Message Freedom in MD4 and MD5 Collisions: Application to APOP

Gaëtan Leurent

Laboratoire d'Informatique de l'École Normale Supérieure,
Département d'Informatique,
45 rue d'Ulm, 75230 Paris Cedex 05, France
gaetan.leurent@ens.fr

Abstract. 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. In this paper, we show how to choose small parts of the colliding messages. Consequently, we break a security countermeasure proposed by Szydlo and Yin at CT-RSA '06, where a fixed padding is added at the end of each block.
Furthermore, we also apply this technique to recover part of the passwords in the Authentication Protocol of the Post Office Protocol (POP). This shows that collision attacks can be used to attack real protocols, which means that finding collisions is a real threat.
Key words: Hash function, MD4, MD5, message modification, meaningful collisions, APOP security

1 Introduction

At EUROCRYPT '05 and CRYPTO '05, Wang *et al.* described a new class of attacks on most of the hash functions of the MD4 family, MD4, MD5, HAVAL, RIPEMD, SHA-0 and SHA-1 in [21,23,24,22], which allows to find collisions for these hash functions very efficiently. However, the practical impact is unclear as many real-life applications of hash functions do not just rely on collision resistance.

One drawback with Wang's attacks when used against practical schemes is that due to the message modification technique, the colliding blocks cannot be chosen and look random. However, these attacks work with any IV, so one can choose a common prefix for the two colliding messages, and the Merkle-Damgård construction allows to add a common suffix to the colliding messages. Therefore, an attacker can choose a prefix and a suffix, but he must somehow hide the colliding blocks (1 block in MD4 and SHA-0, and 2 blocks in MD5 and SHA-1). The poisoned message attack [6] exploits this property to create two different PostScript files that display two different chosen texts but whose digests are equal. In this construction, the two different texts are in both PS files and the collision blocks are used by an `if-then-else` instruction to choose which part to display. This attack was extended to other file formats in [8]. Lenstra and

de Weger also used the free prefix and free suffix property to create different X.509 certificates for the same *Distinguished Name* but with different secure RSA moduli in [12]. Here, the colliding blocks are hidden in the second part of the RSA moduli.

Recently, more concrete attacks have appeared: Stevens, Lenstra and de Weger in [19] found colliding X.509 certificates for two different *Distinguished Name*. In this work, the technique used is far more complex and allows to find messages colliding under MD5 with two *different* chosen prefixes. They used an approach suggested by Wang to find a near-collision for different IVs and used different differential paths to absorb the remaining differences. However, the messages B, B' are not controlled, and this randomness must still be hidden in the moduli.

Other applications of Wang collisions have been proposed to attack HMAC with several hash functions in [3,9]. The techniques use Wang's differential path as a black box but with particular messages to recover some keys in the related-key model or to construct advanced distinguishers.

Our results. In the paper we try to extend Wang's attack to break more hash function uses. More precisely, we address the question of message freedom *inside* the colliding blocks, and we show that this can be used to attack APOP, a challenge-response authentication protocol.

The first contribution of this paper is a technique to gain partial control over the colliding blocks; this can be combined with previous work to make the colliding blocks easier to hide. More concretely, we show that we can select some part at the end of the messages which will collide. Our attack can use any differential path, and only requires a set a sufficient conditions. We are able to choose the last three message words in a one-block MD4 collision, and three specific message words in a two-block MD5 collision with almost no overhead. We are also able to choose the 11 last words of a one-block MD4 collision with a work factor of about 2^{31} MD4 computations.

An important point is that the technique used is nearly as efficient as the best message modifications on MD4 or MD5, even when we choose some parts of the messages. This contradicts the usual assumption that Wang's collisions are mostly random. As a first application of this new message modification technique, we show that a countermeasure recently proposed by Szydlo and Yin at CT-RSA '06 in [20] is almost useless for MD4 and can be partially broken for MD5. This can also be used to handle the padding *inside* the colliding blocks.

The second contribution is a partial password-recovery attack against the APOP authentication protocol, based on the message freedom in MD5 collisions. We are able to recover 3 characters of the password, therefore greatly reducing its entropy. Even though we do not achieve the full recovery of the password, we reduce the complexity of the exhaustive search and it is sufficient in practice to reduce this search to a reasonable time for small passwords, *i.e.* less than 9 characters.

Related work. The first MD4 collision was found by Dobbertin [7], and his attack has a time complexity of about 2^{20} MD4; it combines algebraic techniques and optimization techniques such as genetic algorithms. His attack also allows to choose a large part of the colliding blocks at some extra cost: one can choose 11 words out of 16 with a complexity of about 2^{30} MD4 computations (little details are given and only an experimental time complexity).

Our work is based on Wang's collision attack [21,23], which have the following advantages over Dobbertin's:

- it can be adapted to other hash functions (Dobbertin's method can give collisions on the MD5 compression function, but has not been able to provide MD5 collisions);
- it is somewhat more efficient.

More recently, Yu *et al.*[25] proposed a differential path for MD4 collisions with a small number of sufficient conditions. This allows to build a collision (M, M') which is close to a given message $\bar{M}$ (about 44 different bits). This is quite different from what we are trying to do since the changed bits will be spread all over the message. We are trying to choose many consecutive bits, which is useful for different applications. However we studied their work and propose some improvements in Appendix C.

De Cannière and Rechberger announced at the rump session of CRYPTO '06 that they can find reduced-SHA-1 collisions and chose up to 25% of the message. However, they gave few details on their technique, and the conference version does not talk about this aspect of their work. Their idea seems to be to compute a differential path with the chosen message as conditions.

Organization of the paper. This paper is divided in three sections: in the first part we describe APOP, how the attack works, and give background on MD4, MD5 and Wang's attack; then we describe our new collision finding algorithm and how to choose a part of the message; and eventually we describe some applications of these results, including the practical attack against APOP.

2 Background and notation

2.1 APOP

APOP is a command of the Post Office Protocol Version 3 [13] implementing a simple challenge-response authentication protocol; it was introduced in the POP protocol to avoid sending the password in clear over the network. Servers implementing the APOP command send a challenge (formatted as a message identifier, or 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: MD5($msg\text{-}id||passwd$). The server performs the same computation on his side, and checks if the digests match. Thanks to this trick, an eavesdropper will not learn the password, provided MD5 is a partial one-way function. As there is no integrity protection and no authentication of the server, this protocol

is subject to a man-in-the-middle attack, but the man-in-the-middle should not be able to learn the password or to re-authenticate later. Quoting RFC 1939 [13]:

> It is conjectured that use of the APOP command provides origin identification and replay protection for a POP3 session.

This challenge-response can be seen as a MAC algorithm, known as the suffix method: $\text{MAC}_k(M) = \text{MD5}(M||k)$. This construction is weak for at least two reasons: first, it allows off-line collision search so there is a generic forgery attack with $2^{n/2}$ computations and one chosen-text MAC; second, the key-recovery attack against the envelope method of Preneel and van Oorschot [15] can be used on the suffix method with 2^{67} offline computations and 2^{13} chosen text MACs.

This is a hint that APOP is weak, but these attacks require more computations than the birthday paradox, and the first one is mostly useless in a challenge-response protocol. However, we can combine these weaknesses with the collision attack against MD5 to build a practical attack against APOP.

The APOP attack. In this attack, we will act as the server, and we will send some specially crafted challenges to the client. We will use pairs of challenges, such that the corresponding digest will collide only if some part of the password was correctly guessed.

Let us assume we can generate a MD5 collision with some specific format: $M =$ "`<???...???>x`" and $M' =$ "`<¿¿¿...¿¿¿>x`", where M and M' have both size 128 bytes (2 MD5 blocks). The '`?`' and '`¿`' represent any character chosen by the collision finding algorithm. We will send "`<???...???>`" and "`<¿¿¿...¿¿¿>`" as a challenge, and the client returns MD5("`<???...???>`$p_0p_1p_2...p_{n-1}$") and MD5("`<¿¿¿...¿¿¿>`$p_0p_1p_2...p_{n-1}$"), where "$p_0p_1p_2...p_{n-1}$" is the user password (the p_i's are the characters of the password).

Fig. 1: APOP password recovery through MD5 collisions

$$
\begin{aligned}
M &= \texttt{<???} \quad \ldots \quad \texttt{???>} \; \texttt{x} \qquad c = \texttt{<???...???>} \\
M' &= \texttt{<¿¿¿} \quad \ldots \quad \texttt{¿¿¿>} \; \texttt{x} \qquad c' = \texttt{<¿¿¿...¿¿¿>} \\
\text{Auth}(c) &= \text{MD5}\big(\; \texttt{<???} \quad \ldots \quad \texttt{???>} \; p_0 \quad p_1p_2... \quad pad \; \big) \\
\text{Auth}(c') &= \text{MD5}\big(\; \texttt{<¿¿¿} \quad \ldots \quad \texttt{¿¿¿>} \; p_0 \quad p_1p_2... \quad pad \; \big)
\end{aligned}
$$

As we can see in Figure 1, if $p_0 =$ '`x`', 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 (and it is very unlikely that the two hashes collide for $p_0 \neq$ '`x`'). Therefore we are able to test the first password

character without knowing the others. We will construct pairs of challenge to test the 256 possible values, and stop once we have found the first password character.

Then we generate a new collision pair to recover the second character of the password: $M =$ "<???...???>p_0y" and $M' =$ "<¿¿¿...¿¿¿>p_0y", so as to test if $p_1 =$ 'y'. Thus, we can learn the password characters one by one in linear time.

This motivates the need for message freedom in MD5 collisions: we need to generate collisions with a specific format, and some chosen characters. We will see in section 4.1 that this attack can be used in practice to recover the first 3 characters of the password. We believe most passwords in real use are short enough to be found by exhaustive search once we know these first characters, and this attack will allow us to re-authenticate later. We stress that this attack *is* practical: it needs less than one hour of computation, and a few hundreds authentications.

Since people often read their mail from different places (including insecure wireless networks and Internet cafés), we believe that this man-in-the-middle setting is rather realistic for an attack against APOP. Moreover most mail clients automatically check the mailbox on a regular basis, so it seems reasonable to ask for a few hundred authentications.

2.2 MD4 and MD5

MD4 and MD5 follow the Merkle-Damgård construction. Their compression functions are designed to be very efficient, using 32-bit words and operations implemented in hardware in most processors:

- rotation $\lll$;
- addition mod 2^{32} $\boxplus$;
- bitwise boolean functions Φ_i.

The message block M is first split into 16 words $\langle M_i \rangle_{i=0}^{15}$, then expanded to provide one word m_i for each step of the compression function. In MD4 and MD5 this message expansion is simple, it just reuses many times the words M_i. More precisely, the full message is read in a different order at each round: we have $m_i = M_{\pi(i)}$ (π is given below).

The compression function uses an internal state of four words, and updates them one by one in 48 steps for MD4, and 64 steps for MD5. Here, we will assign a name to every different value of these registers, so the description is different from the standard one: the value changed on step i is called Q_i (this follows the notations of Daum [5]).

Then MD4 is given by:

Step update:	$Q_i = (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_i \boxplus k_i) \lll s_i$
Input:	$Q_{-4} \| Q_{-1} \| Q_{-2} \| Q_{-3}$
Output:	$Q_{-4} \boxplus Q_{44} \| Q_{-1} \boxplus Q_{47} \| Q_{-2} \boxplus Q_{46} \| Q_{-3} \boxplus Q_{45}$

π(0..15):	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
π(16..31):	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15
π(32..47):	0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15

And for MD5, we have:

Step update:	$Q_i = Q_{i-1} \boxplus (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_i \boxplus k_i) \lll s_i$
Input:	$Q_{-4}\|\|Q_{-1}\|\|Q_{-2}\|\|Q_{-3}$
Output:	$Q_{-4} \boxplus Q_{60}\|\|Q_{-1} \boxplus Q_{63}\|\|Q_{-2} \boxplus Q_{62}\|\|Q_{-3} \boxplus Q_{61}$

π(0..15):	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
π(16..31):	1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12
π(32..47):	5	8	11	14	1	4	7	10	13	0	3	6	9	12	15	2
π(48..64):	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

The security of the compression function was based on the fact that the operations are not "compatible" and mix the properties of the input.

We will use $x^{[k]}$ to represent the k+1-th bit of x, that is $x^{[k]} = (x \ggg k) \bmod 2$ (note that we count bits and steps starting from 0).

2.3 Wang's Attack against MD4 and MD5

Wang *et al.* attacks against the hash functions of the MD4 family are differential attacks, and have the same structure with two main parts:

1. A precomputation phase:
   - choose a message difference Δ
   - find a differential path
   - compute a set of sufficient conditions
2. Search for a message M that satisfies the conditions; then $H(M) = H(M + \Delta)$.

The differential path specifies how the computations of $H(M)$ and $H(M+\Delta)$ are related: it tells how the differences introduced in the message will evolve in the internal state Q_i. If we choose Δ with a low hamming weight, and some extra properties, we can find some differences in the Q_i's that are very likely. Then we look at each step of the compression function, and we can express a set of sufficient conditions that will make the Q_i's follow the path. These conditions are on the bits of Q_i, so we can not directly find a message satisfying them; however some of them can be fulfilled deterministically through message modifications, and the rest will be statistical by trial and error.

3 A new approach to collision finding

In this paper we assume that we are given a set of sufficient conditions on the internal state variables Q_i that produces collisions. We will try to find a message M such that when one computes a hash of this message, the conditions on the Q_i's hold. We will first describe the general idea that applies to both MD4 and MD5, and we will then study in more details those two hash functions.

In contrast to previous works [14,11,2], we will not focus on a particular path and give message modification techniques for every single condition, but we will give a generic algorithm that can take any path as input, like Klima in [10] and Stevens in [18]. Our method is based on two important facts:

1. We can search a suitable internal state rather than searching a suitable message, because the step update function is invertible: if Q_{i+1}, Q_{i+2} and Q_{i+3} are known, then we can compute any one of Q_i, Q_{i+4} or m_i from the two others (see Algorithm 2 in Appendix B for explicit formulas).
2. We do not need to search for the internal state from the beginning to the end. If we start from the middle we can satisfy the conditions in the first round and some conditions in the second round at the time; additionally we can choose the end of the message before running the search with little extra cost.

3.1 Previous works

Wang's method to find a message satisfying a set of conditions is roughly described in Algorithm 3 in Appendix B: one basically picks many messages at random, modifies them to fulfill some of the conditions, and checks if the other conditions are fulfilled.

The best message modifications known allow to satisfy every condition up to round 22 in MD4 (which gives a collision probability of 2^{-2}) and up to round 24 in MD5 (which gives a collision probability of 2^{-29}). Basically, message modifications for the conditions in the first round are very easy, but in the second round it becomes much more difficult because we cannot freely change message words without breaking the Q_i in the first round (and therefore also in the second round). At the beginning of the second round it is still possible to find message modifications, but it becomes increasingly difficult as we go forward. Wang's differential paths are chosen with this constraint in mind, and most of their conditions are in the first round and at the beginning of the second.

The algorithm can be rewritten more efficiently: instead of choosing a random message and modify it, we can choose the Q_i in the first round and compute the corresponding message. Since all the conditions in MD4 and MD5 are on the Q_i's, this will avoid the need for message modifications in the first round.

To further enhance this algorithm, Klima introduced the idea of *tunnels* in [11], which is closely related to Biham and Chen's *neutral bits* used in the cryptanalysis of SHA-0 [1]. A tunnel is a message modification that does not affect the conditions up to some step $p_v - 1$ (*point of verification*). Therefore, if we have one message that fulfill the conditions up to $p_v - 1$ and τ tunnels, we can generate 2^τ messages that fulfills conditions up to step $p_v - 1$. This does not change the number of messages we have to try, but it greatly reduces the cost of a single try, and therefore speeds up collision search a lot. This is described in Algorithm 4 in Appendix B.

In MD4 and MD5, the point of verification will be in the second round, and we put it after the last condition in the second round (step 22 in MD4, 24 in MD5). We have message modifications for almost every condition before the point of verification, and it seems impossible to find message modifications for round 3 and later.

3.2 Our method

Our method is somewhat different: we will *not* fix the Q_i from the beginning to the end, but we will start from the middle, and this will allow us to deal with the first round and the beginning of the second round at the same time.

First we choose a *point of choice* p_c of and a *point of verification* p_v. The point of verification is the step where we will start using tunnels, and the point of choice is the first step whose conditions will not be satisfied deterministically. The value of p_c depends on the message expansion used in the second round: we must have $\pi(16) < \pi(17) < ... < \pi(p_c - 1) < 12$, so we will choose $p_c = 19$ in MD4 ($\pi(18) = 8$), and $p_c = 19$ in MD5 ($\pi(18) = 11$).

The key idea of our collision search is to first choose the end of the first round, *i.e.* Q_{12} to Q_{15}. Then we can follow the computations in the first round and in the second round *at the same time*, and choose m_i's that satisfy the conditions in both rounds. There is no difficulty when the first round meets the values we fixed in the beginning: since we only fixed the Q_i's, we just have to compute the corresponding m_i's. More precisely, 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 since we have already fixed Q_{j-4}, Q_{j-3}, Q_{j-2} and Q_{j-1}. Thus, we can fulfill conditions up to round $p_c - 1$ almost for free.

In the end, we will make random choices for the remaining steps ($Q_{\pi(p_c-1)+1}$ to Q_{11}), until we have a message that follows the path up to step $p_v - 1$, and finally we use the tunnels. For a more detailed description of the algorithm, see Algorithm 1 in Appendix B, and take $t = 0$ (this algorithm is more generic and will be described in the next section).

Since we do not choose the Q_i's in the natural order, we have to modify a little bit the set of sufficient conditions: if we have a condition $Q_{12}^{[k]} = Q_{11}^{[k]}$, we will instead use $Q_{11}^{[k]} = Q_{12}^{[k]}$ because we choose Q_{12} before Q_{11}.

Compared to standard message modifications, our algorithm has an extra cost when we try to satisfy conditions in steps p_c to $p_v - 1$. However, this is not so important for two reasons:

- Testing one message only requires to compute a few steps, and we will typically have less than 10 conditions to satisfy, so this step will only cost about a hundred hash computations.
- This cost will be shared between all the messages which we find with the tunnels. In the case of MD5, we have to use a lot of tunnels to satisfy the 29 conditions in round 3 and 4 anyway, and in MD4 we will use a lot of tunnels if we look for many collisions (if one needs a single MD4 collision, the collision search cost should not be a problem).

3.3 Choosing a part of the message

This method can be extended to allow some message words to be fixed in the collision search. This will make the search for a first message following the path

up to the point of verification harder, and it will forbid the use of some tunnels. Actually, we are buying some message freedom with computation time, but we can still find collisions very efficiently. We will show which message words can be chosen, and how to adapt the algorithm to find a first message following the path up to the step $p_v - 1$ with some message words chosen.

Choosing the beginning. If the first steps in the first round are such that in the second round $m_0...m_i$ are only used after the step p_c or in a step j with no condition on Q_j, then we can choose $m_0...m_i$ before running the algorithm. The choice of $m_0...m_i$ will only fix $Q_0...Q_i$ (if there are some conditions on $Q_0...Q_i$, we must make sure they are satisfied by the message chosen), and the algorithm will work without needing any modification.

On MD5, this allows to choose m_0. Using Wang's path, there are no conditions on Q_0 for the first block, so m_0 is really free, but in the second block there are many conditions. On MD4, we have $\pi(16) = 0$, so this can only be used if we use $p_c = 16$, which will significantly increase the cost of the collision search.

Choosing the end. The main advantage of our algorithm is that it allows to choose the end of the message. This is an unsuspected property of Wang's attack, and it will be the core of our attack against APOP. Our idea is to split the search in two: first deal with fixed message words, then choose the other internal state variables. This is made possible because our algorithm starts at the end of the first round; the conditions in those steps do not directly fix bits of the message, they also depend on the beginning of the message.

More precisely, if we are looking for collisions where the last t words are chosen, we begin by fixing Q_{12-t}, Q_{13-t}, Q_{14-t} and Q_{15-t}, and we compute Q_{16-t} to Q_{15} using the chosen message words. We can modify Q_{12-t} if the conditions on the first state Q_{16-t} are not satisfied, but for the remaining $t-1$ steps this is impossible because it would break the previous steps. So, these conditions will be fulfilled only statistically, and we might have to try many choices of Q_{12-t}, Q_{13-t}, Q_{14-t}, Q_{15-t} (note that each choice does not cost a full MD4 computation, but only a few steps).

Once we have a partial state that matches the chosen message, we run the same algorithm as in the previous section, but we will be able to deal with less steps of the second round due to the extra fixed states Q_{12-t} to Q_{11}. The full algorithm is given in Algorithm 1 in Appendix B.

3.4 MD4 message freedom

Using Wang's EUROCRYPT path [21]. If we use Wang's EUROCRYPT path, we will choose $p_v = 23$ so as to use tunnels only for the third round. Therefore, the tunnels will have to preserve the values of m_0, m_4, m_8, m_{12}, m_1, m_5 and m_9 when they modify the Q_i's in the first round.

There are two easy tunnels we can use, in Q_2 and Q_6. If we change the value of Q_2, we will have to recompute m_2 to m_6 as we do not want to change any other

Q_i, but if we look at step 4, we see that Q_2 is only used trough $\mathrm{IF}(Q_3, Q_2, Q_1)$. So some bits of Q_2 can be changed without changing Q_4: if $Q_3^{[k]} = 0$ then we can modify $Q_2^{[k]}$. The same thing happens is step 5: Q_2 is only used in $\mathrm{IF}(Q_4, Q_3, Q_2)$, and we can switch $Q_2^{[k]}$ if $Q_4^{[k]} = 1$. So on the average, we have 8 bits of Q_2 that can be used as a tunnel. The same thing occurs in Q_6: if $Q_7^{[k]} = 0$ and $Q_8^{[k]} = 1$, then we can change $Q_6^{[k]}$ without altering m_8 and m_9. If we add some extra conditions on the path we can enlarge these tunnels, but we believe it's not necessary for MD4.

We can use our collision finding algorithm with up to 5 fixed words; Table 1 gives the number of conditions we will have to satisfy probabilistically. Of course, the cost of the search increases with t, but with $t = 3$ it should be about one 2^9 MD4 computations, which is still very low. Note that this cost is only for the first collision; if one is looking for a bunch of collisions, this cost will be shared between all the collisions found through the tunnels, and we expect 2^{14} of them. Another important remark is that this path has a non-zero difference in m_{12}; therefore when choosing more than 3 words, the chosen part in M and M' will have a one bit difference.

Table 1: Complexity estimates for MD4 collision search with t fixed words, and comparison with Dobbertin's technique [7]. We assume that a single trial costs 2^{-3} MD4 on the average due to early abort techniques.

message words chosen: t	0	1	2	3	4	5	0	11	0	11
path used	[21]						[25]		[7]	
point of choice: p_c	19	19	19	18	18	18	19	17		
point of verification: p_v	23	23	23	23	23	23	23	17		
conditions in steps $17 - t$ to 15	0	0	6	12	18	24	0	17		
conditions in steps p_c to $p_v - 1$	8	8	8	11	11	11	11	0		
conditions in steps p_v to N	2	2	2	2	2	2	17	34		
complexity (MD4 computations $\log_2$)	5	5	5	9	15	21	14	31	20	30

Using Yu *et al.*'s CANS path [25]. To push this technique to the limit, we will try to use $t = 11$: this leaves only m_0 to m_4 free, which is the minimum freedom to keep a tunnel. In this setting, the conditions in steps 6 to 15 can only be satisfied statistically, which will be very expensive with Wang's path [21] (its goal was to concentrate the conditions in the first round). Therefore we will use the path from [25], which has only 17 conditions in steps 6 to 15.

Since we fix almost the full message, the second phase of the search where we satisfy conditions in the first and second round at the same time will be very limited, and we have $p_c = 17$. Then we use the tunnel in Q_0, which is equivalent to iterating over the possible Q_{16}'s, computing m_0 from Q_{16}, and then recomputing Q_0 and m_1, m_2, m_3, m_4. There are 34 remaining conditions,

so we will have to use the tunnel about 2^{34} times. Roughly, we break the message search in two: first find $m_0..m_4$ such that the message follows steps 0 to 16, then modify it to follow up to the end by changing Q_0. This path is well suited for this approach, with few conditions well spread over the first two rounds.

This gives us a lot of freedom in MD4 collisions, but the collision search becomes more expensive. Another interesting property of this path is that it only introduces a difference in m_4, so the 11 chosen words will be the same in M and M'.

3.5 MD5 message freedom

Using Wang's MD5 path [23], we will choose $p_v = 24$ so as to use tunnels only for the third round. Therefore, when we modify the Q_i's in the first round to use the tunnels, we have to keep the values of m_1, m_6, m_{11}, m_0, m_5, m_{10}, m_{15} and m_4. We will not describe the available tunnels here, since they are extensively described in Klima's paper [11].

We use the set of conditions from Stevens [18], which adds the conditions on the rotations that were missing in Wang's paper [23]. We had to remove the condition because it is incompatible with some choices of m_{15}, so we check instead the less restrictive condition on Φ_{15} ($\Phi_{15}^{[31]} = 0$). We also found out that some conditions mark as optimization conditions were actually needed for the set of conditions to be sufficient.

As already stated, we can choose m_0 in the first block, and we will see how many words we can choose in the end of the message. With $t = 0$, we set $p_c = 19$, so we have 7 conditions in steps p_c to $p_v - 1$, and after p_v, there are 29 conditions for the first block, and 22 for the second block. With $t = 1$, we use $p_c = 18$, which increase the number of conditions between steps p_c and $p_v - 1$ to 9, but since we will use a lot of tunnels, this has very little impact on the computing time. We can also try to set $t = 2$, but this adds a lot of conditions when we search the states in the end of the first round. According to our experiments, the conditions on the Q_i's also imply some conditions on m_{14}, so m_{14} could not be chosen freely anyway.

As a summary, in a two-block MD5 collision $(M||N, M'||N')$, we can choose M_0, M_{15} and N_{15}, and the complexity of the collision search is roughly the same as a collision search without extra constraints.

We only implemented a little number of tunnels, but we find the first block in a few minutes with m_0 and m_{15} chosen, and the second block in a few seconds with m_{15} chosen. This is close to Klima's results in [11].

4 Applications

Freedom in colliding blocks can be used to break some protocols and to create collisions of special shape. The applications we show here requires that the chosen part is identical in M and M', *i.e.* the differential path must not use a difference there.

Fixing the padding. We can use this technique to find messages with the padding included in the colliding block. For instance, this can be useful to build pseudo-collision of the hash function: if there is a padding block after the pseudo-colliding messages, the pseudo-collision will be completely broken.

We can also find collisions on messages shorter than one block, if we fix the rest of the block to the correct padding. An example of a 160 bit MD4 collision is given in Table 2 in Appendix A.

Zeroed collisions. Szydlo and Yin proposed some message preprocessing techniques to avoid collisions attacks against MD5 and SHA-1 in [20]. Their idea is to impose some restrictions on the message, so that collision attacks become harder. One of their schemes is called *message whitening*: the message is broken into blocks smaller than 16 words, and the last t words are filled with zeroes. Using our technique we can break this strengthening for MD4 and MD5: in Appendix A we show a 11-whitened MD4 collision in Table 3 and a 1-whitened MD5 collision in Table 4.

4.1 The APOP attack in practice.

To implement this attack, we need to efficiently generate MD5 collisions with some chosen parts; we mainly have to fix the last word, which is precisely what we can do cheaply on MD5.

Additionally, the POP3 RFC [13] 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, but we can not find two colliding ASCII messages with Wang's path.

In practice most mail clients do not check these requirements, leaving us a lot of freedom. According to our experiments with Thunderbird[1] and Evolution[2] there are only four characters which they reject 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

Thunderbird also needs at least one '@' in the msg-id.

We will use Wang's path [23], which gives two-block collisions. The first block is more expensive to find, so we will use the same for every msg-id, and we will fix the first character as a '<', and the last character as a '@'. Then we have to generate a second block for every password character test; each time the last word is chosen, and we must avoid 4 characters in the message. Using the ideas from Section 3.5 we can do this in less than 5 seconds per collision on a standard desktop computer.

[1] available at `http://www.mozilla.com/en-US/thunderbird/`
[2] available at `http://www.gnome.org/projects/evolution/`

Unfortunately, this path uses a difference $\delta M_{14} = 2^{32}$ and this makes a difference in character 60. In order to learn the i-th password character p_{i-1}, we need to generate a collision where we fix the last $i+1$ characters (i password characters, plus a '>' to form a correct msg-id). Therefore, we will only be able to retrieve 3 characters of the password with Wang's path. This points out a need for new paths following Wang's ideas, but adapted to other specific attacks; here a path less efficient for collision finding but with better placed differences could be used to learn more characters of the password.

Complexity. To estimate the complexity of this attack, we will assume the user's password is 8-characters long, and each character has 6 bits of entropy. This seems to be the kind of password most people use (when they don't use a dictionary word...). Under these assumptions, we will have to generate 3×2^5 collisions, and wait for about 3×2^6 identifications. Each collision takes about 5 seconds to generate; if we assume that the client identifies once per minute this will be the limiting factor, and our attack will take about 3 hours. In a second phase we can do an offline exhaustive search over the missing password characters. We expect to find them after 2^{30} MD5 computations and this will take about half an hour according to typical OpenSSL benchmarks.

Recent developments. The attack against APOP had been independently found by Sasaki *et al.* almost simultaneously; they put a summary of their work on the eprint [16]. Moreover, Sasaki announced at the rump session of FSE '07 that he had improved the attack; he can recover 31 password characters using a new differential path.

Recommendations. We believe APOP is to be considered broken, and we suggest users to switch to an other authentication protocol if possible. Since the current attack needs non-RFC-compliant challenges, an easy countermeasure in the mail user agent is to strictly check if the challenge follows the RFC, but maybe this could be defeated by an improved attack.

Acknowledgement

We thank the anonymous reviewers of FSE, for their helpful comments and for pointing out related work. Thanks are also due to Phong Nguyen and Pierre-Alain Fouque for their precious help and proofreading. Finally, we would also like to thank Yu Sasaki for the explanations of his improvements over the APOP attack.

Part of this work is supported by the Commission of the European Communities through the IST program under contract IST-2002-507932 ECRYPT, and by the French government through the Saphir RNRT project.

References

1. Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004.
2. John Black, Martin Cochran, and Trevor Highland. A Study of the MD5 Attacks: Insights and Improvements. In Matthew Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 262–277. Springer, 2006.
3. Scott Contini and Yiqun Lisa Yin. Forgery and Partial Key-Recovery Attacks on HMAC and NMAC Using Hash Collisions. In Xuejia Lai and Kefei Chen, editors, *ASIACYPT*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.
4. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
5. M. Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr-University of Bochum, 2005.
6. Magnus Daum and Stefan Lucks. Hash Collisions (The Poisoned Message Attack) "The Story of Alice and her Boss". Presented at the rump session of Eurocrypt '05. `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/`.
7. Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.
8. Max Gebhardt, Georg Illies, and Werner Schindler. A Note on the Practical Value of Single Hash Collisions for Special File Formats. In Jana Dittmann, editor, *Sicherheit*, volume 77 of *LNI*, pages 333–344. GI, 2006.
9. Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2006.
10. Vlastimil Klima. Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications. Cryptology ePrint Archive, Report 2005/102, 2005. `http://eprint.iacr.org/`.
11. Vlastimil Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105, 2006. `http://eprint.iacr.org/`.
12. Arjen K. Lenstra and Benne de Weger. On the Possibility of Constructing Meaningful Hash Collisions for Public Keys. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2005.
13. J. Myers and M. Rose. Post Office Protocol - Version 3. RFC 1939 (Standard), May 1996. Updated by RFCs 1957, 2449.
14. Yusuke Naito, Yu Sasaki, Noboru Kunihiro, and Kazuo Ohta. Improved Collision Attack on MD4 with Probability Almost 1. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2005.
15. Bart Preneel and Paul C. van Oorschot. On the Security of Two MAC Algorithms. In *EUROCRYPT*, pages 19–32, 1996.
16. Yu Sasaki, Go Yamamoto, and Kazumaro Aoki. Practical password recovery on an md5 challenge and response. Cryptology ePrint Archive, Report 2007/101, 2007. `http://eprint.iacr.org/`.
17. Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18,*

2005, Proceedings, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.
18. Marc Stevens. Fast Collision Attack on MD5. Cryptology ePrint Archive, Report 2006/104, 2006. `http://eprint.iacr.org/`.
19. Marc Stevens, Arjen Lenstra, and Benne de Weger. Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities. Cryptology ePrint Archive, Report 2006/360, 2006. `http://eprint.iacr.org/`.
20. Michael Szydlo and Yiqun Lisa Yin. Collision-Resistant Usage of MD5 and SHA-1 Via Message Preprocessing. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 99–114. Springer, 2006.
21. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [4], pages 1–18.
22. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Shoup [17], pages 17–36.
23. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [4], pages 19–35.
24. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In Shoup [17], pages 1–16.
25. Hongbo Yu, Gaoli Wang, Guoyan Zhang, and Xiaoyun Wang. The Second-Preimage Attack on MD4. In Yvo Desmedt, Huaxiong Wang, Yi Mu, and Yongqing Li, editors, *CANS*, volume 3810 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.

A Collision Examples

Table 2: A 160-bit MD4 collision.

Message M
42 79 2d 65 f0 f8 4f d8 d5 7d 86 bf 78 54 9d 67 3f b3 8c aa
Message M'
42 79 2d 65 f0 f8 4f d8 d5 7d 86 bf 78 54 9d 67 3f b3 8c ac
Message M padded to one block
42 79 2d 65 f0 f8 4f d8 d5 7d 86 bf 78 54 9d 67 3f b3 8c aa 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a0 00 00 00 00 00 00 00
Message M' padded to one block
42 79 2d 65 f0 f8 4f d8 d5 7d 86 bf 78 54 9d 67 3f b3 8c ac 80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a0 00 00 00 00 00 00 00
MD4
46 6d cb bd 04 66 2c 43 75 12 18 f6 f4 e5 68 71

Table 3: A 11-whitened MD4 collision.

Whitened Message M			
b9 39 4f 51	3b 43 68 dd	d6 1d 6f 1c	5d b6 a0 b2
44 d4 69 18	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
Whitened Message M'			
b9 39 4f 51	3b 43 68 dd	d6 1d 6f 1c	5d b6 a0 b2
44 d4 69 1a	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
MD4 without padding			
1d ba e9 89	02 22 9f a6	a9 bb 88 f8	30 c1 38 ab
MD4 with padding			
e1 54 1e 65	46 d8 4b 79	db b3 5b b2	13 00 06 9b

Table 4: A 1-whitened MD5 collision.

Whitened Message M			
00 00 00 00	23 f9 5a 1c	c8 4f 18 59	1b ef 74 a9
02 7a b6 bf	ff 47 53 be	c3 29 a9 dd	b1 1e 62 94
d1 2c 24 05	07 5e b4 42	1b e2 58 72	25 83 b2 52
12 97 d8 24	ca 8c ae 13	e1 e9 34 77	00 00 00 00
54 01 42 6f	b4 b5 4a 77	d2 15 90 5a	7a 42 cf dd
9f 76 5b 37	90 dd 7e 3d	0a fd 77 d7	d1 4c 55 de
49 ff 3e f2	f5 52 b8 86	72 c0 49 7e	80 ac d1 1f
c7 38 b4 96	a8 d3 73 f0	4b 5c d8 f2	00 00 00 00
Whitened Message M'			
00 00 00 00	23 f9 5a 1c	c8 4f 18 59	1b ef 74 a9
02 7a b6 3f	ff 47 53 be	c3 29 a9 dd	b1 1e 62 94
d1 2c 24 05	07 5e b4 42	1b e2 58 72	25 03 b3 52
12 97 d8 24	ca 8c ae 13	e1 e9 34 f7	00 00 00 00
54 01 42 6f	b4 b5 4a 77	d2 15 90 5a	7a 42 cf dd
9f 76 5b b7	90 dd 7e 3d	0a fd 77 d7	d1 4c 55 de
49 ff 3e f2	f5 52 b8 86	72 c0 49 7e	80 2c d1 1f
c7 38 b4 96	a8 d3 73 f0	4b 5c d8 72	00 00 00 00
MD5 without padding			
98 28 90 7a	75 75 ae 7a	25 f9 80 94	62 ea 52 76
MD5 with padding			
7c c4 3c db	a7 a6 f6 9e	c5 10 8e 46	95 00 fd 82

Table 5: APOP MD5 collision. These two msg-id's collide if padded with "`bar`"

Message M				
<xxxÑÕç\ᴴˢØºä4ᴾᴰ<ᴴᴼ	3c 78 78 78	d1 d5 e7 5c	88 d8 ba e4	34 8b 3c 81
mXÀᴬᶜn]ᴱᴮØ\4UᴮˢµiQᴾ¹	6d 58 c0 9f	6e 5d 17 d8	5c 34 55 08	b5 69 51 91
Åì"ᴬᴷˢˣxô!¿ïFsc'áÙ	c5 ec 22 06	02 78 f4 21	bf ef 46 73	63 27 e1 d2
ÝìᵛˢôG^{E}_{C}û49¾Vᴴᴶxxx@	dd ec 8a f4	47 1b fb 34	39 be 56 89	78 78 78 40
ÚcCòßPºᴱᴳÙÜᴴᴛᴺᴴᴿᴵRéý	da 63 43 f2	df 50 ba 05	d9 d6 09 83	8d 52 e9 fd
Äᶜᶜ4$½MH-:y6ÇÊ©Üᴵˢ	c4 94 34 24	bd 4d 48 2d	3a 79 36 c7	ca a9 dc 1c
-0:Þñᴹᵂ|enᴾᵘë5{ᴰᶜþ¾	2d 30 3a de	f1 95 7c 65	6e 8c eb 35	7b 90 fe be
ïL´ïªØ#»ì])ü>	ef 4c b4 ef	aa d8 23 bb	ec 5d 29 fc	3e
Message M'				
<xxxÑÕç\ᴴˢØºä4ᴾᴰ<ᴴᴼ	3c 78 78 78	d1 d5 e7 5c	88 d8 ba e4	34 8b 3c 81
mXÀᵘˢn]ᴱᴮØ\4UᴮˢµiQᴾ¹	6d 58 c0 1f	6e 5d 17 d8	5c 34 55 08	b5 69 51 91
Åì"ᴬᴷˢˣxô!¿ïFsc§áÙ	c5 ec 22 06	02 78 f4 21	bf ef 46 73	63 a7 e1 d2
ÝìᵛˢôG^{E}_{C}û49¾Vᴴᴛxxx@	dd ec 8a f4	47 1b fb 34	39 be 56 09	78 78 78 40
ÚcCòßPºᴱᴳÙÜᴴᴛᴺᴴᴿᴵRéý	da 63 43 f2	df 50 ba 05	d9 d6 09 83	8d 52 e9 fd
Äᶜᶜ4¤½MH-:y6ÇÊ©Üᴵˢ	c4 94 34 a4	bd 4d 48 2d	3a 79 36 c7	ca a9 dc 1c
-0:Þñᴹᵂ|enᴾᵘë5{ᴰᴸþ¾	2d 30 3a de	f1 95 7c 65	6e 8c eb 35	7b 10 fe be
ïL´ïªØ#»ì])|>	ef 4c b4 ef	aa d8 23 bb	ec 5d 29 7c	3e
MD5(M||"`bar`")	b8 98 53 57	f8 06 8c 23	72 cf f8 c2	4c 22 c3 81
MD5(M'||"`bar`")	b8 98 53 57	f8 06 8c 23	72 cf f8 c2	4c 22 c3 81
MD5(M||"`ban`")	40 c6 ee cc	6f e1 e5 2b	53 74 a0 e8	3e f7 4f 54
MD5(M'||"`ban`")	02 c1 8c 29	49 91 04 99	8f 88 33 77	a1 eb 81 be

Table 6: A MD4 collision close to 1^{512}.

Message M			
ff ff ff ff	bf ff ff ff	ff f7 ff ff	ff ff df ff
ff ff ff fd	ff ff df ff	ff ff fd ff	ff ef ff ff
ff ff ff ef	ff ff ff fe	ff ff ef 7f	ff 7f ff ff
7f ff fd 7f	ff bf ff ff	ff ff ff ff	ff bf ff fd
Message M'			
ff ff ff ff	bf ff ff ff	ff f7 ff ff	ff ff df ff
ff ff ff ff	ff ff df ff	ff ff fd ff	ff ef ff ff
ff ff ff ef	ff ff ff fe	ff ff ef 7f	ff 7f ff ff
7f ff fd 7f	ff bf ff ff	ff ff ff ff	ff bf ff fd
MD4 without padding			
ff a3 b5 2d	51 63 59 36	11 e5 9a d0	a6 cf 8b 33
MD4 with padding			
59 93 19 84	d0 6f 55 9f	f3 d0 87 4b	c6 24 f4 8d

B Algorithms

All the algorithms take the set of condition as an implicit input, and will modify a shared state containing the m_i's and the Q_i's.

Algorithm 1 Our collision finding algorithm

```
 1: procedure FindMessage(p_v, p_c, t)
 2:     repeat
 3:         choose Q_{12−t}, Q_{13−t}, Q_{14−t}, Q_{15−t}
 4:         if t ≠ 0 then
 5:             StepForward(16 − t)
 6:             FixState(16 − t)
 7:             StepBackward(16 − t)
 8:             if not CheckConditions(12 − t) then
 9:                 goto 3
10:             for 17 − t ≤ i < 16 do
11:                 StepForward(i)
12:                 if not CheckConditions(i) then
13:                     goto 3
14:         i ← 0
15:         for 16 ≤ j < p_c do
16:             while i < π(j) do
17:                 choose Q_i
18:                 StepMessage(i)
19:                 i ← i + 1
20:             StepForward(j)
21:             FixState(j)
22:             StepMessage(j)
23:             StepForward(i)
24:             if not CheckConditions(i) then
25:                 goto 16
26:         for π(p_c − 1) + 1 ≤ i < 12 − t do
27:             choose Q_i
28:             StepMessage(i)
29:         StepMessage(12-t ... 15-t)
30:         for p_c ≤ i < p_v do
31:             StepForward(i)
32:             if not CheckConditions(i) then
33:                 goto 26
34:         for all tunneled message do
35:             for p_v ≤ i < N do
36:                 StepForward(i)
37:                 if not CheckConditions(i) then
38:                     use the next message
39:     until all conditions are fulfilled
```

Algorithm 2 Step functions

function MD4StepForward(i)
    $Q_i \leftarrow (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_i \boxplus k_i) \lll s_i$
function MD4StepBackward(i)
    $Q_{i-4} \leftarrow (Q_i \ggg s_i) \boxminus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxminus m_i \boxminus k_i$
function MD4StepMessage(i)
    $m_i \leftarrow (Q_i \ggg s_i) \boxminus Q_{i-4} \boxminus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxminus k_i$
function MD5StepForward(i)
    $Q_i \leftarrow Q_{i-1} \boxplus (Q_{i-4} \boxplus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxplus m_i \boxplus k_i) \lll s_i$
function MD5StepBackward(i)
    $Q_{i-4} \leftarrow (Q_i \boxminus Q_{i-1}) \ggg s_i \boxminus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxminus m_i \boxminus k_i$
function MD5StepMessage(i)
    $m_i \leftarrow (Q_i \boxminus Q_{i-1}) \ggg s_i \boxminus Q_{i-4} \boxminus \Phi_i(Q_{i-1}, Q_{i-2}, Q_{i-3}) \boxminus k_i$

Algorithm 3 Wang's message finding algorithm

procedure FindMessageWang
    repeat
        choose a random message
        for $0 \leq i < N$ **do**
            StepForward(i)
            if not CheckConditions(i) **then**
                try to modify the message
    until all conditions are fulfilled

Algorithm 4 Klima's message finding algorithm

procedure FindMessageKlima
    repeat
        for $0 \leq i < 16$ **do**
            choose Q_i
            StepMessage(i)
        for $16 \leq i < p_v$ **do**
            StepForward(i)
            if not CheckConditions(i) **then**
                modify the message
        for all tunneled message **do**
            for $p_v \leq i < N$ **do**
                StepForward(i)
                if not CheckConditions(i) **then**
                use the next message
    until all conditions are fulfilled

C Collisions with a High Number of Chosen Bits

In this paper, we considered the problem of finding collisions with some chosen *words* but some other works addressed the problem of choosing *bits* (mainly [25]). We believe it is more useful to choose consecutive bits and the applications we give in Section 4 all need this property. Specifically, the APOP attack requires to choose consecutive bits in the end of the block; it will fail if one of these bits is uncontrolled. However let us say a few words about collisions with many chosen bits.

Following the ideas of Yu *et al.*[25], we will use a collision path with very few conditions. Such paths are only known in the case of MD4, and we found out that the path from [25] can be slightly enhanced: if we put the difference in the bit 25 instead of the bit 22, we get only 58 conditions (instead of 62). Now the basic idea is to take a message M, and apply message modifications in the first round: this will give a message M^* that has about 10 bit difference from M (there are 20 conditions in the first round) and it gives a collision $(M^*, M^* + \Delta)$ with probability 2^{-38}. Therefore we will generate about 2^{38} messages M_i close to M and the corresponding M_i^*, and one of them will give a collision.

Little detail is given in Yu *et al.* paper, but we can guess from their collision example that they generated the M_i's by changing m_{14} and m_{15}. This makes the attack more efficient since the M_i^* will all have the same first 14 words, but it will modify about 32 extra bits. Actually, one only needs to iterate over 38 bits, which gives on the average 19 modified bits, but Yu *et al.* used the whole 64 bits.

In fact, if the goal is to have a high number of chosen bits, it is better to choose the M_i in another way: instead of iterating over some bits, we will switch a few bits in the whole message, and iterate over the positions of the differences. We have $\binom{512}{5} \approx 2^{38}$, so it should be enough to select 5 positions, but we will have to run the full message modifications in the first round for every message, which is quite expensive (about 2^{37} MD4 computations). Instead, one can choose 4 positions in the first 480 bits, and two in the last 32 bits: we have $\binom{480}{4}\binom{32}{2} \approx 2^{40}$, and the message modification on the first 15 words will only be run every 2^9 messages; the main cost will be that of testing 2^{38} messages for the second and third rounds: using early abort techniques, this cost will be about 2^{33} MD4[3].

On the average, we expect to have 6 bit differences coming from this iteration, plus 10 coming from the message modification in the first round. An example of such message is given in Table 6, it has 18 bit differences from the target (a block consisting only of 1's), which is much better than achieved by Yu *et al.* (43 bit differences).

[3] There are two conditions on step 15, three on step 16 and one on step 17: so $3 \cdot 2^{36}$ messages will stop after 1 step, $7 \cdot 2^{33}$ after 2 steps, 2^{32} after 3 steps, and the remaining 2^{32} messages will need at most 16 steps. This gives less than $95 \cdot 2^{32}$ MD4 steps, that is less than 2^{33} full MD4.