Summer Research Internship Report

A Study on MD-5, Wang's attack on MD-5 and APOP attack

Submitted by-

Raagi Sukhlecha (200901148)

Lalit Agarwal (200901159)

Mentor- Prof. Anish Mathuria, DA-IICT

Start Date of Internship- 14th May, 2012

End Date of internship- 13th July, 2012

Preface

This report documents the work done during the summer internship at DA-IICT under the supervision of Prof. Anish Mathuria. The report will give an overview of the topics we have covered with their technical details. The results and the inferences are discussed and analysed.

This study involved understanding the attack on APOP given by Gaetan Leurent in his paper "Practical key recovery attack against APOP, an MD5 based challenge response authentication", Int. J. of Applied Cryptography, 2008. Understanding the attack required, understanding MD-5, Wang's attack and APOP challenge response mechanism. Apart from this, it also covered a brief study of some practical attacks based on Wang's attack.

Raagi Sukhlecha
Lalit Agarwal

Contents

Introduction 4

The Goals 4

MD-5 4

Length padding in MD-5 7

Why padding length should be as small as possible? 7

Why is the padding necessary? 7

Attack on MD-5 8

Wang's Attack 9

APOP 11

The Mechanism 11

The Attack 12

The Approach 12

The Algorithm 13

The Attack in practice 14

Message Freedom 14

Attack Complexity 14

Conclusion 14

Future Scope 15

References 15

Introduction

APOP ("Authenticated Post Office Protocol") is an extension of the Post Office Protocol (POP) in which the password is sent in encrypted form. With standard POP, the user name and password are sent in plain text over the network which can be intercepted by a malicious third party. APOP provides a simple challenge-response authentication which uses the MD5 hash function in an attempt to avoid replay attacks and disclosure of the shared secret. Some of the mail clients implementing APOP include Mozilla Thunderbird, Opera Mail, Microsoft Outlook, KMail.

However, there have been attacks which use colliding messages to recover part of the password used in the APOP authentication protocol. Since we actually need a little more than mere collisions, we look into the details of MD5 collisions. In Wang's attack, message modifications allow to deterministically satisfy certain sufficient conditions to find collisions efficiently. Unfortunately, message modifications significantly change the messages and one has little control over the colliding blocks. The attack presented in the report allows choosing small parts of the colliding messages, which will allow building the APOP attack. This shows that collision attacks can be used to attack real protocols, which means that finding collisions is a real threat.

The Goals

- To understand the internal working of MD5.
- To find the motivation behind including length padding in the implementation of the MD5 Algorithm.
- To gain an insight into one of the major attacks on MD5- Wang's Attack.
- Understand the APOP and the attack against APOP given by Gaetan Leurent in the paper "Practical key recovery attack against APOP, an MD5 based challenge response authentication", Int. J. of Applied Cryptography, 2008.
- Implement and simulate the attack.

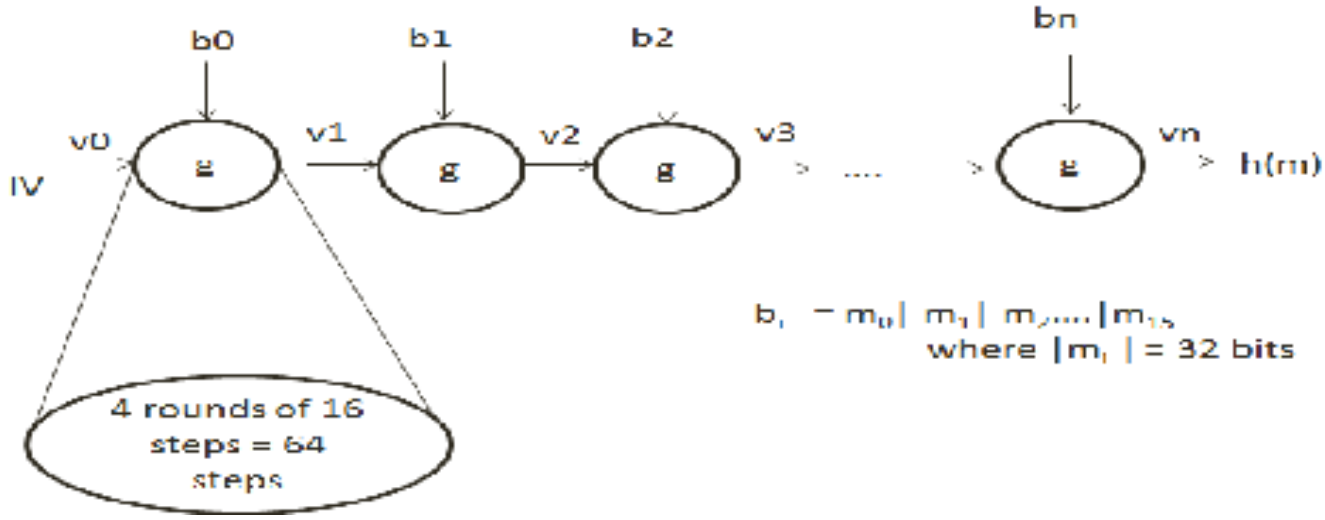
MD-5

MD-5 (Message digest -5) is a hashing algorithm based on Merkle damgard construction[1]. It takes in message blocks, pads them with message length to make them multiple of 512 bits and produces 128 bit output.

The compression function takes an initialization vector of 128 bits and a message block of 512 bits; the output of this compression function (128 bits) is then given as input to the next compression function. Each compression function in turn consists of 4 rounds of 16 steps each (64 steps). Each message block of 512 bits consists of 16 32 bits word; therefore each round uses one word of the message block. Output of round i is Q_i and initialization vector is given as (Q_{-4},Q_{-3},Q_{-2},Q_{-1}).The step function for round i is given as :

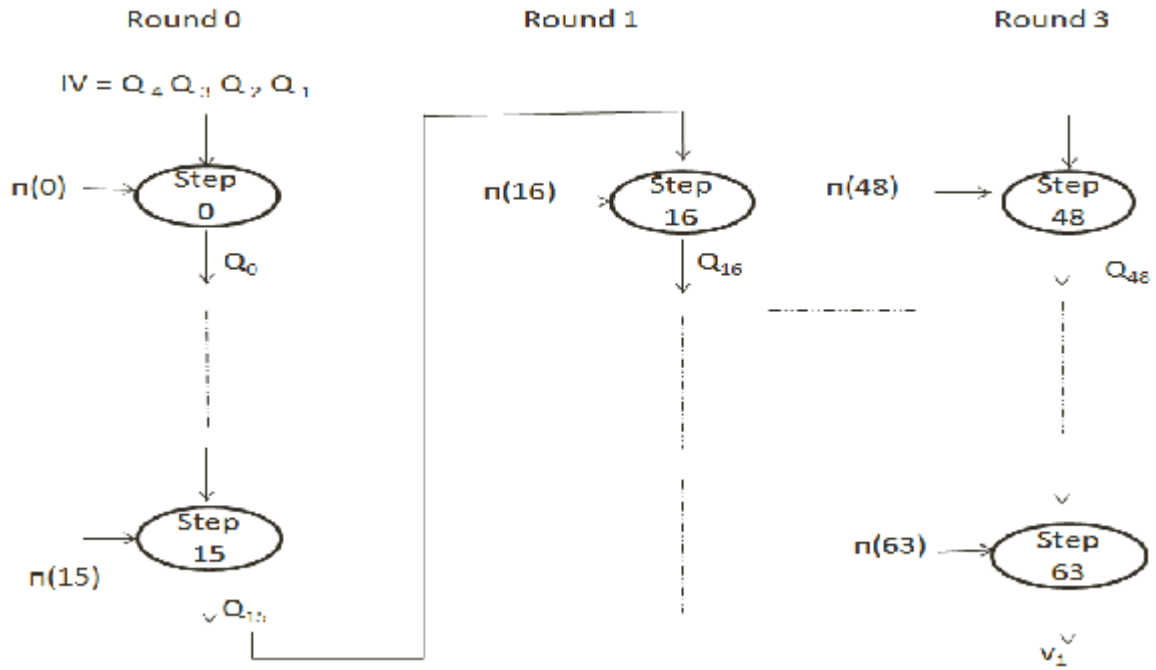
$$Q_i = (Q_{i-4} + f_i + m_i + k_i) <<< s_i$$

Where f_i is function on Q_{i-3}, Q_{i-2}, Q_{i-1}, m_i is the message word and the message word chosen depends upon the predefined permutation, k_i and s_i are constants. The addition is modular addition mod 2^{32}. f_i , m_i , k_i , s_i are so chosen to produce larger avalanche effect and to make hashing function resistant to collisions.



Merkle Damgard Construction

[1] Included in note



Rounds of MD-5

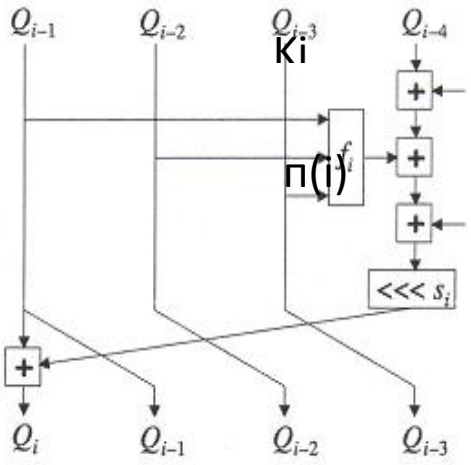


Figure 3: MD 5 Step

Note:

Merkel Damgard Hash Function

Markle-Damgard or MD hash function has three basic components namely,
(1) an underlying compression function f : $\{0, 1\}^{b+t} \rightarrow \{0, 1\}t$ for some $b > 0$,
(2) an initial value IV $\in \{0, 1\}^t$
(3) an easily computable padding rule pad : $M \rightarrow (\{0, 1\}_b)^+$ for some message space M.

Classical iterated function f_{IV}^{+} is defined as:

$(\{0, 1\}^{b})^{+} \rightarrow \{0, 1\}^{t}$
as $f_{IV}(M1, \cdots, M) = f(f(\cdots f(IV, M1), \cdots), M), M1, \cdots, M \in \{0, 1\}$.

The MD hash function is therefore calculated as follows:
- padding(M) $\rightarrow (\{0, 1\}^{b})^{+}$
- f_{IV}^{+} (padding(M))$\rightarrow \{0, 1\}^{t}$

Length padding in MD-5

To make each message block of 512 bits, message block is padded with 1 followed by zeros and the message length. This padding is suffix free and is required to prevent attack on the hash function.

Why padding length should be as small as possible?

If the size of padded bits is more, it may cost more invocations of the underlying compression function. So, in terms of efficiency of hash function, one should try to keep size of pad as small as possible.

Why is the padding necessary?

Suffix free padding: Let X, Y $\in \{0, 1\}^{*}$. We call X a suffix of Y if there exists a binary string Z such that Y = Z X. A padding rule pad is called suffix-free if, for any M != M', pad (M) is not a suffix of pad (M').

The lemma[2] on suffix free padding says that if we have collision for iterated hash f^{+} [3](i.e., $f^{+}_{h}(X) = f^{+}_{h'}(X')$) for distinct inputs of with same length then there must be an intermediate collision during computations of $f^{+}_{h}(X)$ and $f^{+}_{h'}(X')$.

Suffix free padding is sufficient condition for collision preserving hash function

If pad is suffix-free padding rule then given any collision pair (M, M') for MD^{f}_{pad} we can construct a collision pair of compression function f efficiently.
Proof:
Let pad (M) = X and pad (M') = X'. Without loss of generality we assume that $|X| \leq |X'|$.
$X \in (\{0, 1\}^{b})^{k}$ and X' = (Z, Y) where $Y \in (\{0, 1\}^{b})^{k}$ and $Z \in (\{0, 1\}^{b})^{+}$
h'= $f_{h}(Z)$.
Since, $f_{h}(X) = f_{h}(X')$

[2] Refer to References 5.
[3] It is assumed throughout that compression function f is so chosen that it is collision resistant

hence $f_h(X) = f_{h'}(Y)$ where $X,Y \in (\{0,1\}^b)^k$
Since, X != Y as pad is suffix-free.

Therefore by lemma given above for two distinct inputs if a hash function gives a collision then the compression function (f) has a collision and the thus the collision advantage for f is at least the collision advantage of MD hash function.

Suffix free padding is necessary condition for collision preserving hash function

Consider a padding rule which is not suffix free, then the hash function is not collision resistant even though the underlying compression function is collision resistant.

Proof:
Consider a compression function which is collision resistant and let M and M' be messages such the pad (M) is suffix of pad (M') (= Xm || pad (M), where $X = (\{0,1\}^b)^*$ and $m = \{0,1\}^b$).

Case 1 : X = λ (empty string) i.e. pad (M') = m || pad (M)
Define a function f such that

f(x) = f'(x)||0 for x != IV || m
= IV for x = IV || m

Assume that last bit of IV = 1
since f'(x) is collision resistant and for x= IV||m and x!= IV||m there won't be any collision since the last bit of f(x) do not match in that case.
Therefore, there is a collision : $H^f_{pad}(M') = f_{IV}^+(m \,||\, pad(M)) = f_{IV}^+(pad(M)) = H^f_{pad}(M)$
Thus , $H^f_{pad}(M) = H^f_{pad}(M')$

Case 2 : $X = (\{0,1\}^b)^+$

Let IV' = f''(X) where f''(x)= f'(x) || 0 and f'(x)is a collision resistant function.
Let the compression function be

f(x) = f'(x)||0 for x!= IV'||m
= IV for x= IV'||m

$H^f_{pad}(M') = f_{IV}^+(Xm \,||\, pad(M)) = f_{IV'}^+(m \,||\, pad(M)) = f_{IV}^+(pad(M)) = H^f_{pad}(M)$
Thus , $H^f_{pad}(M) = H^f_{pad}(M')$

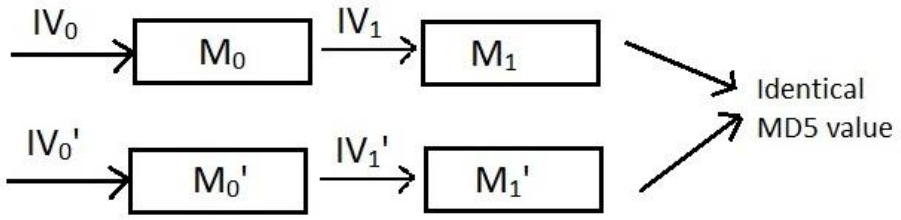
Attack on MD-5

Though Dobertin attack (of MD-4) was not completely successful on MD-5 yet it proved that even MD-5 is not resistant to collision and an algorithm implementing attack on it can be found out. In 2004, Xiaoyun Wang published an attack on MD-5, popularly known as the

"Chinese Attack". Only one pair of colliding block was given and nothing was known about the algorithm behind the attack. Reverse engineering of the attack provided useful information about the computational part of the attack.

Wang's Attack

Wang's attack is a differential attack which finds out two 1024 bits of colliding message blocks and involves differences in output and input blocks[4].



Wang's attack can be viewed as consisting of the following steps:

1. **Pre computational phase** :
    a. Specifying an input difference that is let M and M' be two message blocks then specifying M – M'. In Wang's attack ,
    $\Delta M_0 = M_0' - M_0 = (0, 0, 0, 0, 2^{31}, 0,0,0,0,0,2^{15},0,0,2^{31},0)$
    $\Delta M_1 = M_1' - M_1 = (0, 0, 0, 0, 2^{31}, 0,0,0,0,0,-2^{15},0,0,2^{31},0)$

    b. Specifying the output difference, this is the most mysterious part of the attack and till date all the attacks are based on Wang's differential patterns.

[4] Output differences are difference in output of step i of the compression function ie $Q_i - Q'_i$. Input differences are differences in message block.
Output differences are signed difference and are more restrictive. Input difference is modular difference.
Signed difference, $\nabla y = y' - y$
Denote $y'_i=1$, $y_i=0$ as "+"
Denote $y'_i=0$, $y_i=1$ as "–"
Denote $y'_i=y_i$ as "."
Consider bytes
z' = 10100101 and z = 10010101
Then ∇z is "..+-...."
It is more **restrictive** than modular subtraction.

j	Output	W_j	ΔW_j	ΔOutput	∇Output
4	Q_4	X_4	2^{31}	$\bar{6}$	..... .-++++++ ++++++++ ++.....
5	Q_5	X_5	0	$+\ +\ -$ 31 23 6	+..... +..... ..... .-.....
6	Q_6	X_6	0	$-\ +\ -\ -$ 27 23 6 0	++++++-- -..... ....-+++ ++-+++++
7	Q_7	X_7	0	$-\ -\ -\ +$ 23 17 15 0	..... -..-+++- +..... .....+
8	Q_8	X_8	0	$+\ -\ +$ 31 6 0	-..... ....- ++....+-

Some of the Output Differences

c. Given the differential patterns, derive a set of sufficient conditions on the output (along with the necessary conditions on intermediate values).

	Conditions on M_0				Number
Q_2	.....	....0...	....0...	.0.....	3
Q_3	1.....	0^^^1^^^	^^^^1^^^	^011....	21
Q_4	1000100.	01..0000	00000000	0010.1.1	27
Q_5	0000001^	01111111	10111100	0100^0^1	32
Q_6	00000011	11111110	11111000	00100000	32
Q_7	00000001	1..10001	0.0.0101	01000000	28
Q_8	11111011	...10000	0.1^1111	00111101	28
Q_9	0111....	0..11111	1101...0	01....00	19
Q_{10}	00100000	1...0001	11000000	11000010	29
Q_{11}	000...00	1000	0001...1	0.....	15
Q_{12}	01....01	1111	111....0	0...1...	14
Q_{13}	0.0...00	1011	111....1	1...1...	14
Q_{14}	0.1...01	0	1.....	....0...	7
Q_{15}	0!1.....	.....!.	.....	.....	4
Q_{16}	0!.....	.....0.	^.....	....^...	5
Q_{17}	0.^.....	.....1.	.....	.....	3
Q_{18}	0.....	.....0.	.....	.....	2
Q_{19}	0.....	....!..	.....	.....	2
Q_{20}	0.....	.....^.	.....	.....	2
				Subtotal	287

A subset of the Sufficient Conditions

2. **Computational part of Wang's** : determine a pair of 1024 bits of block for which these conditions will satisfy :
   a. Generate a random 512 message M0.
   b. Use single step modification – direct modification of the message word to satisfy output conditions in the early steps (0-15).
      i. Select a message m_i
      ii. Compute the corresponding Q_i
      iii. Modify Q_i to satisfy the conditions.
      iv. Re-compute m_i
   c. Use multi step modifications – modification of the message such that previously satisfied conditions still hold – to satisfy condition in the steps 16-31.

i. Compute Q_i from the message m_i
ii. Modify Q_i to satisfy the conditions and re-compute m_i
iii. Re-compute Q_i's and m_i's in the first round.

d. Check conditions for all the other steps and if not satisfied go to 4b. This is satisfied probabilistically and input difference was chosen in a way that this probability is high.
e. Similarly, find out M1
Compute $M_0' = M_0 + \Delta M_0$
$M_1' = M_1 + \Delta M_1$

The MD5 of (M_0, M_1) is same as that of (M_0', M_1')

Wang gave only one pair of collision which is-

M_0 = 2dd31d1 c4eee6c5 69a3d69 5cf9af98 **8**7b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a41**7**125 e8255108 9fc9cdf7 **f**2bd1dd9 5b3c3780
M1=d11d0b96 9c7b41dc f497d8e4 d555655a **c**79a7335 cfdebf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 5c15**c**c79 ddcb74ed 6dd3c55f **d**80a9bb1 e3a7cc35

M_0'=2dd31d1 c4eee6c5 69a3d69 5cf9af98 7b5ca2f ab7e4612 3e580440 897ffbb8 634ad55 2b3f409 8388e483 5a41**f**125 e8255108 9fc9cdf7 **7**2bd1dd9 5b3c3780
M1'=d11d0b96 9c7b41dc f497d8e4 d555655a **4**79a7335 cfdebf0 66f12930 8fb109d1 797f2775 eb5cd530 baade822 5c15**4**c79 ddcb74ed 6dd3c55f **5**80a9bb1 e3a7cc35

Hash: 9603161f a30f9dbf 9f65ffbc f41fc7ef

APOP

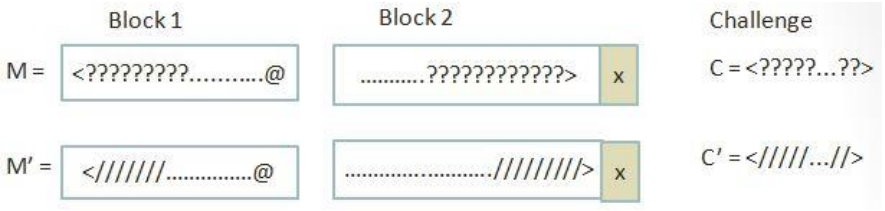
The Mechanism

The server implementing the APOP command sends a challenge (msg-id) in their greeting message and the client authenticates itself by sending the username and the MD5 of the challenge concatenated with the password i.e. MD5(msg||passwd). The server performs the computation on his side and checks if the digests match. Since the message is encrypted using MD5, the eavesdropper will not be able to learn the password.

In the above example, the password is ‘penguin’ and hence the digest is MD5 (‘<171.11776027@pop.mail.com>penguin’).

The Attack

The server creates a pair of specially crafted challenges such that the corresponding digest will collide only if some part of the password was correctly guessed. For each character in the password, the server will create 256 pairs of challenges for each of the possible 256 possible ASCII characters. Thus the server can learn the password in linear time one by one.



Colliding Messages M and M’ when p_0=x, C and C’ are the corresponding challenges

In the above example if the server guesses that the first password character is ‘x’, so it will create the corresponding challenges C and C’ and send the challenges to the client. The client will respond to the challenge with the messages M and M’ where

$$M = \text{<???????.....???????>}p_0p_1p_2p_3...p_{n-1}$$

$$M' = \text{</////////.....//////////>}\ p_0p_1p_2p_3...p_{n-1},$$

where ‘?’ and ‘/’ represent any character chosen by the collision finding algorithm

If p_0=x, the two hashes will collide after the second block the two hashes will collide after the second block, and since the end of the password and the padding are the same, we will see this collision in the full hashes. Therefore we are able to test the first password character without knowing the others.

The Approach

Since we need to generate collisions with a specific format and some chosen characters, message freedom is required. It assumes that we are given a set of sufficient conditions on the internal variables Q_i that produces collision. This Approach tries to find a message M such that when one computes a hash of the message, the conditions on the internal state variables Q_i hold. The approach differs from other approaches as instead of modifying the message to

satisfy some conditions, we first choose the Q_i's and then find the corresponding messages. Also it gives a generic algorithm that can take any path as input like Wang, Klima, Stevens.

The Algorithm

We choose a point of choice, p_c (first step whose condition will not be satisfied deterministically) and a point of verification, p_v (step where we will start using tunnels). Since we want $\pi(16) < \pi(17) < ... < \pi(pc-1) < 12$, so we choose $p_c=19$ and p_v will be 23 for MD4. While for MD5, the respective values of p_c and p_v are 19 and 24. The choice of p_v is such that we will use the tunnels only for the third round.

If we need to generate collisions where the last t words are chosen, the algorithm proceeds as follows-

1) The algorithm begins by fixing the Q_i's from the middle which allows dealing with the first round as well as beginning of the second round at the same round. We begin by fixing Q_{12-t}, Q_{13-t}, Q_{14-t}, Q_{15-t}.
2) We compute Q_{16-t} to Q_{15} if the conditions on the first state Q_{16-t} are not satisfied. We might have to try many choices of Q_{12-t}, Q_{13-t}, Q_{14-t}, Q_{15-t} so that the conditions are fulfilled.
3) We will choose the Q_i from step 0 to $\pi(p_c-1)$ and when we hit a message m_i that is also used in the second round with $i= \pi(j)$, we can modify it to generate a good Q_j. Thus we can fulfil the conditions up to round p_c-1 almost for free.
4) In the end, we will make random choices for the remaining steps ($Q_{(pc-1)}$ to Q_{11}), until we have a message that follows the path up to step p_v-1 and finally we use the tunnels.

In case t=2 i.e. we need to fix the last 2 words, the algorithm will proceed as follows-

Assumption- We are given a set of conditions on the Q_i's which needs to be satisfied.

1) Fix Q_{10}, Q_{11}, Q_{12}, Q_{13} so that the condition on these Q_i's hold.
2) Compute Q_{14}, Q_{15} from the previous Qi fixed in step 1.
3) Check if the condition on Q_{14} is satisfied. If not, try other possible choices for Q_{10}, Q_{11}, Q_{12}, Q_{13}.
4) Choose the Q_i from step 0 to $\pi(p_c-1)$ and when we hit a message m_i that is also used in the second round with $i= \pi(j)$, we can modify it to generate a good Q_j. Thus we can fulfil the conditions up to round p_c-1 almost for free. For MD4 and MD5, the value of p_c is 19.
5) Select random values for the steps ($Q_{(pc-1)}$ to Q_{11}), until we have a message that follows the path up to step p_v-1
6) Finally we use the tunnels for the steps Q_{pv} to N. For MD4 and MD5, the value of p_v is 23 and 24 respectively.

The Attack in practice

In order to implement this attack, we need to efficiently generate MD5 collisions with some chosen parts i.e. we mainly have to fix the last word. Additionally the POP3 RFC requires the challenge to be a msg-id, which means that:-

- It has to begin with '<' and end with '>'.
- It must contain exactly one '@', and the remaining characters are very restricted, in particular, they should be ASCII characters
- There are only four characters which are rejected in the challenge:
    - 0x00 Null: used as end-of-string in the C language
    - 0x3e Greater-Than Sign ('>'): used to mark the end of the msg-id
    - 0x0a Line-Feed: used for end-of-line (POP is a text-based protocol)
    - 0x0d Carriage-Return: also used for end-of-line

A Challenge Response

Message Freedom

Using this approach, we can choose last three message words in a one-block MD4 collision and three specific message words in a two-block MD5 collision. For MD5, the first block is computed only once and includes '<' and '@'. In order to learn the i^{th} password character, we need to fix the last (i+1) characters which are '>' and (i-1) password characters.

Using the attack, we can only recover the first three characters of the password. For MD5, the Wang's path uses a message difference in m_{14}. Due to the message difference we cannot modify the message m_{14} and hence can only recover 3 characters of a password.

Attack Complexity

If we assume that the password is 8 characters long and each characters has 6 bits of entropy, then we need to generate $3*2^5$ collisions and wait for about $3*2^6$ identifications. If each collision takes 5 sec. to generate, then it will take about 3 hours to recover the last three characters of the password. If we try to recover the first three characters of the password using a brute force attack, it will require 256*256*256 computations while this attack only requires 256+256+256 computations.

Conclusion

We studied and understood the theoretical aspect of the attack against APOP. Also in order to understand the attack, we gained an insight into the working of MD-5 and Wang's Attack. Due to security issues, we were not able to get the details of the implementation of the attack from the author and hence were not able to simulate the attack on a PC.

Here are our findings and conclusion

1. Suffix free length padding is necessary and sufficient condition for collision free hash function given that the underlying compression function is collision free.
2. Wang's attack can be used to break security of MD-5 but the colliding blocks look random and therefore have to be hidden.
3. The algorithm by Gaetan Leurent gives some message freedom in the colliding blocks and thus can be used to craft a practical attack against APOP used by most of the mail clients.

Future Scope

Further work in this area of study can help in finding out an algorithm, for practical implementation and simulation of attack which our team was not able to do.

References

1. Ch5, Applied Cryptanalysis – Breaking ciphers in real world, Mark Stamp and Richard M. Low.

2. Hashing in Computer Science: Fifty Years of Slicing and Dicing, Alan G.Konheim.

3. Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications , Klima, IACR Eprint archive, 2005

4. Tunnels in Hash Functions: MD5 Collisions Within a Minute , Klima, Cryptology ePrint Archive, Report 2006/105

5. Characterizing Padding Rules of MD Hash Functions , Preserving Collision Security , Mridul Nandi, ACISP '09 Proceedings of the 14th Australasian Conference on Information Security and Privacy