Attacks on MD5 Hashed Passwords

Antony G. Robertiello, Kiran A. Bandla

***Abstract*—Message Digest 5 is used commonly to create hash of passwords to allow an encrypted form of password to be sent over network or stored in file system. This paper analyses several password-cracking techniques and compares them for exploiting MD5 hashed passwords. Rainbow Tables have been used to successfully crack LAN Manager passwords and may be useful for cracking MD5 hashed passwords**

***Index Terms*—MD5, Password Cracking, Rainbow Tables**

I. Introduction

TODAY, many network protocols that require user authentication (such as those used for instant messaging), utilize Message Digest 5 (MD5) hashes to encrypt passwords sent from a user client to the system server. Often this is the only information that is encrypted in the authentication exchange. When transmitted over wireless networks, for example, these transmissions are subject to intercept, which means a cracker has access to user identification in plain text and password as a MD5 hash. These protocols, such as America Online's Instant Messenger (AIM®) and YAHOO!® Messenger, depend on the strong hashing function provided by MD5, but is hashing the password alone good enough to protect it?

We analyzed the use of two password cracking techniques, Rainbow Tables and Dictionary attacks, as cryptanalytic tools to break password schemes that use MD5 hashed passwords for user authentication. We looked at pre-computation time, analysis time, storage, and other factors relevant to these two attacks.

II. Review of Message Digest 5 Hash

A. Cryptographic hash basics

A cryptographic hash algorithm takes a message of arbitrary size and produces an output of fixed size. The output is the result of a one-way function, which cannot be reversed. These hash algorithms have several other desirable properties. Given a message *M*, hash(*M*) will always produce the same result. Given a hash *h* it should not be possible to determine the message *M*, such that *h* = hash(*M*). The output of the hash function should look random, so that the hashes of two similar messages look very different.

Common cryptographic application of these hash functions is for data integrity and for authentication.

B. Message Digest 5 (MD5)

Message Digest 5 (MD5) hash was developed by Rivest as an update to his previous MD4 hash and published in 1992 [2]. MD5, like other cryptographic hash algorithms, takes a message of arbitrary size and produces an output of fixed size (128 bits).

Figure xx shows how the MD5 algorithm works. A given message is divided into 512-bit chunks and each chunk is processed as a single MD5 operation. The input to the first operation is an initialization vector and the output is used as the starting point for the next chunk's operation. The last part of the message is padded and appended with the length of the message to form the final 512-bit chunk. The output of this last operation is the hash result.

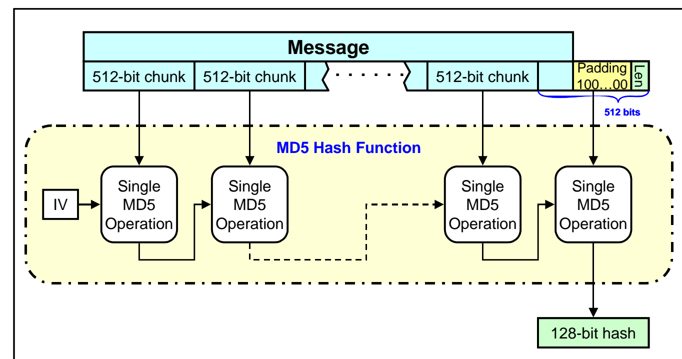


Figure – Illustration of MD5 Hash Function

Within a single MD5 operation, there are 4 rounds of processing, with each round having 16 steps and using a different compression function. MD5 is optimized for 32-bit processors, so the initialization vector and the working state each consist of 4 32-bit words (represented as A, B, C and D) and the 512-bit message chunk is divided into 16 32-bit words. For each round, a different function is performed on 3 of the 4 words (B, C, D) in the state. That result is added (modulo 2^32) with the fourth state word (A), one of the 16 message chunk words and a constant based on the Sine function (which contributes to the randomness of the output). A bitwise shift of the result is performed and then added to state word B. The values of state words B, C, and D are moved to C, D and A respectively and the next step is

Manuscript (initial draft) received December 12, 2005. This work was done as part of ECE 646 at George Mason University (GMU) under the guidance of Professor Kris Gaj.

A. G. Robertiello is a student at GMU and in the United States Air Force currently working for the Defense Information Systems Agency (e-mail: aroberti@gmu.edu).

K. A. Bandla is a student at GMU.

performed for a total of 64 iterations (4 rounds, 16 steps each). Figure xx illustrates a single MD5 operation.

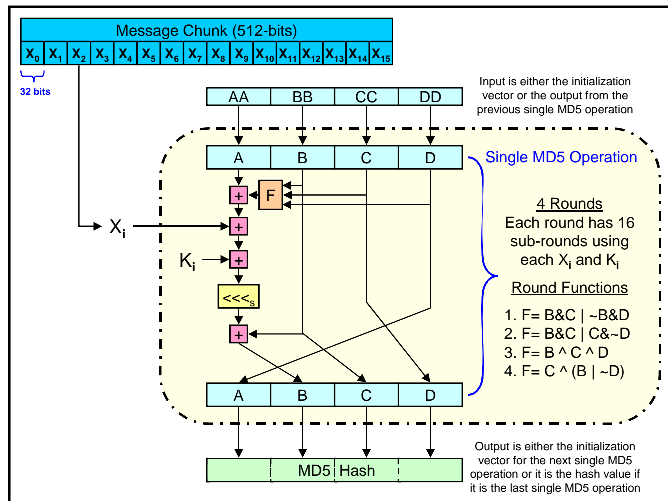


Figure – Single MD5 Operation Illustrated

C. *Uses for MD5 Hashes*

MD5 hashes are used to verify data integrity by providing check-sums of files and with digital signatures of messages. Instead of digitally signing a message under PKI, often a hash of the message is signed instead. MD5 is also used commonly in authentication by hashing passwords. Instead of storing clear text passwords on a system, a hash of the password is stored and during authentication, the system hashes the input password and compares it to the stored hash value. For client server applications, the client sends a hash of the password instead of a clear text password and the server compares that to a hash of the stored password.

D. *Attacks against MD5 hashes*

Most of the current efforts to attack MD5 are based on finding collisions, but for hashed passwords other approaches are better, since the password is a short message, it is easier to use brute force approaches to find the password.

To use brute force requires compute hash value for every value in the key space. This is the classic problem, that it will take too long to compute all possible values.

There are alternative attacks that use tradeoffs in either key space or time/memory to improve the attack success over brute force methods. We look at two methods: Dictionary attacks use select values in the key space and Rainbow Tables use time/memory tradeoffs.

III. Dictionary Attacks

A. *What is a Dictionary Attack*

"A dictionary attack refers to the general technique of trying to guess some secret by running through a list of likely possibilities, often a list of words from a dictionary. It contrasts to a brute force attack in which all possibilities are tried. The attack works because users often choose easy-to-guess passwords, even after being exhorted against doing so." [wikipedia]

B. *How does it work*

Dictionary attacks either pre-compute hash values for a given dictionary file and then compare target password hashes to the pre-computed tables for a match, or words in a dictionary file are hashed and compared against a target password hash for a match.

C. *Advantages and disadvantages of this type of attack*

1) *Advantages*

A dictionary attack does not require pre-computation, but then the lookup/analysis time is much greater. It takes less time to compute values, since not all of the key space is used, since a more likely subset of the key space is used.

2) *Advantages*

The biggest disadvantage is that the entire key space is not used, so probability of success is reduced.

D. *Computation information*

In this section, we try to calculate all the pre-computation time and memory resources for generating a dictionary hash table.

Character set = [abcdefghijklmnopqrstuvwxyz]
Password length = 7
Words = 23109
Dictionary = English
Algorithm = MD5
Record Format = $hash|$password , ordered by $hash
File Size = 947469 bytes
Average lookup time = 1134440021.7656 floating seconds (less than a second).

IV. Rainbow Tables

A. *What are Rainbow Tables*

In 2003, Philippe Oechslin's paper [2] introduced Rainbow Tables, a time-memory trade-off used for cryptanalysis of hashed passwords. Since then, Rainbow Tables have been used extensively to break Windows LAN Manager passwords.

From his work, several applications have been developed that create rainbow tables

-- RTCRACK
-- WINRTGEN

B. *How do they work*

Rainbow tables use a successive reduction function for every point in the chain. They start with a reduction function 1 and end with a reduction size of t-1 (one less chain length). The password is first hashed and then the reduction function is applied to it. Only the first and the intermediate hash (at the length of the chain) are stored in the table.

To crack a hash, we generate a chain from the hash. For each password that we encounter in the chain, we check to see if it at the end of the chain. When we hit a match at the end of

the chain, we know that the hash was part of this chain. Thus we generate the complete chain again and get the password. Note that we can not go backwards from the point where we hit the match.

The time-memory tradeoffs come in by only saving the reduced function values (first and last) in a chain and only re-walking through the computations for that chain when we find a hit that we suspect is in that chain. This greatly saves on memory to store the pre-computed tables and on look-up time.

C. What are their advantages and disadvantages

1) Advantages

The biggest advantage with rainbow tables is less storage space compared to entire key space and less look-up time compared to entire key space. There is also a greater probability success compared to dictionary attack, but not as perfect as brute force, since reduction functions are used.

Once we have the tables generated, it takes little time to crack a password hash.

Rainbow tables can be generated for a wide range of hashing algorithms like MD5, SHA-1, RIPEMD, etc.

Also, the number of table lookups is reduced t times compared to conventional brute force method.

Rainbow chains are merge-free tables compared to Hellman's tables.

2) Disadvantages

The greatest disadvantage is the pre-computation time and effort required. If you change the character set, then you must rebuild the tables.

As the length of the password increases, which in-turn increases the key space, it slowly becomes infeasible for us to mount an attack on hashes using Rainbow Tables. This is because of the increase in table size. Also, if the character set grows beyond a limit, the attack becomes very slow and probability of success is lowered. If the character set is alpha-numeric-symbol then the key space is 76^{14} which takes more than 3 months to generate.

D. Computation information

Rainbow Tables require pre-computation. In this section, we try to calculate all the pre-computation time and memory resources for our sample character set.

Character set = [abcdefghijklmnopqrstuvwxyz]
Password length = 7
Key Space = 267 (8,031,810,176)
Algorithm = MD5
Chain Length= 2100
Chain count per table = 8000000

Pre-computation Information for RTCrack

The following are the details of the test computer that we used:

Processor: Pentium 4, 3.2Ghz , 800Mhz (QDR) FSB , 1MB L2Cache
Memory : 2GB PC3200 RAM
Hard Drive: 76.6 GB , 7200 RPM , ATA/133
Average Disk access time: 0.13 sec
Average MD5 Hash speed : 1599488/sec
Average Step Speed : 1159554/sec

Test computer details

The following are statistics that we calculated using MATLAB.

Statistics for Table precomputation

[key space]	8,031,810,176
[chain length]	2100
[chain count per r.table]	8000000
[table count]	5
[disk usage(MB)]	611
[probability of success]	0.99922
[mean cryptanalysis time(s)]	2.4968
[max cryptanalysis time(s)]	9.508
[mean disk access time(s)]	20.8365
[max disk access time(s)]	79.3457
[precomputation time(days)]	0.83844

Pre-computation Estimates Calculated Using MATLAB

The following graphs were used to pick optimal values. The first graph shows the comparison of chain length versus disk space and is used to pick the optimal chain length. The second and third graph shows a comparison of mean cryptanalysis time and disk usage, which is used to calculate the estimated table size.

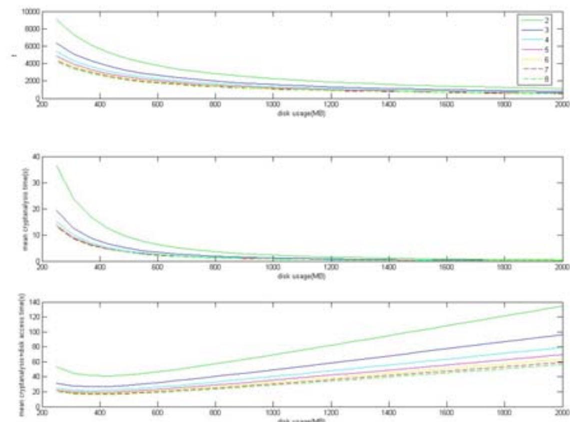


Figure: Mean Cryptanalysis time, Disk usage

The following graph shows the probability of success compared to the size of storage space allocated.

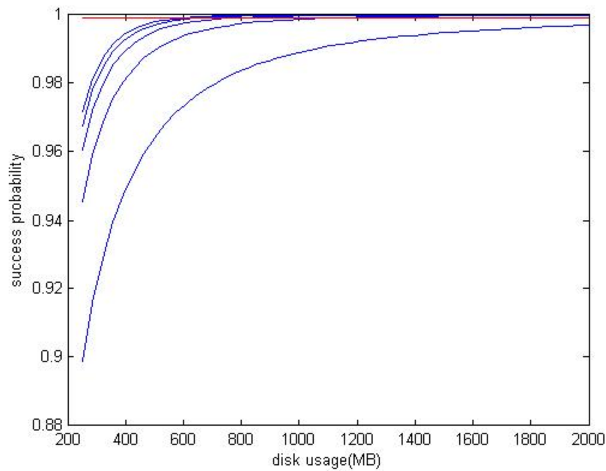


Figure: Probability of success over a disk size from 250MB to 2000MB for a character set of lowercase alpha only with a length of 7.

Rainbow Table generation on the Test machine:

We generated the tables on the test machine based on the parameters from the Pre-computation information. According to the graphs, the optimal table size was 5, which cracks hashes with a probability of success of 0.99922. The generated tables were 122MB each (610MB total table size).

V. Protocols using MD5 Password Hashes

A. America On-line (AOL) Instant Messenger (AIM®)

AOL's Instant Messenger client is available on a number of platforms and is a popular method for communicating.

1) Discussion of Protocol

AIM uses a protocol called OSCAR (Open System for Communication in Realtime). This is a proprietary protocol not published by AOL. It has been documented by others that have analyzed AIM network traffic.

AOL also developed the TOC (Talk to OSCAR) protocol, which they released as a GNU public license in 1999, so that others could develop applications to work with AIM services.

2) Evolution of Protocol

AOL no longer supports the TOC protocol, which only used a "roasted password" (a simple XORing of the password with the roasting string) during authentication.

Versions of AIM prior to 5.2 use a "challenge response" mechanism to authenticate. The user requests authentication and the server sends a challenge string (10 digits). The user concatenates the challenge and the password and the AIM string, which is a constant ("AOL Instant Messenger (SM)") and then hashes that string.

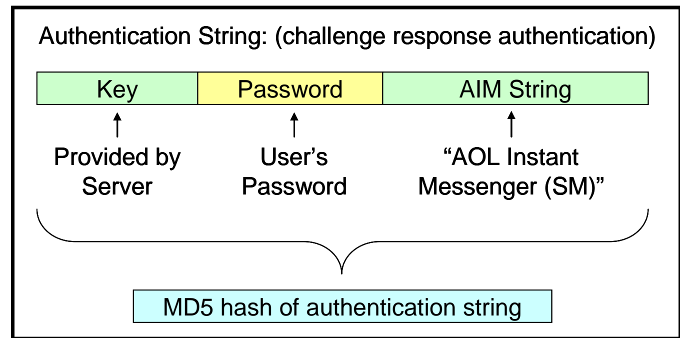


Figure – AIM Protocol Authentication String (prior to Ver 5.2)

In version 5.2 of AIM, that scheme was modified to concatenate the MD5 hash of the password, instead of the just the password, with the challenge string and the AIM string.

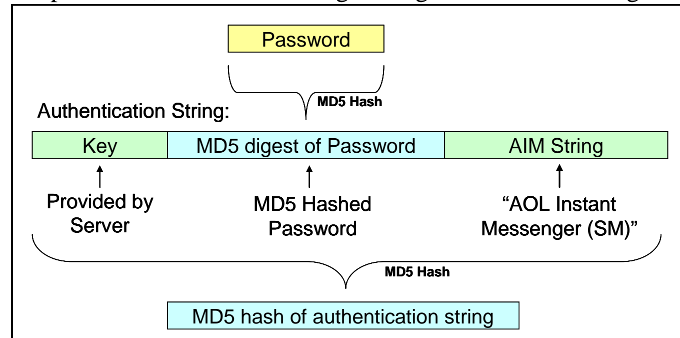


Figure – AIM Protocol Authentication String (Ver 5.2 and later)

B. Other protocols

There are many protocols and applications that use or are capable of using MD5 hashed passwords to support authentication. Some of these applications include: IRC, Jabber, Yahoo 5, Napster, Apache,

Several operating systems support MD5 hashed passwords: UNIX Crypt, Solaris, Sun Messenger Service.

Databases that provide support for MD5 hashed passwords include Oracle, SQL Server and MySQL.

A number of development environments provide readily available MD5 cryptographic modules, such as Microsoft C# and Visual Basic .Net

So, MD5 hashing function is availability and usable in a large range of applications.

VI. Analysis

A. Comparison of two cracking techniques

The following sections compare the two cracking techniques discussed above.

#	Dictionary Attack	Brute Force Attack	Rainbow Table Attack
Key Space	23,109	8,031,810,176	8,031,810,176
Pre-Computation Time	1.05 seconds	96.54 hours (estimated)	20.12 hours
Lookup/ Analysis	< 1 second (0.165)	Depends on sort/search	2.6 seconds max

Time		function	
Storage Requirement	~947K	300GB!	~611MB
Effective Lookup Time Ratio	260		1

* manual work involved in making the dictionary of passwords.

B. *Examples of two cracking techniques against MD5 hashed passwords*

The following are sample results of the two cracking techniques used against MD5 hashed passwords. For this example, we used:

```
Password: zincate
MD5 hash: 9b58c8f1475c261c292c3fbe8dee55cf
```

1) *Dictionary Attack*

```
Looking up 9b58c8f1475c261c292c3fbe8dee55cf in
7hash_sort1.txt...
9b58c8f1475c261c292c3fbe8dee55cf|zincate
time taken to hit = 1134443643.3543 floating seconds
```

2) *Rainbow Table Attack*

```
C:\>rcrack *.rt -h  9b58c8f1475c261c292c3fbe8dee55cf
md5_loweralpha#7-7_0_2100x8000000_all.rt:
128000000 bytes read, disk access time: 0.22 s
verifying the file...
searching for 1 hash...
cryptanalysis time: 3.03 s
md5_loweralpha#7-7_1_2100x8000000_all.rt:
128000000 bytes read, disk access time: 3.78 s
verifying the file...
searching for 1 hash...
plaintext of 9b58c8f1475c261c292c3fbe8dee55cf is zincate
cryptanalysis time: 1.97 s
statistics
-------------------------------------------------------
plaintext found:          1 of 1 (100.00%)
total disk access time:   4.00 s
total cryptanalysis time: 5.00 s
total chain walk step:    3353254
total false alarm:        3450
total chain walk step due to false alarm: 2875139
result
-------------------------------------------------------
9b58c8f1475c261c292c3fbe8dee55cf
   zincate  hex:7a696e63617465
```

Both techniques found our password. It was one of the 7 letter dictionary words in the dictionary, so we expected to find it, but RCRACK found it, too, in less than 2 seconds, but it has the complete key space.

VII. Enhancements and Optimization

A. *Improving Cracking Techniques*

Both methods could benefit form the use of hardware crypto cards instead of software to compute MD5 hashes. This could greatly reduce the pre-computation time for either method.

A study of known passwords could produce algorithms to improve dictionary words (e.g. adding digit to end of word; capitalizing the first letter; combining smaller words; substituting numbers for letters, such as 0 for O and 1 for I), which would create more possible values for use in dictionary attack and increase the probability of success without having to pre-compute all values in the key space.

B. *Improvements for using MD5 hashed passwords*

There are a number of ways to improve using MD5 hashed passwords.

1) *Use Salt*

Standard UNIX salt is 0-4096 represented as 12 bits or 2 bytes. Batch dictionary attacks would require pre-computing all words with all values of Salt. Using pre-computed dictionaries would require pre-computation time and storage requirements to increase by a factor of 2^12.

Dictionary attacks against a single hash (with known salt) would then require computing words plus known salt and comparing to hashed password value. This is the same as attacking a single password without pre-computation, but could only be used against one password at a time.

Rainbow tables would have to be rebuilt to account for the addition of all possible Salt values.

2) *More complex Salt*

If instead of hashing just the password, or adding the traditional UNIX Salt to the password, if the authentication scheme used something in the form: "**user_id : password : domain**", it would significantly improve cryptanalysis attempts using these methods. This is a more complex form of "Salt" and creates more unique hash values for given passwords.

There is no increase in cost to use this in the hashing algorithm, since you still need one single MD5 operation to compute the hash value.

Pre-computed tables would have to factor in possible values for user id and domain OR consider all MD5 message values (for example, user id (8), password (8), domain (7), plus 2 separators (2) == 75^25 = 7.4x10^46 values to pre-computed)

3) *Double hash*

Taking the MD5 hashed password and hashing it again would mean that cryptanalysis techniques would have to find the hashed password value first and then use techniques to find the password from the hashed password value. The key space for that effort would be 2^128!

C. *Possible improvements to protocols that use just MD5 hashed passwords*

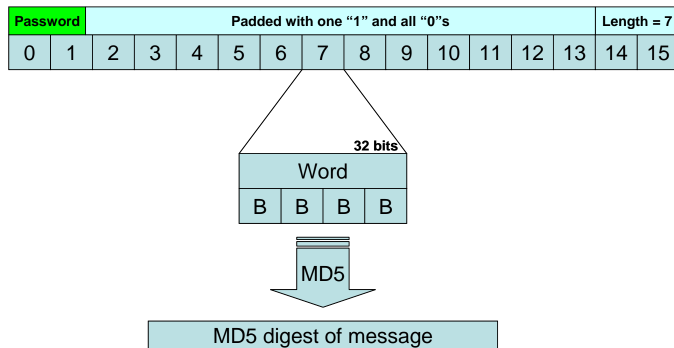
Instead of just sending a hashed password, the authentication scheme could use a challenge response

mechanism. When the client requests authentication, the server returns a challenge string. That sting is concatenated with the password and then hashed and sent back to the server. This is similar to using Salt, but the "Salt" is not known until the authentication sequence is initiated and it is different each time (prevents replay attacks). This means that pre-computation would have to factor in all values of the challenge.

VIII. Other Considerations

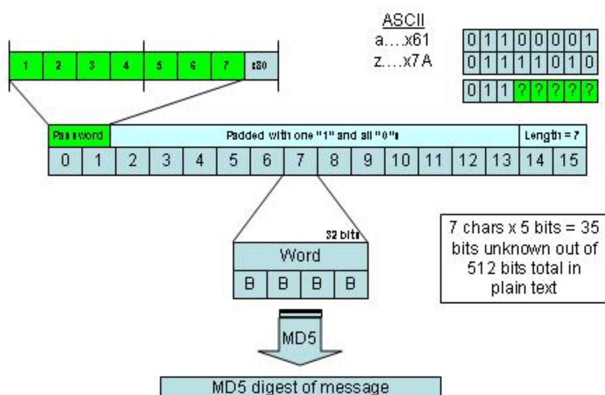
The two cryptanalysis techniques that we looked at are variations of brute force method that use tradeoffs in

Passwords are "short messages" for MD5 hash function, which is typically used to create hash values of large messages. In fact for any password length less than 16 characters, the hash of the password is larger than the password itself. Using MD5 to hash passwords involves just one single MD5 operation. Of the 16 words that make up the 512 bit message input to this single round, all but the first 2 words are known!



Actually, even more is known, since each character is represented by 8 bits and we know the first 3 bits, only 5 are unknown for each of 7 characters… so only 35 bits out of the 512 bit input is unknown.

MD5: 7 character password



What if the single MD5 operation was computed using variables for the 35 unknown bits in the input. If this were accomplished then the output would be 128 equations (one for each bit in the output hash value) with 35 variables. This could be solved to determine the input variables as a form of the output bits.

It would actually be easier to use the determined equations to solve for the variables in the input based on a given hash. If a given input bit resolved to both "0" and "1" from the equations, then the given hash value is not a hash of a 7 character message.

We didn't have time to test 35 bits, but we did develop routines to calculate binary equations using string values (with variables) to represent the bits of words. We then created an MD5 computation and computed the 128 bit hash values for an input that had 5 variables (1 character). Those results are in the appendix.

Given improvements in how to represent and process these binary equations with variables, it could be possible to solve for more variables and thus more characters in a message. This effort would be heavy on pre-computation time, but could have potentially fast cryptanalysis time.

This could become a "partial known message" attack against MD5 hashes, but that's for another paper.

IX. Summary and Conclusion

MD5 hashes of passwords are only as strong as the passwords themselves. Weak passwords can be easily attacked. We demonstrated this using 2 cryptanalysis techniques that are readily available.

Using Rainbow Tables is an effective method for cracking passwords. This methodology increases the time for cryptanalysis over brute force methods while using less storage space for the pre-computed tables. The biggest challenge is the time needed to generate the tables. It's only a matter of time before someone generates tables that can crack longer passwords with a larger character set, if they haven't already.

It is easier to change the password hashing scheme associated with a given application than it is to change the password practices of millions of users.

Increasing the message length used in the hash function, such as by adding a Salt or using a challenge response scheme, greatly increases the effort required to crack these passwords using the methods that we used. There is no increased computation cost, since the larger message still requires only one single MD5 operation.

Appendixes

See attached.

Acknowledgment

The authors would like to acknowledge Elizabeth Goff for the inspiration for the idea.

REFERENCES

[1] R. Rivest, *RFC 1321: The MD5 Message-Digest Algorithm*, Internet Activities Board, 1992. Available: ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt

[2] P. Oechslin, "Making a Faster Cryptanalytic Time-Memory," Proceedings of Crypto'03, 2003. Available: http://lasecwww.epfl.ch/~oechslin/publications/crypto03.pdf

[3] M. E. Hellman. A cryptanalytic time-memory trade off. IEEE Transactions on Information Theory, IT-26:401–406, 1980.

[4] Z. Shuanglei. (2004, January 1) "Parameter optimization of time-memory trade-off cryptanalysis in RainbowCrack," Project RainbowCrack [Online] Available: www.antsight.com/zsl/rainbowcrack.

[5] X. Wang, et. al. "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD" - http://eprint.iacr.org/2004/199.pdf

[6] D. Kaminsky 2004, Dec 6). "MD5 to be considered harmful someday" Doxpara Research [Online] Available: http://www.doxpara.com/md5_someday.pdf.

[7] Neal Hindocha (2003, January). "Threats to Instant messaging", Symantec Security Response Whitepaper [Online]. Available: http://securityresponse.symantec.com/avcenter/reference/threats.to.instant.messaging.pdf

[8] "Yahoo Messenger Protocol" [Online]. Available: http://www.venkydude.com/articles/yahoo.htm

[9] X. Wang and S. Manoharan. "Comparative Analysis of Instant Messaging." From Proceeding (431) Advances in Computer Science and Technology - 2004. Available: http://www.actapress.com/PaperInfo.aspx?PaperID=17278

[10] B. Pinkas and T. Sander, "Securing passwords against dictionary attacks," Proceedings of the 9th ACM conference on Computer and communications security, Sesssion - Authentication and authorization: 161 - 170, 2002.

[11] I. Thomson (2005, March 14) "Microsoft to abandon passwords," CeBIT [Online] Available: http://www.vnunet.com/vnunet/news/2126966/microsoft-abandon-passwords.

[12] J. Davis (2005, March 12). "Password Cracking and Time-Memory Trade Off" [Online] Available: http://neworder.box.sk/newsread_print.php?newsid=13362

[13] M. Stamp (2003, July 26). "Once Upon a Time-Memory Tradeoff" [Online] Available: http://www.cs.sjsu.edu/faculty/stamp/RUA/TMTO.pdf

[14] A. Biryukov (2005). "Some Thoughts on Time-Memory-Data Tradeoffs," Cryptology ePrint Archive [Online]. Available: http://eprint.iacr.org/2005/207.pdf

[15] Gaim Source code – http://gaim.sf.net

[16] OSCAR http://iserverd.khstu.ru/oscar/

Appendix #1: MATLAB caluculations for Rainbow Table Pre-Computation

Statistics for Table precomputation

[key space]	8031810176
[chain length]	2100
[chain count per r.table]	8000000
[table count]	5
[disk usage(MB)]	611
[probability of success]	0.99922
[mean cryptanalysis time(s)]	2.4968
[max cryptanalysis time(s)]	9.508
[mean disk access time(s)]	20.8365
[max disk access time(s)]	79.3457
[precomputation time(day)]	0.83844

Appendix #2: Success probability for table count from 1 to 8

```
MD5 | plaintext = lower-alpha | length=7
Variations using table count from 1 to 5
-----estimates-----
[key space]                     8031810176
[chain length]                  2100
[chain count per r.table]       8000000


Statistics for Table precomputation (table count = 1)
[table count]                   1
[disk usage(MB)]                123
[probability of success]        0.76101
[precomputation time(day)]      0.16769

Statistics for Table precomputation(table count = 2)
 [table count]                   2
[disk usage(MB)]                245
[probability of success]        0.94288
 [precomputation time(day)]      0.33538

Statistics for Table precomputation(table count = 3)
[table count]                   3
[disk usage(MB)]                367
[probability of success]        0.98635
[precomputation time(day)]      0.50307

Statistics for Table precomputation(table count = 4)
[table count]                   4
[disk usage(MB)]                489
[probability of success]        0.99674
[max disk access time(s)]       63.4766
[precomputation time(day)]      0.67076

Statistics for Table precomputation(table count = 5)
[table count]                   5
[disk usage(MB)]                611
[probability of success]        0.99922
[precomputation time(day)]      0.83844

Statistics for Table precomputation(table count = 6)
[table count]                   6
[disk usage(MB)]                733
[probability of success]        0.99981
[precomputation time(day)]      1.0061

Statistics for Table precomputation(table count = 7)
[table count]                   7
[disk usage(MB)]                855
[probability of success]        0.99996
[precomputation time(day)]      1.1738

Statistics for Table precomputation(table count = 8)
[table count]                   8
[disk usage(MB)]                977
[probability of success]        0.99999
[precomputation time(day)]      1.3415
```

Appendix #3: Dictionary Attack Code

Sort.c: To sort input text & create list (1word per line)
Hash.pl: Hash each line, create record $hash|$password. Then sort the records based on $hash
Hash_get.pl: Looks up given hash in given hash dictionary, shows lookup time in Hi-resolution.

Appendix #4: Output of MD5 hash with binary variables (5 variables A0-A4 representing 1 character message)

000:
A0&A1&!A3&!A4|A0&!A1&A2&A4|A0&!A1&A3&A4|!A0&A1&A2&A4|!A0&A1&A3&A4|!A0&!A1&A2&!A4|!A0&!A2
&A3&!A4|!A1&!A2&!A3&A4|A0&!A1&!A2&A3&!A4 001:
A0&!A2&!A3&!A4|!A0&A1&!A2&A4|!A0&!A1&!A3&A4|!A0&A2&A3&A4|!A1&A2&A3&A4|A0&A1&!A2&!A3&A4
002: !A0&A1&A3&!A4|!A0&A2&!A3&A4|A0&A1&A2&A3&A4 003:
A0&A1&!A2&A3|A0&!A2&!A3&!A4|!A0&A1&A2&A4|!A0&A1&A3&A4|A1&!A2&!A3&!A4|A0&!A1&A2&A3&A4|A0
&!A1&A2&!A3&!A4|!A0&!A1&A2&A3&!A4|!A0&!A1&!A2&!A3&!A4 004:
A0&A1&!A2&!A4|A0&!A1&!A2&!A3|A0&!A1&A3&A4|A1&A2&A3&A4|!A1&A2&!A3&!A4 005:
A0&!A1&A3&A4|!A0&A1&A2&A4|!A0&!A1&A3&!A4|!A1&!A2&A3&A4 006:
!A0&A1&!A2&!A4|!A0&!A1&A2&A4|!A0&A2&A3&!A4|A1&A2&!A3&!A4|A0&!A1&A2&!A3&A4 007:
!A0&A4|A2&A4|A0&A1&A2|A0&A1&A3|A0&A1&!A4|A0&A2&A3|A0&A3&A4|!A0&!A1&!A3|!A0&A2&!A3|A1&A2&!
A3|A1&!A2&A3|!A1&!A2&!A3|A0&!A2&!A3&!A4 008:
A0&!A1&A2&!A3|!A0&A1&!A3&A4|!A0&A2&!A3&!A4|A0&A1&A2&A3&A4|A0&!A1&!A2&A3&!A4|!A0&!A1&A2&
A3&A4|!A0&!A1&!A2&!A3&!A4 009:
A0&A1&A2|A0&!A1&A3|A0&A3&A4|A0&!A1&A2&!A3|!A0&!A2&A3&A4|!A1&A2&A3&!A4|!A0&A1&A2&!A3&!A4|
!A0&A1&!A2&A3&!A4|!A0&!A1&A2&!A3&A4|!A0&!A1&!A2&!A3&!A4 010:
!A0&A1&A2|!A0&A1&!A3|!A0&A1&A4|!A0&!A1&!A2|!A0&!A2&!A3|!A0&!A2&A4|!A0&A3&A4|!A0&!A3&!A4|!A2&!
A3&A4|A0&!A1&A2&!A3|A0&!A1&!A3&A4 011:
A0&!A1&!A2&A3|A0&A2&A3&!A4|!A1&!A2&!A3&!A4|A0&A1&!A2&!A3&!A4|!A0&A1&A2&!A3&A4|!A0&!A1&A2
&!A3&!A4|!A0&!A1&!A2&!A3&A4 012:
A0&!A2&!A3&A4|A1&A2&A3&!A4|A0&!A1&A2&!A3&A4|!A0&A1&!A2&A3&A4|!A0&!A1&!A2&A3&!A4 013:
A0&!A1&!A2&!A4|A0&!A1&!A3&!A4|!A0&A1&A3&A4|!A0&!A1&A3&!A4|A1&A2&!A3&!A4|A1&!A2&A3&!A4|!A1&
!A2&!A3&A4|A0&!A1&!A2&A3&A4|!A0&!A1&A2&A3&A4 014:
A1&!A2&A4|A0&A1&!A3&A4|A0&!A1&A2&A4|A0&A2&!A3&A4|!A0&A1&A2&A3|!A0&A1&A3&A4|!A0&!A1&!A3&
!A4|!A1&A2&!A3&A4|A0&A1&A2&A3&!A4|!A0&A1&A2&!A3&!A4 015:
A1&!A2|A2&A4|A0&A1&!A3|A0&!A2&!A4|!A0&A1&A3|!A0&A1&A4|!A0&!A1&!A3&!A4|!A0&!A2&!A3&!A4 016:
A0&!A1&!A3&A4|!A0&A1&!A2&A4|!A0&!A1&A2&A4|!A0&!A2&A3&A4|A1&!A2&!A3&!A4|A0&A1&A2&!A3&A4|!
A0&A1&A2&A3&!A4 017:
!A1&A2|A0&!A1&A4|A0&A2&!A4|!A0&!A1&!A4|!A0&A2&!A3|A2&!A3&!A4|A0&!A2&!A3&A4 018:
!A0&A1&A2&!A3|!A0&A2&A3&!A4|A0&!A1&A2&!A3&!A4|A0&!A1&!A2&A3&A4 019:
A0&!A1&!A2&!A3|!A0&A1&!A3&A4|!A1&!A2&!A3&A4|A0&A1&!A2&A3&!A4|A0&A1&!A2&!A3&A4|A0&!A1&!A2
&A3&A4|!A0&!A1&A2&A3&!A4|!A0&!A1&A2&!A3&A4 020:
A0&A1&A3&!A4|A1&A2&A3&A4|A0&A1&!A2&!A3&!A4|A0&!A1&!A2&!A3&A4|!A0&A1&!A2&A3&A4 021:
A0&!A1&A3|!A0&!A1&!A2|!A0&!A2&!A3|!A0&!A2&!A4|!A0&!A3&!A4|A1&A2&A4|!A1&!A2&A3|!A1&!A2&!A4|!A2
&!A3&!A4|!A0&A1&!A3&A4|!A0&A1&!A2&A3&A4 022:
A0&!A1&!A2&A4|A0&!A2&!A3&!A4|A1&!A2&A3&!A4|!A1&!A2&A3&A4|A0&!A1&A2&A3&!A4|!A0&A1&!A2&!A3
&A4|!A0&!A1&A2&A3&A4|!A0&!A1&!A2&!A3&!A4 023:
A0&A2|A1&A2|A2&!A3|A0&A1&A3|A0&A1&A4|A0&A3&!A4|!A0&A1&!A4|!A0&!A1&!A3|!A0&!A3&!A4|A1&A3&!A
4|!A2&A3&!A4 024:
!A0&!A2&!A4|A0&!A1&!A3&A4|A0&A2&!A3&!A4|!A0&A1&!A3&!A4|!A0&!A1&A2&!A3|!A0&!A1&A2&A4|!A0&!A2
&!A3&A4|!A1&A2&!A3&!A4|A0&A1&!A2&!A3&A4|!A0&A1&A2&!A3&A4 025:
!A0&A1&!A2&A4|!A0&A1&A3&A4|!A0&!A1&A2&A4|!A0&A2&A3&!A4|!A0&!A2&!A3&A4|!A1&A2&!A3&!A4|A0&
A1&A2&!A3&A4|A0&!A1&A2&A3&!A4|!A0&A1&!A2&!A3&!A4 026:
A0&A2&A4|!A0&A1&A3|!A0&A1&A4|!A0&!A2&A4|A0&!A2&!A3&!A4|!A0&A2&A3&!A4|A1&A2&!A3&A4|!A1&!A2
&!A3&A4|A0&A1&!A2&!A3&A4|A0&!A1&A2&A3&!A4 027:
A1&A2&A3|A0&A1&A2&!A4|A0&A2&!A3&!A4|!A0&!A1&!A2&!A4|!A0&!A1&!A3&A4|!A0&A2&A3&!A4|A0&A1&!
A2&A3&!A4|A0&!A1&!A2&A3&A4|A0&!A1&!A2&!A3&!A4 028:
!A0&A1&A2&!A4|!A0&!A1&!A2&!A3|!A0&!A1&!A3&A4|!A0&!A2&!A3&A4|!A1&A2&!A3&A4|A0&A1&A2&A3&!A4|
A0&A1&!A2&A3&A4|A0&A1&!A2&!A3&!A4 029:
A0&!A1&A3&!A4|A0&!A1&!A3&A4|A0&A2&A3&!A4|!A0&A1&A2&A4|!A0&!A1&!A2&A3|!A0&A1&!A2&!A3&!A4

030: A0&A1&!A2&!A4|A0&!A1&A3&!A4|!A0&!A1&A3&A4|A0&A1&A2&!A3&!A4|A0&A1&!A2&A3&A4 031:
!A1&!A3|A2&!A3|A2&A4|!A0&A1&A2|!A0&A1&A3|!A0&A1&!A4|A1&!A2&!A4|A0&A1&!A2&A3&A4 032:
A2&!A3|A2&!A4|A0&A1&A2|A0&A1&A3|!A0&A3&!A4|A1&!A2&A3|A1&A3&!A4 033:
A0&!A1&!A2&A3|A0&!A1&A3&!A4|!A0&A1&!A3&A4|!A0&A2&!A3&!A4|!A1&!A2&A3&!A4|!A0&A1&A2&A3&!A4
034:
!A0&!A3&A4|A0&A1&A2&A3|A0&A1&!A3&!A4|A0&A2&A3&A4|!A0&!A1&!A2&!A4|A0&A1&A2&!A3&A4|!A0&A1
&!A2&A3&A4|!A0&A1&!A2&!A3&!A4|!A0&!A1&A2&A3&!A4 035:
A0&A1&A2&A3|A0&A1&!A2&!A3|!A0&!A2&A3&A4|A1&A2&!A3&!A4|A0&!A1&A2&!A3&A4|A0&!A1&!A2&A3&!A
4|!A0&A1&!A2&A3&!A4 036: !A0&!A2&!A4|!A0&A3&!A4|A0&!A1&!A2&!A3|!A0&A1&A2&A4 037:
A0&A1&!A4|A1&A2&!A4|A1&!A2&!A3&A4|!A1&A2&A3&A4|A0&!A1&!A2&!A3&A4|!A0&A1&A2&!A3&A4|!A0&A1
&!A2&A3&A4|!A0&!A1&!A2&!A3&!A4 038:
A0&A1&A3&!A4|!A0&A1&A2&A4|!A0&A1&!A2&!A3|!A0&!A1&A2&!A3|!A0&A2&A3&A4|A0&A1&A2&A3&A4|A0&
!A1&!A2&!A3&!A4|!A0&!A1&!A2&A3&A4 039:
!A0&A1|!A0&A2|A1&!A2|A1&!A3|A1&!A4|A2&!A3|A2&!A4|A3&!A4|A0&!A1&A3|A0&!A2&A3 040:
!A3&!A4|A0&A1&!A2|A0&A1&!A4|A0&!A1&A2|A0&!A1&!A3|A0&!A1&A4|A0&A2&A3|A0&A2&!A4|A0&!A2&A4|A0
&A3&A4|!A0&!A2&!A4|!A1&!A2&!A3|!A0&A1&A2&A3|!A0&A1&A2&A4 041:
!A1&!A2&!A4|A0&!A1&A2&!A3&A4|!A0&!A1&A2&!A3&!A4 042:
A0&A1&!A2|A0&!A2&A4|A0&!A1&A2&A3|A0&!A1&A3&A4|!A0&A2&!A3&A4|A1&!A2&A3&!A4|!A1&A2&A3&A4|!
A0&!A1&A2&!A3&!A4 043:
A0&A1&!A2|!A1&!A2&A4|!A2&!A3&A4|!A0&!A1&!A2&A3|A0&A1&A2&A3&A4|A0&A1&A2&!A3&!A4|A0&!A1&A2
&A3&!A4 044:
!A2&A3|!A2&A4|A0&A1&!A2|A0&!A1&A3|A0&A1&!A3&A4|A0&!A1&A2&!A4|!A0&A1&A3&!A4|!A0&A2&A3&A4
045:
A0&A1&A2&!A3|A0&A1&!A2&A3|A0&!A2&!A3&A4|!A0&!A1&!A3&A4|A1&!A2&!A3&A4|A0&!A1&!A2&A3&A4
046:
A0&!A1&A2&!A3|A0&!A1&A2&!A4|!A0&A1&A2&!A3|!A0&A1&A2&!A4|!A0&A1&!A2&A3|!A0&A1&A3&!A4|!A0&!
A1&!A2&!A3|A0&A1&A2&A3&!A4|A0&A1&!A2&!A3&!A4 047:
!A0&A2&!A3|A1&!A2&A3|A1&A3&!A4|A0&!A1&A2&!A4|A0&A2&A3&!A4|!A0&A1&A2&!A4|!A0&!A2&A3&!A4|!A1
&A2&!A3&!A4|A0&!A1&A2&A3&A4|A0&!A1&!A2&A3&!A4 048:
A0&A2&A4|A0&A1&A2&!A3|A0&A1&!A3&A4|!A0&!A2&A3&A4|!A1&A2&A3&A4|!A0&A1&!A2&!A3&!A4|!A0&!A1
&!A2&A3&!A4 049:
A1&!A3|A0&A2&A4|!A0&!A3&A4|A1&A2&!A4|!A1&A2&A4|A2&!A3&A4|!A0&!A1&!A2&A3&!A4 050:
A0&!A2&A3|A2&A3&!A4|!A2&A3&A4|A0&A1&A2&!A3|A0&!A1&A2&A3|A0&A2&!A3&A4|!A0&!A1&!A2&A4|A1&
A2&!A3&!A4 051: A0&A1&!A2&!A3|!A0&A1&A2&A3|A0&!A1&A2&A3&!A4|!A0&A1&A2&!A3&A4 052:
A0&A1&A3&A4|A0&!A1&A2&!A3&A4|!A0&A1&!A2&!A3&A4 053:
A0&A1|A0&!A3&!A4|A0&!A2&A3&A4|A1&!A2&!A3&A4|!A1&!A2&!A3&!A4|A0&!A1&!A2&!A3&A4|!A0&A1&A2&
A3&!A4|!A0&!A1&A2&!A3&!A4 054:
A0&!A1&A3&A4|A0&!A2&A3&A4|!A0&A2&!A3&A4|A0&A1&A2&!A3&A4|A0&!A1&A2&!A3&!A4|!A0&!A1&!A2&!
A3&!A4 055:
A1&!A3&!A4|A0&A1&!A2&!A3|A0&A2&!A3&A4|A0&!A2&!A3&!A4|!A0&A1&!A2&!A4|A0&!A1&!A2&A3&!A4|!A0&
!A1&A2&A3&!A4|!A0&!A1&!A2&A3&A4 056: A0&!A1&!A2&A4|!A0&A2&!A3&!A4|A0&!A1&A2&!A3&A4 057:
!A0&!A1&!A2|A0&A1&!A2&!A3|A0&A1&A3&!A4|A0&A1&!A3&A4|!A0&A2&!A3&!A4|!A0&A1&!A2&A3&A4 058:
!A0&A1&!A3|!A0&A1&!A4|!A0&!A1&!A2|!A0&!A1&A3|!A0&A2&A3|!A0&!A2&!A4|!A1&!A2&!A3|A0&!A1&!A3&!A4
|A0&!A2&!A3&A4 059:
!A0&!A2&A4|!A1&!A2&!A3|!A1&!A3&!A4|A0&A1&!A3&A4|A0&!A1&A2&!A4|!A0&!A1&A2&!A3|A1&A2&A3&!A4|!
A0&A1&!A2&A3&!A4 060:
A0&A1&A2&A4|A0&!A1&!A2&A3|A0&!A1&!A2&!A4|A0&!A1&!A3&!A4|A0&!A2&!A3&!A4|!A0&!A2&A3&!A4|A0&
A1&A2&A3&!A4|!A0&!A1&A2&A3&!A4 061:
A3&A4|!A0&!A2&A4|A0&A1&A2&A4|A0&!A1&!A2&A3|!A0&!A1&!A2&!A3|!A0&!A1&!A3&!A4|A1&!A2&A3&!A4|!
A1&A2&!A3&!A4|A0&A1&!A2&!A3&A4 062:
A0&!A1&!A2&!A3|!A0&!A2&A3&!A4|!A1&!A2&!A3&!A4|A0&!A1&A2&!A3&!A4|!A0&!A1&!A2&!A3&A4 063:
A0&!A1&A2&!A3&A4 064:
A0&A1&A3&A4|A0&!A1&A2&A4|A0&!A1&!A3&A4|!A0&A1&A2&A4|!A0&!A1&A3&!A4|!A1&A2&!A3&!A4|A0&A1
&A2&!A3&!A4|A0&A1&!A2&A3&!A4 065:
!A0&!A1&A4|A0&!A1&A2&!A3|A0&!A1&A2&A4|A0&A2&A3&A4|!A0&A1&!A2&!A4|!A0&A2&!A3&A4|!A1&!A2&!
A3&!A4|A0&A1&!A2&A3&!A4|!A0&A1&!A2&A3&A4|!A0&!A1&!A2&A3&!A4 066:
A0&A1&!A2&!A3|A1&A2&!A3&!A4|A0&!A1&A2&A3&!A4|A0&!A1&!A2&!A3&!A4|!A0&A1&A2&A3&A4 067:
A0&A1&!A2&!A3|A0&A1&!A3&!A4|!A0&!A1&A2&A4|!A0&!A1&!A3&A4|!A0&A2&A3&!A4|A0&!A1&!A2&!A3&!A4
068:

!A1&!A2&A4|A0&A1&!A2&!A3|A0&A1&A3&!A4|A0&!A1&!A3&!A4|!A0&!A1&!A3&A4|A0&A1&!A2&A3&A4|A0&!
A1&A2&A3&!A4|!A0&A1&!A2&A3&!A4 069:
A2&A3&A4|A0&A1&!A2&!A3|!A0&!A1&A2&A4|!A0&!A1&A3&A4|A0&A1&A2&!A3&A4|A0&!A1&!A2&!A3&!A4|!A
0&A1&A2&!A3&!A4 070:
!A0&!A1&!A3|!A0&!A1&A4|A2&A3&A4|A0&A1&A2&A3|A0&A1&!A2&!A4|A0&!A1&!A2&A4|A0&!A2&!A3&A4|!A0
&A2&!A3&!A4 071:
!A0&A1&!A2&A3|A0&A1&!A2&!A3&A4|!A0&A1&A2&A3&!A4|!A0&A1&A2&!A3&A4|!A0&!A1&A2&!A3&!A4|!A0&
!A1&!A2&A3&A4 072:
A0&A1&A2&!A4|A0&A1&!A3&!A4|!A0&!A1&A2&A4|!A1&A2&!A3&A4|A0&A1&A2&!A3&A4 073:
!A0&A1&A2|!A0&A1&!A3|!A0&A1&!A4|!A0&A2&A4|!A0&!A3&A4|A1&A2&A3|A1&!A2&!A3|A1&!A3&!A4|!A1&!A3
&A4|!A2&!A3&A4 074: A0&A1&A2&A3|!A1&A2&A3&!A4|!A1&A2&!A3&A4|!A0&A1&A2&!A3&!A4 075:
!A0&A1&A2|!A0&A1&!A3|!A0&!A1&!A2|!A0&!A2&!A3|!A0&!A2&!A4|A1&!A3&!A4|!A1&!A2&A4|A0&!A1&A2&A4|!
A0&!A1&A3&A4|!A0&A2&A3&A4|A0&A1&A2&A3&!A4|A0&A1&!A2&A3&A4 076:
!A0&A1|!A0&!A2|A1&A2&!A3|A1&A2&A4|!A1&!A2&A4|A0&A1&!A3&!A4|A1&!A2&!A3&!A4|!A0&!A1&A2&A3&!A
4 077:
!A0&A1|!A0&!A2|A1&A2|!A0&A3&A4|A1&!A2&!A4|!A1&!A2&A3|!A1&!A2&A4|!A2&A3&!A4|A0&A1&!A3&!A4 078:
!A1&!A3&A4|A0&A1&A2&!A3|!A0&!A1&!A3&!A4|A0&A1&!A2&A3&A4 079:
A0&!A1&!A2|A0&A1&A2&A3|A0&A1&A2&!A4|A0&A1&A3&A4|A0&A2&A3&!A4|A0&!A2&!A3&A4|!A0&A1&!A2&!
A3|!A0&A1&!A2&!A4|!A0&!A1&A2&A4|!A0&!A1&!A2&A3|!A0&!A1&A3&A4|A1&A2&A3&!A4|A1&!A2&A3&A4
080:
A0&!A1&A3|A0&!A2&A3|A0&A3&!A4|!A1&A3&A4|A2&A3&!A4|A0&A1&A2&!A3|!A0&!A1&A2&A4|!A0&A1&!A2&
!A3&!A4 081:
A0&!A1&!A2&!A3|!A0&A2&A3&!A4|A1&!A2&!A3&A4|A0&A1&A2&A3&!A4|A0&A1&!A2&A3&A4|!A0&!A1&!A2&
A3&A4 082:
A0&!A1&A4|!A0&!A2&A3|!A1&A3&A4|A0&A1&A2&!A3|A0&!A1&A2&A3|A0&!A1&!A2&!A3|A0&A2&!A3&A4|!A0
&A1&!A2&!A4|!A0&A1&A2&A3&A4 083:
A0&A1&!A3&A4|A0&!A1&!A3&!A4|A0&A2&!A3&!A4|!A0&!A2&A3&A4|A1&A2&!A3&!A4|!A1&!A2&A3&A4|A0&!
A1&A2&A3&!A4 084:
A0&A1&A4|A0&!A2&A4|A0&A1&A2&!A3|A0&!A1&!A2&!A3|!A0&A1&!A3&!A4|A1&A2&!A3&!A4|!A0&!A1&A2&A
3&!A4 085: A0&A1&A2&!A3&!A4|A0&A1&!A2&!A3&A4|A0&!A1&!A2&A3&A4|!A0&A1&!A2&A3&A4 086:
!A0&!A2|A0&A1&A4|A0&A3&A4|A1&!A2&A4|!A2&A3&A4|!A0&A1&!A3&!A4|!A0&!A1&!A3&A4 087:
A1&!A2&A3|!A0&!A1&!A3&!A4|A1&A2&A3&A4|A0&A1&!A2&!A3&!A4|!A0&A1&A2&!A3&A4|!A0&!A1&!A2&!A3
&A4 088:
A0&!A1&!A2|A0&!A1&!A3|A0&!A1&A4|A0&!A2&A3&!A4|!A0&!A1&A2&A3|!A0&A2&A3&A4|!A1&A2&A3&A4|!A0
&A1&!A2&!A3&!A4 089:
A0&!A1&A2&A4|!A0&A1&!A2&A4|!A0&!A1&A2&!A3|A0&A1&A2&!A3&!A4|A0&A1&!A2&A3&A4|A0&!A1&A2&A
3&!A4|!A0&A1&A2&!A3&A4|!A0&!A1&A2&A3&A4 090:
A0&A2&A3&!A4|!A1&A2&!A3&A4|!A1&!A2&A3&!A4|A0&A1&A2&A3&A4|A0&!A1&!A2&!A3&!A4|!A0&!A1&A2&
A3&!A4|!A0&!A1&!A2&A3&A4 091:
A0&!A2&!A3|!A0&A1&A4|A1&!A3&A4|!A2&!A3&A4|A0&A1&!A2&!A4|A0&A1&A3&!A4|A0&!A1&A2&A3|A0&!A1
&!A2&A4|A0&!A1&A3&A4|A0&!A1&!A3&!A4|A0&A2&A3&!A4|A1&!A2&A3&!A4 092:
A1&A2|A1&!A3|A1&!A4|!A2&!A3|!A3&!A4|A0&A2&A3|A0&A2&!A4|!A0&!A2&A4|!A1&!A2&A4|A0&!A1&A3&A4
093:
!A0&!A1&!A2&!A3|!A0&!A1&!A3&A4|A1&!A2&A3&A4|A0&!A1&A2&!A3&!A4|A0&!A1&!A2&A3&!A4|A0&!A1&!A
2&!A3&A4|!A0&A1&A2&A3&!A4|!A0&A1&A2&!A3&A4 094:
A0&A1&A2&!A3|A0&A1&!A2&A4|!A0&A1&A2&!A4|!A0&A1&A3&A4|!A0&!A1&A2&A4|!A0&A2&A3&A4|A1&!A2&
A3&!A4|A0&!A1&!A2&!A3&!A4|!A0&!A1&!A2&A3&!A4 095:
!A0&A1&A3&!A4|!A0&!A1&!A3&!A4|A1&A2&A3&A4|A0&!A1&A2&A3&!A4|!A0&A1&!A2&!A3&A4 096:
A0&!A2&A3&A4|!A0&A1&!A2&!A3|!A0&!A1&A3&A4|!A0&!A1&!A3&!A4|!A0&A2&A3&A4|!A1&A2&!A3&!A4|!A1&
!A2&A3&!A4|A0&A1&A2&A3&A4|A0&A1&!A2&!A3&!A4|!A0&A1&!A2&A3&A4|!A0&!A1&A2&!A3&A4 097:
!A1&A2&A3|A0&!A1&A2&A4|!A0&A1&A2&!A4|!A1&A2&!A3&A4|!A0&A1&!A2&A3&A4 098:
A0&!A3&A4|A0&A1&A2&!A3|A1&A2&!A3&!A4|!A1&A2&!A3&A4|A0&!A1&A3&!A4|!A1&!A2&A3&!A4|A0&!A1&A
2&A3|!A0&A1&A2&A3|!A0&A1&A3&A4|A0&!A1&A2&A4 099:
A0&A4|A0&A1&!A3|A0&!A1&A3|A0&A2&A3|!A0&A2&!A3|A1&A2&A4|A1&A3&A4|A2&!A3&A4|!A0&!A2&A3&!A4|
A0&A1&!A2&A3&!A4|!A0&A1&!A2&!A3&!A4 100:
!A0&!A1&!A3|!A0&A2&!A3|A0&!A2&A3&A4|A0&!A1&A2&A3&!A4 101:
A0&A1&A2|!A1&A3&!A4|A0&A1&!A3&!A4|A0&A2&A3&!A4|A0&A2&!A3&A4|!A0&A1&!A3&A4|A0&!A1&!A2&!A3
&!A4|!A0&A1&!A2&A3&!A4|!A0&!A1&!A2&A3&A4 102:
A0&A1|A0&A2&A4|A0&A3&!A4|!A0&!A1&!A4|!A1&A3&!A4|!A2&A3&!A4|!A0&A2&!A3&!A4|A1&A2&A3&A4 103:

A0&!A2&A3&A4|!A0&A1&!A2&A3|!A1&!A2&A3&!A4|A0&A1&!A2&!A3&A4|A0&!A1&A2&A3&A4|!A0&A1&A2&A3&!A4|!A0&!A1&!A2&!A3&!A4 104:
A0&A1&A3&!A4|A0&!A1&!A2&A4|!A0&A2&!A3&A4|A0&A1&A2&!A3&A4|A0&!A1&A2&!A3&!A4|!A0&A1&A2&!A3&!A4|!A0&A1&!A2&!A3&A4|!A0&!A1&A2&A3&A4 105:
A0&A1&A3|!A0&!A1&!A4|!A0&A2&A3|!A0&A2&!A4|!A0&!A3&!A4|A1&A2&A3|A1&A2&!A4|!A1&!A2&!A4|!A1&!A3&!A4|A2&A3&A4|A2&!A3&!A4|A0&!A1&A2&A4|!A0&A1&!A2&!A3|A0&!A1&A2&!A3|A0&A1&!A2&A4 106:
A0&!A1&!A2&A3|A0&!A2&!A3&A4|!A0&A2&A3&A4|A1&A2&!A3&!A4|A1&!A2&A3&A4|A0&A1&A2&A3&!A4|A0&!A1&A2&A3&A4|A0&!A1&A2&!A3&!A4|!A0&A1&!A2&!A3&A4|!A0&!A1&A2&!A3&A4|!A0&!A1&!A2&A3&!A4
107:
!A0&!A3|!A0&A1&!A4|!A0&A2&!A4|A0&A1&A2&A3|A0&!A1&A2&A4|!A0&!A1&!A2&A4|A1&A2&!A3&!A4|!A1&!A2&A3&A4 108:
!A0&!A2&!A3|A0&A1&A3&!A4|A0&!A1&!A2&!A4|!A0&A1&A3&A4|!A0&!A2&A3&!A4|!A0&A1&A2&!A3&!A4|!A0&!A1&A2&A3&!A4|!A0&!A1&!A2&A3&A4 109:
A0&A1&!A4|A0&!A1&A4|!A0&!A1&A2|!A0&A2&A3|!A0&A2&A4|!A0&A3&A4|!A1&A2&A3|!A1&A2&A4|!A1&A3&A4|A2&A3&!A4 110: A0&A1&!A3|A0&A1&!A2&A4|!A0&!A2&A3&A4|!A0&A1&!A2&A3&!A4 111:
!A2&!A3&A4|!A0&A1&!A2&!A3|A0&A1&A2&A3&A4|A0&A1&A2&!A3&!A4|A0&A1&!A2&A3&!A4|A0&!A1&!A2&!A3&!A4 112:
!A0&A3&!A4|!A0&A1&!A3&!A4|!A1&!A2&A3&A4|A0&A1&A2&A3&A4|A0&A1&A2&!A3&!A4|A0&A1&!A2&A3&!A4|A0&!A1&A2&A3&!A4|A0&!A1&A2&!A3&A4|!A0&!A1&A2&A3&A4 113:
!A0&A1&A2&!A3|!A0&A1&A3&!A4|!A0&A1&!A3&A4|!A0&!A1&A3&A4|A0&A1&A2&A3&!A4|A0&A1&A2&!A3&A4|A0&A1&!A2&!A3&!A4|A0&!A1&A2&!A3&!A4|A0&!A1&!A2&A3&!A4|!A0&!A1&!A2&!A3&!A4 114:
!A1&A3&!A4|A0&!A2&A3&!A4|!A0&A2&A3&A4|!A0&!A2&!A3&!A4|A1&A2&A3&A4|!A1&A2&!A3&A4 115:
!A0&A3|!A0&!A1&!A4|!A0&!A2&!A4|!A1&A3&!A4|!A2&A3&!A4|A1&A2&A3&A4 116:
!A0&!A1&!A3&!A4|!A0&A2&A3&!A4|!A0&!A2&!A3&A4|A1&A2&A3&A4|!A1&A2&!A3&A4|A0&A1&!A2&!A3&A4
117:
A0&A1&A2&!A3|A0&A2&A3&!A4|A0&!A2&!A3&!A4|!A0&!A2&!A3&A4|!A1&!A2&A3&A4|A0&!A1&A2&!A3&A4|!A0&A1&!A2&A3&A4 118:
!A0&!A4|!A0&A1&A2|!A0&A2&A3|A1&!A2&A3|A1&A3&A4|!A2&A3&!A4|A0&A1&!A2&A4|A0&!A2&!A3&A4 119:
A0&A2&!A3|!A0&!A2&!A4|!A0&A1&!A2&A4|A1&A2&!A3&!A4|A0&A1&A2&A3&A4|A0&!A1&A2&A3&!A4 120:
!A0&!A1&!A3&A4|A0&A1&!A2&!A3&A4|A0&!A1&A2&!A3&!A4|!A0&A1&!A2&A3&A4|!A0&!A1&!A2&A3&!A4
121:
A0&A1&A2&A3|A0&A2&A3&!A4|A0&A1&!A2&!A3&!A4|!A0&A1&A2&!A3&A4|!A0&!A1&A2&!A3&!A4|!A0&!A1&!A2&A3&A4 122:
A0&A1&A2&!A4|A0&A1&!A3&!A4|A0&!A1&A2&A4|A0&!A1&!A2&A3|!A0&A1&!A2&A4|!A0&!A1&A2&!A3|A0&A1&A2&A3&A4|!A0&!A1&!A2&A3&!A4 123:
A0&!A1&A2|!A0&!A3&!A4|!A0&A1&A2&!A3|!A0&!A1&!A2&!A4|A1&A2&!A3&A4|!A1&A2&A3&A4|!A1&A2&!A3&!A4 124: A1&!A3&!A4|A0&A2&A3&!A4|A0&!A2&!A3&A4|!A0&A1&A3&!A4|!A0&!A1&A2&!A3|A1&!A2&!A3&A4
125: A0&!A3&!A4|!A0&A2&A3&!A4|!A0&!A2&A3&A4|A0&!A1&!A2&!A3&A4 126:
A0&!A1&A2&A4|A0&!A1&A3&!A4|!A0&A1&A2&!A4|!A0&!A1&!A2&A4|!A1&A2&A3&A4|A0&A1&A2&A3&A4|A0&A1&!A2&!A3&A4|A0&!A1&!A2&!A3&!A4|!A0&A1&!A2&!A3&!A4 127:
!A1&A3&!A4|A0&A2&A3&!A4|!A0&A1&A3&A4|A0&A1&!A2&A3&A4|!A0&!A1&!A2&!A3&A4