2006

A Meaningful MD5 Hash Collision Attack

Narayana D. Kashyap
San Jose State University

A Meaningful MD5 Hash Collision Attack

A Writing Project Presented to the Faculty of the

Department of Computer Science

San Jose State University

In Partial Fulfillment of the

Requirements for the Degree

Master of Science

By

Narayana D Kashyap

Aug 2006

Dedicated to

My parents *Hema* and *Datha*

And my sweet Grandma *Ajji*

ACKNOWLEDGEMENTS

I would like to thank Dr. Mark Stamp for his guidance, insights and immense patience, without which my project would have been impossible to complete. His suggestions and his work in this field helped me tremendously in understanding and working on the topic. Dr. Stamp also provided appropriate research papers, including his own book, which aided me in identifying the areas to concentrate and consequently write a sound project statement.

I would also like to thank Dr. Sami Khuri and Prof. David Blockus for agreeing to be the committee members to review and certify my project.

Finally, I would like to express my gratitude to Asif, Vinod, Venkat, Bharath, Joshi, Vamsi, Karan, Lakshmi, Amulya, Pavan, Chakki, Manu, my brothers Chythanya and Vinay and all my other friends and family members who have supported me immensely both in technical and moral spheres.

ABSTRACT

It is now proved by Wang et al., that MD5 hash is no more secure, after they proposed an attack that would generate two different messages that gives the same MD5 sum. Many conditions need to be satisfied to attain this collision. Vlastimil Klima then proposed a more efficient and faster technique to implement this attack. We use these techniques to first create a collision attack and then use these collisions to implement meaningful collisions by creating two different packages that give identical MD5 hash, but when extracted, each gives out different files with contents specified by the atacker.

Keywords: MD5, hash, collision, Wang, attack

Table of Contents

Table of Illustrations

1 Introduction to cryptography

The evolution of cryptography was led by the idea of information security, i.e., cryptography is the science of securing the information. It involves encryption and decryption of messages. Encryption is the process of converting a plain text into cipher text and decryption is the process of getting the original message back from the encrypted text. Cryptography also provides Integrity, Authentication, and Non-repudiation, in addition to confidentiality [2].

There are many known cryptographic algorithms. The basis of any cryptographic algorithm is the "key" used for encrypting/decrypting the information. Many of the cryptographic algorithms are available publicly, though some believe in having a secret algorithm. The general method has been to use a publicly known algorithm while maintaining a secret key [7].

Based on the key used, there are two categories of cryptosystems: S*ymmetric* and A*symmetric*. In Symmetric Key Cryptosystems, the same key is used for both Encryption and decryption. i.e. if K and M were the key and the message, then, we have $D_K(E_K(M)) = M$ where D and E denotes decryption and encryption. Advantages of this system are speed and security based on the strength of key.
There are some disadvantages too. Exchange and administration of the key gets complicated and non-repudiation is impossible. Examples: DES, 3-DES, RC4, RC5 etc [2][7].

In Asymmetric, also called shared key or Public key cryptosystems, two different but interchangeable keys are used for encryption and decryption. The two keys are linked mathematically. One of the keys is made public (shared) while the other is kept secret. i.e. if k1 and k2 are public and private keys, respectively and M be the message, then $D_{k2}(E_{k1}(M)) = D_{k1}(E_{k2}(M)) = M$ [2].

Public key systems are considered to be very secure and encourages non-repudiation. Key exchange is not required thus minimizing the key administration. But the ciphertext tend to be much bigger than plaintext and is much slower than Symmetric systems.
Examples: Diffie-Hellman, RSA and Elliptic Curve Cryptography.

The idea of using Elliptic curves in cryptography was first suggested by Victor Miller and Neal Koblitz. This was introduced as an alternative to established public-key systems such as RSA and DSA. The Elliptical curve Discrete Log Problem (ECDLP) makes it more complicated to break an ECC as compared to RSA and DSA where the problems of the discrete log problem or factorization can be solved in sub-exponential time. This implies that ECC can be built with much smaller parameters than in other systems like RSA and DSA [27].

2 Cryptosystems and Public key cryptography

The word "cryptography" is derived from the Greek word "kryptos", which means "secret writing". Cryptography has been around for over a thousand years and the Romans were thought to be the masters of cryptography since they used simple cipher techniques to conceal the real meaning conveyed by the messages [12].

Cryptographic systems are generally categorized on the following basis:

1. **Method used for encryption**: Most encryption algorithms are based on two principles,
    a. ***Substitution***, in which each character in plain text is replaced by some other character to get the cipher text
    b. ***Transposition,*** in which characters in plain text are reorganized to get cipher text.
2. **Processing of Plain text**: A Stream cipher processes the input characters continuously producing the output. A Block cipher processes the input one block at a time, with one output block for each input block.
3. **Number of keys used**: If a single key is used by both the sender and receiver, then it is called a Symmetric or conventional system. If different keys are used, then it is an Asymmetric or public-key system [7].

2.1 Outline of some cryptographic algorithms

2.1.1 Diffie-Hellman (DH) public-key algorithm:

This was the first public-key algorithm invented in 1976. The discrete logarithm problem forms the basis of this algorithm. Alice and Bob both agree on two large prime numbers n and g such that g is primitive mod n, over an unprotected channel. This is the basic algorithm:

1. Alice chooses a large random number a and sends Bob $x = g^a$.
2. Bob picks another large random integer b and sends Alice: $y = g^b$
3. Alice computes k from y, $k = y^a$
4. Similarly, Bob computes $k' = x^b$

k and k' both will be = g^{ab} mod n. An evesdropper Eve would have the knowledge of only n, g, x and y. She cannot get a and b due to the Discrete Logarithm problem. The security depends on picking large values of g and n.

If random number generators are used whose outputs are not entirely random but are predictable, then the task for Eve becomes easy. This is susceptible to "Man in the Middle Attack" which can be explained as follows –

When Alice asks Bob for his public key, Eve can intercept Bob's key and sends another public key to Alice for which she has a private key. Alice encrypts her message with Eve's key and sends the ciphertext to Bob which is again intercepted by Eve who decrypts it with her key and encrypts again with Bob's key and sends it to Bob who believes was sent to him by Alice.

This attack can be significantly defeated by using digital signatures [4].

2.1.2 RSA

RSA is a public-key cryptosystem invented by – Rivest, Shamir and Adleman from MIT in 1977. It has endured extensive cryptanalysis for many years. It is used in many secure communications over the internet. It is a block cipher in which the original message and ciphertext are integers between 0 and n – 1 for some integer n. The difficulty in factoring large numbers builds the security of RSA [16].

2.1.2.1 Algorithm

Choose two large random prime numbers p and q of almost equal length. Let n = pq. Get the Euler's Totient function $\phi(n)$ which is computed as $\phi(n) = (p - 1)(q - 1)$. Select two keys a and b such that, $ab \equiv 1 \pmod{\phi(n)}$. One of them is made public while the other secret.

The message M is encrypted as $C = M^a \bmod n$.

C is decrypted back to M by using $M' = C^b \bmod n$.

So, $M' = (M^{a)b} \bmod n = M^{ab} \bmod n = M^{k\phi(n) + 1} \bmod n$ (as $ab \equiv 1 \pmod{\phi(n)}$)

Therefore $M' = M \cdot M^{k\phi(n)} \bmod n = M \bmod n$

(Fermat's Little Theorem $a^{\phi(n)} \equiv 1 \pmod{n}$) [6]

2.1.2.2 RSA Security

Three potential attacks to the RSA are:

- ***Mathematical attacks:*** Basically factoring the two primes.
- ***Brute Force:*** Attempting all likely private keys.
- ***Timing attacks:*** Based on the running time of decryption algorithm.

Timing Attack

A cryptosystem takes different amounts of time to process different inputs in giving the outputs. This depends on optimization techniques, instruction processing and branching and other various reasons. Here the input parameters are the plain text and the encryption key or the ciphertext and decryption key. This non-constancy of running times, when carefully analyzed, can reveal significant information about the secret key. Measuring carefully, the amount of time taken for various known data by the vulnerable systems and studying meticulously the statistics acquired, an attacker can extract the hidden secrets of the system and gain control over it [8].
This idea was first propounded by Kocher[11]. According to him, a 512 bit key can be broken in a few minutes if three hundred thousand timing measurements can be obtained [8].

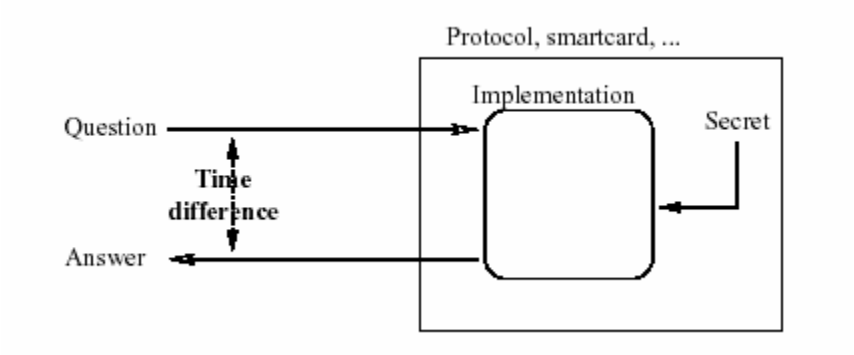

Figure 1: The principle behind timing attack

Source: [8] J.-F. Dhem et al.

Timing attack on RSA: RSA operations on private key is a simple modular exponentiation of the form $R = y^x \bmod n$, where n is public and y can be found out. The objective of an attacker is to figure out x. If a new x is chosen for every operation, this attack fails. This attack can be made to work with any implementation that runs in non-constant time, but is first drafted using the simple modular exponentiation algorithm that computes $R = y^x \bmod n$. The exponent bits can be found by running the algorithm as many number of times as the number of bits of the exponent [11].

3 Hash Functions

A hash *h* is generated by a hash function H of the form

$$h = H(M)$$

where M is a message of variable length and H(M) is the hash value of fixed length.

A hash function should satisfy the following properties to be useful:

1. A hash function can be applied to a data block of any size.
2. It always produces an output of fixed length.
3. It must be easy and efficient to compute H(x) for any given x. Though the effort depends on the length of x, it should not be a function of its length.
4. One-way: It should not be possible to find x for any given value h such that h = H(x).
5. Weak collision resistance: Given x, it is computationally infeasible to find $y \neq x$ such that H(y) = H(x).
6. Strong collision resistance: It is computationally infeasible to find any pair (x,y) such that H(x) = H(y) [10].

All hash functions are operated using the following principle: The input is divided into sequence of n-bit blocks and is then processed one block at a time iteratively which produces an m-bit hash value. A simple hash function is bitwise exclusive-OR of every block [2].

Generally, the message is padded before processing by the hash function so that the message can be split into blocks of equal size. Most hash functions use Merkle-Damgard construction in which the input message M is partitioned into L blocks of fixed size, say *b* bits. If needed, the final block is padded to make it *b* bits long. The value of total length of the input message is also included in the final block [3].

A compression function, *f,* is used repeatedly in the hash algorithm, that takes two inputs, an n-bit input called the chaining value, from the previous step, and a b-bit block, and gives out an n-bit output. The chaining value has an initial value, specified by the algorithm, at the start of the first iteration. Generally, n< b, and hence the term compression.

This can be summarized as given below:

$CV_0 = IV$ = initial value

$CV_i = f(CV_{i-1}, Y_{i-1})$ $1 \leq i \leq L$

$H(M) = CV_L$

where the input message M is divided into blocks Y_0, Y_1,…..Y_{L-1}.

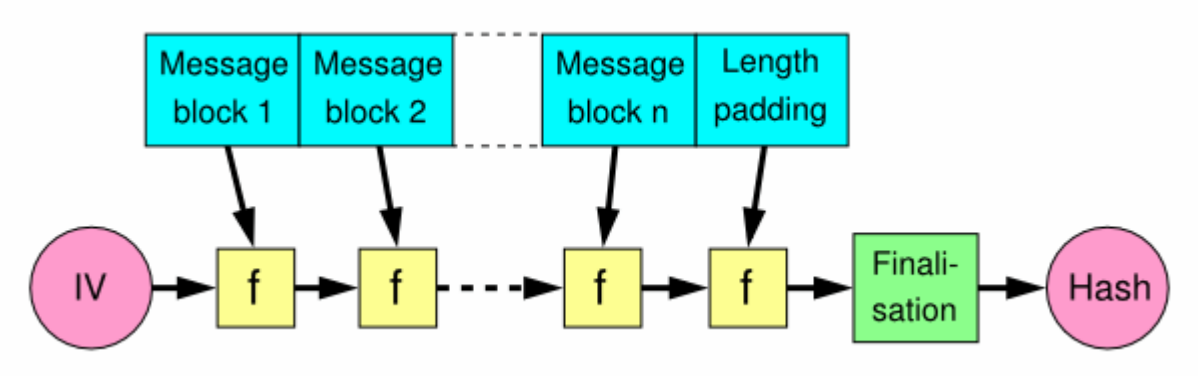

Figure 2: Merkle-Damgard Construction

Source: [3] Wikipedia

3.1 Application of Hash Functions

Hash functions are used in different contexts, like digital signatures, password protection, message authentication codes and as pseudo-random number generators.

3.1.1 Digital Signature

Digital signatures are used for various purposes including the following:

To guarantee authenticity: The recipient is assured that the message was indeed sent by the claimed sender.

To avoid repudiation: The sender cannot claim that he did not sign that message.

Generally, a message is first hashed before signed so as to reduce the size of the signature [26].

3.1.2 Password Protection

Passwords are stored after hashing instead of plaintext for obvious reasons. When a password is typed, the newly computed hash is compared with the existing hash and if it matches, then the password is declared correct [26].

4 MD5

MD5 is a hash function that operates on 512-bit blocks of data and gives a 128-bit hash value. The message is padded to get the length in bits to be congruent to 448 modulo 512. It's always padded even if the message is of the desired length. The length of the original message in bits is represented by 64 bits and is appended to the padded message to make it an integer multiple of 512 bits [3].

The 128-bit initial values IV for the compression function of MD5 are:

$Q_{-3} = 0x67452301$

$Q_{-2} = 0x10325476$

$Q_{-1} = 0x98badcfe$

$Q_0 = 0xefcdab89$

The compression function also takes as input, a 512-bit message block split into 16 words of 32 bits. It involves 64 steps and in each step *i,* an additive constant K_i, is used. This is computed using the following formula

$$K_i = \lfloor 2^{32} \times \text{abs}(\sin(i+1)) \rfloor$$

where *i* is in radians [2].

Each step involves a bitwise function $f_i(X,Y,Z)$ which takes the form of one of the following four functions:

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z) \quad \text{for } 0 \leq i \leq 15$$

$$G(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z) \quad \text{for } 16 \leq i \leq 31$$

$$H(X, Y, Z) = X \oplus Y \oplus Z \quad \text{for } 32 \leq i \leq 47$$

$$I(X, Y, Z) = Y \oplus (X \vee \neg Z) \quad \text{for } 48 \leq i \leq 63$$

Where X, Y, and Z are 32-bit words, and $\oplus, \vee, \wedge, \neg$ denote XOR, OR, AND, and NOT operations respectively.[3] The steps 0 to 15 is often referred as Round 1, steps 16 to 31 as Round 2, steps 32 to 47 as Round 3 and steps 48 to 63 as Round 4 [10].

The order of message word m_i that is processed in each of the 64 steps is given by the following formula [26]. If W_i is the word used in step i, then $W_i = m_i$ where *i* is:

i	for $0 \leq i \leq 15$
$(5i + 1) \bmod 16$	for $16 \leq i \leq 31$
$(3i + 5) \bmod 16$	for $32 \leq i \leq 47$
$7i \bmod 16$	for $48 \leq i \leq 63$

Every step also has a rotation or shift value S_i which is derived from the following table:

S_i	*i mod 4*				
		0	**1**	**2**	**3**
i div 16	**0**	7	12	17	22
	1	5	9	14	20
	2	4	11	16	23
	3	6	10	15	21

Table 4.1: Shift value for each step i in the MD5 compression function [26]

The compression function can now be summarized as follows:

Algorithm for the compression function of MD5 [26]

Input: $m_0,\ldots\ldots,m_{15}$ and $(Q_{-3}, Q_{-2}, Q_{-1}, Q_0) = IV$

for i = 0 to 63 do

$$Q_{i+1} = Q_i + (f_i(Q_i, Q_{i-1}, Q_{i-2}) + Q_{i-3} + k_i + W_i) <<< S_i$$

end

return $(Q_{61} + Q_{-3}, Q_{62} + Q_{-2}, Q_{63} + Q_{-1}, Q_{64} + Q_0)$

Figure 3 illustrates the MD5 processing of a single 512-bit block [2], where

CV_q denotes the chaining variable processed with the q^{th} message block

Y_q is the q^{th} 512-bit block

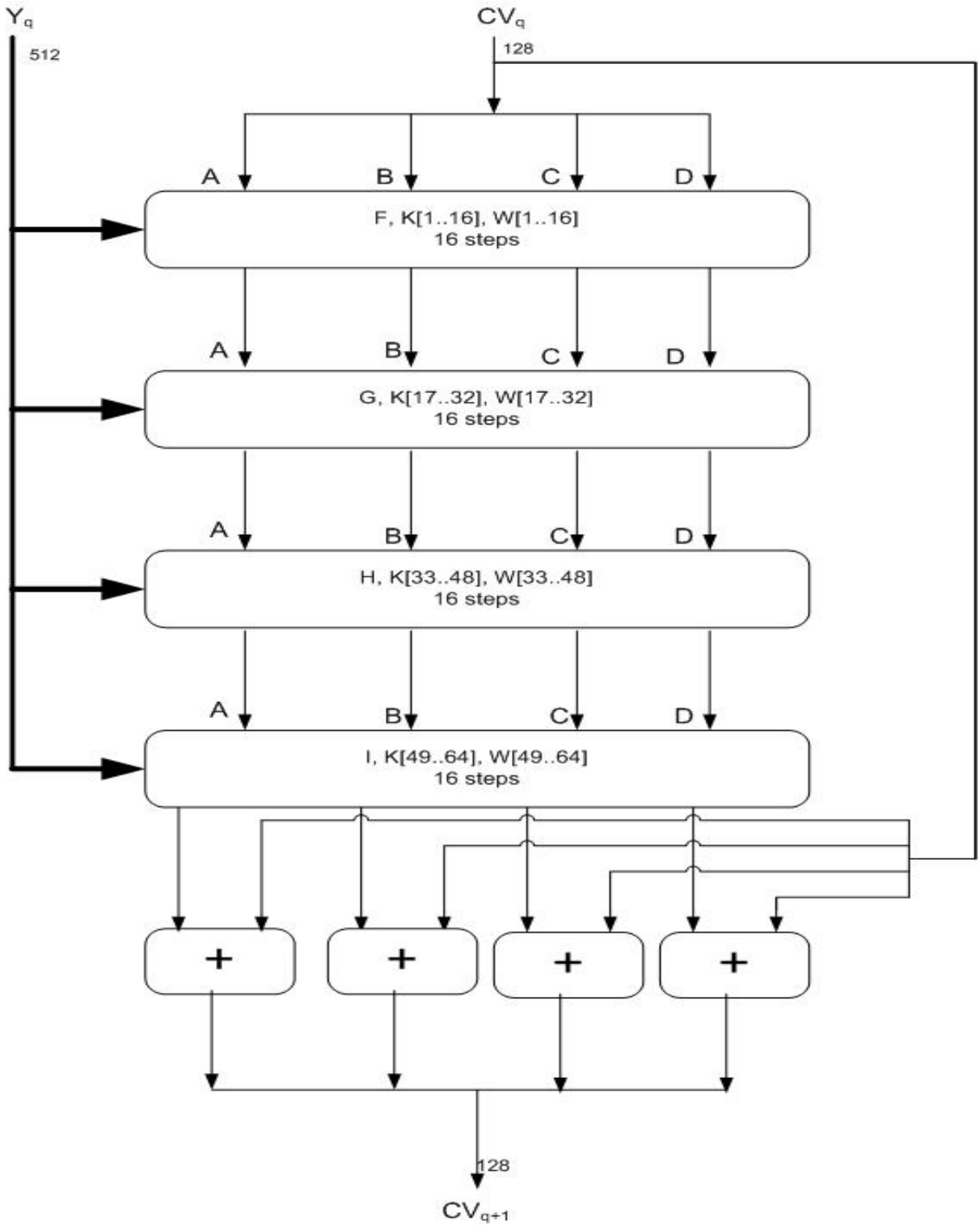

Figure 3: MD5 processing of a single 512-bit block.

Source: [2] William Stallings

5 Wang's Attack on MD5

Cryptanalysis of hash functions concentrates on the structure of the compression function and is based on finding techniques to create collisions for a single execution of the function. It should also take into account the initial values. The attack involves analyzing the changes in bit pattern from round to round.

Three types of attacks are often considered on a hash function. They are:

Collision attack: Finding two different messages that gives the same hash value

Preimage attack: Finding a message that maps to a given hash value

Second Preimage attack: Finding another message that hashes to the same value as the given message [26].

The MD5 attack explained here, proposed by Wang [20], finds a pair of 1024-bit messages, that gives the same MD5 hash value. The two messages are denoted by $M = (M_0, M_1)$ and $M' = (M'_0, M'_1)$, where each M_i and M'_i is a 512-bit block having 16 words of 32 bits each.

Initially, a collision was given by Wang without explaining any underlying technique. This led to a remarkable attempt to reverse engineer Wang's method and this has provided the foundation for further improvements in the attack.

5.1 Differential cryptanalysis

Wang's MD5 attack is a differential attack. The basic kind of difference operator is the *modular difference* which is subtraction modulo 2^{32}. Wang's attack uses this for inputs. Nonetheless, some parts of the attack require a more elaborate information than modular subtraction offers. This calls for the use of "precise differential" [20] which includes modular difference as well as XOR difference and also provides extra information beyond what these two standard differentials offer [10] .

Consider the pair of bytes x' = 01010010 and x = 01000010 and the pair y' = 10100011 and y = 10110011. We have

$$x' - x = y' - y = 00010000 = 2^4$$

which implies that these two pairs are similar as far as modular difference is concerned. But, MD5 attack requires more information to clearly distinguish even between these kinds of pairs. Though XOR difference provides information on the differing bit positions, we require more

information than the combined effect of these two differences. To be specific, we need to know whether the positional difference in bit is a +1 or a -1 [10].

Let $y = (y_0, y_1, \ldots, y_7)$ and $y' = (y'_0, y'_1, \ldots, y'_7)$ where each y_i and y'_i is a bit. Now, working on the same examples given above

$$y' - y = 2^4$$

and

$$y' \oplus y = 10110011 \oplus 10100011 = 00010000$$

The non-zero bit in the XOR difference is because $y'_3 = 1$ and $y_3 = 0$. But, this would result in the same XOR difference if $y'_3 = 0$ and $y_3 = 1$. Wang's attack requires even this difference to be distinguished. For this purpose, *signed difference* is used which is basically a signed version of XOR difference. That is, if the i^{th} bit of the XOR difference is 1, then we put a "+" if $y'_i = 1$ and $y_i = 0$ and a "-" implies $y'_i = 0$ and $y_i = 1$. We use a "." if the bit if $y'_i = y_i$, i.e., the XOR difference is 0. The signed difference is denoted by ∇y[1] and modular difference as Δy. So now, $\Delta y = 2^4$ and $\nabla y =$ "…+…." [10].

Now consider

$$x' - x = 00101000 - 00011000 = 2^4$$

and

$$x' \oplus x = 00101000 \oplus 00011000 = 00110000$$

Therefore, $\nabla x =$ "..+-….".

From ∇x, the XOR difference can be easily derived and although ∇x contains all the necessary difference information, modular difference is still retained for convenience.

For a specified ∇x, each "+" denotes a 1 in the corresponding bit of x' and 0 in x and similarly, a "-" indicates the presence of 0 in x' and 1 in x. But for any bit position which is neither a "+", nor a "-", the bits in x' and x should agree but the value is arbitrary. Both can be either 1 or 0.

Therefore, it should be noted that while the signed differential is more restrictive then the standard differentials, it however allows for significant freedom in choosing the values that satisfy a given differential [10].

---

[1] The name of the symbol ∇ is *nabla*. It is also called by the names *del* and *grad*. It is used as a vector differential operator in Vector Calculus. On the lighter side, the author likes to call it *oolta*, which means "upside down" in the author's language.

5.2 Wang's Differentials for MD5 Attack

Input Differential Pattern:

Let the MD5 initial values be denoted as IV = (A, B, C, D) and the vector for the second block M_1 be $IV_1 = (A_1, B_1, C_1, D_1)$. Then,

$$IV_1 = (A_1, B_1, C_1, D_1) = (Q_{61}, Q_{64}, Q_{63}, Q_{62}) + (A, B, C, D)$$

where $(Q_{61}, Q_{64}, Q_{63}, Q_{62}) = MD5_{1..64}(IV, M_0)$. Then the hash value of (M_0, M_1) is

$$h = MD5_{1..64}(IV_1, M_1) + (A_1, B_1, C_1, D_1).$$

Similarly, define IV'_1 and h' for M'_0 and M'_1. The input modular differences for Wang's attack are specified as

$$\Delta M_0 = M'_0 - M_0 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0)$$

$$\Delta M_1 = M'_1 - M_1 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0)$$

This implies that the messages M_0 and M'_0 differ only in words 4, 11 and 14 and similarly does M_1 and M'_1 differ. It is also required that,

$$\Delta IV_1 = IV'_1 - IV_1 = (2^{31}, 2^{25} + 2^{31}, 2^{25} + 2^{31}, 2^{25} + 2^{31})$$

$$\Delta h = h' - h = (0, 0, 0, 0).$$

Intermediate collisions can be observed in both the blocks from step 23 to 34. This implies $Q_i = Q'_i$ for $24 \leq i \leq 35$ in both iterations [10].

Output Differential Pattern:

Wang's output differentials are provided in the Appendix A in Tables A-1 and A-2. The following description of the columns facilitates the reading of the two tables: The column *j* specifies the step, *Output* refers to the output while processing M_0 and M_1 respectively, W_j is the data element used at step *j*, ΔW_j is the modular difference in the outputs for M'_0 and M_0 in A-1 and M'_1 and M_1 in A-2, Δ*Output* is the modular difference between the input for M'_0 and M_0 in A-1 and M'_1 and M_1 in A-2 and ∇*Output* is the signed difference corresponding to Δ*Output* [10].

Wang did not provide much information about many critical points in her attack and it appears from the brief descriptions she provided, that her approach was a very intuitive one. The real essence in Wang's attack lies in the choice of difference patterns. The selection of input differences was made in such a way so as to behave nicely in later rounds. The bigger mystery lies in the choice of the output differences. There is no proven method of constructing effective

difference patterns. Daum [14] proposes building a "tree of difference patterns", that includes both input and output differences. Enumerating the input difference pattern will limit the branching to something manageable, but the tree should still be trimmed to stop the exponential growth. Since most branches will have low probabilities, a cost function that uses probability can be used to trim the tree. Daum suggested to use a meet-in-the-middle approach to find the inner collisions and these could be woven together to create collision for a whole message block. Nevertheless, neither this approach nor any other technique has yet generated a useful difference pattern, apart from Wang's intuition [10].

5.3 The Outline of the Attack

Let us first distinguish between input differences and output differences. Input differences are the modular differences between input words of the two messages M and M', while output differences are differences between corresponding intermediate values Q_i and Q'_i. Wang's attack is a pure differential attack [10]. It thoroughly specifies the input differences and applies tight constraints on the output differences.

The general idea behind the attack is:

a) Generate a random 512-bit message block M_0.

b) Perform the step operation for each step and modify M_0 to make sure that all the conditions on the step variable are satisfied. If some of the conditions cannot be satisfied, then start again.

c) Verify that all the required differences are satisfied and if so, then we have found M_0 that is needed for the attack.

d) Use the MD5 output obtained after processing M_0, to set the initial values for M_1.

e) After M_0 is found, generate a random 512-bit message block M_1.

f) Perform the step operation for each step and modify M_1 to make sure that all the conditions on the step variable are satisfied. If some of the conditions cannot be satisfied, then start again.

g) Verify that all the required differences are satisfied and if so, then we have found a collision.

h) Calculate $M'_0 = M_0 + \Delta M_0$ and $M'_1 = M_1 + \Delta M_1$.

i) The MD5 hash of $M = (M_0, M_1)$ will be equal to the MD5 hash value of $M' = (M'_0, M'_1)$ [10] [26].

5.4 Reverse Engineering Wang's Attack

Wang and her team did not supply any information on how the collision was achieved, which led Hawkes, Paddon and Rose [24] to do some investigation based on the only collision presented. This is a significant work since the most effective attacks are based on this work rather than the consequent details Wang provided. Carefully analyzing the differential conditions at every step, they obtained conditions on outputs that, if satisfied, can yield a collision. Let us consider a few steps for the first message block [10].

Let us define

$$T_j = f_j(Q_j, Q_{j-1}, Q_{j-2}) + Q_{j-3} + k_j + W_j$$
$$R_j = T_j <<< S_j$$
$$Q_{j+1} = Q_j + R_j$$

If Δ is the notation for modular difference modulo 2^{32}, then

$$\Delta T_j = \Delta f_j(Q_j, Q_{j-1}, Q_{j-2}) + \Delta Q_{j-3} + k_j + \Delta W_j$$
$$\Delta R_j = (\Delta T_j) <<< S_j$$
$$\Delta Q_{j+1} = \Delta Q_j + \Delta R_j$$

and

$$\Delta f(Q_j, Q_{j-1}, Q_{j-2}) = f(Q'_j, Q'_{j-1}, Q'_{j-2}) - f(Q_j, Q_{j-1}, Q_{j-2}) = \Delta f_j$$

The authors of [24] calculated ΔQ_j, Δf_j, ΔT_j and ΔR_j for every j. Then they obtained the conditions on ΔT_j bits to make sure that the preferred differential path is maintained. The conditions for the first round of M_0 block are given in Table B-1 in Appendix B. For the purpose of saving space, the table has a "+" or "—" on top of n to represent 2^n or -2^n and "$\pm$" to indicate the number that can be 2^n or -2^n [10].

Let us examine the first few rows of the table to ascertain the conditions on T_j that makes sure the rotation yields the desired result:

$\Delta T_j = T_j' - T_j$, which implies

$(\Delta T_j) <<< S_j = (T_j') <<< S_j - (T_j) <<< S_j = \Delta R_j$

But, this may not hold true always, if a carry propagates from the lower- order bits past $(31 - S_j)^{th}$ bit or a carry from higher-order bits goes past the 31^{st} bit and hence proper measures must be taken so that the carries are not propagated [26].

Suppose, $T' = 2^{20}$ and $T = 2^{19}$ and $S = 10$, then $\Delta T = T' - T = 2^{19}$ and

$$(\Delta T) <<< S = (T' - T) <<< S = (T' <<< S) - (T <<< S) = 2^{29}$$

Now, consider this:

$T' = 2^{18}$ and $T = 2^{19}$ and $S = 9$. Then,

$$(\Delta T) <<< S = (T' <<< S) - (T <<< S)$$
$$= 2^{27} - 2^{28}$$
$$= -2^{27}$$

But, if $S = 18$, then

$$(\Delta T) <<< S = -2^5$$

but, $(T' <<< S) - (T <<< S) = 2^4 - 2^5 = -2^4$

Thus, with ΔT given, different values can still be obtained for $\Delta R = \Delta T <<< S$ because of many restrictions on T [10].

Now, we will look into a few steps where the restrictions on T_j are deduced from the collision given. Before that, note should be made that $-2^{31} = 2^{31} \pmod{2^{32}}$ and $2^{31} + 2^{31} = 0 \pmod{2^{32}}$.

Let us consider step 5 in Table B-1, where

$$\Delta f_5 = 2^{19} + 2^{11},\ \Delta Q_5 = -2^6 \text{ and } \Delta Q_6 = \pm 2^{31} + 2^{23} - 2^6$$

So,

$$\Delta R_5 = \Delta Q_6 - \Delta Q_5 = \pm 2^{31} + 2^{23}$$

We have $S_5 = 12$ and therefore, we would like to have

$$\Delta R_5 = \Delta T_5 <<< 12 = \pm 2^{31} + 2^{23}$$

which implies that $\Delta T_5 = 2^{19} + 2^{11}$ will not propagate any carry into higher-order bits.

Now consider step 6 where

$\Delta f_6 = -2^{14} - 2^{10}$ and since $S_6 = 17$, we would like to have

$$\Delta R_6 = \Delta T_6 <<< 17 = \Delta Q_7 - \Delta Q_6 = \pm 2^{31} - 2^{27} - 2^0$$

However, a simple rotation of $\Delta T_6 = -2^{14} - 2^{10}$ by 17 positions will give us $-2^{31} - 2^{27}$, which does not equal the desired result. Therefore, ΔT_6 must be modified in such a way that a bit wraps around into the 0^{th} position. Thus $\Delta T_6 = -2^{15} + 2^{14} - 2^{10}$ [10].

[24] explains in complete detail how to obtain all the conditions on T_j that should be met for the differential to hold.

Conditions on Q_j

Let us now consider the conditions for the outputs, Q_j. The efficiency of the eventual attack depends on the number of these conditions being satisfied. Analysis of output differences requires the use of "signed difference", ∇X which provide more information than the combined information that modular and XOR differences provide [10].

The authors of [24] computed the values of ΔQ_j, ∇Q_j, Δf_j and ∇f_j for all the steps of MD5 collision provided by Wang. Those values for the first round of first block M_0 are given in Table 6.1.

t	δQ_t	∇Q_t	∇f_t	δf_t
0-4	-	.....	.....	-
5	$\bar{6}$	.....-++++++++++++++++.....	.....+.....+.....	$\overset{+}{19}, \overset{+}{11}$
6	$\overset{\pm}{31}, \overset{+}{23}, \overset{-}{6}$	±.....+.....-.....	.....-+++++++.++++.....	$\overset{-}{14}, \overset{-}{10}$
7	$\overset{-}{27}, \overset{+}{23}, \overset{-}{6}, \overset{-}{0}$	±+++++---.....-+++++-+++++	....-.-.....+.....+....+..-..	$\overset{-}{27}, \overset{-}{25}, \overset{+}{16}, \overset{+}{10}, \overset{+}{5}, \overset{-}{2}$
8	$\overset{-}{23}, \overset{-}{17}, \overset{-}{15}, \overset{+}{0}$	.....-..-+++-+.....+	±.....-.....+.....+.+.+.....	$\overset{\pm}{31}, \overset{-}{24}, \overset{+}{16}, \overset{+}{1}., \overset{+}{8}, \overset{+}{6}$
9	$\overset{\pm}{31}, \overset{-}{6}, \overset{+}{0}$	±.....-++....+-	±....+..-..-.....+.....+	$\overset{\pm}{31}, \overset{+}{26}, \overset{-}{23}, \overset{-}{2}., \overset{+}{6}, \overset{+}{0}$
10	$\overset{\pm}{31}, \overset{+}{12}$	±.....+-.....	.....-.....+.....+.....+	$\overset{-}{23}, \overset{+}{13}, \overset{+}{6}, \overset{+}{0}$
11	$\overset{\pm}{31}, \overset{+}{30}$	±+.....	.....-.....-	$\overset{-}{8}, \overset{-}{0}$
12	$\overset{\pm}{31}, \overset{-}{13}, \overset{-}{7}$	±.....-++++++....-+.....	±.....+--.....+.....	$\overset{\pm}{31}, \overset{+}{17}, \overset{+}{7}$
13	$\overset{\pm}{31}, \overset{+}{24}$	±.....+-.....	±.....-++++++.....	$\overset{\pm}{31}, \overset{-}{13}$
14	$\overset{\pm}{31}$	±.....	±.....+.....	$\overset{\pm}{31}, \overset{+}{18}$
15	$\overset{\pm}{31}, \overset{-}{15}, \overset{+}{3}$	±.....-.....+...	±.....+.....	$\overset{\pm}{31}, \overset{+}{25}$

Table 6.1: Propagation of differences through the functions f_j in the first round of the first block for the collision provided by Wang et al. [24]. [2]

The first round uses the function F which is

$$F(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z).$$

[2] Notice that the symbol δ is used in place of Δ. There is no distinction between the two symbols in this context.

This function makes use of the bits in X to decide between the corresponding bits of Y and Z. If i^{th} bit of X is 1, then the i^{th} bit of F is i^{th} bit of Y; else it is the i^{th} bit of Z. The conditions on the bits of Q_j can be derived by using the information in Table 6.1 along with the definition of the function F.

Let us consider step 5 where the attacker has $\Delta Q_3 = 0$, $\Delta Q_4 = 0$ and $\Delta Q_5 = -2^6$ and wants $\Delta f_5 = 2^{19} + 2^{11}$. The only way to achieve this from the differences available is if the carry in $\Delta Q_5 = -2^6$ propagates into higher order bits. The relevant information for f_5 can be observed in Table 6.2.

	δ	∇
Q_3		. . . . .
Q_4		. . . . .
Q_5	$\bar{6}$	. . . . . −+++++++++++++++ . . . . .
f_5	$\overset{+}{19}, \overset{+}{11}$	. . . . . + ... + ...

Table 6.2: Computation of f_5 [24]

From ∇Q_5, we have $Q_5[22] = 1$ and $Q_5[21 - 6] = 0$. The bits 6, 7, …, 21 are called the *non-constant* bits and the remaining bits are called *constant* bits. That is $Q_5 = Q'_5$ for constant bits and $Q_5 \neq Q'_5$ on non-constant bits [10].

Consider the constant bits first: The function $F(Q_5, Q_4, Q_3)$ selects bits of Q_4 or Q_3 depending on the bit of Q_5. From ∇Q_5, we have $Q_5[i] = Q'_5[i]$ for bit i = [0..5, 23..31] and

If $Q_5[j] = 1$, then $F_5[j] = Q_4[j]$ and $F'_5[j] = Q'_4[j]$

If $Q_5[j] = 0$, then $F_5[j] = Q_3[j]$ and $F'_5[j] = Q'_3[j]$.

Now, for the non-constant bits: For ∇F_5 to have the value specified in Table 6.2, we need $F'_5[6..10, 12..21] = F_5[6..10, 12..21]$. However, we have $Q'_5[6..10, 12..21] = 1$ such that $F'_5[6..10, 12..21] = Q'_4[6..10, 12..21]$. Since $Q_5[6..10, 12..21] = 0$, we have $F_5[6..10, 12..21] = Q_3[6..10, 12..21]$. And since $Q_4 = Q'_4$, for the conditions of ∇F_5 to hold for bits 6 through 10, 12 through 18, 20, 21, the conditions $Q_4[6..10, 12..21] = Q_3[6..10, 12..21]$ are enough [10] [24].

To consider the non-constant bits 11 and 19, we need $F'_5[11, 19] = 1$ and $F_5[11, 19] = 0$. From ∇Q_5, we have $Q'_5[11, 19] = 1$ and $Q_5[11, 19] = 0$ which in turn means that

$F'_5[11, 19] = Q'_4[11, 19]$ and $F_5[11, 19] = Q_3[11, 19]$. From ∇F_5, we wish to have $F'_5[22] = F_5[22]$

Since $Q_3 = Q'_3$, it is required to have the condition $Q_3[22] = Q_4[22]$.

Thus, the summary of conditions derived from this step 5 is:

$Q_5[6..21] = 0$ $\quad$ $Q_5[22] = 1$

$Q_4[11, 19] = 1$ $\quad$ $Q_3[11, 19] = 0$

$Q_3[6..10, 12..21] = Q_4[6..10, 12..21]$

5.5 Message Modification

The authors of [24] have obtained a set of conditions on the outputs Q_j by continuing the method described in the previous section. A collision will result if all these conditions are met. And luckily, these conditions in the first round (initial 16 steps) can be met by simple modifications of the message words, called the *Single Step Modification*. Then the technique *Multi Message Modification* is applied to satisfy the conditions on Q_j for $j > 15$, while all the conditions for $j < 16$ still hold. The two techniques of message modification are elaborated in the following sections.

5.5.1 Single Step Modification

This approach is also known as single-message modification. This technique is based on the fact that each of the 16 messages appears once in the first 16 steps, and that the output Qj can be changed by modifying a message word Wj.

For example, a message block is randomly selected. Using the single-step modification technique, the message words are modified to force all the conditions on Qj to hold,
for j = 0, 1, …, 15. It is important to note that if M0 = (X0, X1, …, X15), $W_i = X_i$,
for i = 0, 1, …, 15.

Suppose $M'_0 = (X'_0, X'_1, ..., X'_{15})$ was randomly selected as the first message block, and let W'_i, for i = 0,1, …, 63 be the corresponding input words to the MD5 algorithm. The goal is to modify M0 to obtain a message block $M_0 = (X_0, X_1, ..., X_{15})$ for which all of the first round output conditions hold, i.e., all of the conditions on Qi, for i < 16 hold [10].

Considering step 2 with an assumption that X0 and X1 have been found, and that IV consists of (Q_{-4}, Q_{-3}, Q_{-2}, Q_{-1}). Then, using M_0', we have

$$Q_2' = Q_1 + (F_1 + Q_{-2} + W_2' + K_2) <<< s_2$$

The idea is to change Q'$_2$ to Q_2 such that bits of Q_2 and Q'$_2$ are identical, ensuring that $\langle Q_2 = 0 \rangle_{12,20,25}$. i.e., the 12th, 20th and 25th bits of Q_2 are 0. For i = 0, 1, …, 31, let E_i be a 32-bit word defined by

$$\langle E_i = 1 \rangle_i \text{ and } \langle E_i = 0 \rangle_j \text{ for } j \neq i,$$

i.e., E_i is 0 except for bit i, which in this case is 1. Thus, $E_i = 2^{31 - i}$.

Furthermore, let the bits of Q_2' be

$$Q_2' = (q_0, q_1, q_2, ..., q_{31}).$$

Also let $D = -q_{12}E_{12} - q_{20}E_{20} - q_{25}E_{25}$. Thus, to satisfy the desired condition on Q_2, let

$$Q_2 = Q_2' + D.$$

Suppose that W_2' in the above equation is replaced with the value of W_2 for which

$$Q_2 = Q_1 + (F_1 + Q_{-2} + W_2 + K_2) <<< s_2.$$

This value of W_2 can be algebraically determined as,

$$W2 = ((Q_2 - Q_1) >>> s_2) - F_1 - Q_{-2} - K_2.$$

Notice that from the above equation Q_2 is known, and all terms on the right-hand side of the last equation are known. Therefore, W2 is now known. Furthermore, by letting $X_2 = W_2$, the value of Q_2 can be determined, thereby satisfying the output conditions at step 2.

Applying similar process to steps 0 through 15 results in the message $M_0 = (X_0, X_1, …, X_{15})$ for which all the output conditions in these steps will hold. The remaining conditions are tested to see if they all hold. If they do hold, it means that a collision has been identified. However, there might be a case when a condition beyond step 15 does not hold. In such a scenario, a new

random number M_0' is selected and the entire process is repeated. Given the fact that the probability of each condition being held is about 1/2, an attack can be expected with a work factor of about 2^c, where c is the number of conditions in steps 16 through 63.

Thus, single-step modification provides an effective shortcut for simulating an attack, although a multi-step modification could further reduce the work factor as can be seen in the next section [10].

5.5.2 Multi-Step Modification

When it is required to satisfy some of the conditions in steps beyond 15, a multi-step modifications (or multi-message modifications) can be adopted. However, care must taken to ensure that when using this technique the outcomes from previous steps are not violated. This renders the multi-step modification technique more complicated and rigorous that the single-step modifications technique. Sometimes a multi-step modification technique could be not deterministic to a certain extent, i.e., a condition could fail with a small probability. The paper by Daum [14] provides a good description of several such techniques [10].

Let $M_0' = (X_0', X_1', ..., X_{15}')$ be the first message block after the single-step modifications. Assuming that the desired output condition $\langle Q_{16} = 0 \rangle_0$ should hold, for step 16 we have

$$Q_{16}' = Q_{15} + (f_{15} + Q_{12} + W_{16}' + K_{16}) <<< s_{16},$$

where $W_{16}' = X_1'$ and $f_{15} = G(Q_{15}, Q_{14}, Q_{13})$.

Also, let $Q_{16}' = (q_0, q_1, ..., q_{31})$ and $D = -q_0 E_0$, where $\langle E_i = 1 \rangle_i$ and $\langle E_i = 0 \rangle_j$ for $j \neq i$. These two variables can be used to show that $Q_{16} = Q_{16}' + D$ will satisfy the required condition at step 16. Again, replacing W_{16}' with W_{16} results in

$$Q_{16} = Q_{15} + (f_{15} + Q_{12} + W_{16} + K_{16}) <<< s_{16}.$$

$$\Rightarrow W_{16} = ((Q_{16} - Q_{15}) >>> s_{16}) - f_{15} - Q_{12} - K_{16}.$$

However $W_{16} = X_1$, and therefore it much be ensured that all of the conditions in the first round involving X_1 hold. The fact that Q_i, for i = 1, 2, 3, 4, 5, also depend on X_1 calls for each of these steps to be analyzed, except Q_1, since no condition was initially specified for the same [10].

Applying the single-step modification with the new input at step 16, i.e., $X_1 = W_{16}$,

$$Q_1 = Q_0 + (f_0 + Q_{-3} + X_1' + K_1) <<< s_1 .$$

Thus,

$$Z = Q_0 + (f_0 + Q_{-3} + X_1 + K_1) <<< s_1 .$$

In other words, the modified X_1 from step 16 gives a new value for Q_1, which is the Z.

Also,

$$Q_2 = Z + (f_1(Q_1, Q_0, Q_{-1}) + Q_{-2} + X_2' + K_2) <<< s_2 .$$

Again, choosing X_2 such that,

$$Q_2 = Z + (f_1(Z, Q_0, Q_{-1}) + Q_{-2} + X_2 + K_2) <<< s_2$$

$$\Rightarrow X_2 = ((Q_2 - Z) >>> s_2) - (f_1(Z_1, Q_0, Q_{-1}) - Q_{-2} - K_2 .$$

It is important to notice that using X_2 in the above form eliminates any effect on the output condition from step 2 when modification is made in X_1 selection process. In other words, all of the conditions on Q_2, before and after the single-step modification process, hold true [10].

Similarly, choosing

$$X_3 = ((Q_3 - Q_2) >>> s_3) - (f_2(Q_2, Z, Q_0) - Q_{-1} - K_3$$

$$X_4 = ((Q_4 - Q_3) >>> s_4) - (f_3(Q_3, Q_2, Z) - Q_0 - K_4$$

$$X_5 = ((Q_5 - Q_4) >>> s_5) - (f_4(Q_4, Q_3, Q_2) - Z - K_5 .$$

Notice that any other X_i need not be modified, since Z – the new Q_1 – is not used in calculating any other Q_i.

Thus, all the conditions on step 16 have been deterministically satisfied, while maintaining all of the conditions on steps o through 15 resulting from the single-step modifications. There are several other multi-step modification techniques than the one explained above that have reduced work factor of Wang's attack. While Wang's attack is currently highly efficient, and it has been very difficult to find improvised multi-step modifications, there are some advanced modification

techniques although their effectiveness and efficiencies hold low probabilities. As such, further advancements in this area are expected to be incremental [10].

5.6 Klima's technique

Vlastimil Klima published a paper [33] that explained some changes to find the first block of the collision, utilizing the pattern Wang et al. found. It has been proven that the first block collisions can be found in a few minutes, much faster than the original Wang's approach [26].

The outline:

A smart use of the fact that no conditions are specified for Q_1 and Q_2 in the first block, is made by Klima. He therefore lets Q_{17} and Q_{20} to decide the values of m_0 and m_1. This also makes sure that the condition on Q_{20} is held and the selection of new values for Q_{17}, which determines Q_{18} and Q_{19}, till all the conditions on the three values are satisfied, leaves us with thirty-three conditions to take care of. There will be 31 bits of Q_{20} that are free to try and this means that we will be required, on an average, to select new values for Q_1 to Q_{19} four times prior to obtaining a near collision. So, the time required to choose the first 19 Q values and calculate the appropriate message words will be very negligible [26].

This simple technique is best described in the form of Algorithm 5.1.

A few conditions on the step variables in second iteration can be modified to improve the attack by a small factor. First, it is not necessary to have $\nabla Q_{64} = -2^{25} \pm 2^{31}$. The previous step variable needs this requirement to make it possible to control the distribution of bit differences through the compression function. Nevertheless, Q_{64} is never used in an *f* function and only the modular difference $-2^{25} + 2^{31}$ is required so that this is cancelled out while calculating the final addition with the chaining variables. Thus, the process can be speeded up by a factor of 2, by discarding the condition $Q_{64}[25] = 1$.

The condition on $Q_{63}[25]$ can also be discarded as the compression function in the 63^{rd} step is $I(Q_{63}, Q_{63}, Q_{61})$, and $Q_{61}[25] = 1$ is required to hold true. If $Q_{63}[25] = 1$, then the 25^{th} bit of the output of *I* will not be affected by the changes on the 25^{th} bit of Q_{63} and Q_{62} because of the fact that

$$I(1,1,1) = I(0,0,1) = 0.$$

Algorithm 5.1: Klima's technique of MD5 attack [26]

Make sure that M and M' with the difference defined form a near-collision
repeat
  Arbitrarily choose $Q_3, Q_4, \ldots, Q_{16}$, but fulfilling conditions, including T-conditions [24]
  Compute $m_6, m_7, \ldots, m_{15}$ from the just chosen Q-values
  repeat
    Arbitrarily choose Q_{17}, but fulfilling the conditions
    $Q_{18} \leftarrow Q_{17} + (G(Q_{17}, Q_{16}, Q_{15}) + Q_{14} + m_6 + k_{17}) <<< 9$
    $Q_{19} \leftarrow Q_{18} + (G(Q_{18}, Q_{17}, Q_{16}) + Q_{15} + m_{11} + k_{18}) <<< 14$
  until all conditions on Q_{17}, Q_{18} and Q_{19} are fulfilled
  $m_1 \leftarrow (Q_{17} - Q_{16}) >>> 5 - G(Q_{16}, Q_{15}, Q_{14}) - Q_{13} - k_{16}$
  if $Q_{19}[31] = 0$ **then**
    $Q_{20} = 0$
    $Z = 2^{31} - 1$
  else
    $Q_{20} = 2^{31}$
    $Z = 2^{32} - 1$
  end if { To make sure that the single condition on Q_{20} is met}
  while $Q_{20} \leq Z$ and all the conditions are not yet met **do**
    $m_0 = (Q_{20} - Q_{19}) >>> 20 - G(Q_{19}, Q_{18}, Q_{17}) - Q_{16} - k_{19}$
    $Q_1 = Q_0 + (F(Q_0, Q_{-1}, Q_{-2}) + Q_{-3} + m_0 + k_0) <<< 7$
    $Q_2 \leftarrow Q_1 + (F(Q_1, Q_0, Q_{-1}) + Q_{-2} + m_1 + k_1) <<< 12$
    Calculate m_2, m_3, m_4, m_5 according to steps 2 to 5
    Calculate all the remaining step variables. If a condition is not met, then
    let $Q_{20} = Q_{20} + 1$ and continue
  end while
until all the conditions are met

The 25th bit of I's output will still not change if $Q_{63}[25] = 0$ instead of 1, since

$$I(0,1,1) = I(1,0,1) = 1.$$

Here, subtracting 2^{25} from Q_{63} will propagate a carry to bit 26 or higher. Suppose it propagates to bit s, which can be any bit between and including bit 26 and 31. Then, the output of I does not change unless $1 \in Q_{61}[s - 26]$, as every time $Q_{61}[i] = 0$, $Q_{63}[i]$ will not have any effect. Hence, if $Q_{63}[25] = 0$, then we need $Q_{61}[i] = 0$, for i increasing from 26 and $Q_{63}[i]$ remains 0. It should still be 0 for the first i where $Q_{63}[i] = 1$. This will make the possibility of the requirements being met, improve from ½ to 2/3.

If the carry is propagated to bit 31 and stops, then $Q_{63}[31] = 1$, and as we need $Q_{63}[31] = Q_{61}[31]$, we have $Q_{61}[31] = 1$. This case is a success since we need a difference in the 31st bit of I's output, and this is accomplished because

$$I(1,A,1) = \neg I(1, \neg A,0)$$

for any value of A.

Care should be taken to see that the carry is not propagated past 31, because

$$I(0,A,0) = I(0, \neg A,1)$$

for any value of A.

The complexity of finding the second block is reduced by a factor of 3/8 with these improvements [26].

5.7 Implementation of Wang's Attack

An optimal implementation of Wang's MD5 attack would constitute two parts: first one being the finding of first block, optimally implemented by using Klima's algorithm [33], and the second part is finding the second block using the approach of Wang et al.. The two blocks can definitely be found by different techniques since they are independent of each other, but for the chaining values used in the second part which are generated during the first iteration. Any first block that satisfies the conditions will also satisfy the conditions on those chaining values and hence, this can be used to find the second block [26].

Our attack has been implemented in Java. The program was run on a desktop computer with AMD64 3000+ (1.83 GHz) on Windows XP as well as a virtual machine with Fedora Core 4 on the same computer. We could find 25 collisions in little less than 10 hours which averages out to 24 minutes for one collision. The first part took an average of 17 minutes and the second one needed 7 minutes. The fastest find took 11 minutes and the slowest was of 101 minutes. The variations in the timing can be attributed to the arbitrariness of the random number generator used in the program.

6 A Practical Attack on MD5 by Constructing Meaningful Collisions

Collision resistant hash functions are one of the very important elements of modern day cryptography. Hash function produces a fixed size output for variable length inputs. Collision resistance implies that it is not feasible to generate two messages which give the same hash value. Although, many collisions do exist, it must not be possible to really find a collision [15].

Nevertheless, there are cases where security vulnerability may be created by using an apparently useless collision. Magnus Daum and Stefan Lucks [19] described a method called *Poisoned Message Attack,* to exploit the programming language constructs of some standard documents (postscripts in this example) to create meaningful collisions that, when viewed using a standard viewer of that file format, look very genuine.

6.1 Poisoned Message Attack

The core idea of the p*oisoned message attack* is to exploit the "if-then-else" constructs available in most of the advanced document languages. For instance, in postscript, the command

$(A_1)\ (A_2)$ eq $\{B_1\}\{B_2\}$ ifelse

executes B_1 string if the two strings A_1 and A_2 are equal, else executes the string B_2.

Aware of the weakness of all the hash functions that involves iteration, that with a collision obtained h(X) = h(X'), all extensions XS and X'S by a random common string S also collide, a meaningful collision can be created in postscript documents, the procedure [14] for which is explained below –

Let W be the first 64 bytes of the file, which means that W contains all bytes up to and including the opening "(" in "(A_1) (A_2) eq {". The hash value $MD5_{0..63}$(IV, W) is the result of the initial block of the file compressed. Using Wang's attack on MD5, a collision is found where this hash value is used as the IV. The two colliding messages will be named M and M'.

Let C be the file got by replacing "(A_1) (A_2) eq {" with "(M) (M) eq {" and C' be the file got by replacing "(A_1) (A_2) eq {" with "(M') (M) eq {". Since the two strings before the "eq" are same (M in both) in C, the postscript interpreter displays only the contents of the first file. But, the two strings before the "eq" are not the same (M' and M) in C', and this causes the postscript interpreter to display only the contents of the second file. It should be noted that the hashes of the

two files C and C' are identical because M and M' have same MD5 hash value as a result of having W as the initial block in both the files.

Of course, the forgery can be easily perceived if one inspects the source code of the documents. But one generally trusts the viewer that displays a particular document type and thus makes the probability of such an attack higher [10] [14].

6.2 Other document file formats

Gebhardt, Illies and Schindler [29] investigated Merkle-Damgard hash functions and various other file formats to construct meaningful hash collisions. They first showed how it was constructed in postscript format as explained above and went on to examine PDF, TIFF and MS Word 97 (.doc) file formats. They also touched upon executables and packages. They summarized the strategy to construct meaningful collisions as follows: Find a suitable string *a,* such that for a significant segment of pair *b and b*', which are the outputs of a collision search, a string *c* exists such that

$$M = a||b||c, \qquad M' = a||b'||c$$

Represent meaningful messages for the file format specified. The pair (b, b') is a *universal collision* if it is possible to pick a string c such that M and M' have any predetermined meaning [29].

The threat this attack brings upon is that if anyone creates two documents with same hash values and if he gets one of them to be digitally signed by a person, then he can obtain a valid signature for the other document and this could pose serious threat to the signer in the future. Likewise, an insincere signer can create two such documents, sign one of them and claim that he signed the other [29].

Unlike Postscript, there are no programming language features like control constructs and procedures in **PDF** because of which, finding a collision in more difficult. [29] explains that the color strings in PDF are used to create poisoned messages.

TIFF (Tagged Image File Format) is a standard image file format used for scanning paper documents and the pages of a TIFF document are described by *Image File Directories*(IFD). [29] explains the method of constructing poisoned messages in TIFF by using the offsets to the IFDs.

7 Implementation of a Practical Attack

From the above attack on MD5 proposed by Wang et al., it is inferred that MD5 hash is no more a secure way of confirming a file's integrity, in view of the fact that colliding files/messages can be generated that give the same MD5 hash.

Here, we will use the colliding messages generated by our implementation of Wang's MD5 attack, to create two meaningful files with matching hash. This is also implemented in Java and the program creates two packages that contain any files of attacker's preference. The two packages created will give the same MD5 hash, although when extracted gives different files, with the same name suggested by the attacker. The inputs for the program are two different files specified by the attacker which are supposed to be later extracted from the packages the program creates. The attacker will also suggest a name for the output file. Figure 4 illustrates this. The program then creates two packages which are different but still gives the same MD5 sum. There is another program which is used to extract these packages and when this is run on the two packages separately, each will extract a file with the name specified for the output file and the contents will be that of the two input files, respectively.

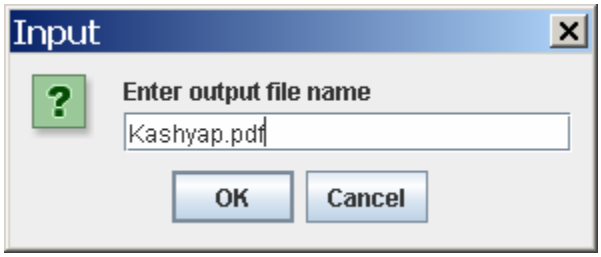

Figure 4: Packager program asking for the name of the final output file

When the two packages are given to two different users with the extraction program, both believe that they have received the same file since the MD5 sum matches, but when extracted, using the extraction program, each gets a different file. This idea of collision attack can be used in many practical purposes [30].

The idea behind this attack is simple. Each of the packages contains one of the colliding blocks at the beginning. The colliding blocks are obtained from the output of the Wang's attack which gives two 1024-bit messages. The rest of the data in both the packages are similar. They actually contain the contents of both the input files and also the information about the length of each file, the name of the output file and the length of the name. That explains the matching of MD5 hash

for both the packages. The two packages created are then renamed to a standard name that is readable by the extractor program and is stored in separate folders. When the extraction program is then executed on either of the packages, it reads the package file and looks for a specific bit in the colliding block as a pointer to select which file to extract. We know from Wang's theory that

$$\Delta M_0 = M'_0 - M_0 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, 2^{15}, 0, 0, 2^{31}, 0)$$

$$\Delta M_1 = M'_1 - M_1 = (0, 0, 0, 0, 2^{31}, 0, 0, 0, 0, 0, 0, -2^{15}, 0, 0, 2^{31}, 0)$$

This means that the first difference between the two colliding strings will be available in the 32^{nd} bit of the fifth word, i.e., the MSB of the 20^{th} byte. Therefore, the program uses two variables *colliding_byte = 19* and *collision_bitmask = 0x80,* which indicate the position and the mask of the differing bit. Using these two values, the extractor decides which file to extract and seeks the right position in the package to get the contents of that file and the right amount of bytes are extracted onto the output file.

Here is the blueprint of the package file:

BYTES	128	1	lenOut	10	10	size1	size2
Info	Collision block	Length of the output file name – *lenOut*	Name of the output file to be extracted	Size of file 1 in integer – *size1*	Size of file 2 in integer – *size2*	Contents of file 1	Contents of file 2

Table 7.1: Layout of the package

7.1 A practical scenario of the attack

Let us first create two files we want to be extracted. One of them will have the genuine contents and the other will have the contents of attacker's choice. Let us for example assume the contents of file 1 to be "I owe Mr. Moonlight $100 and will pay this amount on or before 12/25/2006" and the contents in file 2 to be "I owe Mr. Moonlight $1000 and will pay this amount on or before 12/25/2006". When these two files are submitted as input for the packager program, it creates two packages, which when extracted will give out a file with the name specified by the attacker with the contents of file 1 and file 2 respectively. If the first package is sent out to the person

who owes the money and get him to digitally sign the document, the attacker can then claim that he signed the second document using the fact that the two packages give the same hash value.

Another practical example will be this: A company distributes its product through package that consists of the installation files of that product which will be later extracted and installed on the client site. If there is a corrupt packager, he can create packages with defective files using the above mentioned attack, so that it gives the same MD5 sum when passed through the quality control department, and the responsibility falls upon the testing department as it was they who certified the product as genuine. This could also play heavily on the client as the company may charge the client extra for the maintenance and troubleshooting of the defect [30].

8 Conclusion and Future Work

We have shown that MD5 is no more secure and foolproof, thanks to Wang et al. They provided a set of conditions, which when satisfied, will create two different message blocks that give the same MD5 sum. We then showed how these colliding strings can be used to create meaningful collisions by creating two colliding packages that extracts to different files. Therefore, we can conclude that –

- MD5 should not be used anymore, especially in places where collisions are a serious threat to the security. It should be gradually phased out
- Further research on hash functions should be encouraged to understand the existing weaknesses and come up with better hash designs
- Alternatives and/or fixes to the Merkle-Damgard construction technique should be developed [26]

It will be very interesting and challenging to follow the progress of hash functions and their defense in the next few years.

Improvements may be achieved with respect to the speed in calculation of the colliding blocks by probably using a better random number generator. One could also attempt at combining or reducing the number of conditions required to be satisfied in Wang's attack using the example given in section 5.6.

With respect to the packager or meaningful attacks, one could try to get a colliding pair of files like PDF, WORD or even executables (EXE) that perform different actions of attacker's choice [30], although having same MD5 hash. The packages that are created in our program are actually native to the program. One could explore the possibility of creating a pair of colliding .zip or .tar files that can be distributed and extracted globally without the aid of the attacker's program.

9 References

[1] B.Schneier. *Applied Cryptography*. John Wiley and Sons, second edition, 1996

[2] William Stallings, *Cryptography and Network Security-Principles and Practice*, Third edition, Prentice Hall publications 2004.

[3] Wikipedia http://www.wikipedia.org

[4] WordIQ http://www.wordiq.com/

[5] Ross J. Anderson, *Security Engineering*

[6] Wolfram Math World http://mathworld.wolfram.com/

[7] Kumar, Satish and Zabeer, *Design and Analysis of Algorithms – ECC*, ASU, 2004.

[8] J.F.Dhem et al., *A Practical Implementation of the Timing Attack*

[9] Mah et al., *Timing Attack on ECC*

[10] M. Stamp and R. M. Low, *Applied Cryptanalysis: Breaking Ciphers in the Real World*, Wiley 2007.

[11] Paul C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman,RSA, DSS and other systems*

[12] Michael Rosing, *Implementing Elliptic Curve Cryptography*, Manning Publications Co., 1998

[13] Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart, HP Labs, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999

[14] Magnus Daum, *Cryptanalysis of Hash Functions of the MD4 Family*, 2005

[15] The Poisoned Message Attack Website, http://www.cits.rub.de.MD5Collisions/

[16] Trappe and Washington, University of Maryland, *Introduction to Cryptography with Coding Theory*, Prentice Hall, 2002.

[17] Sun Microsystems http://research.sun.com/projects/crypto/

[18] Certicom Corp. http://www.certicom.com/

[19] S. Lucks and M. Daum, *The Story of Alice and her Boss*. Presented at the rump session of Eurocrypt '05, May 2005

[20] X. Wang and H. Yu, *How to Break MD5 and Other Hash Functions.*

[21] http://world.std.com/~franl/crypto.html

[22] X. Wang, D. Feng, X. Lai, H. Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, rump session, CRYPTO 2004, Cryptology ePrint Archive, Report 2004/199, http://eprint.iacr.org/2004/199

[23] Rivest, R.: "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt

[24] P. Hawkes, M. Paddon, and G. G. Rose. *Musings on the Wang et al. MD5 Collision*. Cryptology ePrint Archive, Report 2004/264, 2004.

[25] Robert Milson, "Introduction to Public Key Cryptography and Modular Arithmetic"

[26] Soren Steffen Thomsen, *Cryptographic Hash Functions*, 2005.

[27] V. Miller, *Uses of elliptic curves in cryptography*, Advances in Cryptology - CRYPTO'85, LNCS 218, 1986.

[28] Soren Steffen Thomsen, MD5 Collison Finder, http://www2.mat.dtu.dk/people/S.Thomsen/wangmd5/

[29] M. Gebhardt, G. Illies., and W. Schindler, *A Note on the Practical Value of Single Hash Collisions for Special File Formats,* 2005.

[30] Ondrej Mikel, *Practical Attacks on Digital Signatures Using MD5 Message Digest,* 2004.

[31] Marc Stevens, *Fast Collision Attack on MD5,* 2006

[32] Vlastimil Klima, *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, 2006

[33] Vlastimil Klima*, Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications,* 2005

[34] Ondrej Mikel, Practical Attacks on Digital Signatures Using MD5 Message Digest, http://cryptography.hyperlink.cz/2004/collisions.htm

Appendix A

Wang's Output Differentials

j	Output	W_j	ΔW_j	ΔOutput	∇Output
4	Q_4	X_4	2^{31}	-2^6	..... .-++++++ ++++++++ ++.....
5	Q_5	X_5	0	$-2^6+2^{23}+2^{31}$	+..... +..... ..... .-.....
6	Q_6	X_6	0	$-1-2^6+2^{23}-2^{27}$	++++++-- -..... ....-+++ ++-+++++
7	Q_7	X_7	0	$1-2^{15}-2^{17}-2^{23}$ ..... -..-+++- +..... .....+
8	Q_8	X_8	0	$1-2^6+2^{31}$ -..... ....- ++....+-
9	Q_9	X_9	0	$2^{12}+2^{31}$ +..... ....+-.... .....
10	Q_{10}	X_{10}	0	$2^{30}+2^{31}$ ++..... ....
11	Q_{11}	X_{11}	2^{15}	$-2^7-2^{13}+2^{31}$ +..... ....-+++ +++....- +.....
12	Q_{12}	X_{12}	0	$2^{24}+2^{31}$ +.....+- ....
13	Q_{13}	X_{13}	0	2^{31} +..... ....
14	Q_{14}	X_{14}	2^{31}	$2^3-2^{15}+2^{31}$ +..... ..... -..... ....+...
15	Q_{15}	X_{15}	0	$-2^{29}+2^{31}$ +.-..... ....
16	Q_{16}	X_1	0	2^{31} +..... ....
17	Q_{17}	X_6	0	2^{31} +..... ....
18	Q_{18}	X_{11}	2^{15}	$2^{17}+2^{31}$ +..... ......+. ....
19	Q_{19}	X_0	0	2^{31} +..... ....
20	Q_{20}	X_5	0	2^{31} +..... ....
21	Q_{21}	X_{10}	0	2^{31} +..... ....
22	Q_{22}	X_{15}	0	0 ....
23	Q_{23}	X_4	2^{31}	0 ....
24	Q_{24}	X_9	0	0 ....
25	Q_{25}	X_{14}	2^{31}	0 ....
⋮	⋮	⋮	⋮	⋮ ⋮
34	Q_{34}	X_{11}	2^{15}	2^{31} ±..... ....
35	Q_{35}	X_{14}	2^{31}	2^{31} ±..... ....
36	Q_{36}	X_1	0	2^{31} ±..... ....
37	Q_{37}	X_4	2^{31}	2^{31} ±..... ....
38	Q_{38}	X_7	0	2^{31} ±..... ....
⋮	⋮	⋮	⋮	⋮ ⋮
44	Q_{44}	X_9	0	2^{31} ±..... ....
45	Q_{45}	X_{12}	0	2^{31} +..... ....
46	Q_{46}	X_{15}	0	2^{31} +..... ....
47	Q_{47}	X_2	0	2^{31} +..... ....
48	Q_{48}	X_0	0	2^{31} +..... ....
49	Q_{49}	X_7	0	2^{31} -..... ....
50	Q_{50}	X_{14}	2^{31}	2^{31} +..... ....
51	Q_{51}	X_5	0	2^{31} -..... ....
⋮	⋮	⋮	⋮	⋮ ⋮
57	Q_{57}	X_{15}	0	2^{31} -..... ....
58	Q_{58}	X_6	0	2^{31} +..... ....
59	Q_{59}	X_{13}	0	2^{31} +..... ....
60	$Q_{60}+A$	X_4	2^{31}	2^{31} +..... ....
61	$Q_{61}+D$	X_{11}	2^{15}	$2^{25}+2^{31}$ +.....+. ....
62	$Q_{62}+C$	X_2	0	$2^{25}+2^{31}$ +....+-. ....
63	$Q_{63}+B$	X_9	0	$2^{25}+2^{31}$ -.....+. ....

Table A-1: Wang's ΔM_0 Differential. [10]

j	Output	W_j	ΔW_j	ΔOutput	∇Output
0	Q_0	X_0	0	$2^{25}+2^{31}$	-.....+. ....
1	Q_1	X_1	0	$2^5+2^{25}+2^{31}$	-.+...+. ....
2	Q_2	X_2	0	$2^5+2^{11}+2^{16}+2^{25}+2^{31}$	-+-----. ..+----- ...+-... +--.....
3	Q_3	X_3	0	$-2^1+2^5+2^{25}+2^{31}$	-....+-. ....+-+++.
4	Q_4	X_4	2^{31}	$1+2^6+2^8+2^9+2^{31}$	+..... ....+---+ +-.....+
5	Q_5	X_5	0	$-2^{16}-2^{20}+2^{31}$	+..... ..-+..-+ ....
6	Q_6	X_6	0	$-2^6-2^{27}+2^{31}$	-..-+... ....-+ ++.....
7	Q_7	X_7	0	$2^{15}-2^{17}-2^{23}+2^{31}$	-....-++ +.....-+ -..... .....
8	Q_8	X_8	0	$1+2^6+2^{31}$	-..... ....+- --....+-
9	Q_9	X_9	0	$2^{12}+2^{31}$	-..... ....+.... .....
10	Q_{10}	X_{10}	0	2^{31}	-..... ....
11	Q_{11}	X_{11}	-2^{15}	$-2^7-2^{13}+2^{31}$	-..... ....-+++ +++..... .....
12	Q_{12}	X_{12}	0	$2^{24}+2^{31}$	++------ ....
13	Q_{13}	X_{13}	0	2^{31}	+..... ....
14	Q_{14}	X_{14}	2^{31}	$2^3+2^{15}+2^{31}$	+..... ..... +..... ....+...
15	Q_{15}	X_{15}	0	$-2^{29}+2^{31}$	+.-..... ....
16	Q_{16}	X_1	0	2^{31}	+..... ....
17	Q_{17}	X_6	0	2^{31}	+..... ....
18	Q_{18}	X_{11}	-2^{15}	$2^{17}+2^{31}$	+..... .....+. ....
19	Q_{19}	X_0	0	2^{31}	+..... ....
20	Q_{20}	X_5	0	2^{31}	+..... ....
21	Q_{21}	X_{10}	0	2^{31}	+..... ....
22	Q_{22}	X_{15}	0	2^{31}	+..... ....
23	Q_{23}	X_4	2^{31}	2^{31}	+..... ....
24	Q_{24}	X_9	0	0	....
25	Q_{25}	X_{14}	2^{31}	0	....
⋮	⋮	⋮	⋮	⋮	⋮
34	Q_{34}	X_{11}	-2^{15}	2^{31}	±..... ....
35	Q_{35}	X_{14}	2^{31}	2^{31}	±..... ....
36	Q_{36}	X_1	0	2^{31}	±..... ....
37	Q_{37}	X_4	2^{31}	2^{31}	±..... ....
38	Q_{38}	X_7	0	2^{31}	±..... ....
⋮	⋮	⋮	⋮	⋮	⋮
48	Q_{48}	X_0	0	2^{31}	+..... ....
49	Q_{49}	X_7	0	2^{31}	-..... ....
50	Q_{50}	X_{14}	2^{31}	2^{31}	+..... ....
51	Q_{51}	X_5	0	2^{31}	-..... ....
⋮	⋮	⋮	⋮	⋮	⋮
58	Q_{58}	X_6	0	2^{31}	+..... ....
59	Q_{59}	X_{13}	0	2^{31}	+..... ....
60	$Q_{60}+AA$	X_4	2^{31}	0	....
61	$Q_{61}+DD$	X_{11}	-2^{15}	0	....
62	$Q_{62}+CC$	X_2	0	0	....
63	$Q_{63}+BB$	X_9	0	0	....

Table A-2: Wang's ΔM_1 Differential. [10]

Appendix B

Add-differences provided by Hawkes et al.

t	δQ_t	δf_t	δQ_{t-3}	δW_t	δT_t	$S(t)$	δR_t
0-3	-	-	-	-	-	.	-
4	-	-	-	$\bar{31}$	$\bar{31}$	7	$\bar{6}$
5	$\bar{6}$	$\pm 19, \pm 11$	-	-	$\pm 19, \pm 11$	12	$\pm 31, \pm 23$
6	$\pm 31, \pm 23, \bar{6}$	$\bar{14}, \bar{10}$	-	-	$\bar{15}, \pm 14, \bar{10}$	17	$\pm 31, \bar{27}, \bar{0}$
7	$\bar{27}, \pm 23, \bar{6}, \bar{0}$	$\bar{27}, \bar{25}, \pm 16, \pm 10, \pm 5, \bar{2}$	-	-	$\bar{27}, \bar{25}, \pm 16, \pm 10, \pm 5, \bar{2}$	22	$\pm 27, \bar{24}, \bar{17}, \bar{15}, \pm 6, \pm 1$
8	$\bar{23}, \bar{17}, \bar{15}, \pm 0$	$\pm 31, \bar{24}, \pm 16, \pm 10, \pm 8, \pm 6$	$\bar{6}$	-	$\bar{31}, \bar{24}, \pm 16, \pm 10, \pm 8$	7	$\bar{31}, \bar{23}, \pm 17, \pm 15, \bar{6}$
9	$\pm 31, \bar{6}, \pm 0$	$\pm 31, \pm 26, \bar{23}, \bar{20}, \pm 6, \pm 0$	$\pm 31, \pm 23, \bar{6}$	-	$\pm 26, \bar{20}, \pm 0$	12	$\pm 12, \pm 6, \bar{0}$
10	$\pm 31, \pm 12$	$\bar{23}, \pm 13, \pm 6, \pm 0$	$\bar{27}, \pm 23, \bar{6}, \bar{0}$	-	$\bar{27}, \pm 13$	17	$\pm 30, \bar{12}$
11	$\pm 31, \pm 30$	$\bar{8}, \bar{0}$	$\bar{23}, \bar{17}, \bar{15}, \pm 0$	± 15	$\bar{23}, \bar{17}, \bar{8}$	22	$\bar{30}, \bar{13}, \bar{7}$
12	$\pm 31, \bar{13}, \bar{7}$	$\pm 31, \pm 17, \pm 7$	$\bar{31}, \bar{6}, \pm 0$	-	$\pm 17, \pm 6, \pm 0$	7	$\pm 24, \pm 13, \pm 7$
13	$\pm 31, \pm 24$	$\pm 31, \bar{13}$	$\pm 31, \pm 12$	-	$\bar{12}$	12	$\bar{24}$
14	± 31	$\pm 31, \pm 18$	$\pm 31, \pm 30$	$\bar{31}$	$\bar{30}, \pm 18$	17	$\bar{15}, \pm 3$
15	$\pm 31, \bar{15}, \pm 3$	$\pm 31, \pm 25$	$\pm 31, \bar{13}, \bar{7}$	-	$\pm 25, \bar{13}, \bar{7}$	22	$\bar{29}, \pm 15, \bar{3}$
16	$\pm 31, \bar{29}$	± 31	$\pm 31, \pm 24$	-	± 24	5	± 29
17	± 31	± 31	± 31	-	-	9	-
18	± 31	± 31	$\pm 31, \bar{15}, \pm 3$	± 15	± 3	14	± 17
19	$\pm 31, \pm 17$	± 31	$\pm 31, \bar{29}$	-	$\bar{29}$	20	$\bar{17}$
20-21	± 31	± 31	± 31	-	-	.	-
22	± 31	± 31	$\pm 31, \pm 17$	-	± 17	14	± 31
23	-	-	± 31	$\bar{31}$	-	20	-
24	-	± 31	± 31	-	-	5	-
25	-	-	± 31	$\bar{31}$	-	9	-
26-33	-	-	-	-	-	.	-
34	-	-	-	± 15	± 15	16	± 31
35	± 31	$\bar{31}$	-	$\bar{31}$	-	23	-
36	± 31	-	-	-	-	4	-
37	± 31	± 31		$\bar{31}$	-	11	-
38-49	± 31	± 31	± 31	-	-	.	-
50	$\bar{31}$		$\bar{31}$	$\bar{31}$	-	15	-
51-59	$\bar{31}$	± 31	± 31	-	-	.	-
60	± 31		$\bar{31}$	$\bar{31}$	-	6	-
61	$\bar{31}$	$\bar{31}$	$\bar{31}$	± 15	± 15	10	± 25
62-63	$\pm 31, \pm 25$	± 31	± 31	-	-	.	-

Table B-1: First Block of the differential. [24]