 15 characters long. In this case, Windows stores an LM hash value that cannot be used to authenticate the user.

To get an idea of the power of using NTLM for your hash algorithm lets see how long it will take to generate a NTLM mixed-alphanumeric rainbow table with Winrtgen:



Rainbow Table properties

Hash	Min Len	Max Len	Index	Chain Len	Chain Count	N° of tables
ntlm	1	7	0	4000	40000000	1

Charset: mixalpha-numeric Edit

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Table properties

Key space: 3579345993194 keys
Disk space: 610.35 MB
Success probability: 0.043235 (4.32%)

Benchmark

Hash speed: 1721763 hash/sec
Step speed: 723588 step/sec
Table precomputation time: 2.55926 days
Total precomputation time: 2.55926 days
Max cryptanalysis time: 11.056 seconds

Benchmark OK Cancel

For a mixed-alphanumeric NTLM table it will take 2.5 days to generate one table with a 4.32% success rate.

Rainbow Table properties

Hash	Min Len	Max Len	Index	Chain Len	Chain Count	N° of tables
ntlm	1	7	0	4000	40000000	100

Charset: mixalpha-numeric Edit

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

Table properties

Key space: 3579345993194 keys
Disk space: 59.60 GB (610.35 MB each table)
Success probability: 0.987963 (98.80%)

Benchmark

Hash speed: 1721763 hash/sec
Step speed: 734214 step/sec
Table precomputation time: 2.52222 days
Total precomputation time: 252.222 days
Max cryptanalysis time: 18.16 minutes

Benchmark OK Cancel

It will take over 100 rainbow tables, 60 GB of space, and 252 days to create the tables to crack the same passwords (with a 98.80% success rate) we have been attacking throughout this paper if they were stored as NTLM instead of LM!



Conclusion

As you can see Rainbow Tables and RainbowCrack are powerful password auditing tools. The best course of action to protect yourself is to not allow the storage and use of LAN Manager (LM) passwords on your network if you don't absolutely need to and create and enforce a strong password policy that will force the storage and use of passwords as NTLM and not LM. Additionally, the time to compute and space requirements of complex Rainbow Tables should limit the use of them to only determined attackers or auditors. A strong password policy, strong domain security policy, and keeping up with your patches and updates is your best safeguard against password attacks.

References

Password Attack Discussion & Benchmarks by Alan Amesbury **Where I got the benchmarking numbers (slightly modified) <http://www1.umn.edu/oit/security/passwordattackdiscussion.html>

RainbowCrack--Not a New Street Drug
<http://redmondmag.com/columns/article.asp?EditorialsID=736>

Rainbow Tables: Nature, Use, and Generation
<http://security.the-engine.org/documents/48/rainbow-tables-nature-use-and-generation>

"Faster Cryptanalytic time – memory trade off" paper by Philippe Oechslin
http://lasecwww.epfl.ch/php_code/publications/search.php?ref=Oech03

Project RainbowCrack
<http://www.antsight.com/zsl/rainbowcrack/>

Get Free Rainbow Tables via torrent files
<http://rainbowtables.shmoo.com/>

The Tactical Use of RainbowCrack to Exploit Windows Authentication in a Hybrid Physical-Electronic Attack by Mike Mahurin
http://www.giac.org/practicals/GCIH/Mike_Mahurin_GCIH.pdf

Password Cracking: Rainbow Tables Explained
<https://www.isc2.org/cgi-bin/content.cgi?page=738>

How to prevent Windows from storing a LAN manager hash of your password in Active Directory and local SAM databases
<http://support.microsoft.com/default.aspx?scid=KB;EN-US;q299656&>

How to disable LM authentication on Windows NT
<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q147706>

How to enable NTLM 2 authentication
<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239869>



About the Author

Chris Gates, CISSP serves as the operations manager and course mentor for [LearnSecurityOnline.com](https://www.learnsecurityonline.com/). Feel free to email comments and suggestions on the tutorial to [chris \[at\] learnsecurityonline \[dot\] com](mailto:chris@learnsecurityonline.com).