Practical and Secure Message Authentication *

S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk
E-mail: [shahram, rei, josef]@cs.uow.edu.au

Centre for Computer Security Research,
Department of Computer Science,
University of Wollongong, Wollongong,
NSW 2522, Australia

April 27, 1995

Abstract

Use of encryption algorithms in message authentication is replaced by secure hash functions which are often faster than encryption algorithms. Tsudik [14] has proposed three methods on message authentication which are only based on one-way hash functions and use some keys to make them secure. In this paper, we give a set of practical methods, each of which uses a fast collision free hash function (such as MD5) and provides secure message authentication. The idea of the proposed methods is almost similar to that of Tsudik's, but we are able to reduce the key length eight times compared to the Tsudik's constructions, while maintaining the same security. In our methods, the secret key is added using exclusive-or or assign operators (instead of concatenation) to make them faster. We also have proved that our methods belong to the Secure Keyed One-Way Hash Function (SKOWHF) group, if the underlying hash function is secure.

1 Introduction

In today's communication, existence of a fast method for message authentication is of high interest. A secure message authentication should provide integrity and assure that the message is protected against impersonation, modification, and substitution. By secure authentication, we mean that only legitimate parties should be able to generate (or change) the message. For instance, use of parity bits in communication provides authenticity (since every one can test the originality of the message), but we do not call it secure authentication, because, everyone can change some bits and find the corresponding parity bits; the authenticity does not depend on a secret key. Sometimes in the literature, secure authentication is called *Message Authentication Code (MAC)* [10], or *Secured Keyed One-Way Hash Functions (SKOWHF)* [3].

*Support for this project was provided in part by the Australian Research Council under the reference number A49530480.

Encryption algorithms, such as DES ([8]), are usually used in different applications to provide the secrecy (based on a secret key). One of the most popular kinds of message authentication is to send a message followed by a secure message digest. A secure message digest should have a fix length and can be produced, for example, by performing DES in Cipher Block Chaining (CBC) mode on a message [9, page 26]. As another example, a hash function followed by an encryption algorithm can be used to compute the digest. Then, the pair [message , digest] is used to send the message authentically through the channel [13, Section 4.3]. Since encryption algorithms are invertible, they are often complex and slow and, as the result, existence of a method based on one-way functions, which are usually faster than encryption algorithms, is well appreciated.

In [14], Tsudik has proposed three methods on message authentication which are based on one-way hash functions and some secret keys. He has named the methods as *Secret Prefix*, *Secret Suffix*, and *Envelope Method*, and used MD4 ([11]) as the hashing function.

In this paper we modify Tsudik's methods by reducing the key length from 512 to 128 bits. This reduction not only increases the speed of the process, but also makes easier to handle the key. Furthermore, a 128-bit key is only 16 bytes and might be memorized by the legitimate parties. MD5[1] is used in our methods, while some secret keys increase its security. Instead of concatenating the key to the message, we exclusive-or it to the message or use it directly as the initial vector. This allows us to save one block in the hash function process. (In Envelope Method two blocks are saved.) Other hash functions can be used instead of MD5, depending on the level of the security and the speed.

In this paper, we have also used Berson *et al.* ([3]) definition of *Secured Keyed One-Way Hash Functions (SKOWHF)* and have proved that our methods hold the properties of such hash functions.

The organization of the paper is as follows. Next section provides the notations, assumptions, and preliminaries. Section 1.2 gives a summary of MD5. Tsudik's methods are introduced in Section 2. Our proposed methods (six methods) are described in Section 3. The definition of SKOWHF and the proof of the claim that our methods are SKOWHF, are given in Section 4. Section 5 is the security analysis of the proposed methods in terms of the related possible attacks. Section 6 gives the discussions and conclusions.

1.1 Notations and Preliminaries

In our discussion, we use the following notations:

- A and B are two communicants (Alice and Bob).
- M is an arbitrary length message which is needed to be sent from one party to another.
- IV is the initial vector (128 bits) for MD5 algorithm.
- MD denotes the 128-bit message digest of MD5 for a given message M ($MD = \mathbf{MD5}(M)$).
- 'P' is the padding bits in MD5 algorithm.

[1]MD5 is briefly described in Section 1.2. More details on MD5 can be found in [12]. In Section 6 we suggest using other hash functions, depending on the different demands.

- S^p_{AB} and S^s_{AB} are 512-bit secret keys shared between A and B (In Tsudik's methods).
- K_{AB} is a 128-bit secret key shared between A and B (In our methods).
- '$||$' denotes concatenation.
- '$X \oplus Y$' denotes addition modulo 2 (bit-wise exclusive-or), when X and Y have the same length. Sometimes we use the term XOR instead.
- '$X \overline{\oplus} Y$' denotes addition modulo 2, when X and Y have (probably) different lengths. The short one (between X and Y) will be padded by zero bits from the right hand side to make its length the same as the other's. In other words, the short one will be XORed to the beginning of the other one.
- '$X \underline{\oplus} Y$' is the same as above, but the padding will be done in the left hand side. That is, the short one will be XORed to the end of the other one.

In our discussion, we assume that there is an insecure channel between A and B. By insecure channel we mean that the intruder has access to the channel and can read, analyze, change, and substitute the content of the stream which is passing through the channel. We also assume that, A wants to send a message M authentically to B. If the intruder is able to produce a pair $[M, MD]$ which is not generated but accepted as a genuine pair of [message , digest] by the legitimate parties, we say that he/she has broken the method. Note that, we ignore the case that the intruder might replace the pair of [message , digest] with another pair which had been sent before (*Replay Attack*). Usual method of avoiding this attack is using *time stamps*. We also assume that the intruder never deletes the content of a communication, as it is always possible to thwart this attack. For instance, a serial number can be added to the message or, as another example, sender can wait for the acknowledgment of the message.

Note that, the security of our methods depends on that of the hash function, therefore, we assume that MD5 is a secure hash function (refer to the next section). We use the term *hard* in this paper as *computationally infeasible*. The terms *cryptanalyst, intruder*, and *enemy* have the same meaning.

In our methods, we have supposed that the two parties (A and B) share a 128-bit secret key K_{AB} (and K'_{AB} in Method 6). It is not always possible (or is difficult) to provide a secret key between each two parties (eg, on a wide computer network with many users). In this case, existence of an arbiter becomes useful. To find more about the basic key distribution schemes refer to [13].

1.2 MD5 Summary

MD5 is a hashing algorithm that maps an arbitrary length message to a 128-bit message digest (hash value). It is build to be fast on machines with 32-bit registers. MD5 always pads some bits (P), consisting of a '1' followed by zeros and the length of the message, to the end of the input message and makes its length an exact multiple of 512. Each 512-bit block of the message is processed in one MD5 loop which is constructed of four rounds (64 steps). The 128-bit result of each step is called an *intermediate value* of the MD5 algorithm. In the process of the first 512-bit block, MD5 uses a 128-bit buffer, called *initial vector (IV)*. For the other blocks, MD5 uses the output of the previous block (loop), and finally, the last output is the message digest. In [12], the author has conjectured that:

1. The difficulty of finding a message with a given message digest is on the order of 2^{128} operations.
2. The difficulty of finding two messages with the same message digest is on the order of 2^{64} operations.

We assume that the above conjectures, which keep MD5 secure, are true. The details about MD5 can be found in [12].

2 Tsudik's Proposed Methods

Tsudik [14] has proposed three methods for message authentication. They are *Secret Prefix*, *Secret Suffix*, and *Envelope Method* that are briefly explained in this section. More details on these methods and the possible attacks are given in [14].

1. *Secret Prefix Method:* When A wants to send a message M to B, she follows the following steps:
   - Use the 512-bit secret key S_{AB}^{p},
   - Compute $MD = \mathbf{MD4}(S_{AB}^{p} \parallel M)$,
   - Send the pair $[M, MD]$ to B.

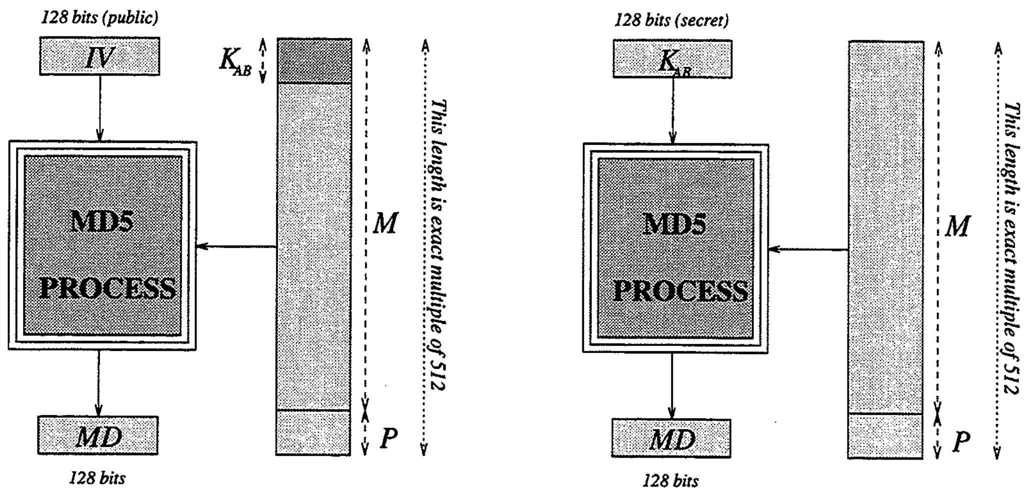
   Since B knows the key S_{AB}^{p}, he can verify MD by recomputing $\mathbf{MD4}(S_{AB}^{p} \parallel M)$. Note that, the key length is 512 bits and MD4 needs to process this extra 512 bits in 64 steps (one loop).

2. *Secret Suffix Method:* A does the following steps:
   - Use the 512-bit secret key S_{AB}^{s},
   - Compute $MD = \mathbf{MD4}(M \parallel S_{AB}^{s})$,
   - Send the pair $[M, MD]$ to B.

   B can again verify MD, since he knows the key S_{AB}^{s}. Same as the previous method, 512 bits is appended to the message (one extra block).

3. *Envelope Method:* This is the combination of the previous two methods, where A:
   - Uses the secret keys S_{AB}^{p} and S_{AB}^{s} (1024 bits),
   - Computes $MD = \mathbf{MD4}(S_{AB}^{p} \parallel M \parallel S_{AB}^{s})$, and
   - Sends the pair $[M, MD]$ to B.

   In this method the message M is surrounded by S_{AB}^{p} and S_{AB}^{s}. Two blocks (1024 bits) that are added to the message cause the MD4 to perform $2 \times 64 = 128$ extra steps.



(a)- K_{AB} (128 bits) as the prefix of M. *(b)- K_{AB} (128 bits) as the initial vector of MD5.*

Figure 1: Method 1 and Method 2

It is obvious that increasing the key length in the above methods will not increase the security as it is still equivalent to the cryptanalysis of MD4. We show that reducing the key length to 128 bits can still hold the security of the system. Also, the key does not need to be appended to the message.

In the next section, we modify Tsudik's methods and give six methods that provide different environments depending on the user choice. The key length is up to 8 times smaller than that of used in Tsudik's methods. We have XORed the key to the message and sometimes used it as the initial vector.

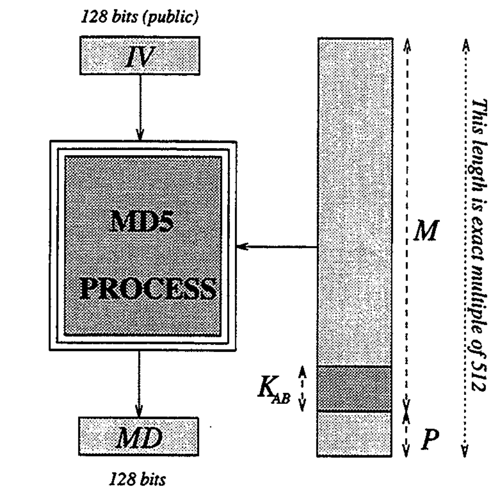
3 The Improved Methods (Using Small Keys)

It is believed that a 128-bit key is strong enough for a system to remain secure against the *Exhaustive Search.* We propose the modified methods corresponding to the methods explained in the previous section and give some new methods which are more efficient in terms of the speed and the key length, and are as secure as the Tsudik's. This section will only give a brief description of the proposed methods. The details about the security of our methods can be found in Section 5.

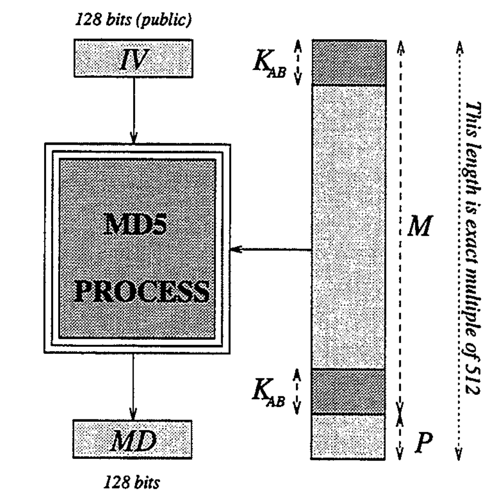
3.1 Method 1: K_{AB} as the Prefix of M

In this method (Figure 1(a)), a 128-bit key K_{AB} is used as the prefix of the message M to compute the message digest. That is, A follows the following steps to send M to B:

1. Use the 128-bit secret key K_{AB},
2. Compute $MD = \mathbf{MD5}(K_{AB} \overline{\oplus} M)$,
3. Send the pair $[M, MD]$ to B.



(a)- K_{AB} (128 bits) as the suffix of M.



(b)- Surrounding the message M with two same 128-bit keys.

Figure 2: Method 3 and Method 4

This method is the modified Secret Prefix method, where we have reduced the key length from 512 to 128 bits (4 times) and have XORed (instead of prepending) the key to the beginning of the message to make the process faster (one block is saved).

3.2 Method 2: K_{AB} as the Initial Vector of MD5

Instead of adding a key to the message, the initial vector of MD5 can be set to a 128-bit secret key. Then, the hash function starts processing the message. This method is illustrated in Figure 1(b) and can be summarized as:

1. Use the 128-bit secret key K_{AB},
2. Set $IV = K_{AB}$,
3. Compute $MD = \mathbf{MD5}(M)$,
4. Send the pair $[M, MD]$.

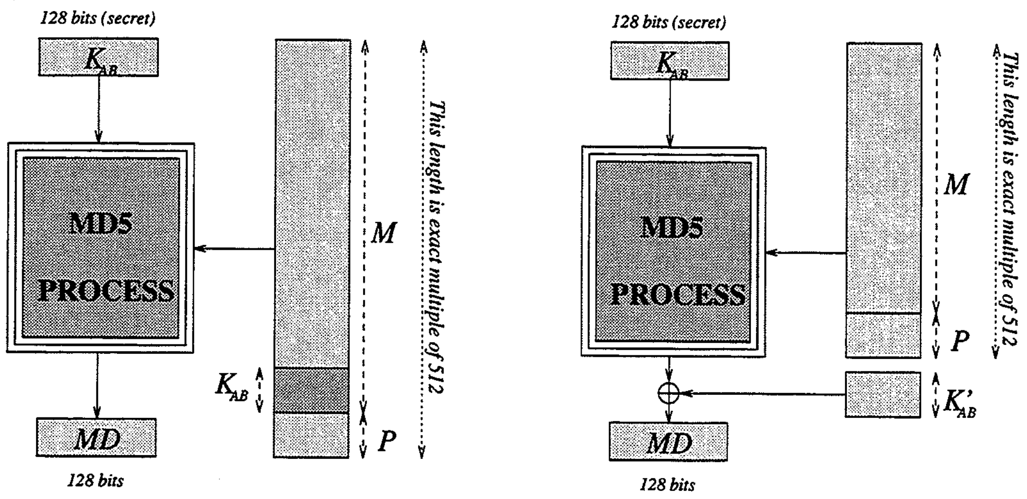
Since no concatenation is done in this method, it is as fast as the MD5 algorithm.

3.3 Method 3: K_{AB} as the Suffix of M

As shown in Figure 2(a), a 128-bit key K_{AB} is XORed to the end of the message M before computing the message digest. When A wants to send M to B, she:

1. Uses the 128-bit secret key K_{AB},
2. Computes $MD = \mathbf{MD5}(M \underline{\oplus} K_{AB})$, and
3. Sends the pair $[M, MD]$ to B.

This method is the modified Secret Suffix method, where the key length is reduced from 512 to 128 bits (4 times) and the key is XORed to the message instead of being appended.



(a)- A 128-bit key as both the initial vector of MD5 and the suffix of M.

(b)- A 128-bit key as the initial vector and another 128-bit key for masking the message digest.

Figure 3: Method 5 and Method 6

3.4 Method 4: Surrounding M with K_{AB}

Figure 2(b) illustrates this method, where the secret key is used twice to provide more secrecy (combination of the first and the third methods). This method can be described as:

1. Use the 128-bit secret key K_{AB},
2. Compute $MD = \mathbf{MD5}(K_{AB} \overline{\oplus} M \underline{\oplus} K_{AB})$,
3. Send the pair $[M, MD]$.

This method uses only a 128-bit key which is 8 times smaller than that of used in Tsudik's Envelope Method. This length of the key also speed up the process by two blocks. Note that, if length of M is less than 256 bits, two keys will overlap. A padding can be done, when this happens.

3.5 Method 5: K_{AB} as Both Initial Vector and Suffix of M

This method is the combination of Method 2 and Method 3. Figure 3(a) illustrates this method, where A follows these steps:

1. Use the secret key K_{AB},
2. Set $IV = K_{AB}$,
3. Compute $MD = \mathbf{MD5}(M \underline{\oplus} K_{AB})$,
4. Send the pair $[M, MD]$ to B.

Similar to Method 4, the key length in this method is only 128 bits, and the speed is almost the same as that of plain MD5.

3.6 Method 6: K_{AB} as Initial Vector and K'_{AB} as a Mask for Message Digest

In this method (Figure 3(b)), instead of having a secret suffix, we use a 128-bit key K'_{AB} to XOR to the result of MD5. The method can be summarized as:

1. Use the secret keys K_{AB} and K'_{AB} (256 bits),
2. Set $IV = K_{AB}$,
3. Compute $MD = \mathbf{MD5}(M) \oplus K'_{AB}$,
4. Send the pair $[M, MD]$.

This method protects the message digest by adding security at the beginning and the end of the process. The speed of the method is almost the same as that of the MD5 algorithm, but 256-bit key is needed. This is the only method that uses 256 bits as the secret key; nevertheless, the key length is 4 times smaller than that of used in Tsudik's Envelope Method. Furthermore, in Section 6 we explain how K'_{AB} can be constructed from K_{AB} to keep the key length equal to 128 bits.

4 Secure Keyed One-Way Hash Function (SKOWHF)

In [3], Berson *et al.* give a formal definition for *Secure Keyed One-Way Hash Function (SKOWHF)*. SKOWHF uses a secret key to increase the security of digest, and can be used for message authentication. (In the literature, it is also called *Message Authentication Code (MAC)*.) The following is the definition of SKOWHF.

Definition 1 *A function $f()$ that maps a fixed length key K and an arbitrary length message M to a fixed length message digest MD is a SKOWHF, if it satisfies the following properties:*

1. *Given K and M, it is easy to compute $MD = f(M, K)$,*
2. *Given K and MD, it is hard to find M with $MD = f(M, K)$,*
3. *Given K, it is hard to find two values M and M' ($\neq M$) such that $f(M, K) = f(M', K)$,*
4. *Given (possibly many) pairs of $[M, MD]$, it is hard to find the secret key K,*
5. *Without knowledge of K, it is hard to compute $f(M, K)$ for any M.*

In general, the above properties are necessary for a keyed hash function to thwart the possible attacks. The first property ensures that the algorithm is easy to compute. The second property is the one-wayness property of the function, when the key is given. Third one relates to the collision freeness of the function. Fourth property keeps the key secret among the legitimate parties. The fifth property prevents the enemy to generate a non original digest. Based of the above definition, when K is fixed, SKOWHF becomes the normal One-Way Hash Function (OWHF). Note that, in keyed hash functions, because of the inclusion of a secret key, the designer can tailor the algorithm according to the security or efficiency requirements. For example, the hash function can be designed to be secure against the outsiders (who do not know the key) or be secure for all people (including the key holders). In the first case, the designer can construct an efficient algorithm, using a well studied secret key in the process. It is clear that the security is increased if the secret key contributes in the process of the first and the last round as well as the body of the round function.

If the designer intend to design a keyed hash function that is secure when the key is known (public), the design procedure becomes more complex. It is not clear whether it is necessary to make a keyed hash function secure against attack by legitimate parties (those who know the key) [10]. Therefor, we can leave the second and the third properties as optional ones, but, they should exist when the key is unknown (for outsiders).

Proposition 1 *The methods described in Section 3 hold the SKOWHF properties, where the function $f()$ is MD5, the message is M, and the key is K_{AB} (and K'_{AB} in Method 6).*

Proof Sketch: MD5 is relatively a fast hash function and $MD = f(M, K_{AB})$ can be easily computed in all methods. This proves the first property.

The second property of SKOWHF emphasize that the function should not be reversible. In our methods, since the message M always accompany the message digest MD, one-wayness is not required (while in general, our methods are one-way). Nevertheless, it should be infeasible to find a different message M' that is mapped to the same message digest MD. In other words, when $[M, MD]$ with $MD = f(M, K)$ is sent, it does not need to be hard to compute M when K and MD are given, but, it should be hard to find an M' with $f(M', K) = f(M, K)$. The latter case is, in fact, the third property, which provides the collision freeness:

Consider Method 1, where $f(M, K_{AB}) = \mathbf{MD5}(K_{AB} \overline{\oplus} M)$. (Proof for the other methods is fairly the same.) If it is easy to find $M \neq M'$ such that $f(M, K_{AB}) = f(M', K_{AB})$, we set $X = (K_{AB} \oplus M)$ and $X' = (K_{AB} \overline{\oplus} M')$. This implies that it is easy to find $X \neq X'$ such that,

$$\mathbf{MD5}(X) = \mathbf{MD5}(K_{AB} \oplus M) = f(M, K_{AB}) = f(M', K_{AB}) = \mathbf{MD5}(K_{AB} \overline{\oplus} M') = \mathbf{MD5}(X').$$

Therefore, it is easy to find a collision for MD5, a contradiction. This proves the third property.

For the fourth property, we prove that it is hard to compute the key when some pairs of [message , digest] are given. In those methods that the key is XORed to the message (such as Method 1), it is hard to compute K_{AB} when some pairs of $[M, MD]$ are given. This is true since MD5 is one-way. [2]

In those methods that the key is used as the initial vector of MD5, it is also hard to find the key when pairs of [message , digest] are given. This is true because in the process of each block the initial value will be added to the result of that block (loop) and therefore causes going backward impossible (one-wayness).

In Method 6, the result of the algorithm, before XORing to the key K_{AB}, is unknown (because the initial vector is secret). We call this fixed intermediate value as X. Now, the question can be simplified to *"Is it hard to find K'_{AB} from $X \oplus K'_{AB} = MD$ when only*

[2]We have also had some practical research to find M when MD and t bits of M are given, with $MD = \mathbf{MD5}(M)$ and $t = (\mathit{length\ of\ } M) - 128$. It is hard to find the unknown 128 bits of the message, because, each bit of the message contributes four times in the process of MD5 blocks (loops).

MD is known?" It is obvious that the answer is yes and therefore the fourth property is proved.

The similar proof that was mentioned above, proves the fifth property. Note that, in MD5 the message digest is heavily depending on both the initial vector and the message. Therefore, wrong choice of even a few bits of the input will affect every bit of the message digest. This makes sure that nobody can compute $f(M, K)$ without knowledge of K. □

5 Security Analysis of the Proposed Methods

In the previous section we proved that our methods hold the SKOWHF properties. That means we have started from MD5 (believed to be OWHF), and constructed six SKOWHFs. Now we would like to examine our methods against the related possible attacks.

The length of the key assures that *Exhaustive Key Search* (or *Brute Force Attack*) does not work on the methods (for fast implementation of this kind of attack refer to [6]). Therefore, we examine *Pseudo Attack*, *Padding Attack*, and *Birthday Attack*.

5.1 Pseudo Attack

We have called this attack *Pseudo Attack*, since the cryptanalyst tries to find a pseudo key $\widehat{K}$ with $f(M,K) = f(M,\widehat{K})$, while K is the real key. For example in Method 1, the intruder should find a $\widehat{K}$ such that $\mathbf{MD5}(\widehat{K} \overline{\oplus} M) = \mathbf{MD5}(K_{AB} \overline{\oplus} M)$. It is obvious that this is equivalent to finding collisions for MD5, and so, it is not practical.

In Method 2, this attack corresponds to the problem of finding *pseudo collisions* for MD5. Pseudo collision for hash functions (MD5 in particular) can be defined as finding two different initial vectors that under a message give a same message digest. The first pseudo collision for MD5 has been found by Boer and Bosselaers in [5]. Their attack cannot give a pseudo collision when one of the initial vectors is fixed, because, they start from a middle point and go forward and backward to find two initial vectors that are mapped to one message digest. This implies that the found initial vectors, message, and message digest look like random numbers. Therefore, their attacks are not applicable to our methods. (Also refer to [7].)

In Method 6, the enemy cannot find a pseudo value for K'_{AB}, because, it is processed separately under an exclusive-or operation. That is, if $X \oplus K'_{AB} = MD$, where X is the intermediate value before the XOR operation, it is not possible to find a $\widehat{K}$ such that $X \oplus \widehat{K} = MD$. (If so, we have $X \oplus K'_{AB} = X \oplus \widehat{K}$, therefore, $K'_{AB} = \widehat{K}$.)

The other methods are the combination of the above ones and, as the result, are secure against this attack.

Note that, even if a pseudo key $\widehat{K}$ for some given pairs of $[M, MD]$ is found, it does not necessarily imply that $\widehat{K}$ can be used with another message M' to generate a genuine pair $[M', MD']$. In other words, suppose we have used the key K to generate $[M_1, MD_1]$, $[M_2, MD_2]$, ..., $[M_t, MD_t]$, for some integer t ($MD_i = f(M_i, K)$, $i = 1, 2, \ldots, t$). Now if the intruder can find a pseudo key $\widehat{K}$ with $MD_i = f(M_i, \widehat{K})$, $i = 1, 2, \ldots, t$, it does not necessarily imply that for an $M' (\neq M_i,\ i = 1, 2, \ldots, t)$, we should have $f(M', K) = f(M', \widehat{K})$.

5.2 Padding Attack

In this attack, the intruder tries to append (or prepend) a message to the existing one such that the result remains acceptable by the legitimate parties. For example, if $[M, MD]$ with $MD = f(M, K)$ is sent by a party to another, the cryptanalyst tries to find an M' such that he/she can evaluate MD' as a genuine message digest, when $(M \parallel M')$ or $(M' \parallel M)$ is used instead of M. Then, $[(M \parallel M'), MD']$ or $[(M' \parallel M), MD']$ is sent instead of $[M, MD]$.

This attack is not applicable to the methods from 3 to 6, since a secret part is added to the end. That is, the secret part does not allow the intruder to evaluate the new message digest MD' and form the pair $[(M \parallel M'), MD']$ or $[(M' \parallel M), MD']$. There remains only one chance that the enemy can search for an M' such that the appending gives the same message digest result as before. This chance is equivalent to the chance of breaking MD5. For example in Method 3, the intruder should find an $\widehat{M}$ such that $\mathbf{MD5}((M \parallel \widehat{M}) \underline{\oplus} K_{AB}) = \mathbf{MD5}(M \underline{\oplus} K_{AB})$ (or $\mathbf{MD5}((\widehat{M} \parallel M) \underline{\oplus} K_{AB}) = \mathbf{MD5}(M \underline{\oplus} K_{AB})$). Finding such an $\widehat{M}$ is equivalent to finding a collision for MD5, that is hard.

In Method 1 and Method 2, it is sometimes possible to append to the message. For example in Method 1, after receiving the pair $[M, MD]$ with $MD = \mathbf{MD5}(K_{AB} \overline{\oplus} M)$, the intruder can set $IV = MD$ and compute $\widehat{MD} = \mathbf{MD5}(\widehat{M})$, where $\widehat{M}$ is an arbitrary message chosen by the intruder and the length of M is bigger than 128 bits. This is equivalent to computing $\mathbf{MD5}(K_{AB} \overline{\oplus} (M \parallel P \parallel \widehat{M}))$, while IV is set to the default value. Then, the pair $[(M \parallel P \parallel \widehat{M}), \widehat{MD}]$ can be sent instead of $[M, MD]$.

There are some ways to avoid the above attack. The followings are three ways, each of which can protect the method against the Padding Attack:

1. Since P is a string of '1' followed by zeros plus the length of $(K_{AB} \overline{\oplus} M)$, sending $(M \parallel P \parallel \widehat{M})$ allows everyone to distinguish the padding P in the middle of the message and find out that the massage is not genuine. (Specially when M is a plain text, such as English text.)

2. Length of the message M can be attached as the header of the message M (or XORed to the beginning of the message) before computing the message digest. The receiver can check this length by doing the same procedure and then can accept the message as genuine.

3. One can always append a text T with a fix length to the message M and check it at the receiving point. That means, B verifies MD by computing $\mathbf{MD5}(K_{AB} \overline{\oplus} (M \parallel T))$. [3]

Note that, the intruder is not able to append anything to the beginning of the message M when Method 1 or Method 2 is used, because, the intermediate values in MD5 algorithm are unknown after the process of the secret key.

Also note that, the methods from 4 to 6 can be considered as the improved versions of Method 1 and Method 2, where instead of attaching the length string, a secret key is used to thwart the Padding Attack.

[3] This text can be, for example, a random string of characters. It does not need to be secret, but should be known exactly by A and B. Also, it should not appear within the message M. A suggestion for the length of T is 16 bytes.

5.3 Birthday Attack

Suppose the message digest MD is produced by applying a hash function on a message M. The enemy, in this attack, generates a pool consisting of many pairs of [message , digest]. When he/she intercepts a message digest, he/she compares the message digest with all message digests in the pool, and in case of a match, sends the corresponding message (which hopefully differ from the real one) instead.

This attack is one of the most powerful attacks on the hash functions with uniform message digest distribution and short message digest length [9, 10]. In [9, Section 3.2], the authors have recommended that the length of the hash value should be around 128 bits to avoid this attack. Therefore, our methods are safe against this attack. [4]

Furthermore, in each method there is a secret part which does not allow the intruder to produce the pairs $[M, MD]$. He/she should wait for these pairs to be transmitted by the legitimate parties. This makes the construction of the required pool more difficult.

6 Discussion and Conclusion

In this paper we showed a new set of methods which were based on a collision free hash function and explained the security of our methods against possible attacks. Security of the methods heavily depends on the security of the hash function. We used the terminologies in [3] to define Secure Keyed One-Way Hash Function (SKOWHF), and proved that our methods are SKOWHF.

In Tsudik's methods, a 512-bit key is added to the message before applying MD4 (the key is 1024 bits in the Envelope Method). This length of the key not only makes the system slow, but also requires unnecessarily large tamper proof memory to store the key. Appending 512-bit information to the input message causes the MD5 (or MD4) algorithm to loop one more to process the extra 512 bits (one block) and therefore, decreases the speed of the system.

In our methods, we not only have reduced the key length by up to %87.5, but also have used XOR operation to prevent concatenation of the key to the message. Existence of the secret keys in our methods does not increase the length of the input, and therefore, does not decrease the speed of the process. (XOR operation takes no time, comparing with the process of one block in MD5.) Note that, speed was, in fact, the main reason that we replaced hash-then-encrypt algorithms with keyed hash functions.

In our proposed methods, we believe that the fifth method (Figure 3(a)) is the best one. This method is safe enough against the possible attacks and, at the same time, is efficient. Figure 3(a) shows that the only operation added to the normal process of the hash function (MD5) is the XOR operation on 128 bits. Therefore, the method is fast. On the other hand, the message is surrounded by the secret key and this thwarts any possible attack.

In Method 6, it is possible to have $K'_{AB} = K_{AB}$ but, since the initial vector (K_{AB}) is used as a mask for the result of MD5, when the length of M is small (< 448), the addition

[4]If length of the messages is on average 16 bytes, each pair of [message , digest] needs $16 + 16 = 2^5$ bytes. Therefore, if the intruder wants to have a pool containing about 2^{64} pairs of $[M, MD]$ to get a probability of around $\frac{1}{2}$, he/she needs $2^5 \times 2^{64} = 2^{69}$ bytes for storing the data (this is almost 6×10^{11} Gbytes). On the other hand, if process of each pair takes 2^{-20} ($\simeq 10^{-6}$) seconds, process of the pool takes $2^{64} \times 2^{-20} = 2^{44}$seconds (this is almost 557844 years).

modulo 2^{32} at the end of the MD5 process and the XOR operation in our method affect each other and reveal some key bit information.[5] To avoid this problem, an arbitrary text should be added to the input message to increase its length to more than 512 bits. As a suggestion, a 512-bit text consisting of 64 space characters can be added to the message M, whenever the length is less than 448. Nevertheless, we do not recommend using this modified method.

It is useful to mention that there is another method which is completely insecure: A might use the default IV and compute $MD = \mathbf{MD5}(M) \oplus K_{AB}$. In this case, since the pair $[M, MD]$ is known to everyone, The cryptanalyst can compute $\mathbf{MD5}(M) \oplus MD$ and find the secret key. This idea caused Method 6 to be invented. In fact, the enemy cannot find the intermediate value before the XOR operation (since the initial value is secret). Therefore, the linear dependency of K'_{AB} and MD will not help the intruder to get any information about the key.

In Method 4 and Method 5, we have used K_{AB} twice in the process. The security of MD5 allows us to have this iteration. Meanwhile, notice that we can have the second key different from the first one. They can be completely independent, such as K_{AB} and K'_{AB} in Method 6, or different but dependent, such as $K'_{AB} = \mathbf{MD5}(K_{AB})$. The latter technique can be also applied to Method 6 to reduce the key length from 256 to 128 bits.

Another option that makes the methods more secure but slower is to XOR the key to the all message blocks. (Note that, for example in Method 1, we XORed the key to the beginning of the message. That means, The key is XORed to the beginning of the first message block.)

We also suggest using other hash functions depending on the different demands. MD4 can be used when a faster implementation is needed. Note that, MD4 is less complicated than MD5 and seems to be less secure than MD5 (refer to [4]). HAVAL [15], is another hash function that can be used when a more secure keyed hash function is demanded. The interesting point with HAVAL is its ability to manipulate the message digests with different sizes. For instance, one can change our methods by increasing the key length and the message digest length from 128 to 256 bits, using HAVAL.

References

[1] S. Bakhtiari. Report on Hash Functions. Internal report in the Department of Computer Sience, University of Wollongong, March 1995.

[2] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. A Fast Keyed Hash Function (KHF). In *Submitted to: Cryptography — Policy and Algorithms Conference, Queensland University of Technology*, 1995.

[3] T. A. Berson, L. Gong, and T. M. A. Lomas. Secure, Keyed, and Collisionful Hash Functions. Technical Report (included in) SRI-CSL-94-08, SRI International Laboratory, Menlo Park, California, December 1993. Last revised version (September 2, 1994).

[5]If length of the message M is less than 448, four bits of the key will be revealed. This is because, addition modulo 2^{32} is like XOR operation on the least significant bit, and as the result, addition modulo 2^{32} followed by the XOR operation reveals the four least significant bits of the four 32-bit message digest words.

[4] B. Boer and A. Bosselaers. An Attack on the Last Two Rounds of MD4. In *Advances in Cryptology, Proceedings of CRYPTO '91*, pages 194–203, 1991.

[5] B. Boer and A. Bosselaers. Collisions for the Compression Function of MD5. In *Advances in Cryptology, Proceedings of EUROCRYPT '93*, pages 293–304, 1994.

[6] H. Eberle. A High-Speed DES Implementation for Network Applications. In *Advances in Cryptology, Proceedings of CRYPTO '92*, pages 521–539, 1992.

[7] B. S. Kaliski Jr. and M. J. B. Robshaw. On "Pseudocollisions" in the MD5 Message-Digest Algorithm, April 1993. RSA Laboratories, 100 Marine Parkway, Redwood City, CA 94065.

[8] National Bureau of Standard. *Data Encryption Standard.* FIPS publication, June 1977. No. 46, U.S. Department of Commerce.

[9] J. Pieprzyk and B. Sadeghiyan. *Design of Hashing Algorithms.* Springer-Verlag, 1993.

[10] B. Preneel. *Analysis and Design of Cryptographic Hash Functions.* PhD thesis, Katholieke University Leuven, January 1993.

[11] R. Rivest. The MD4 Message-Digest Algorithm. RFC 1320, April 1992. Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc.

[12] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, April 1992. Network Working Group, MIT Laboratory for Computer Science and RSA Data Security, Inc.

[13] J. Seberry and J. Pieprzyk. *Cryptography: An Introduction to Computer Security.* Prentice Hall, 1989.

[14] G. Tsudik. Message Authentication with One-Way Hash Functions. *IEEE INFOCOM*, May 1992.

[15] Y. Zheng, J. Pieprzyk, and J. Seberry. HAVAL - A One-Way Hashing Algorithm with Variable Length of Output. In *Advances in Cryptology, Proceedings of AUSCRYPT '92*, pages 83–104, December 1992.

The Compression Function of MD2 is not Collision Free.

N. ROGIER , P. CHAUVAUD

CNET PAA/TSA/SRC

38-40, rue du Général Leclerc

92131 ISSY LES MOULINEAUX (France)

Abstract : In 1989, B. Kaliski introduced the MD2 Message Digest Algorithm which takes as input a message of arbitrary length and produces as output a 128-bit message digest, by appending some redundancy to the message and then iteratively applying a 32 bytes to 16 bytes compression function [1].

This function has been updated in 1993 in the RFC 1423 document. It was conjectured that the number of operations needed to get two messages having the same message digest is on the order of 2^{64} (using the birthday paradox), and that the complexity of inverting the hash function is on the order of 2^{128} operations. No attack against this function has been published so far. In this paper, we propose a low complexity method to find collisions for the compression function of MD2.

1. Description of MD2.

The MD2 hash function accepts a b-byte message as input, and produces a 16-bytes output. The MD2 computations are byte oriented. They involve a byte-permutation S of the {0..255} set, and consist of the three following steps:

• Padding.

The message is padded by appending i bytes, where i is comprised between 1 and 16, as to obtain a L-bytes message, where L is congruent to 0 modulo 16. The value of each of the i padding bytes is taken equal to i. A message consists of L/16 16-bytes blocks $M_0 ... M_{L/16-1}$ is thus obtained.

• Appending a checksum block.

A 16-bytes checksum block $M_{L/16}$ is appended to the previous message, thus providing a L/16+1 blocks message $M_0 ... M_{L/16}$.
The algorithm for calculating $M_{L/16}$ is the following (an auxiliary byte t is used):

```
t:=0;
for j:=0 to 15 do $M_{L/16}$:=0;
for k:=0 to L/16-1 do
    for j:=0 to 15 do
    begin
        $M_{L/16}[j]$:= $M_{L/16}[j]$ XOR S[t XOR $M_k[j]$];
        t:=$M_{L/16}[j]$;
    end;
```

Remark: $M_k[j]$ denotes the byte number j, j in [0..15], of the M_k block.

• Computing the digest of the $M_0 ... M_{L/16}$ message.

A compression function md2(.,.) from two 16-bytes blocks to a 16-bytes block is iteratively applied to the $M_0 ... M_{L/16}$ message, using the recurrence:

```
    H:=0; (initial 16-byte hash value equal to zero)
    for k:=0 to L/16 do H:=md2(H,M_k)
```

The MD2 hash value is the final value of the 16-bytes block H.

Description of the md2 compression function of MD2:

Let H= (H[0], ..., H[15]) and M= (M[0], ..., M[15]) be two 16-bytes blocks. The 16-bytes block md2(H,M) is given by the following algorithm, where an auxiliary matrix T[0..18,0..48] of 19*49 bytes is used.

```
    (* initialisation *)
    for j:=1 to 16 do
    begin
        T[0,j]:=H[j];
        T[0,j+16]:=M[j];
        T[0,j+32]:=H[j] xor M[j];
    end;
    T[1,0]:=0;

    (* main loop *)
    for i:=1 to 18 do
    begin
        if i>1 then T[i,0]:=T[i-1,48]+i-1 mod 256;
        for j:=1 to 48 do T[i,j]:=T[i-1,j] XOR S(T[i,j-1]);
    end;

    (* output *)
    for j:=0 to 15 do md2(H,M)[j]:=T[18,j+1];
```

2. Principles used in our Attack

2.1 A General Collision Search Method.

Let h be a compression function from m bytes to n bytes and n_1 and n_2 be integers such that $n_1 + n_2 = n$.

We denote by $h_1(x)$ (respectively $h_2(x)$), where x is a m-bytes word, the n_1-bytes word (respectively the n_2-bytes word) given by the n_1 first bytes (respectively the n_2 last bytes) of h(x).

The following property essentially says that if it is easy to find multiple collisions for h_1, i.e. several values providing the same h_1 output, then this can be used to find collisions for the h function.

Proposition (P)

If it is computationally easy to find a multiple collision of size $1.17*256^{n_2/2}$ for h_1(i.e. if $1.17*256^{n_2/2}$ x_i values such that all the $h_1(x_i)$ values are equal can be found in no more than $O(256^{n_2/2})$ operations), then if h_2 behaves as a random mapping, a collision for h can be found with a 0.5 probability in time $O(256^{n_2/2})$ operations.

Proof:

This is a straightforward application of the "birthday paradox" to the h_2 function.

2.2 General Properties of the MD2 Compression Function.

We consider three matrices T_1,T_2 and T_3 of 19*17 bytes defined as follows :

$$T = (T_1|T_2|T_3)$$

with these conventions:

The columns of T_1 are the columns 0 to 16 of T.

The columns of T_2 are the columns 16 to 32 of T.

The columns of T_3 are the columns 32 to 48 of T.

First we will study some specific properties of one of those submatrices. To simplify the writing we note it B as block.

The definition of the compression function gives the following three relations :

$\forall i \in [1..18]\ \forall j \in [1..16]\ B[i,j] = B[i-1,j]\ \text{XOR}\ S(B[i,j-1])$ (1)

$$\begin{tikzcd}
 & & & B[i-1,j] \arrow[dl] \\
B[i,j-1] \arrow[r] & S \arrow[r] & \oplus \arrow[r] & B[i,j]
\end{tikzcd}$$

$\forall i \in [1..18]\ \forall j \in [0..15]\ B[i,j] = S^{-1}(B[i,j+1]\ \text{XOR}\ B[i-1,j+1])$ (2)

$$\begin{tikzcd}
 & & & B[i-1,j+1] \arrow[dl] \\
B[i,j] & S^{-1} \arrow[l] & \oplus \arrow[l] & B[i,j+1] \arrow[l]
\end{tikzcd}$$

$\forall i \in [0..17]\ \forall j \in [1..16]\ B[i,j] = B[i+1,j]\ \text{XOR}\ S(B[i+1,j-1])$ (3)

$$\begin{tikzcd}
 & & & B[i,j] \\
B[i+1,j-1] \arrow[r] & S \arrow[r] & \oplus \arrow[ur] & B[i+1,j] \arrow[l]
\end{tikzcd}$$

Applying those three relations, we obtain the five following properties, which are useful in the sequel.

Property 1 (forward) : (p1)

If the column 0 and the row 0 of the B matrix are known then all the elements of the matrix B can be computed.

<u>Proof :</u>

By using (1) the row 1 can be computed from the row 0 and B[1,0].

$$\begin{tikzcd}
 & B[0,0] \arrow[d] & & & B[0,16] \arrow[d] \\
B[1,0] \arrow[r] & B[1,1] & \cdots & B[1,15] \arrow[r] & B[1,16]
\end{tikzcd}$$

By repeating this operation 18 times, we get all the elements of B.

Property 2 (backward 1) : (p2)

If the column 16 of the B matrix is known, then all the elements of the B matrix under the oblique row (B[0,16] ... B[16,0]), in other words

B[i,j]   j=0..16   i=16-j..18.

can be computed.

<u>Proof</u> :

By using (2) the column 15 from the row 1 to the row 18 can be computed from the whole column 16.

Let's suppose now that the column j, where 0<j<16, is known from the row 16-j to the row 18.

By using (2) the column j-1 can be computed from the row 15-j to the row 18.

By repeating this for each column j from j=15 to j=1, we get all the elements of B under the oblique row (B[0,16] ... B[16,0]).

$$\begin{tikzcd}
 & B[0,16] \arrow[dl] \arrow[d, dash] \\
B[16,0] \arrow[d, dash] & B[16,16] \arrow[l] \arrow[d, dash] \\
B[18,0] & B[18,16] \arrow[l]
\end{tikzcd}$$

Property 3 (p3)

If the column 16 and the row 0 of the matrix B are known then all the elements of the matrix B can be computed.

Proof :

By using (2) the row 1 can be computed from the row 0 and B[1,16].

$$\begin{tikzcd}
 & B[0,0] \arrow[dl] & & & B[0,16] \arrow[dl] \\
B[1,0] & B[1,1] \arrow[l] & \cdots & B[1,15] & B[1,16] \arrow[l]
\end{tikzcd}$$

By repeating this 16 times, we get all the elements of B.

Property 4 (backward 2) : (p4)

If the row 18 of the matrix B is known, then all the elements of the matrix B under the oblique row (B[2,16] ... B[18,0]), in other words

B[i,j] i=2..18 j=18-i..16.

can be computed.

Proof :

By using (3) the row 17 from the column 1 to the column 16 can be computed from the whole row 18.

Let's suppose now that the row i, where 3<i<17, is known from the column 18-i to the column 16.

By using (3) the row i-1 can be computed from the column 17-i to the column 16.

By repeating this for each row i from i=17 to j=1, we get all the elements of B under the oblique row (B[2,16] ... B[18,0]).

$$\begin{tikzcd}
 & B[2,16] \\
B[18,0] \arrow[ur] \arrow[r, dash] & B[18,16] \arrow[u]
\end{tikzcd}$$

Property 5 (backward 3) : (p5)

If the column 0 and the row 18 of the matrix B are known, then all the elements of the matrix B can be computed.

Proof :

Let apply (p3) to the B[18,0], B[18,1], B[17,1] elements: we obtain B[17,1].

By using this process, the column 1 can be computed from the column 0 and B[18,1].

$$\begin{tikzcd}
 & B[1,0] \\
B[1,0] \arrow[ur] & B[1,1] \arrow[u] \\
 & \vdots \\
 & B[17,1] \\
B[18,0] \arrow[ur] & B[18,1] \arrow[u]
\end{tikzcd}$$

We can determine by the same way each column j, where 1<j<17, from the column j-1 and B[18,j].

The above last five properties can be applied to each of the three blocks T_1, T_2 and T_3. Moreover, the following proposition establishes a relation between the blocks T_1 and T_3.

Property 6: (p6)

If the column 16 of T_3 is known, then the columns of T_1 can be calculated from row 2 to 18.

proof :

This property results directly from the definition of the compression function. Indeed, the design of the compression function leads to the following relation :

for each i = 2..18, $T_1[i,0] = T_3[i-1,0] + i-2 \mod 256$.

2.3 General Method for the md2 Collision Search.

Although a general collision for md2 consists of two distinct inputs (H,M) and (H',M') such that md2(H,M) = md2(H',M'), in our attack, we further require H = H': given H, we want to find M and M' such that md2(H,M) = md2(H,M').

Due to (p1), a sufficient condition for md2(H,M) and md2(H,M') to collide is that the 18 T[j,0] bytes (j=1..18) are equal in both computations, i.e. (M,M') provide a collision for the following h function

$$\begin{array}{lcl} \{0..255\}^{16} & \rightarrow & \{0..255\}^{18} \\ M & \mapsto & T[1,0]\cdots T[18,0] \end{array}$$

All the collision attacks developed in this paper consist in applying the method of the proposition (P) to the h function.

3. How to find Collisions for the Compression Function of MD2

The above last six properties are the basis for our attack of the md2(H,M) compression function.

3.1. First case : H = 0.

This situation occurs for example at the beginning of the first round of the compression function.

Remark : Since H = 0, then the row 0 of block T_1 (the M block) and the row 0 of block T_2 (the M XOR H block) are identical.

Our goal is to find M values of such that the column 16 of T_1 is identical with the column 16 of T_2 and T_3, for rows 1 to 15. For that purpose, we perform the following two steps:

First step :

- We calculate this column 16. The $T_1[1,0]$ value is an input value for the compression function. (This value is the initialisation value of "t" used in the algorithm).

The values $T_1[0,j]$ for $j = 0 .. 16$ and $T_1[1,0]$ are then known.

- Using (p1), we calculate $T_1[1,j]$ for $j = 1 .. 16$.
- By hypothesis, $T_1[1,16] = T_2[1,16] = T_3 [1,16]$.
- Using (p6) : the $T_1[2,0]$ value is then calculated.

Similarly, the values of $T_1[i,16]$ $(0<i<15)$ can easily be found from row i and from $T_1[i+1,0]$.

The results of this computation give the values of the column 16 of each T_1, T_2, T_3 block, for rows 1 to 14.

The values $T_2[i,j]$ (for $i = 1..14$ and $j = 17-i,..,16$) are determined using (p2) applied to the column 16 of block T_2.

Second step :

We now consider the 256^2 possible values for the pair $T_2[14,1]$ and $T_2[14,2]$.

- The column $T_2[.,0]$ is known as it is equal with the column 16 of T_1.
- (p5) enables us to find the 256^2 values of M so that the column 16 of T_1, T_2, T_3 are equal from row 1 to 14.
- Using Proposition (P), the number of (M,M') pairs of M values leading to collisions on $T_1[15,0]$, $T_1[16,0]$, $T_1[17,0]$ and $T_1[18,0]$, should be about 128.

Let (M,M') be one of the above pairs of T_1 values. Let notice respectively by T and T' the matrix associated to M and M'.

Remark : The first row and column of T_1 and T_1' are equal.

- Using (p1), the row 18 of T_1 and T_1' are equal. This means that the values M and M' lead to a collision of the compression function: for each (M,M') pair, md2(H,M) = md2(H,M')

Experimental results :

We found experimentally 141 pairs of such M values. Some examples of such pairs of values are given in Section 4.

3.2. Second case : H contains a Sequence of z Consecutive Zero Bytes.

We now consider the more generic case H[j] = 0 for j = d .. 16 with 0<d<16. H contains z = 17-d consecutive zero bytes.

Remark : Since H[j] = 0 for j = d .. 16 with d>0, $T_2[0,j] = T_3[0,j]$ for j = d .. 16.

We start by finding values for the row 0 of T_2, such that $T_1[i,16] = T_2[i,16] = T_3[i,16]$ for i = 1..n_1, with $n_1<z$.

We arbitrarily fix the values of $T_2[0,j]$ for j = 1..d-1.

- Using (p3), then for a given value of $T_2[0,j]$ (or $T_3[0,j]$) for j=d..16, we determine one and only one submatrix of T (we denote by SM this submatrix):

$$SM = \begin{pmatrix} T_2[0,d] & \dots & T_2[0,16] \\ \dots & \dots & \dots \\ T_2[n_1,d] & \dots & T_2[n_1,16] \end{pmatrix} \qquad \text{with } n_1 < z.$$

Therefore for i = 1..n_1, $T_2[i,d-1] = T_3[i,d-1]$.

Using (p1) with values $T_1[0,.]$ and $T_2[0,j]$ for j= 0..d-1, we compute $T_2[1,d-1]$. Then, using (p3) with $T_3[1,d-1]$ (= $T_2[1,d-1]$) and $T_3[0,j]$ for j=0..d-1, we compute $T_2[1,16]$.

Since $T_3[1,16] = T_3[1,16]$ is known, then $T_1[2,0]$ is known using (p6).

These computations are repeated in order to find :

- $T_1[i,j]$ for $i=0..n_1$ and $j=0..16$
- $T_1[n_1+1,0]$
- $T_2[i,j]$ for $i=0..n_1$ and $j=0..d-1$
- $T_2[i,16]$ for $i=0..n_1$
- $T_3[i,j]$ for $i=0..n_1$ and $j=0..d-1$
- $T_3[i,16]$ for $i=0..n_1$

We repeat the same calculations as in the first case with the submatrix SM :

From the $(256)^{z-n_1}$ possible values of the z-n_1-uple ($T_2[n_1,d],T_2[n_1,d+1],...,$
$T_2[n_1,d+z-n_1-1]$), $(256)^{z-n_1}$ values of ($T_2[0,d],...,T_2[0..16]$) are calculated.

We have then a multi-collision on the n_1 first steps of the compression function. Using (P), it is then possible to find a collision if $\left[(256)^{z-n_1}\right]^2 \geq (256)^{17-n_1}$.

As n_1 is strictly positive, we have the condition: $2z-17>0$.

The choice $n_1 = 2z-17$ leads to a 256^{17-z} complexity. Just for instance, $z = 12$ leads to 256^5.

This method provides improved bounds only if H contains at least nine consecutive zero bytes.

3.3. Bounds for Other Values of H

The methods introduced in 3.1 and 3.2 are no longer applicable when H does not contain any sufficiently long sequence of consecutive zero bytes. However, in the general case a 2^{56} bound can still be found on the complexity of finding, for a given H value, a (M,M') pair such that md2(H,M) = md2(H,M').

The algorithm for obtaining this bound can be outlined as follows :

Step 1 :

A "meet in the middle" method is used to obtain a multicollision of size $(256)^7$ on the three bytes T[2,0], T[3,0], T[4,0], in time 256^7 using $(256)^6$ bytes of memory.

Step 2 :

A collision on the 14 bytes T[5,0] to T[18,0] is then searched among the $(256)^7$ blocks contained at Step 1.

Since $(256)^7 = (256)^{14/2}$ the success probability is about 0,5 (by the birthday paradox).

The detail of Step 1 is the following :

- Six arbitrarily values T[1,32], T[2,32], T[3,32], T[1,48], T[2,48], T[3,48] are selected;

- The corresponding values of T[2,0], T[3,0], T[4,0] are deduced from H by applying (p6). By using (p1) in the T_1 submatrix, then these values and H enables to find T[1,16], T[2,16] and T[3,16].

- For $(256)^6$ different values of the (M[0],..,M[7]) 8-uple of bytes, the three bytes T[1,24]; T[2,24]; T[3,24] are computed from T[1,16], T[2,16] and T[3,16], using (p1) in the T_2 submatrix, and the three bytes T[1,40]; T[2,40], T[3,40] are computed from T[1,32]; T[2,32]; T[3,32] using (p1) in the T_3 submatrix.

- The (M[0], ..,M[7]) 8-uples are stored such a way that for each (T[1,24], T[2,24], T[3,24], T[1,40],T[2,40], T[3,40]) 6-uples, the list of corresponding M[0],...,M[7] 8-uples can be accessed in very few elementary operations.

- For $(256)^7$ different values of the (M[8],...,M[15]) 8-uple of bytes, the 3 bytes T[1,24], T[2,24], T[3,24] are computed from T[1,32], T[2,32], T[3,32] using (p3) in the T_2 submatrix, and the 3 bytes T[1,40], T[2,40], T[3,40] are computed from T[1,48], T[2,48],T[3,48] using (p3) in the T_3 submatrix; all the M[0],..., M[7] messages of the list associated to the (T[1,24], T[2,24], T[3,24], T[1,40], T[2,40], T[3,40]) 6-uple provide a contribution to the multicollision.

4. Implementation

A program has been written implementing the algorithm finding collisions for the compression function in the H=0 case. It establishes a work load of finding 141 collisions for the compression function of MD2.

collision 1 :

M = 2EC90ABB 41FCD859 AE7E83A8 D02B835B

M'= 0C7F5F73 82DAB197 5F5D7A8C BF588B86

collision 2 :

M = 02F1473A 6F942524 C017C0DC EF8DBA5B

M'= F96F15D2 5C908A65 BE538043 71B60781

collision 3 :

M = B8833EBB 390DB95A DF649A23 FB95725B

M'= BF99E8D0 3AAE8739 591A71B4 F3E92734

collision 4 :

M = 6B2FC868 D1562335 C3A8AA3F 79F8DB44

M'= 4B33ABDC FCE255EA 19C73DEB 6645ACD4

collision 5 :

M = 690A85B6 E8769D72 A1469F40 FC5EB971

M'= 84423007 339002EF 210AA8F6 0E7D7883

Conclusion.

The weaknesses we have found in the MD2 compression function are not sufficient to prove that MD2 is a weak hash function. However it has become an usual requirement for iterative hash functions that the compression function be conjectured collision free [2]. Since the MD2 compression function is not collision free, the whole security of MD2 rests on the redundancy introduced by the checksum block.

Acknowledgements.

The authors would like to thank Henri Gilbert for many helpful discussions and comments on the MD2 compression function.

Bibliography.

[1] B. Kaliski, "The Message Digest Algorithm MD2", RFC1115, RSA Laboratories, April 1992.

[2] Y. Damgard, "Design Principles for Hash Functions", Advances in Cryptology, Proceedings of Crypto'89, Springer Verlag.

[3] B. Preneel, "Analysis and Design of Cryptographic Hash Functions", Katholieke Universiteit Leuven, Thesis 1993.