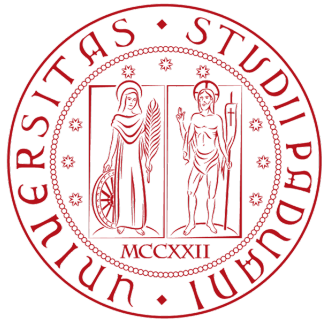
Cryptanalysis of Hash Functions

Sebastiano Degan



Kongens Lyngby 2012

DTU Matematik
Danmarks Tekniske Universitet
Matematiktorvet, Building 303B
DK-2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
Fax +45 4588 1399
MAT-INSTADM@mat.dtu.dk
http://www.mat.dtu.dk/

Summary

Despite recent breakthrough discoveries of practical collision attacks against MD5 [WY05, SLdW07], the latter is still very widely used. The main reason is likely as follows: When it comes to practical applications, retro-compatibility is a desirable property; furthermore, some protocols only require preimage resistance. Hence, research on preimage attacks and the security margin of hash functions against them is well motivated, especially if those hash functions are in practical use.

The aim of this thesis is to evaluate the applicability of the recently developed *biclique* [KRS11] to the preimage attack performed by Sasaki and Aoki in [SA09]. This led to a slightly improved time complexity of $2^{121.3}$ compression function evaluations and a greatly improved memory complexity of $2^{20.7}$ 32-bit memory words. Thanks to this reasonable memory requirement, an attack faster than brute force can be actually implemented, though its execution time would still be infeasible.

Preface

This thesis was prepared at the Department of Mathematics at the Technical University of Denmark under the supervision of Professor Christian Rechberger, in fulfilment of the requirements for acquiring a M.Sc. in Mathematical Modelling and Computation at the Technical University of Denmark and a Laurea Magistrale in Ingegneria Informatica at the University of Padua in the frame of TIME double-degree program.

The thesis deals with the cryptanalysis of hash functions, with a focus on `MD5`. It consists of an overview of cryptanalytic techniques concerning Meet-in-the-Middle attacks, which have recently received increasing attention. Some of those techniques will be further extended and applied in new contexts, thus improving known preimage attacks on `MD5`.

Lyngby, 31-August-2012

Sebastiano Degan

Contents

Chapter 1

Introduction

The word *cryptanalysis* refers to a research activity on a particular cryptographic component aiming to discover any weakness capable of compromising the security of the latter. It consists of both an evaluation of the applicability of known *cryptanalytic techniques* and the development of new ones when possible and/or necessary. Hash functions will be the target of the techniques presented in this thesis, with a focus on `MD5`.

1.1 Hash Functions and Their Applications

Here hash functions are defined together with some examples of their applications, motivating the need for the security requirements listed below.

Definition 1.1 (Cryptographic Hash Function)
A *cryptographic hash function* H is a function mapping arbitrary binary strings into binary strings of fixed length:

$$\begin{array}{rccc} H: & \{0,1\}^* & \longrightarrow & \{0,1\}^n \\ & x & \longmapsto & H(x) \end{array}$$

$H(x)$ is commonly referred to as *message hash*, *message digest*, *hash* or *digest*. Furthermore, for H to be considered secure, the following properties must hold:

Security Property 1 (Collision Resistance)
It is difficult to find distinct $x, x' \in \{0,1\}^*$ *such that:*

$$H(x) = H(x')$$

Security Property 2 (Second Preimage Resistance)
Given $x \in \{0,1\}^*$*, it is difficult to find* $x' \in \{0,1\}^*$, $x' \neq x$ *such that:*

$$H(x) = H(x')$$

Security Property 3 (Preimage Resistance)
Given $y \in \{0,1\}^n$*, it is difficult to find* $x \in \{0,1\}^*$ *such that:*

$$H(x) = y$$

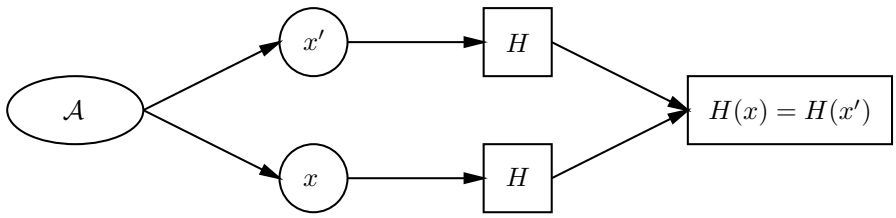


Figure 1.1: Collision Attack

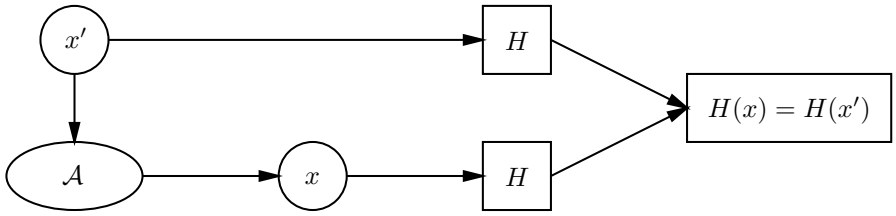


Figure 1.2: 2nd Preimage Attack

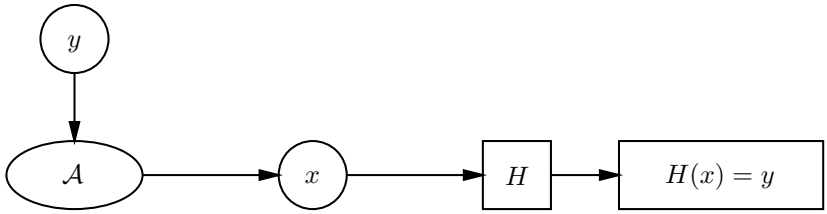


Figure 1.3: Preimage Attack

1.1.0.1 Safe Password Storage

One of the most common ways users validate their credentials is by providing a secret password. As a consequence, this sensitive information must be stored so that:

- It cannot be accessed by unauthorized entities.
- It is available for verification.

Exploiting access control lists and/or other similar features provided by the operating system is certainly a good idea; however, it is possible to add another layer of security by using hash functions. Instead of saving the password itself, the following value is stored:

$$\{salt, H(salt||password)\}$$

Figure 1.4: Safe Password Storage - *salt* is a random value for each entry.

When a user enters his password, the value $H(salt||password)$ is computed and compared with the stored one; if they match, the password is correct. Using this scheme, even if an attacker gains access to the information in Figure 1.4, the secrecy of the password will not be compromised since he will not be able to compute a valid password thanks to the preimage resistance of H. Furthermore, the use of a salt prevents common passwords to be detected.

1.1.0.2 Message Authentication

There is more to cryptography than encryption and confidentiality issues: Being able to determine by whom a particular message was generated is equally important. This goal can be achieved through a process called *message authentication*. Message authentication protocols can use either public key cryptography or symmetric key cryptography, in both cases hash functions are likely to be a fundamental building block.

In Figure 1.5 message authentication is achieved using symmetric key cryptography: A and B share the secret $K = (K_1, K_2)$ and use it to generate a *Message Authentication Code (MAC)*.

$$\text{Example of MAC codes} \begin{cases} MAC_K(m) = H(K_1||m||K_2) \\ MAC_K(m) = H(K_1||H(m||K_2)) \end{cases}$$

$$A \xrightarrow{(m, MAC_K(m))} B$$

Figure 1.5: Message Authentication - Symmetric Key

When B receives the message, he can check that it was indeed created by A by computing $MAC_K(m)$ and comparing it with the one provided. Since the key is known only to A and B, if the two MACs are equal then A must have created the message.

In Figure 1.6 message authentication is achieved using public key cryptography and digital signatures. The encryption algorithm is applied to the digest of the message rather than the message itself, this approach has two advantages:

- Encryption and decryption in public key cryptography are usually computationally expensive procedures, hash functions allow them to be applied on a fixed-size input, which is easier to handle.

- Hash functions disrupt any mathematical property linked to the particular primitive used. In RSA, for example, the encryption of the product is the product of the encryptions.

$$A \xrightarrow{(m, dec_A(H(m)))} B$$

Figure 1.6: Message Authentication - Public Key

Once B receives the message, he can check its authenticity by verifying that $H(m) = enc_A(dec_A(H(m))$; since only A knows her own private key, she must be the one who signed the message.

In this context, all of the security properties are crucial: A preimage or second preimage attack would allow to construct a message m' such that $H(m') = H(m)$, hence it would appear as A had signed it. The importance of collision resistance arises when A is asked to sign documents created by third parties: An attacker could create two different messages with the same hash, then he would ask A to sign one of them and reuse the same signature for the other.

1.1.0.3 Randomness in Cryptography

Random numbers are extensively used in cryptography and the security of many systems depends on the availability of good sources of randomness. One example is key generation: If the probability distribution is not uniform, an attacker could exploit this fact in order to speed-up the key recovery procedure by testing only the most likely keys.

Hardware random number generators are devices exploiting particular physical phenomena, such as thermal noise, which according to our understanding of physics may be considered as random processes. These devices are not always practical due to their price or limited throughput; for these reasons, *pseudo-random number generators* are used instead. These latter are deterministic procedures capable of generating random-looking bit-streams, which can be distinguished from real random bits only with a huge computational effort. One way to construct such procedures is by using hash functions as showed in Figure 1.7.

$$\begin{cases} H(seed+0) = r_0 \\ H(seed+1) = r_1 \\ H(seed+2) = r_2 \\ \quad \cdots \\ H(seed+c) = r_c \\ \quad \cdots \end{cases}$$

Figure 1.7: PRNG - The concatenation $r_0||r_1||r_2||\ldots||r_c||\ldots$ gives a random-looking stream of bits. Each sequence is defined by a *seed*, which must be carefully chosen and possibly random.

By combining both HRNG and PRGN, it is possible to generate real random seeds with the former and a long pseudo-random bit-stream with the latter, thus solving the throughput problem while keeping a reasonable security margin.

1.2 Attacks on Hash Functions

According to the definitions of the security requirements, it is not clear what should be considered an attack and what should not. This sort of gray line is due to the use of the sentence "*it must be difficult to*" without specifying how to measure the difficulty of a procedure.

The reason why the sentence "*it must not be possible to*" is not used is because it is *always* possible to compute preimages and collisions for any hash function. Let us consider a hash function H with n-bit-long digest:

Collisions Thanks to the birthday paradox, it is sufficient to examine a set of $2^{\frac{n}{2}}$ random messages in order to find a collision among them with high probability.

Preimages Random messages can be tested until the correct digest is found, after 2^n trials the correct digest is obtained with high probability.

Since these procedures do not depend on particular weaknesses of the hash function but rather on its very definition, their performances are considered the standard against which other attacks are compared. Hence, a preimage attack can be defined as any procedure requiring less than 2^n hash evaluations (or an equivalent workload) to be performed. Similarly, a collision attack can be defined as any procedure requiring less than $2^{\frac{n}{2}}$ hash evaluations to be performed.

1.3 Brief History of MD5

MD5 was first introduced in 1991 by Ronald L. Rivest and intended to be a replacement to MD4, which allowed for fast implementation both in hardware and in software but whose design had shown some weaknesses [dBB91]. Despite being slightly more complex, MD5 is still very similar to MD4.

Collisions In [dBB93] Boer and Bosselaers provided an algorithm capable of fining a pseudo-collision [1] within minutes. The security implications were not clear and in a subsequent report [Rob94] Robshaw commented as follows:

> "As it stands, the pseudo-collision arises from initializing the four-word buffer at the start of MD5 to two different values. These values differ only in the MSB of each of the four words. The same message is used for both sets of buffer values and the same message digest is obtained. A far more serious flaw would be if it were possible to choose one initial starting value for the buffer, not necessarily the one given in the algorithm, and then choose two different messages, perhaps differing in only a few bits of one word, so that the same message digest is obtained."

Three years later Dobberin achieved exactly this [Dob96].

In 2004 Wang et al. discovered actual collisions [WFLY04], their method could be extended to other hash functions such as SHA-0 and RIPEMD and these are the main characteristics of the attack on MD5:

- Uses differential cryptanalysis.
- Colliding messages are 128 bytes long and random-looking.
- Works for any initialization vector.
- At the time of discovery, collisions could be computed within one or two hours.

[1] A pseudo-collision is a collision obtained for two different initialization vectors, see Chapter 2

By exploiting this attack, it is possible to create two colliding messages M_A and M_B such that:

$$M_A = M_1||M_{cA}||M_2$$

$$M_B = M_1||M_{cB}||M_2$$

Where M_1 and M_2 are arbitrary while M_{cA} and M_{cB} are generated by the attack algorithm. Such a degree of freedom and the computational feasibility of the attack allowed for practical applications: It was possible to generate colliding text documents [DL05], executables [Dia05] and also public key certificates [LWdW05].

In the following years the efficiency of the attack was further improved, by 2006 it was possible to obtain collisions on a regular notebook pc in less than a minute [Kli06]. In 2007 a new type of collision called *chosen-prefix collision* was discovered [SLdW07], this was a way more powerful attack capable of making two arbitrary messages collide by appending a 718-byte-long random-looking message, which means that it is possible to create two colliding messages M_A and M_B such that:

$$M_A = M_1||M_{cA}||M_3$$

$$M_B = M_2||M_{cB}||M_3$$

Where M_1, M_2 and M_3 are arbitrary while M_{cA} and M_{cB} are generated by the attack algorithm. In order to prove the hazards of such an attack, in 2008 a group of researchers succeeded in creating a rouge CA certificate [SSA+09], clearly showing that `MD5` should not be used anymore in public key certificates.

Preimages During recent years, Sasaki and Aoki provided several attacks on step-reduced versions of `MD5`, and in 2008 they came up with an optimized procedure for a brute force approach [AS08]. Finally, in 2009 they presented an algorithm that can be considered a theoretical attack on full `MD5` [SA09]; one of the major contributions of their work is the so-called *initial structure*, which is a generalization of *local collisions*. Despite being an important breakthrough, the attack does not represent an immediate threat to preimage resistance for the following reasons:

- The speed-up is very limited.
- Memory complexity is too high to allow for an efficient implementation of the attack.
- The length of the preimage cannot be chosen.

Current Situation MD5 should be considered broken for any application requiring collision resistance, such as public key certificates. Its use is generally discouraged, however, due to the lack of relevant attacks and other practical reasons such as its speed as well as retro-compatibility with previous protocols, it is still widespread in contexts requiring only preimage resistance.

1.4 Contribution of this Thesis

Inspiration In [KRS11] a new cryptanalytic tool called *biclique* was introduced in the framework of Meet-in-the-Middle attacks, it has been used to attack both block ciphers and hash functions and it makes use of *differential cryptanalysis*. Since an initial structure is in some way analogous to a biclique, it is natural to ask whether the latter can be applied to MD5 and what kind of improvements, if any, it is possible to obtain on the attack discovered by Sasaki and Aoki.

Original Work In this thesis, MD5 is attacked with a MitM approach using bicliques instead of initial structures. Some issues arisen during this process suggested that it was possible and convenient to generalize the concept of bicliques, thus leading to the definition of *bigraphs*. Bicliques and bigraphs proved to be easily applicable in the context of Multi-Target Pseudo-Preimage attacks [Leu08, MR07]. Furthemore, bicliques have been applied together with local collisions in order to obtain one-block preimages.

Results Known attacks on MD5 are improved: Theoretically, the estimated time complexity of a pseudo-preimage attack is about 2^{115} compression function evaluations whereas a preimage requires $2^{121.3}$ compression function evaluations; the respective memory complexities are $2^{19.8}$ and $2^{20.7}$ 32-bit words. Thanks to this reasonable memory requirement, an attack faster than brute force can be actually implemented, though its execution time would still be infeasible. A new optimized brute force procedure for one-block preimages is discovered as well, with a time and memory complexity of $2^{126.4}$ and $2^{10.3}$, respectively.

1.4.1 Thesis Organization

In the first part of the thesis, Chapter 2 describes the design principles at the core of many hash functions as well as how those principles are implemented by

MD5; Chapter 3 introduces Meet-in-the-Middle attacks on block ciphers together with a series of enhancements while Chapter 4 presents the natural extension of those attacks to hash functions, together with further improvements on Meet-in-the-Middle attacks.

In the second part of the thesis, Chapter 5 presents a preimage attack on MD5 using bigraphs and MTPP techniques; Chapter 6 describes two variants of the attack, allowing for a comparison between bicliques and bigraphs; Chapter 7 presents an optimized brute force procedure for one-block preimages. Finally, Chapter 8 summarizes the results and compares them with the best attacks known for MD5.

Part I

An Overview of Cryptanalytic Techniques

Chapter 2

MD5 Description

2.1 Design Principles

Whenever a cryptographic primitive is designed, it must be possible to provide compelling arguments for its security. This can be done mainly in two ways:

- Establishing an equivalence between breaking the system and some difficult computational problem.
- Failure of cryptanalysis techniques to provide relevant attacks.

Ideally, the first way is the most elegant since there are many well studied computational problems for which no efficient solution is known. Hash functions allowing such equivalences to be proven are called *provably secure hash functions*. Throughout the years several hash functions of this type have been proposed, like [Dam87, Gir87, Gib91] in the late '80-early '90 or the more recent [LMPR08, CLS06, AFS96]. However, when practical aspects are considered, speed is a key factor and good performances often cannot be achieved together with provable security, or can be achieved only on particular architectures.

2.1.1 Merkle–Damgård Construction

One of the main issues in providing security proofs/arguments for hash functions is dealing with arbitrarily long input strings. With this problem in mind, in [Dam89] Damgård introduced a new way of constructing hash functions based on the following components:

Definition 2.1 (Compression Function)
A *compression function* C is a function:

$$\begin{array}{rccc} C\colon & \{0,1\}^n \times \{0,1\}^l & \longrightarrow & \{0,1\}^n \\ & (CV, M) & \longmapsto & C(CV, M) \end{array}$$

CV and M are referred to as *chaining value* and *message block*, respectively.

Definition 2.2 (MD-Compliant Padding)
A *MD-Compliant Padding* is a function:

$$\begin{array}{rccc} pad\colon & \{0,1\}^* & \longrightarrow & \left\{\{0,1\}^l\right\}^* \\ & M & \longmapsto & pad(M) \end{array}$$

such that $\forall\ M_1, M_2, M \in \{0,1\}^*$:

1. M is a prefix of $pad(M)$
2. $|M_1| = |M_2| \Rightarrow |pad(M_1)| = |pad(M_2)|$
3. $M_1 \neq M_2 \Rightarrow$ The last message block of $pad(M_1)$ is different from the last message block of $pad(M_2)$

Remark In the original work of Damgård the padding rule was less generic, this extension is due to Bellare in [GB01, § 8.5].

These two elements, together with a constant IV called *initialization vector*, define a hashing algorithm in the following way:

$$\begin{array}{c} pad(M) = M' = (M_1, M_2, M_3, \cdots, M_N), \quad |M_i| = l \\ \left\{ \begin{array}{l} H_0 = IV \\ H_i = C(H_{i-1}, M_i) \\ H(M) = H_N \end{array} \right. \end{array} \tag{2.1}$$

A graphic description is given in Figure 2.1: The padding of M returns a message

M' whose length is a multiple of l, the latter is then split into message blocks of the right size and fed to the compression functions. The output of the last compression step is possibly further processed.

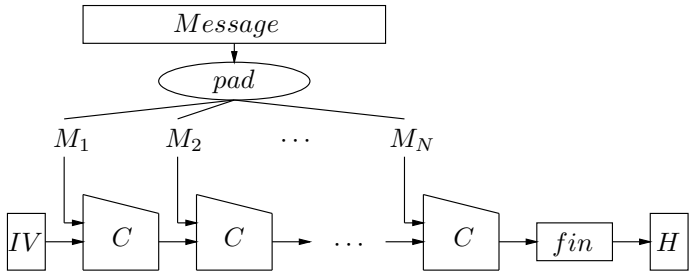


Figure 2.1: Merkle–Damgård Construction

The reason why this design is so interesting and used in practice, is that if security properties 1, 2 and 3 hold for the compression function, then they hold for the resulting hash function as well. This clearly simplifies the effort required for the design and cryptanalysis of hash functions, allowing cryptographers to focus the attention on a much simpler component operating on fixed-size strings.

2.1.2 Compression Function

In the following, some of the most common ways to construct/describe compression functions are presented. They all make use of this component:

Definition 2.3 (Block Cipher)
A *block cipher* or *cipher* E is a family of permutations indexed by a *set of keys* $\mathcal{K}$, mapping a *set of plaintexts* $\mathcal{P}$ onto a *set of ciphertexts* $\mathcal{C}$:

$$\begin{aligned} E\colon \quad & \mathcal{K} \times \mathcal{P} \longrightarrow \mathcal{C} \\ & (k, m) \longmapsto E_k(m) = c \end{aligned}$$

The inverse permutation $(E_k)^{-1}$ will be denoted as D_k.

It must be clear though, that the concept of block cipher is used as a purely descriptive mean and it does not imply that every compression function uses it.

Definition 2.4 (Davies–Meyer Mode)
A *Davies–Meyer compression function* is a function

$$C(CV, M) = E_M(CV) \boxplus M$$

where E is a *cipher* and $\boxplus$ is a group operation.

Definition 2.5 (Matyas–Meyer–Oseas Mode)
A *Matyas–Meyer–Oseas compression function* is a function

$$C(CV, M) = E_{g(CV)}(M) \boxplus M$$

where E is a *cipher*, $\boxplus$ is a group operation and $g()$ is a function formatting CV so that it can be used as a key for E.

Definition 2.6 (Miyaguchi–Preneel Mode)
A *Miyaguchi–Preneel compression function* is a function

$$C(CV, M) = E_{g(CV)}(M) \boxplus M \boxplus CV$$

where E is a *cipher*, $\boxplus$ is a group operation and $g()$ is a function formatting CV so that it can be used as a key for E.

Remark In all three cases, $\boxplus$ is referred to as *feed forward*.

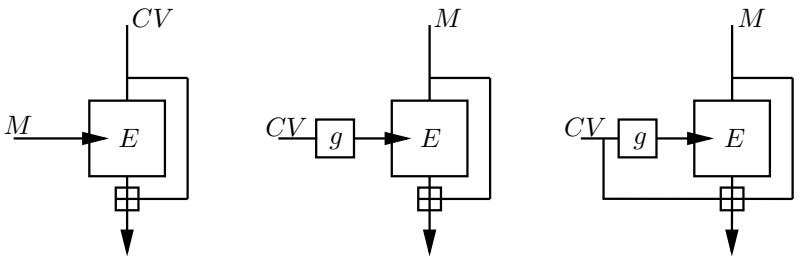


Figure 2.2: Davies–Meyer, Matyas–Meyer–Oseas and Miyaguchi–Preneel compression functions.

2.2 MD5 Hashing Algorithm

Cryptographic hash function `MD5` adopts a Merkle–Damgård design, so it is entirely specified by its compression function, a padding rule and an initialization vector.

2.2.1 Compression Function

The compression function can be viewed as a Davies–Meyer compression function operating on 512-bit-long message blocks and 128-bit-long chaining values.

$$\begin{array}{rccc} C\colon & \{0,1\}^{128} \times \{0,1\}^{512} & \longrightarrow & \{0,1\}^{128} \\ & (CV, M) & \longmapsto & E_M(CV) \boxplus CV \end{array}$$

Analogously to the Merkle–Damgård construction, cipher E operates by iterating a simple step several times, updating an internal state after each iteration. The chaining value is divided into four 32-bit registers and forms the initial internal state, the final internal state constitutes the cipher output.

$$\{0,1\}^{4\times 32} \ni (Q_{-3}, Q_0, Q_{-1}, Q_{-2}) = CV \in \{0,1\}^{128}$$

The whole computation is divided into 4 rounds, which are further subdivided into 16 steps each. Each step uses a portion of the message block and operates by mixing modular addition, rotations and Boolean functions, see Figure 2.3.

$$\{0,1\}^{16\times 32} \ni (m_0, m_1, m_2, \ldots, m_{15}) = M \in \{0,1\}^{512}$$

$$Q_{i+1} = \left[Q_{i-3} + k_i + m_{\pi(i)} + \Phi_i(Q_i, Q_{i-1}, Q_{i-2})\right]_{<<<s_i} + Q_i, \quad 0 \le i < 63$$

The rule π governing how the message block is used at each step is called *message expansion*. In `MD5` the message expansion is composed of four permutations of the message words m_i; each permutation is assigned to a round. Finally, the group operation $\boxplus$ is a register-wise modular addition.

$$\begin{array}{rccc} \boxplus\colon & \{0,1\}^{4\times 32} \times \{0,1\}^{4\times 32} & \longrightarrow & \{0,1\}^{4\times 32} \\ & ((a,b,c,d),(a',b',c',d')) & \longmapsto & (a+a', b+b', c+c', d+d') \mod 2^{32} \end{array}$$

Step constants k_i, s_i, Boolean functions Φ_i and the message expansion π are defined in Appendix C.

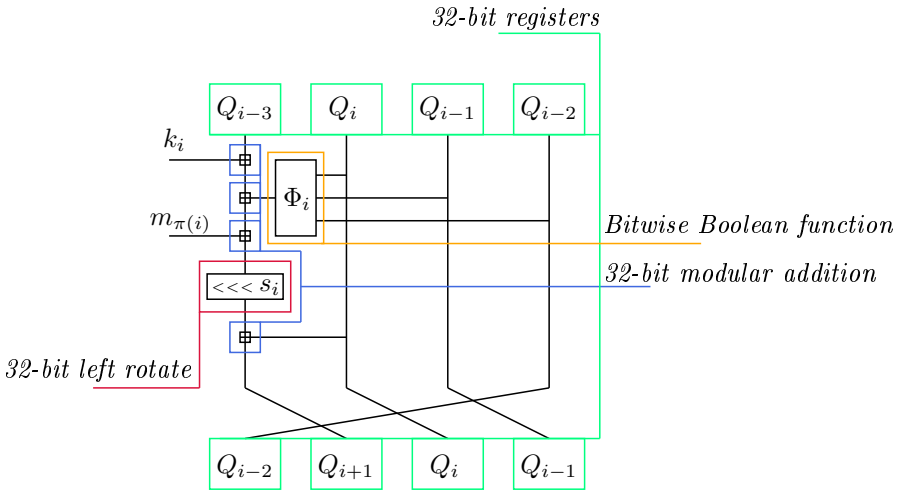


Figure 2.3: MD5 i-th Compression Step

2.2.2 Message Padding

Let M be the message to be padded and L its length expressed in bits. The padding rule, also called *MD strengthening* or *length padding* is the following:

Algorithm 2.1 MD Strengthening

Input:
A binary string M of length L
Output:
A padded binary string M'

Description:

1: $M' \leftarrow M$
2: $M' \leftarrow M'||1$
3: $t \leftarrow \min\{x \mid (L + 1 + x) = 448 \mod 521\}$
4: $M' \leftarrow M'||0^t$
5: Let l_{low} and l_{high} be 32-bit values such that the binary representation of L mod 2^{64} is $l_{high}||l_{low}$
6: $M' \leftarrow M'||l_{low}||l_{high}$
7: **return** M'

Chapter 3

Meet-in-the-Middle Framework

A MitM attack was used in [DH77] by Diffie and Hellman to show that double encryption under two different keys does not double the security level. In this chapter MitM is explained together with a series of techniques capable of extending its applicability beyond double encryption schemes and/or step-reduced ciphers.

3.1 Basic MitM Attack

Let us assume $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$ and $E_k = E^1_{k_1} \circ E^2_{k_2}$, in other words: E is the composition of two ciphers whose keys are independent of each other. Let us now consider a couple $(m, E_k(m))$, the computation of the ciphertext is split as follows:

$$m \xrightarrow[E^1]{k_1} c' \xrightarrow[E^2]{k_2} c$$

This implies that an attacker, given m and c, can perform the following computations:

$$\begin{cases} m \xrightarrow[E^1]{k_1} v_1 \ \forall \ k_1 \in \mathcal{K}_1 \\ c \xrightarrow[D^2]{k_2} v_2 \ \forall \ k_2 \in \mathcal{K}_2 \end{cases}$$

If a match between any of the v_1 and any of the v_2 is found, the corresponding key (k_1, k_2) encrypts m to c. The attack procedure is formally described in Algorithm 3.1 and graphically represented in Figure 3.1.

Algorithm 3.1 MitM Key Recovery Algorithm

Input:
A (plaintext,ciphertext) couple (m, c)
Output:
Keys k such that $E_k(m) = c$

Description:
1: **for all** $k_1 \in \mathcal{K}_1$ **do**
2: $v_1 \leftarrow E^1_{k_1}(m)$
3: Store v_1 in a table
4: **end for**
5: **for all** $k_2 \in \mathcal{K}_2$ **do**
6: $v_2 \leftarrow D^2_{k_2}(c)$
7: Check whether v_2 matches any of the v_1
8: **end for**
9: **return** all (k_1, k_2) for which there was a match

In the remainder of the thesis, E^1, D^2, v_1 and v_2 will be referred to as *forward chunk*, *backward chunk*, *forward candidate* and *backward candidate*, respectively.

Attack Complexity The attack requires $O(|\mathcal{K}_1| + |\mathcal{K}_2|)$ encryptions and $O(|\mathcal{K}_1|)$ memory.

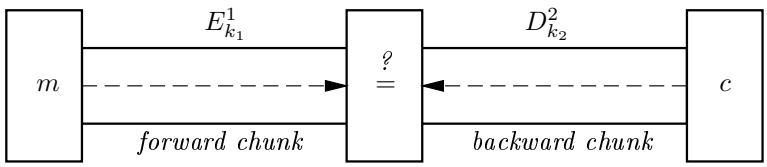


Figure 3.1: Key Recovery Through Meet-in-the-Middle

3.2 Beyond Double Encryption Schemes

Being able to partition the set of keys as indicated in Section 3.1 is rather unlikely, even with step-reduced ciphers. A far more common situation is a partitioning $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$ and $E_k = E^1_{k_1,k_3} \circ E^2_{k_2,k_3}$, where the two ciphers are independent of each other with respect to a portion of the key. The key recovery procedure in this case is outlined in Algorithm 3.2.

Algorithm 3.2 MitM Key Recovery Algorithm

Input:
A (plaintext,ciphertext) couple (m, c)
Output:
Keys k such that $E_k(m) = c$

Description:
1: **for all** $k_3 \in \mathcal{K}_3$ **do**
2: **for all** $k_1 \in \mathcal{K}_1$ **do**
3: $v_1 \leftarrow E^1_{k_1,k_3}(m)$
4: Store v_1 in a table
5: **end for**
6: **for all** $k_2 \in \mathcal{K}_2$ **do**
7: $v_2 \leftarrow D^2_{k_2,k_3}(c)$
8: Check whether v_2 matches any of the v_1
9: **end for**
10: **end for**
11: **return** all (k_1, k_2, k_3) for which there was a match

Attack Complexity The attack requires $O(|\mathcal{K}_3|(|\mathcal{K}_1| + |\mathcal{K}_2|))$ encryptions and $O(|\mathcal{K}_1|)$ memory.

3.2.1 Splice-and-Cut

Splice-and-Cut is a more general MitM type of attack introduced by Merkle and Hellman in [MH81] in order to argue that triple encryption under two different keys does not double the security level. The term was used for the first time by Sasaki and Aoki in [AS08], where this technique has been extended to hash functions. In [WRG+11] it has been applied for the first time on a cipher not making use of triple encryption.

Splice-and-Cut works by considering the computations performed by the cipher as a loop, this comes at a cost: An *oracle* is needed to perform the attack. Let us assume that it is possible write E_k as $E^1_{k_1,k_3} \circ E^2_{k_2,k_3} \circ E^3_{k_1,k_3}$ and to obtain encryptions of arbitrary plaintexts, the following attack is then possible:

Algorithm 3.3 MitM + Splice-and-Cut Key Recovery Algorithm

Input:
An oracle $\mathcal{O}$ encrypting messages with a secret key k
Output:
Set of keys containing k

Description:
1: **for all** $k_3 \in \mathcal{K}_3$ **do**
2: Randomly choose a starting state s
3: **for all** $k_2 \in \mathcal{K}_2$ **do**
4: $v_2 \leftarrow E^2_{k_2,k_3}(s)$
5: Store v_2 in a table
6: **end for**
7: **for all** $k_1 \in \mathcal{K}_1$ **do**
8: $m \leftarrow D^1_{k_1,k_3}(s)$
9: $c \leftarrow \mathcal{O}(m)$
10: $v_1 \leftarrow D^3_{k_1,k_3}(c)$
11: Check whether v_1 matches any of the v_2
12: **end for**
13: **end for**
14: **return** all (k_1, k_2, k_3) for which there was a match

Attack Complexity The attack requires $O(|\mathcal{K}_3|(|\mathcal{K}_1| + |\mathcal{K}_2|))$ encryptions, $O(|\mathcal{K}_2|)$ memory and $|\mathcal{K}_3||\mathcal{K}_1|$ calls to the oracle.

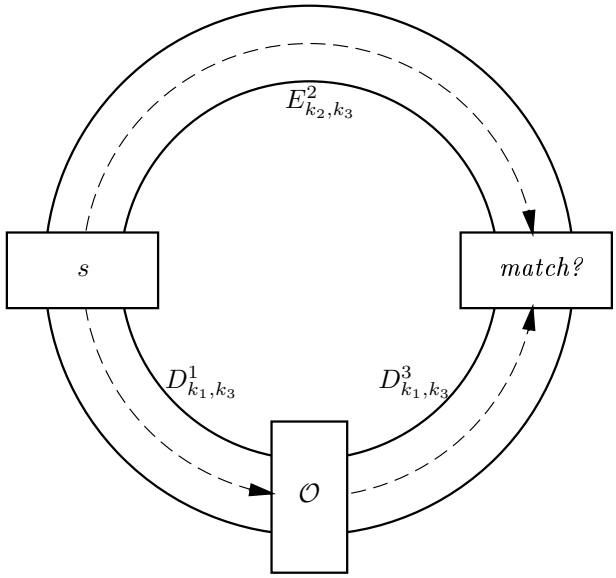


Figure 3.2: MitM + Splice-and-Cut

3.2.2 Bicliques

Bicliques are a recently developed cryptanalytic tool [KRS11] used to solve a problem that can be informally described as "chunk overlapping". Let us assume that for a key partitioning $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2 \times \mathcal{K}_3$ the correspondent cipher partitioning is $E_k = E^1_{k_1,k_3} \circ f_k \circ E^2_{k_2,k_3} \circ E^3_{k_1,k_3}$. The issue in this case is that since f depends on both k_1 and k_2, the computations are not divided into two independent parts.

If a MitM attack as described in Section 3.2.1 were to be performed, two starting states Q and P would be needed:

$$\begin{cases} Q \xrightarrow[D^1]{k_1,k_3} m \xrightarrow{\mathcal{O}} c \xrightarrow[D^3]{k_1,k_3} v_1 \quad \forall\, k_1 \in \mathcal{K}_1 \\ P \xrightarrow[E^2]{k_2,k_3} v_2 \quad \forall\, k_2 \in \mathcal{K}_2 \end{cases} \tag{3.1}$$

However, f cannot be neglected, and for any matching pair (v_1, v_2) it would be

necessary to verify that:

$$f_{k_1,k_2,k_3}(Q) = P \tag{3.2}$$

which unless f is independent of the key cannot be true for every couple (k_1, k_2). This implies that if there is any possibility of achieving (3.2) for every couple of keys, the starting states Q and P have to be chosen as a function of k_1 and k_2. However, since the computations in (3.1) have to be independent of each other, the only possibility is to pick Q as a function of k_1 and P as a function of k_2. This leads quite naturally to the following definition:

Definition 3.1 (Sub-Cipher)
Let E be a cipher and its set of keys $\mathcal{K}$ be divided as $\mathcal{K}' \times \mathcal{K}_f$, a cipher f is called a *sub-cipher of* E if there exist ciphers E^1 and E^2 such that:

$$E_k = E^1_k \circ f_{k_f} \circ E^2_k$$

Definition 3.2 (Biclique)
Let f be a sub-cipher of E and $\mathcal{N} = \{N[i,j]\}$ be a subset of $\mathcal{K}_f$. A *biclique of dimension* d over f for $\mathcal{N}$ is a pair of sets $\{Q_i\}$ and $\{P_j\}$ of 2^d states each such that

$$Q_i \xrightarrow[f]{N[i,j]} P_j, \quad \forall i,j$$

If such a construction were possible, computations in (3.1) could be modified in order to take the biclique into account:

$$\begin{cases} Q_{k_1} \xrightarrow[D^1]{k_1,k_3} m \xrightarrow{\mathcal{O}} c \xrightarrow[D^3]{k_1,k_3} v_1 \quad \forall\ k_1 \in \mathcal{K}_1 \\ P_{k_2} \xrightarrow[E^2]{k_2,k_3} v_2 \quad \forall\ k_2 \in \mathcal{K}_2 \end{cases} \tag{3.3}$$

Hence, for any matching pair (v_1, v_2) the following would be true by definition:

$$f_{k_1,k_2,k_3}(Q_{k_1}) = P_{k_2}$$

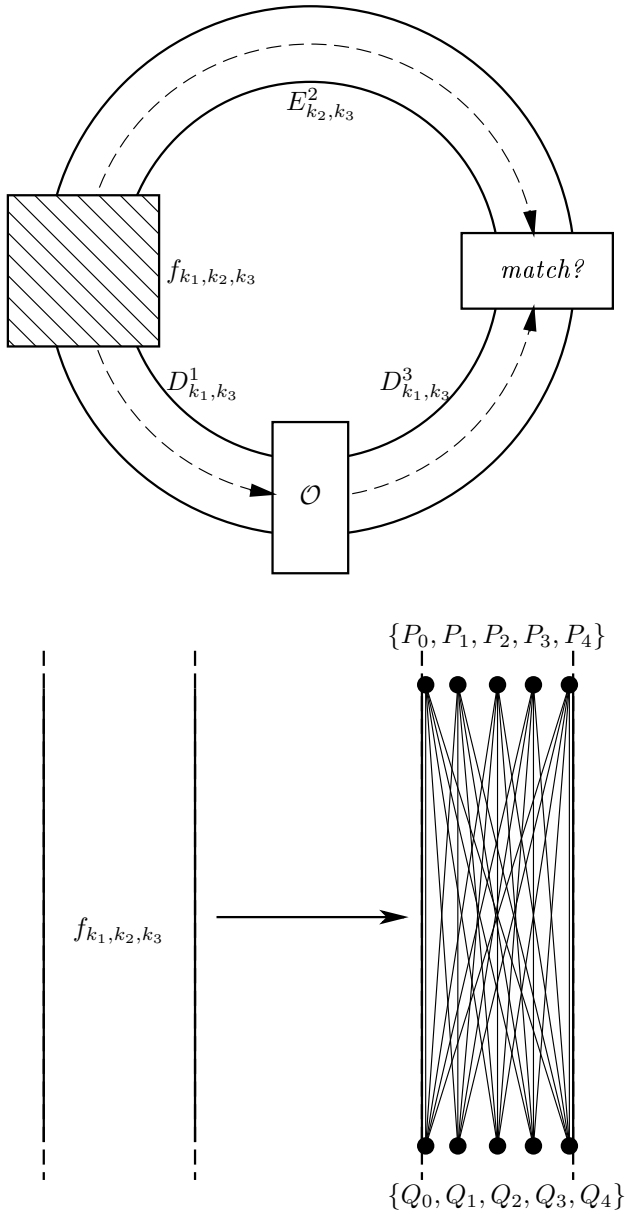


Figure 3.3: Biclique used in the Splice-and-Cut framework. When starting states and keys are properly initialized, the resulting structure assures that $f_{i,j,k_3}(Q_i) = P_j \ \forall \ i, j$.

3.2.2.1 Construction Algorithms

By adopting a differential view, bicliques can also be considered as a two sets of differential trails together with a couple of starting states:

$$\begin{cases} \{* \xrightarrow[f]{N[*,j]} \Delta_j \mid \forall j\} \\ \{\Delta_i \xleftarrow[f^{-1}]{N[i,*]} * \mid \forall i\} \\ \\ \qquad (Q_0, P_0) \end{cases} \tag{3.4}$$

With this information, it is possible to construct a biclique according to Definition 3.2:

$$\{P_j = P_0 + \Delta_j \mid \forall j\}$$
$$\{Q_i = Q_0 + \Delta_i \mid \forall i\}$$

Assuming that the trails *simultaneously* hold, the biclique property is satisfied:

$$Q_0 + \Delta_i \xrightarrow[f]{N[i,j]} P_0 + \Delta_j \quad \forall\ i, j$$

This implies that any construction algorithm can exploit insights from differential cryptanalysis in order to provide trails in (3.4).

3.2.2.2 Differential Trails and Bigraphs

Since differential cryptanalysis is well known, cryptographic primitives are specifically designed to withstand it. This is usually done by mixing several type of operations, thus introducing strong non-linearities and achieving the so-called *avalanche effect*. The latter can be informally defined as the tendency of a function to output significantly different values even if only a small portion of its input is modified. For these reasons, when the biclique dimension is high and f is a relatively large portion of the cipher, finding differential trails as indicated in (3.4) is not an easy task.

By allowing the biclique property not to be always satisfied, namely:

$$\exists\ \bar{i}, \bar{j}\ \textit{such that}\ f_{N[\bar{i},\bar{j}]}(Q_0 + \Delta_{\bar{i}}) \neq P_0 + \Delta_{\bar{j}}$$

Definition 3.2 can be made less restrictive.

Definition 3.3 (Bigraph) Let f be a sub-cipher of E, and $\mathcal{N} = \{N[i,j]\}$ be a subset of $\mathcal{K}_f$. A *biclique of dimension* $d_1 \times d_2$ *and efficiency* e over f for $\mathcal{N}$ is a pair of sets $\{Q_i\}$ and $\{P_j\}$ of 2^{d_1} and 2^{d_2} states respectively such that

$$\frac{\left|\left\{(i,j) : Q_i \xrightarrow[f]{N[i,j]} P_j\right\}\right|}{2^{d_1+d_2}} = e \tag{3.5}$$

Remark The biclique efficiency represents the probability of the biclique property being satisfied for a uniformly random couple (i,j):

$$e = Prob\left[f_{N[i,j]}(Q_i) = P_j\right]$$

Biclique Efficiency and Attack Complexity If the procedure outlined in (3.3) needs to be repeated T times before finding a match, then $\frac{T}{e}$ repetitions are needed in order to find a match and satisfy the biclique property. Hence, the time complexity increases by a factor $\frac{1}{e}$ whereas the memory complexity remains unchanged.

Chapter 4

Preimage Attacks on Hash Functions

This chapter shows how MitM techniques for block ciphers can be used to attack hash functions, assuming these latter adopt a Merkle–Damgård design and a Davies-Meyer mode for the compression function. The notation is modified accordingly: $\mathcal{M}$ and M will be used in place of $\mathcal{K}$ and k.

Furthermore, if not otherwise specified, H denotes the hash function being attacked and C its compression function.

4.1 From Ciphers to Compression Functions

A key recovery attack on a cipher can be easily transformed into a preimage attack on the compression function: Since $H = C(CV, M) = E_M(CV) \boxplus CV$, finding a preimage is equivalent to finding *any* key M such that $C = E_M(CV) = H \boxminus CV$.

No Need for an Oracle In this setting, there is a strong relation between the chaining value, the cipher output and the message digest, namely:

$$C \boxplus CV = H$$

This means that in order to use Splice-and-Cut, the need for an oracle mentioned in Section 3.2.1 translates to the need to freely choose the chaining value. This implies that the attack will obtain a *pseudo-preimage*, where the term "pseudo" indicates that a different initialization vector must be used in order to get the desired digest.

4.2 From Compression Functions to Hash Functions

4.2.1 Converting Pseudo-Preimages to Preimages

[MvOV96, Fact 9.99] describes a way to convert pseudo-preimage attacks into preimage attacks; the procedure is generic in the sense that it does not depend on the way a pseudo-preimage is obtained as long as the length specified in the padding is known in advance. The algorithm is based on a MitM approach and it works as follows:

Algorithm 4.1 Conversion of Pseudo-Preimages to Preimage

Input:
A hash function H with n-bit-long digest
A pseudo-preimage attack $\mathcal{A}$ of time complexity 2^s
A target hash h
Output:
A preimage for h

Description:
1: Run $\mathcal{A}(h)$ $2^{\frac{n-s}{2}}$ times and store its output
2: Let b be the extra message blocks necessary to satisfy the padding
3: Freely choose the first $b-1$ message blocks.
4: $CV \leftarrow H(M_1||M_2||\cdots||M_{b-1})$
5: **while** no match **do**
6: $M_b \leftarrow$ `random()`
7: Check if $C(CV, M_b)$ matches any of the stored chaining values
8: **end while**
9: Let (CV_j, M_j) be the matched pseudo-preimage
10: **return** $M_1||M_2||\cdots||M_b||M_j$

Attack Complexity $2^{\frac{n+s}{2}}$ iterations of the while loop are necessary to obtain a match with high probability, which means $2^{\frac{n+s}{2}}$ calls to the compression function. The cost of obtaining $2^{\frac{n-s}{2}}$ pseudo-preimages is $2^{\frac{n+s}{2}}$, the total is then $2^{\frac{n+s}{2}+1} + (b-1)$ compression function evaluations. Furthermore, $O(\frac{n-s}{2})$ extra memory is required to store the pseudo-preimages.

4.2.2 Expandable Messages

Some attacks do not allow to choose the length specified in the padding, so Algorithm 4.1 is unlikely to produce a message correctly padded. However, it is possible to produce arbitrarily long messages all yielding the same hash by using the following property of compression functions:

Definition 4.1 (Fixed-Point)
A *fixed-point* for a compression function C is a couple (CV, M) such that $C(CV, M) = CV$.

For a Davies-Meyer compression function it is particularly easy to compute a fixed-point, although there is no control over the chaining value: Let $C(CV, M) = E_M(CV) \boxplus M$, in order to obtain the correspondent chaining value forming a fixed-point it is sufficient to compute $CV = D_M(0)$[1].

Exploiting Fixed-Points Let us assume that for a message $M = M_0||M_1$, M_1 is a fixed-point with respect to its chaining value $CV = C(IV, M_0)$; this implies that any message $M' = M_0||(M_1)^i$ will have the same hash as M. Algorithm 4.2 shows how to create such messages.

[1] "0" is the zero with respect to "$\boxplus$".

Algorithm 4.2 Generation of Expandable Messages

Input:
A Davies-Meyer compression function C with n-bit long hash
Output:
An expandable message $(M_0||M_e)$

Description:
1: Compute $C(IV, M)$ for $2^{\frac{n}{2}}$ random messages and store the results
2: **while** (no match) **do**
3: Compute a fixed point (CV, M_e)
4: Check if CV matches any of the stored hashes
5: **end while**
6: **return** a matching couple $(M||M_e)$

Attack Complexity In order to obtain a matching with high probability the loop must iterate $2^{\frac{n}{2}}$ times, for a total complexity of $2^{\frac{n}{2}}+2^{\frac{n}{2}} = 2^{\frac{n}{2}+1}$ evaluations of the compression function. $O(2^{\frac{n}{2}})$ memory is needed as well.

Availability of Fixed-Points If fixed-points are not easily computable, [KS05] provides a general procedure to obtain expandable messages.

4.2.3 MTPP: Multi-Target Pseudo-Preimage

Algorithm 4.1 involves many repetitions of a pseudo-preimage attack, this implies that a growing number of pseudo-preimages is potentially available repetition after repetition. MTPP techniques aim at exploiting this fact in order to speed-up the preimage search, hence improving the total complexity of the attack.

4.2.3.1 General Overview

Let us consider the following algorithm:

Algorithm 4.3 MTPP Algorithm

Input:
A target hash H_0
A algorithm $\mathcal{A}$ returning a pseudo-preimage for one of the specified targets
An integer k
Output:
A list containing 2^k pseudo-preimages and the original target H_0

Description:
1: $\mathcal{H} \leftarrow \{H_0\}$
2: $\mathcal{T} \leftarrow \{H_0\}$
3: **while** $|\mathcal{H}| \leq 2^k$ **do**
4: $(CV, M) \leftarrow \mathcal{A}(\mathcal{H})$
5: $\mathcal{H} \leftarrow \mathcal{H} \cup \{CV\}$
6: $\mathcal{T} \leftarrow \mathcal{H} \cup \{(CV, M)\}$
7: **end while**
8: **return** $\mathcal{T}$

As the list $\mathcal{T}$ of pseudo-preimages gets built, it is possible to organize it in a tree structure with the following properties:

1. Every node is labelled by a target hash.
2. Every edge is labelled by a message.
3. If a node H_i is connected to its parent H_j by an edge M, then $C(H_i, M) = H_j$.
4. The concatenation of all the messages found in the path between any node H_i and the root node H_0 yields a pseudo-preimage with respect to H_0 when H_i is used as chaining value.

Such a structure is be called *layered hash tree* [Leu08], an example of it is shown in Figure 4.1: It provides 2^k pseudo-preimages of variable length for the specified target hash H_0. In order to deal with variable length preimages it is possible to use expandable messages, see Algorithm 4.4.

The whole attack procedure is now more complex, but if the attack algorithm $\mathcal{A}$ is capable of taking advantage of multiple targets, the construction of 2^k pseudo-preimages needed for the conversion will be less computationally expensive.

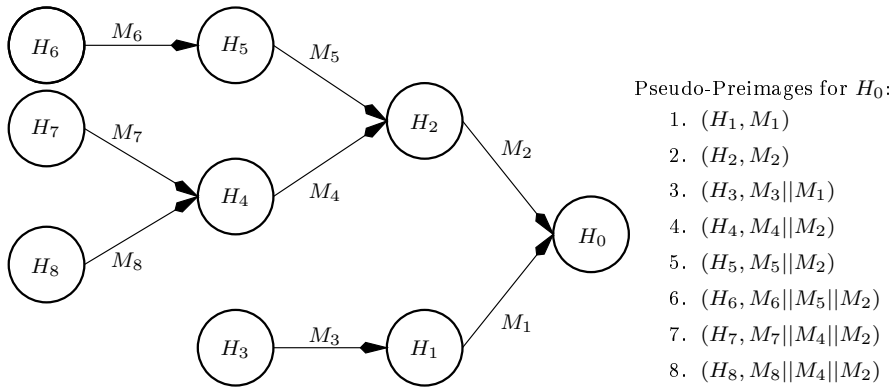


Figure 4.1: This tree provides 9 possible targets for a new attack and 8 pseudo-preimages of variable length.

Algorithm 4.4 Conversion of Pseudo-Preimages to Preimage using MTPP

Input:
A hash function H with n-bit long digest
A pseudo-preimage attack $\mathcal{A}$ exploiting MTPP
A target hash h
An integer k
An expandable message $(M_0||M_e)$

Output:
A preimage

Description:
1: Generate a layered hash tree for 2^k targets using algorithm $\mathcal{A}$
2: $CV \leftarrow H(M_0||M_e)$
3: **while** no match **do**
4: $M \leftarrow$ `random()`
5: Check if $C(CV, M)$ matches any of the stored targets
6: **end while**
7: Let $M_1||M_2||\cdots||M_b$ be the correspondent message for the matched target
8: Let i be the extra message blocks needed to satisfy the padding
9: **return** $M_0||(M_e)^i||M||M_1||M_2||\cdots||M_b$

Attack Complexity In order to obtain a matching with high probability the loop must iterate 2^{n-k} times, for a total complexity of

$$C_{\mathcal{A}} + 2^{n-k}$$

evaluations of the compression function, where $C_{\mathcal{A}}$ represents the complexity of obtaining 2^k targets using $\mathcal{A}$. Furthemore, $\mathcal{O}(2^k)$ extra memory is required to store the layered hash tree.

4.2.3.2 MTPP: Some Examples

Preimages on MD4 In [Leu08] the author presents an algorithm performing a partial pseudo-preimage attack on `MD4`, namely: given a target hash H, the algorithm finds a couple (CV, M) such that a part of $md4(CV, M)$ is equal to H. The remaining bits are random, so the algorithm is repeated until there is a full match. If many targets are available, assuming they differ only in the part on which the algorithm has no control, the probability of matching at least one of them is higher.

MitM In [GLRW10] MTPP is introduced in the framework of Meet-in-the-Middle attacks, in this case multiple targets have no direct impact on the matching probability but can be exploited to increase the number of candidates computed:

$$\begin{cases} Q_{M_1} \xrightarrow[D^1]{M_1, M_3} CV \xrightarrow[C=H \boxminus CV]{} C \xrightarrow[D^3]{M_1, M_3} v_1 \quad \forall\ (M_1, H) \in \mathcal{M}_1 \times \mathcal{H} \\ P_{M_2} \xrightarrow[E^2]{M_2, M_3} v_2 \quad \forall\ M_2 \in \mathcal{M}_2 \end{cases}$$

By redefining $\mathcal{M}_1$ and $\mathcal{M}_2$ and using the extra targets available, it is possible to compute more candidates for each chunk and lower the attack complexity, see Figure 4.2.

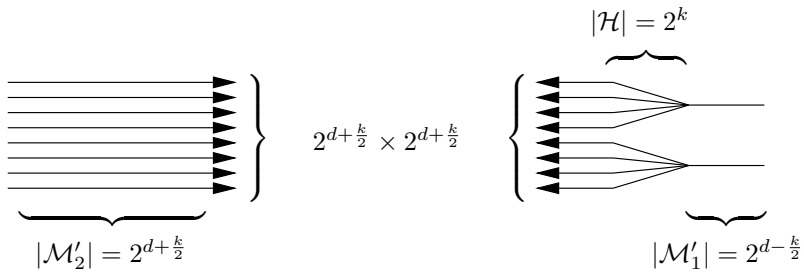


Figure 4.2: When 2^k targets are available, the number of possible couplings for the candidates becomes 2^{2d+k}. In this example $d = 2$ and $k = 2$.

MTPP and Bicliques In Section 3.2.2.2 it was mentioned that a high biclique dimension may have negative effects on the efficiency; hence, when 2^k targets are available, depending on which chunk includes the feed forward, it is possible to redefine either $\mathcal{M}_1$ or $\mathcal{M}_2$ so that the number of candidates stays constant and the biclique dimension decreases (see Figure 4.3). This will result in a higher biclique efficiency and lower time complexity of the attack.

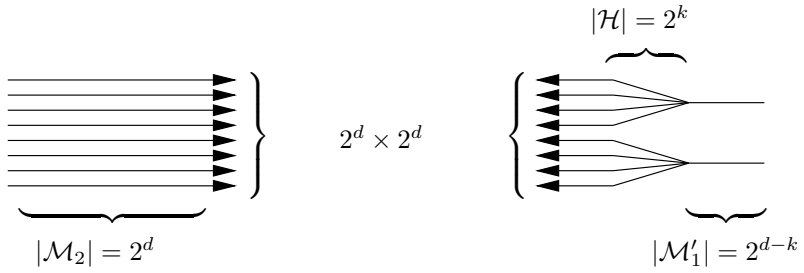


Figure 4.3: In this example $d = 3$ and $k = 2$, the biclique dimension goes from $d \times d$ to $d \times (d - k)$.

4.3 Matching Techniques

Let us assume that for a partitioning $\mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2$ the correspondent cipher partitioning is $E_M = E^1_{M_1} \circ f_{M_1,M_2} \circ E^2_{M_2}$. This setting is similar to the one

introduced in Section 3.2.2: In that situation, where E^1 was the backward chunk and E^2 was the forward chunk, the problem was how to choose the initial states.

Now the dual problem is considered, where forward chunk and backward chunk are swapped, hence the issue is how to perform the matching given that f cannot be computed due to its dependency on both M_1 and M_2:

$$CV \xrightarrow[E_1]{M_1} v_1' \xleftrightarrow[f]{M_2,M_1} v_2' \xleftarrow[E_2]{M_2} C$$

This problem can be addressed by using techniques called *partial matching* and *partial fixing*. The main idea behind them can be traced back to [CE85] but their names are due to Sasaki and Aoki, who successfully applied them to `MD5` in [AS08] and [SA09], where they also introduced *partial fixing for unknown carry behaviour*.

Partial Matching As described in Section 2.1.2, `MD5` works by iterating a simple set of operations which update an internal state. This is not just a peculiarity of `MD5`, other hash functions such as `MD4`, `SHA-0`, `SHA-1`, `SHA-2` and `RIPEMD` are similarly defined. The compression step often does not update the whole internal state, but rather only a portion of it, hence several iterations are needed before the internal state is completely refreshed. This means that at any point in the computation it is possible to know the value of part of the internal state several steps ahead, without the need to actually computes those steps. Hence it is possible to partially match v_1' and v_2' if f does not include too many steps.

Partial Fixing Even without the knowledge of part of the message, it might still be possible to compute parts of f to some extent, namely:

$$\begin{cases} v_1' \xrightarrow[f]{?,M_1} v_1 \\ v_2' \xrightarrow[f^{-1}]{M_2,?} v_2 \end{cases} \tag{4.1}$$

where v_1 and v_2 are two partially known states. The name "partial fixing" derives from the fact that even if backward and forward chunks are independent of bigger parts of the message, those parts are kept fix on purpose so that (4.1) can be computed.

Unknown Carry Behaviour When considering bitwise Boolean operations, it is rather easy to carry out the computations in Equations 4.1 since it is

always possible to compute a part of the output, provided all inputs are known for the correspondent bit positions. Slightly more complex is the case of modular addition, where carries generated by unknown parts of the input can propagate and possibly affect the whole result. Whenever the partial evaluation of f presents such ambiguities, it is possible to carry on the computations for both possibilities and check later on which one is correct, see Figure 4.4.

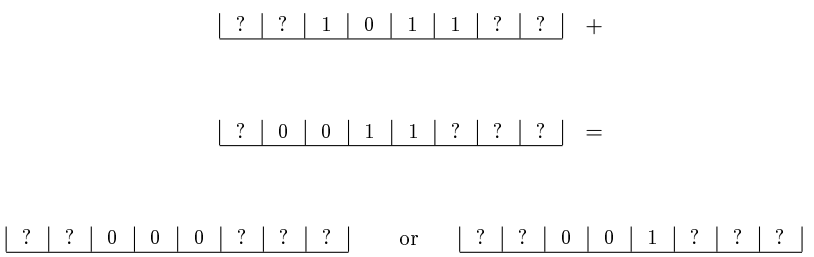


Figure 4.4: 8-bit Modular Addition - In this example two results are computed in order to account for a possible carry propagation.

Let c_1 and c_2 be the number of ambiguities in the computation of a candidate for the forward chunk and backward chunk, respectively. This implies that if $|\mathcal{M}_1| = 2^{d_1}$ and $|\mathcal{M}_2| = 2^{d_2}$, then there will be $2^{d_1+c_1}$ forward candidates and $2^{d_2+c_2}$ backward candidates.

Verifying the Match All these techniques allow for new configurations of the chunks, but require additional work in order to check whether a partial match is indeed a full match. Since this will be done by actually computing f for every couple of candidates that partially match, it is necessary that this does not happen for too many couples, otherwise the complexity will be extremely close to brute force. Let us assume for example that d_m bits of the candidates v_1 and v_2 can be matched, i.e. they represent the same portions of the same internal state: If the compression function behaves randomly enough, the *(partial) matching probability* is 2^{-d_m}, so if $|\mathcal{M}_1| = |\mathcal{M}_2| = 2^d$ the extra work required by the matching procedure is :

$$2^{2d-d_m} C_f$$

where C_f is the complexity of computing f.

4.4 One-Block Preimages

Preimages that fit into a single message block are called *one-block preimages* and are particularly interesting for password retrieval (see Section 1.1.0.1). However, as a consequence of this restriction, any pseudo-preimage attack is useless since its conversion to a preimage attack with the methods described in this chapter would yield messages of at least two blocks.

4.4.1 Fixed CV and Splice-and-Cut

As explained in Section 4.1, when using a Splice-and-Cut approach it is not possible to fix the chaining value. This is true in general, but there are techniques that can overcome this issue.

Definition 4.2 (Local Collision)
A *local collision* is a collision happening for the internal state of the compression function.

Remark The use of local collisions can be traced back to [CJ98]

When performing the attack outlined in Figure 3.3, the chaining value is obtained by the following computations:

$$CV = D^1_{M_1,M_3}(Q_{M_1}) \tag{4.2}$$

Where Q_{M_1} is the starting state given by the biclique for the correspondent message difference. Since both Q_{M_1} and D^1 depend on M_1, so does the chaining value. Hence, sufficient conditions for CV to be fixed are the following:

Condition 4.3 $D^1_{M_1,M_3}$ *is independent of* M_1.

Condition 4.4 Q_{M_1} *is independent of* M_1.

Condition 4.3 can be fulfilled by properly redefining the chunks, Condition 4.4 implies that the differential trails in the biclique have to have the following from:

$$\begin{cases} \{* \xrightarrow[f]{N[*,j]} \Delta_j \mid \forall j\} \\ \{0 \xleftarrow[f]{N[i,*]} * \mid \forall i\} \end{cases} \tag{4.3}$$

The second set of trails in (4.3) represents a local collision. If such a construction were possible, the following starting states would be obtained:

$$\{P_j = P_0 + \Delta_j \mid \forall j\}$$
$$\{Q_i = Q_0 \mid \forall i\}$$

Hence, Equation 4.2 would be independent of M_1. Now, assuming that it is possible to choose M_3 so that $D^1_{M_3}(Q_0)$ gives the right chaining value, the attack can be carried out as described in Section 3.2.1, and the result will be a one-block preimage.

Part II

Improved Attacks on MD5

Chapter 5

Pseudo-Preimage and Preimage Attacks on MD5

This chapter shows how the techniques discussed in the first part of this thesis can be implemented into a pseudo-preimage attack on MD5 and the correspondent conversion algorithm.

Section 5.1 provides a high-level description of the attack, Sections 5.2, 5.3 and 5.4 focus respectively on matching, biclique and padding. Sections 5.5 and 5.6 provide an attack algorithm and a conversion algorithm together with their complexity analysis.

5.1 Attack Outline

$\mathcal{M}_1$ and $\mathcal{M}_2$ are bit positions $[31-24, 8-0]$ and $[31-15]$ of message words m_6 and m_{14}, respectively. This determines position and length of the chunks, biclique and matching part:

Forward Chunk Steps $18, 19, 20, \ldots, 42$ (25 in total)

Backward Chunk Steps $13, 12, 11, \ldots, 0, 1, 63, 62, \ldots, 51$ (27 in total)

Biclique Steps $14, 15, 16, 17$ (4 in total)

Matching Steps $43, 44, 45, \ldots, 50$ (8 in total)

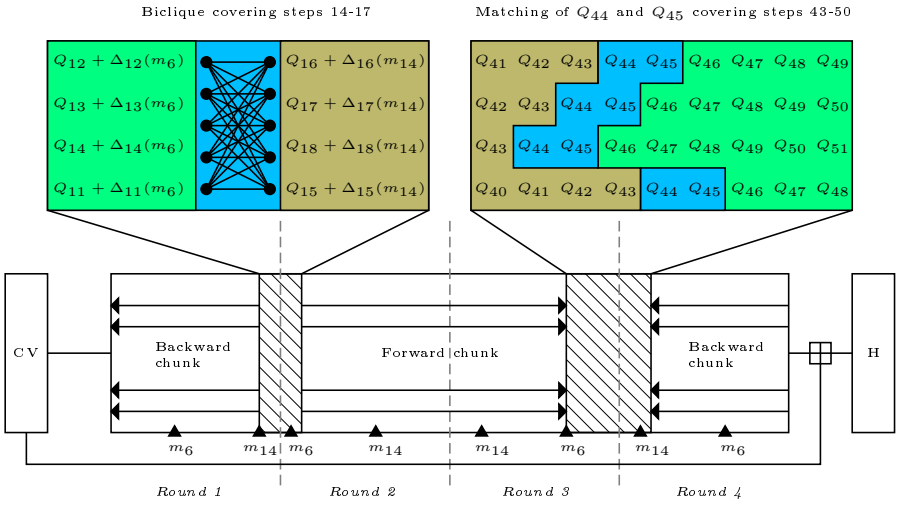


Figure 5.1: Attack Outline - Green background indicates computations affected by $\mathcal{M}_1$, brown background indicates computations affected by $\mathcal{M}_2$ and blue background indicates computations affected by both of them.

5.2 Matching

This section describes how forward and backward candidates can be computed and provides a procedure checking whether a partial match is indeed a full match.

5.2.1 Candidates Generation

In the case of MD5, the compression function updates the internal state 32 bits at a time, this implies that it is possible to skip up to 3 compression steps and perform the matching only on one 32-bit register instead of the full 128-bit internal state. Furthermore, since parts of m_{14} and m_6 are fixed, it is also possible to carry on the computations when one of those words is used in the compression step, as described in Section 4.3.

Sections 5.2.1.1 and 5.2.1.2 show step by step how candidates are computed; in order to do so, the following notation is introduced:

- $A[x-y]$ indicates the value of bit positions $[x-y]$ of A.
- $A^{[x-y]}$ indicates that only $A[x-y]$ is known.
- $A^{[x-y]\times z}$ indicates that only $A[x-y]$ is known, and because of uncertainties on the carries there are z possible values for it.

5.2.1.1 Backward Candidates

Backward candidates consist of two partially known states, namely: $Q_{44}[14-9]$ and $Q_{45}[14-0]$.

Step 50

$$Q_{47} = (Q_{51} - Q_{50})_{>>>15} - \Phi_{50}(Q_{50}, Q_{49}, Q_{48}) - m_{14}^{[14-0]} - k_{50} =$$
$$= [(Q_{51} - Q_{50})_{>>>15} - \Phi_{50}(Q_{50}, Q_{49}, Q_{48}) - m_{14} - k_{50}]^{[14-0]}$$

Step 49

$$Q_{46} = (Q_{50} - Q_{49})_{>>>10} - \Phi_{49}(Q_{49}, Q_{48}, Q_{47}^{[14-0]}) - m_7 - k_{49} =$$
$$= (Q_{50} - Q_{49})_{>>>10} - \Phi_{49}(Q_{49}, Q_{48}, Q_{47})^{[14-0]} - m_7 - k_{49} =$$
$$= [(Q_{50} - Q_{49})_{>>>10} - \Phi_{49}(Q_{49}, Q_{48}, Q_{47}) - m_7 - k_{49}]^{[14-0]}$$

Step 48

$$Q_{45} = (Q_{49} - Q_{48})_{>>>6} - \Phi_{48}(Q_{48}, Q_{47}^{[14-0]}, Q_{46}^{[14-0]}) - m_0 - k_{48} =$$
$$= (Q_{49} - Q_{48})_{>>>6} - \Phi_{48}(Q_{48}, Q_{47}, Q_{46})^{[14-0]} - m_0 - k_{48} =$$
$$= [(Q_{49} - Q_{48})_{>>>6} - \Phi_{48}(Q_{48}, Q_{47}, Q_{46}) - m_0 - k_{48}]^{[14-0]}$$

Step 47

$$Q_{44} = (Q_{48} - Q_{47}^{[14-0]})_{>>>23} - \Phi_{47}(Q_{47}^{[14-0]}, Q_{46}^{[14-0]}, Q_{45}^{[14-0]}) - m_2 - k_{47} =$$
$$= [(Q_{48} - Q_{47})_{>>>23}]^{[23-9]} - \Phi_{47}(Q_{47}, Q_{46}, Q_{45})^{[14-0]} - m_2 - k_{47} =$$
$$= [(Q_{48} - Q_{47})_{>>>23} - \Phi_{47}(Q_{47}, Q_{46}, Q_{45}) - m_2 - k_{47}]^{[14-9]\times 2^1}$$

Complexity Analysis Steps 50,49 and 48 have to be computed only once since there is no uncertainty about the carry, whereas step 47 has to be computed for the two possible carry patterns. The complexity of generating backward candidates is then:

$$C_{bc} = \frac{3}{64} + 2^1 \frac{1}{64} = \frac{5}{64} \tag{5.1}$$

compression function evaluations and 2^1 candidates are generated.

5.2.1.2 Forward Candidates

Forward candidates consist of two partially known states, namely: $Q_{44}[14-0]$ and $Q_{45}[14-4]$.

Step 43

$$Q_{44} = Q_{43} + \left[Q_{40} + \Phi_{43}(Q_{43}, Q_{42}, Q_{41}) + k_{43} + m_6^{[23-9]}\right]_{<<<23} =$$
$$= Q_{43} + \left\{[Q_{40} + \Phi_{43}(Q_{43}, Q_{42}, Q_{41}) + k_{43} + m_6]_{<<<23}\right\}^{[14-0]\times 2^1} =$$
$$= \left\{Q_{43} + [Q_{40} + \Phi_{43}(Q_{43}, Q_{42}, Q_{41}) + k_{43} + m_6]_{<<<23}\right\}^{[14-0]\times 2^1}$$

Step 44

$$Q_{45} = Q_{44}^{[14-0]\times 2^1} + \Big[Q_{41} + \Phi_{44}(Q_{44}^{[14-0]\times 2^1}, Q_{43}, Q_{42}) + k_{44} + m_9\Big]_{<<<4} =$$
$$= Q_{44}^{[14-0]\times 2^1} + \big\{[Q_{41} + \Phi_{44}(Q_{44}, Q_{43}, Q_{42}) + k_{44} + m_9]_{<<<4}\big\}^{[18-4]\times 2^1} =$$
$$= \big\{Q_{44} + [Q_{41} + \Phi_{44}(Q_{44}, Q_{43}, Q_{42}) + k_{44} + m_9]_{<<<4}\big\}^{[14-4]\times 2^2}$$

Complexity Analysis Step 43 has to be computed for two possible carry patterns and step 44 has to be computed for four possible patterns. The complexity is then

$$C_{fc} = 2^1 \frac{1}{64} + 2^2 \frac{1}{64} = \frac{6}{64} \tag{5.2}$$

compression function evaluations and 2^2 candidates are generated.

5.2.2 Matching Procedure

Once a couple of candidates partially match, i.e. forward and backward candidates for $Q_{44}[14-9]$ and $Q_{45}[14-4]$ have the same value, Algorithm 5.1 need to be run in order to check for a full match.

Algorithm 5.1 Matching Procedure

Input:
A couple of message words (m_{14}, m_6) for which there is a partial match
State $(Q_{40}, Q_{43}, Q_{42}, Q_{41})$ correspondent to m_{14}
State $(Q_{48}, Q_{51}, Q_{50}, Q_{49})$ correspondent to m_6
Candidate $(Q_{44}[14-0], Q_{45}[14-4])$ correspondent to m_{14}
Candidate $(Q_{44}[14-9], Q_{45}[14-0])$ correspondent to m_6
Output:
`true` in case of full match, `false` otherwise.

Description:

1: */* If any of the checks fail, the algorithm will stop and return* `false` **/*
2: Fully Compute Q_{44} (step 43)
3: Check carry hypothesis of forward candidate $Q_{44}[14-0]$
4: Fully Compute Q_{45} (step 44)
5: Check carry hypothesis of forward candidate $Q_{45}[14-4]$
6: Check for further matching of backward candidate $Q_{45}[3-0]$
7: Fully Compute remaining steps (45 to 50)
8: Check for full match on state $(Q_{48}, Q_{51}, Q_{50}, Q_{49})$
9: **return** `true`

Complexity Analysis Carry hypotheses are correct with probability 2^{-1} and a matching on $Q_{45}[3-0]$ will happen with probability 2^{-4}, the expected running time is then:

$$C_{match} = \left(\frac{1}{64}\right) 2^{-1} + \left(\frac{2}{64}\right) 2^{-1}(1 - 2^{-5}) + \left(\frac{8}{64}\right) 2^{-6} = 2^{-5}\frac{51}{64} \qquad (5.3)$$

compression function evaluations.

5.3 Biclique

This section provides a high-level description of the differential trails and the construction algorithm for the biclique. A complete proof of their soundness and an accurate computation of the biclique efficiency is provided in Appendix A.

5.3.1 Absorption Properties in Boolean Functions

In Boolean algebra the following relations are called *absorption properties*:

$$a \vee (a \wedge b) = a$$
$$a \wedge (a \vee b) = a$$

On some Boolean functions, these particular relations between inputs and outputs are often exploited in order to obtain differential trails that are simpler and/or hold with higher probability. Table 5.1 shows some examples of absorption properties for the Boolean functions used in MD5 and other hash functions as well.

	1st Absorption	2nd Absorption	3rd Absorption
IF	$\texttt{IF}(X, \mathbb{C}, \mathbb{C}) = \mathbb{C}$	$\texttt{IF}(\mathbb{0}, X, \mathbb{C}) = \mathbb{C}$	$\texttt{IF}(\mathbb{1}, \mathbb{C}, X) = \mathbb{C}$
IF3	$\texttt{IF3}(X, \mathbb{C}, \mathbb{0}) = \mathbb{C}$	$\texttt{IF3}(\mathbb{C}, X, \mathbb{1}) = \mathbb{C}$	$\texttt{IF3}(\mathbb{C}, \mathbb{C}, X) = \mathbb{C}$
ONX	$\texttt{ONX}(X, \mathbb{C}, \mathbb{0}) = \neg\mathbb{C}$	-	$\texttt{ONX}(\mathbb{1}, \mathbb{C}, X) = \neg\mathbb{C}$

Table 5.1: Absorption properties for IF,IF3 and ONX. X can be any n-bit value, $\mathbb{C}$ is a constant, $\mathbb{1} = 1^n$ and $\mathbb{0} = 0^n$

Since all these functions operate bitwise, it is possible to exploit several absorption properties at the same time, each acting on a different part of the input.

This technique was used by Sasaki and Aoki in [SA09] for the construction of the *initial structure*, they called it *cross absorption properties*.

5.3.2 Trails Interaction and Absorption Properties

Let us consider the Boolean function

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

and the problem of achieving $IF(X, Y, Z) = Z$ even if X and Y are not constant: This is possible if and only if $(X \wedge Y) = (X \wedge Z)$, which translated in equations on single bits gives:

$$[x_i \wedge y_i = x_i \wedge z_i \ \forall i] \Leftrightarrow [y_i = z_i \ \forall i \text{ such that } x_i = 1] \tag{5.4}$$

This means that if a particular X has k zeroes in its binary representation, there exist 2^k values of Y verifying (5.4). Hence, assuming that X and Y span form 0 to $2^n - 1$, the number of couples (X, Y) such that $IF(X, Y, Z) = Z$ is:

$$\sum_{k=0}^{n} \binom{n}{k} 2^k = 3^n$$

So, if X and Y are uniformly random values, the absorption probability is:

$$p_{abs} = \left(\frac{3}{4}\right)^n$$

$$\begin{array}{lll}
Z : z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0 & & \\
\qquad\qquad \wedge \text{———} & (\neg X \wedge Z) : 0\ z_6\ 0\ \ 0\ z_3 z_2\ 0\ z_0 & \\
\neg X : 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1 & & \\
 & & \wedge \text{———} \quad IF(X, Y, Z) : z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0 \\
Y : z_7\ y\ z_5 z_4\ y\ \ y\ z_1\ y & & \\
\qquad\qquad \wedge \text{———} & (X \wedge Y) : z_7\ 0\ z_5 z_4\ 0\ \ 0\ z_1\ 0 & \\
X : 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0 & &
\end{array}$$

Figure 5.2: Example of absorption when X, Y, X are 8 bits variables. $X =$ `0x4d` $\Rightarrow$ there are 2^4 possible values for Y that assure absorption.

5.3.3 Trails Description

Differential trails are defined with respect to modular addition, hence, rotations and boolean functions are a source of non-linearities. These latter will be handled by properly initializing states and messages so that:

- Rotations can be considered as multiplications or divisions.
- Boolean functions absorb differences.
- Backward and forward trails affect different portions of the states.

Figure 5.3 provides a visual description of the biclique.

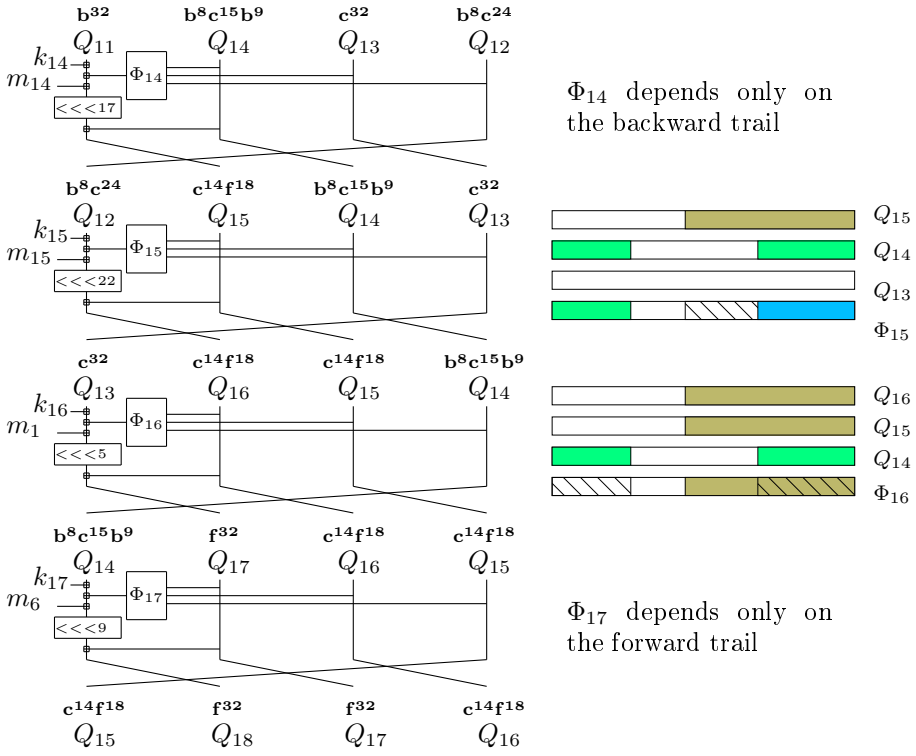


Figure 5.3: Biclique for Steps 14-17 - Above each state its configuration is indicated. Notations like $f^x c^y b^z$ mean that x bits are possibly affected by the forward trail, followed by y constant bits and z bits possibly affected by the backward trail. On the right side, Φ functions and their inputs: Colours white, green, brown, blue and oblique lines indicate respectively constant parts, influence of backward trail, influence of forward trail, influence of both and absorption of differences.

Trails Interaction and Biclique Efficiency Φ_{15} is supposed to absorb differences of both the forward trail and a part of the backward trail. Forward trail affects bits $[17-0]$ of Q_{15} and backward trail affects bits $[8-0]$ of Q_{14}; hence, for what concerns the nine least significant bits, this is the same situation

depicted in Section 5.3.2. As result, the biclique efficiency is[1]:

$$e = \left(\frac{3}{4}\right)^9 \tag{5.5}$$

5.3.4 Construction Algorithm

The procedure used to construct the biclique, presented in Algorithm 5.2, can be divided into three parts:

Message and State Initialization [Lines 2-6]
The starting state $(Q_{11}, Q_{14}, Q_{13}, Q_{12})$ is initialized so that the differential trails behave as expected.

Forward Trail [First Loop]
For every possible difference on m_{14}, the correspondent starting state $(Q_{15}, Q_{18}, Q_{17}, Q_{16})$ is found by computing steps 14 to 17, assuming that at step 15 absorption takes place.

Backward Trail [Second Loop]
For every possible difference on m_6, the correspondent starting state $(Q_{11}, Q_{14}, Q_{13}, Q_{12})$ is found by computing the inverse steps 17 to 14, assuming that at step 15 absorption takes place.

[1] The exact efficiency is slightly lower because of a non-linear interaction due to the rotation at step 14, see Appendix A

Algorithm 5.2 Biclique construction

Input:
None
Output:
Arrays of starting states $Q[]$ and $P[]$

Description:

1: /* *Any non-initialized variable can be freely chosen* */
2: $Q_{13}[17-0] \leftarrow$ `0x1ff`
3: $Q_{14}[17-0] \leftarrow$ `0x1ff`
4: $Q_{12} \leftarrow -(\Phi_{15}(Q_{14}, Q_{14}, Q_{13}) + m_{15} + k_{15})$
5: $Q_{11} \leftarrow -(\Phi_{15}(Q_{14}, Q_{13}, Q_{12}) + m_{14} + k_{14})$
6: $Q[0] \leftarrow (Q_{11}, Q_{14}, Q_{13}, Q_{12})$
7: **for** $(i = 0; i < 2^{17}; i++)$ **do**
8: $\quad \Delta_F^{m_{14}} \leftarrow 2^{15} i$
9: $\quad$ *CurrentState* $\leftarrow Q[0]$
10: $\quad$ Compute steps 14 to 17, assuming $\Phi_{15}[8-0] = Q_{13}[8-0]$
11: $\quad P[i] \leftarrow$ *CurrentState*
12: **end for**
13: **for** $(i = 0; i < 2^{9}; i++)$ **do**
14: $\quad$ **for** $(j = 0; j < 2^{8}; j++)$ **do**
15: $\quad\quad \Delta_B^{m_{14}} \leftarrow i + 2^{24} j$
16: $\quad\quad$ *CurrentState* $\leftarrow P[0]$
17: $\quad\quad$ Compute inverse steps 17 to 14, assuming $\Phi_{15}[8-0] = Q_{13}[8-0]$
18: $\quad\quad Q[2^9 j + i] \leftarrow$ *CurrentState*
19: $\quad$ **end for**
20: **end for**
21: **return** $(Q[], P[])$

Complexity Analysis The biclique consists of 4 steps, which have to be computed 2^{17} times both ways. Time complexity is then

$$C_{biclique} = \frac{8}{64} 2^{17} \tag{5.6}$$

compression function evaluations. For what concern memory complexity, two lists of 2^{17} internal states each have to be stored, for a total of 2^{20} 32-bit memory words.

5.4 Padding

According to the MD5 padding rule (see Section 2.2.2), the shortest padding is appended when the length L of the message is congruent 447 mod 512. In this case the padding is just 65 bits long and it affects the last message block in the following way:

- $m_{13}[31] = 1$
- $m_{14}[8-0] = 447$
- m_{15} and remaining parts of m_{14} must be chosen according to L

The first two conditions can be easily verified since the attacker is allowed to choose those bit values, the third condition needs more attention since $m_{14}[31-15]$ is not known in advance.

Fixing the Padding When pseudo-preimages are converted to a pre-image, 2^K of them are computed using the technique described in Section 4.2.3, which means that a pseudo-preimage can be as long as 2^K message blocks, the last of which has to be 447 bits long according to the padding. So, the maximum length of a pseudo-preimage obtained by the attack is

$$512 \times (2^K - 1) + 447$$

The length specified in the padding must be bigger than that: This can be obtained by picking $m_{15}[x] = 1$ such that

$$2^{32+x} > 512 \times (2^K - 1) + 447$$

When the number of extra message blocks needed is known, expandable messages can be used to fill the gap.

Table 5.2 shows how message words can be properly initialized in order to have a correct padding, since for this particular attack $K = 8$ it is sufficient to pick $m_{15}[0] = 1$.

m_{13}	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1
m_{14}	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	1	0	1	1	1	1	1	1	1
m_{15}	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1

Table 5.2: Initialization of m_{13}, m_{14} and m_{15} - Bit positions marked x indicate that the value can bee freely chosen or otherwise determined by the attack algorithm.

5.5 Attack Algorithm

The procedure used to attack `MD5`, presented in Algorithm 5.2, can be divided into the following parts:

MTPP Initialization [Lines 1-4]
Sets $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ are defined according to the number of targets available. In particular, $\mathcal{M}_1$ is chosen so that $|\mathcal{M}_1| \times |\mathcal{H}| = 2^{17}$ and the biclique efficiency increases.

MitM Initialization [Lines 6-7]
Before each iteration of the `repeat` loop, the message block is reinitialized and a biclique is built.

Backward Computations [Lines 9-15]
For every $(M_1, H) \in \mathcal{M}_1 \times \mathcal{H}$, backward candidates are computed and stored.

Forward Computations [Lines 19-21]
For every $M_2 \in \mathcal{M}_2$, forward candidates are computed.

Candidates Testing [Lines 22-27]
Determines whether or not a partial match yields a pseudo-preimage.

Algorithm 5.3 MTPP Attack

Input:
A list of 2^k targets $\mathcal{H} = \{H_0, H_1, \ldots, H_{2^k-1}\}$
Output:
A pseudo-preimage (CV, M) for one of the targets

Description:

$x \leftarrow (8-k)$
$\mathcal{M}_1 \stackrel{def}{=}$ bit positions $[31-24]$ and $[x-0]$ of m_6
$\mathcal{M}_2 \stackrel{def}{=}$ bit positions $[31-15]$ of m_{14}
$\mathcal{M}_3 \stackrel{def}{=}$ remaining parts of the message block
repeat
  Randomly initialize M_3 but satisfy the padding
  Build biclique
  for all $M_1 \in \mathcal{M}_1$ **do**
    $Q \leftarrow$ starting state corresponding to M_1
    $CV \leftarrow D^1_{M_1,M_3}(Q)$
    for all $H \in \mathcal{H}$ **do**
      $C \leftarrow H \ominus CV$
      $v_1' \leftarrow D^3_{M_1,M_3}(c)$
      Generate candidates v_1
      Store (v_1, v_1', M_1)
    end for
  end for
  for all $M_2 \in \mathcal{M}_2$ **do**
    $P \leftarrow$ starting state corresponding to M_2
    $v_2' \leftarrow E^2_{M_2,M_3}(P)$
    Generate candidates v_2
    if partial match **then**
      Check for full match
    end if
    if full match **then**
      Check biclique property
    end if
  end for
until a preimage is found
return any (M_1, M_2, M_3) for which there is a full match and the biclique property is satisfied

5.5.1 Complexity Analysis

Both time complexity and memory complexity are analysed, the former is measured in compression functions evaluations and the latter in 32-bit words. When not explicitly specified, the term complexity always refers to time complexity.

Biclique Construction [Line 7]
Four compression steps need to be computed $|\mathcal{M}_2|$ times in the forward direction and $|\mathcal{M}_1|$ times in the backward direction:

$$C_{biclique} = \frac{4}{64}2^{17} + \frac{4}{64}2^{17-k}$$

Furthermore, two lists of 2^{17} and 2^{17-k} full states are stored, hence the memory complexity is:

$$4 \times (2^{17} + 2^{17-k}) = 2^{19} + 2^{19-k}$$

For what concerns the efficiency, when 2^k targets are available the trails interaction affects only the $(9-k)$ least significant bits of Φ_{15}, resulting in an efficiency of:

$$e_k = \left(\frac{3}{4}\right)^{9-k}$$

Backward Computations [Lines 9-15]
A part of the backward chunk is computed $|\mathcal{M}_1| = 2^{17-k}$ times, whereas the second part is computed $|\mathcal{M}_1| \times |\mathcal{H}| = 2^{17}$ times. For sake of simplicity, this is considered equivalent to 2^{17} full computations of the chunk, which is composed of 27 steps. This, together with the generation of the candidates, has a total complexity of

$$2^{17}C_b = 2^{17}\left(\frac{27}{64} + C_{bc}\right) = 2^{17}\frac{32}{64} \tag{5.7}$$

obtained using Equation (5.1).

Furthermore, at each iteration two candidates are stored together with a full state and a message word, hence the memory complexity is

$$2^{17+1} \times 6 = 3 \times 2^{19}$$

Forward Computations [Lines 19-21]
25 compression steps are computed and 2^2 candidates are generated. Since this procedure is repeated $|\mathcal{M}_2| = 2^{17}$ times, the complexity is:

$$2^{17}C_f = 2^{17}\left(\frac{25}{64} + C_{fc}\right) = 2^{17}\frac{31}{64} \tag{5.8}$$

obtained using Equation (5.2).

No memory is needed.

Matching Procedure and Biclique Property Verification [Lines 22-27]
Throughout one iteration of the `repeat` loop, 2^{17+2} forward candidates and 2^{17+1} backward candidates are generated, for a total of 2^{34+3} possible couplings. Since the matching probability is 2^{-17}, for 2^{17+3} couples the matching procedure need to be run, hence the resulting complexity is

$$2^{17+3}C_{match} = 2^{17+3}\left(2^{-5}\frac{51}{64}\right) = 2^{17}\frac{12.75}{64} < 2^{17}\frac{13}{64} \tag{5.9}$$

Amongst the 2^{17+3} partially matched couples, only for 2^{17} of them the carry hypotheses are correct. Furthermore, since the probability of a full match is 2^{-111}, the biclique property need to be verified on 2^{-94} of them on average. The complexity of such a procedure is negligible.

No memory is needed.

Total Complexity To sum up, the complexity of one iteration of the `repeat` loop is:

$$C_{loop} = C_{biclique} + 2^{17}(C_f + C_b) + 2^{17+3}C_{match} <$$
$$< 2^{17}\frac{80}{64} + 2^{17-k}\frac{4}{64}$$

As argued before, during one execution of the `repeat` loop, 2^{34+3} couples of candidates are generated. Only for $2^{34}e_k$ of them the carry hypotheses are correct and the biclique property hold. So, in order to obtain a pseudo-preimage with high probability, the loop needs to be repeated $2^{94}\frac{1}{e_k}$ times, for a total complexity of:

$$C_{ppi}(k) = \frac{2^{94}}{e_k}C_{loop} = 2^{111}\left(\frac{80}{64} + 2^{-k}\frac{4}{64}\right)\left(\frac{4}{3}\right)^{(9-k)} \tag{5.10}$$

For what concerns the total memory complexity, the only steps requiring memory are biclique construction and backward candidates generation, for total of

$$2^{19} + 2^{19-k} + 3 \times 2^{19} = 2^{21} + 2^{19-k}$$

Table 5.3 shows the attack complexities for different k.

Available Targets	Time Complexity	Memory Complexity
2^0	$2^{115.1}$	$2^{21.3}$
2^1	$2^{114.7}$	$2^{21.2}$
2^2	$2^{114.2}$	$2^{21.1}$
2^3	$2^{113.8}$	$2^{21.0}$
2^4	$2^{113.4}$	$2^{21.0}$
2^5	$2^{113.0}$	$2^{21.0}$
2^6	$2^{112.6}$	$2^{21.0}$
2^7	$2^{112.2}$	$2^{21.0}$
2^8	$2^{111.7}$	$2^{21.0}$
2^9	$2^{111.3}$	$2^{21.0}$

Table 5.3: Complexity of a Multi-Target-Pseudo-Preimage Attack

5.6 Conversion to Preimage

Equation 5.10 gives the complexity of computing one pseudo-preimage when 2^k targets are available. In order to double them, the complexity is

$$2^k C_{ppi}(k) = 2^{111} \left(2^k \frac{80}{64} + \frac{4}{64}\right) \left(\frac{4}{3}\right)^{(9-k)}$$

So, in order to obtain 2^K targets starting from a single one, the total complexity is

$$\sum_{k=0}^{K-1} \left[2^k C_{ppi}(k)\right] = 2^{111} \left(\frac{4}{3}\right)^9 \sum_{k=0}^{K-1} \left[\frac{80}{64} \left(\frac{3}{2}\right)^k + \frac{4}{64} \left(\frac{3}{4}\right)^k\right] =$$

$$= 2^{112} \frac{80}{64} \left(\frac{4}{3}\right)^9 \left[\left(\frac{3}{2}\right)^K - 1\right] + 2^{113} \frac{4}{64} \left(\frac{4}{3}\right)^9 \left[1 - \left(\frac{3}{4}\right)^K\right]$$

Eventually, to get a preimage with high probability other 2^{128-K} hashes need to be computed (see Algorithm 4.4); the lowest complexity is obtained with $K = 8$ and it is

$$2^{112}\frac{80}{64}\left(\frac{4}{3}\right)^9\left[\left(\frac{3}{2}\right)^8 - 1\right] + 2^{113}\frac{4}{64}\left(\frac{4}{3}\right)^9\left[1 - \left(\frac{3}{4}\right)^8\right] + 2^{120} \approx 2^{121.4} \quad (5.11)$$

For what concerns memory complexity, this procedure requires 20×2^8 extra memory in order to store the presudo-preimages, which is negligible with respect to the memory complexity of the pseudo-preimage attack.

Time Complexity	**Memory Complexity**
$2^{121.4}$	$2^{21.3}$

Table 5.4: Preimage Attack with MTPP Approach

Chapter 6

Pseudo-Preimage and Preimage Attacks on MD5: Two Variants

Two variants of the attack described in the previous chapter are now presented; the aim of both of them is to increase the biclique efficiency by reducing its dimension: One variant will always achieve an efficiency of 1, whereas the other one will still allow the efficiency to be less than 1.

6.1 Variant I: Biclique Efficiency 1.0

A biclique efficiency of 1 can always be achieved regardless of the number of available targets: It is sufficient to redefine $\mathcal{M}_1$ and $\mathcal{M}_2$ so that there is no trail interaction in the biclique. Algorithm 6.1 shows how this this can be done.

Algorithm 6.1 MTPP Attack - Variant I

Input:
A list of 2^k targets $\mathcal{H} = \{H_0, H_1, \ldots, H_{2^k-1}\}$
Output:
A pseudo-preimage (CV, M) for one of the targets

Description:

if k is even **then**
  $x \leftarrow (3 - \frac{k}{2})$
  $y \leftarrow (19 - \frac{k}{2})$
else
  $x \leftarrow (3 - \frac{k-1}{2})$
  $y \leftarrow (19 - \frac{k-1}{2})$
end if
$\mathcal{M}_1 \stackrel{def}{=}$ bit positions $[31-24]$ and $[x-0]$ of m_6
$\mathcal{M}_2 \stackrel{def}{=}$ bit positions $[31-y]$ of m_{14}
$\mathcal{M}_3 \stackrel{def}{=}$ remaining parts of message block
/ From now on, apart from the missing verification of the biclique property, the attack algorithm is identical */*
repeat
  Randomly initialize M_3 but satisfy the padding
  Build biclique
  for all $M_1 \in \mathcal{M}_1$ **do**
    $Q_{M_1} \leftarrow$ starting state corresponding to M_1
    $CV \leftarrow D^1_{M_1,M_3}(Q)$
    for all $H \in \mathcal{H}$ **do**
      $C \leftarrow H \boxminus CV$
      $v_1' \leftarrow D^3_{M_1,M_3}(c)$
      Generate candidates v_1
      Store (v_1, v_1', M_1)
    end for
  end for
  for all $M_2 \in \mathcal{M}_2$ **do**
    $P_{M_2} \leftarrow$ starting state corresponding to M_2
    $v_2' \leftarrow E^2_{M_2,M_3}(P)$.
    Generate candidates v_2
    if partial match **then**
      Check for full match
    end if
  end for
until a preimage is found
return (M_1, M_2, M_3) for which there is a full match.

6.1.0.1 Complexity Analisys

Both time complexity and memory complexity are reanalysed, the former is measured in compression functions evaluations and the latter in 32-bit words. When not explicitly specified, the term complexity always refers to time complexity.

During the analysis, variables h_k and r_k will be used:

$$k = 2 \times h_k + r_k \ , \ 0 \leq r_k < 2 \quad and \quad h_k \geq 0$$

Biclique Construction [Line 14]
Four steps have to be computed for $|\mathcal{M}_1| = 2^{12+k_h}$ times in the backward direction and $|\mathcal{M}_2| = 2^{13+k_h}$ times in the forward direction:

$$C_{biclique} = \frac{4}{64}2^{13+h_k}\frac{4}{64}2^{12-h_k} \leq \frac{6}{64}2^{13+h_k} \tag{6.1}$$

Furthermore, the starting states of the biclique need $2^{15+h_k} + 2^{14-h_k}$ memory to be stored.

For what concerns the efficiency, $\mathcal{M}_1$ and $\mathcal{M}_2$ are defined so that there is no trail interaction in Φ_{15}, hence the efficiency is 1.

Backward Computations [Lines 16-22]
A part of the backward chunk is computed $|\mathcal{M}_1| = 2^{12-h_k}$ times, whereas the second part is computed $|\mathcal{M}_1| \times |\mathcal{H}| = 2^{12+h_k+r_k}$ times. For sake of simplicity, this is considered equivalent to $2^{12+h_k+r_k}$ full computations of the chunk, which is composed of 27 steps. This, together with the generation of the candidates, has a total complexity of

$$2^{12+h_k+r_k}C_b = 2^{12+h_k+r_k}\left(\frac{27}{64} + C_{bc}\right) = 2^{12+h_k+r_k}\frac{32}{64} \tag{6.2}$$

Furthermore, at each iteration two candidates are stored together with a full state and a message words, hence the memory complexity is

$$2^{12+h_k+r_k+1} \times 6 = 3 \times 2^{14+h_k+r_k}$$

Forward Computations [Lines 26-28]
25 compression steps are computed and 2^2 candidates are generated. Since this procedure is repeated $|\mathcal{M}_2| = 2^{13+h_k}$ times, the complexity is:

$$2^{13+h_k} C_f = 2^{13+h_k} \left(\frac{25}{64} + C_{fc} \right) = 2^{13+h_k} \frac{31}{64} \tag{6.3}$$

No memory is needed.

Matching Procedure [Lines 29-31]
Throughout one iteration of the `repeat` loop, 2^{13+h_k+2} forward candidates and $2^{12+h_k+r_k+1}$ backward candidates are generated, for a total of 2^{25+k+3} possible couplings. Since the matching probability is 2^{-17}, for 2^{8+k+3} couples the matching procedure need to be run, hence the resulting complexity is

$$2^{8+k+3} C_{match} = 2^{8+k+3} \left(2^{-5} \frac{51}{64} \right) = 2^{8+k} \frac{12.75}{64} < 2^{8+k} \frac{13}{64} \tag{6.4}$$

No memory is needed.

Total Complexity To sum up, the complexity of one iteration of the `repeat` loop is:

$$\begin{aligned} C_{loop}(k) = C_{biclique} + 2^{13+h_k} C_f + 2^{12+h_k+r_k} C_b + 2^{8+k+3} C_{match} < \\ < 2^{13+h_k} \frac{6}{64} + 2^{13+h_k} \frac{31}{64} + 2^{12+h_k+r_k} \frac{32}{64} + 2^{8+k} \frac{13}{64} = \\ = 2^{13+h_k} \frac{37}{64} + 2^{12+h_k+r_k} \frac{32}{64} + 2^{8+k} \frac{13}{64} \end{aligned}$$

During one execution of the loop, 2^{25+k+3} couples of candidates are generated. Only for 2^{25+k} of them the carry hypotheses are correct. So, in order to obtain a pseudo-preimage with high probability, the loop needs to be repeated 2^{103-k} times, for a total complexity of:

$$C_{ppi}(k) = 2^{103-k} C_{loop}(k) = 2^{116-k+h_k} \frac{37}{64} + 2^{115-k+h_k+r_k} \frac{32}{64} + 2^{111} \frac{13}{64} \tag{6.5}$$

For what concerns the total memory complexity, the only steps requiring memory are biclique construction and backward candidates generation, for total of

$$2^{15+h_k} + 2^{14-h_k} + 3 \times 2^{14+h_k+r_k}$$

Table 6.1 shows the attack complexities for different k.

Available Targets	Time Complexity	Memory Complexity
2^0	$2^{115.7}$	$2^{16.6}$
2^1	$2^{115.1}$	$2^{17.2}$
2^2	$2^{114.7}$	$2^{17.4}$
2^3	$2^{114.1}$	$2^{18.0}$
2^4	$2^{113.8}$	$2^{18.4}$
2^5	$2^{113.2}$	$2^{19.0}$
2^6	$2^{112.8}$	$2^{19.3}$
2^7	$2^{112.2}$	$2^{20.0}$
2^8	$2^{111.9}$	$2^{20.3}$
2^9	$2^{111.3}$	$2^{21.0}$

Table 6.1: Complexity of a Multi-Target-Pseudo-Preimage Attack - Variant I

Conversion to Preimage Equation 6.5 gives the complexity of computing one pseudo-preimage when 2^k targets are available. In order to double them, the complexity is

$$2^k C_{ppi}(k)$$

So, in order to obtain 2^K targets starting from a single one, the total complexity is:

$$\sum_{k=0}^{K-1} \left[2^k C_{ppi}(k)\right] = \sum_{k=0}^{K-1} \left[2^{116+h_k}\frac{37}{64} + 2^{115+h_k+r_k}\frac{32}{64} + 2^{111+k}\frac{13}{64}\right] =$$

$$= \sum_{h=0}^{\frac{K}{2}-1} \left(2^{116+h}\frac{53}{64}\right) + \sum_{h=0}^{\frac{K}{2}-1} \left(2^{116+h}\frac{79}{64}\right) + \sum_{k=0}^{K-1} \left(2^{111+k}\frac{13}{64}\right) =$$

$$= \sum_{h=0}^{\frac{K}{2}-1} \left(2^{117+h}\frac{66}{64}\right) + \sum_{k=0}^{K-1} \left(2^{111+k}\frac{13}{64}\right)$$

$$= 2^{117}\frac{66}{64}(2^{\frac{K}{2}} - 1) + 2^{111}\frac{13}{64}(2^K - 1)$$

Eventually, to get a preimage with high probability, other 2^{128-K} hashes need to be computed (see Algorithm 4.4); the lowest complexity is obtained with $K = 8$ and it is:

$$2^{117}\frac{66}{64}(2^4 - 1) + 2^{111}\frac{13}{64}(2^8 - 1) + 2^{120} = 2^{121.5} \tag{6.6}$$

For what concerns memory complexity, this procedure requires 20×2^8 extra memory in order to store the presudo-preimages, which is negligible with respect to the memory complexity of the pseudo-preimage attack.

Time Complexity	Memory Complexity
$2^{121.5}$	$2^{20.0}$

Table 6.2: Preimage Attack with MTPP Approach - Variant I

6.2 Variant II: Optimized Biclique

The biclique efficiency is greatly affected by the 9-bit trail interaction in Φ_{15}. For the trails not to interact, these two equivalent conditions must be fulfilled:

Condition 6.1 *Whenever* $Q_{15}[8-0]$ *has a one at some bit position,* $Q_{14}[8-0]$ *must have a one at the same position*

Condition 6.2 *Whenever* $Q_{14}[8-0]$ *has a zero at some bit position,* $Q_{15}[8-0]$ *must have a zero at the same position*

This implies that the lower the hamming weight of $Q_{15}[8-0]$, the higher the number of non-interacting $Q_{14}[8-0]$; similarly, the higher the hamming weight of $Q_{14}[8-0]$ the higher the number of non-interacting $Q_{15}[8-0]$. The idea, graphically represented in Figure 6.1, is then to reduce the number of computed candidates by filtering away those differences responsible for low efficiency. If the gain in efficiency is enough to compensate for the loss of candidates, the overall complexity will be lower.

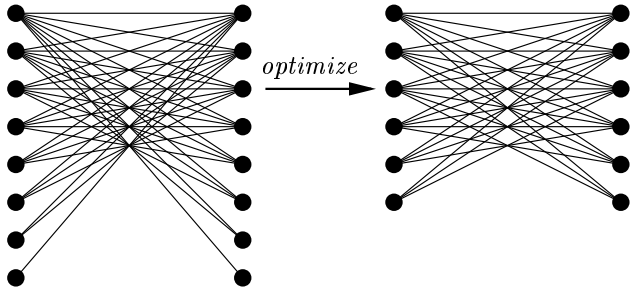


Figure 6.1: The biclique on the left has an efficiency of $e_l = \frac{36}{64} = 0.5625$, the one on the right of $e_r = \frac{30}{36} = 0.8\overline{3}$. Let C_r be the complexity of the attack when using the biclique on the right, when the other one is used instead, the complexity is $C_r = C_l \frac{e_l}{e_r} \frac{8}{6} = C_l \frac{9}{10}$

.

6.2.1 Biclique optimization

Let us assume that only differences capable of granting these properties are applied:

Condition 6.3 *$Q_{15}[8-0]$ has hamming weight $\leq 9-k$.*

Condition 6.4 *$Q_{14}[8-0]$ has hamming weight $\geq k$.*

This means that instead of using the whole range of 2^{17} possible differences on each side, only

$$2^8 \sum_{i=k}^{9} \binom{9}{i}$$

of them are used. This affects the number of candidates on each side by a factor

$$\frac{2^8 \sum_{i=k}^{9} \binom{9}{i}}{2^{17}} = \frac{\sum_{i=k}^{9} \binom{9}{i}}{2^9}$$

The latter can be generalized for the case when the interaction affects n bits, giving:

$$c_{n,k} = \frac{\sum_{i=k}^{n} \binom{n}{i}}{2^n} \tag{6.7}$$

Recomputing Biclique Efficiency In order to compute the efficiency it is convenient to distinguish two cases:

Case 1 If $Q_{15}[8-0]$ has hamming weight $(9-h) \geq k$, because of Condition 6.1 any non-interacting $Q_{14}[8-0]$ will have $t = (9-h)$ ones in fixed positions, hence its hamming weight will be $\geq t \geq k$. This implies that Condition 6.4 is already fulfilled and the remaining bits can be freely chosen, giving 2^h possible values.

Case 2 If $Q_{15}[8-0]$ has hamming weight $(9-h) < k$, any non-interacting $Q_{14}[8-0]$ will have $t = (9-h)$ ones in fixed positions, since $t < k$ Condition 6.4 requires other $k-t$ ones to be present, giving $\sum_{i=k-t}^{h} \binom{h}{i}$ possible values instead of 2^h.

The total number of couples verifying the biclique property is then:

$$\sum_{h=k}^{9} \left\{ 2^8 \binom{9}{h} \left[2^8 \sum_{i=\max(0,k-t)}^{h} \binom{h}{i} \right] \right\}$$

The latter can be generalized for the case when the interaction affects n bits, giving a biclique efficiency of:

$$\frac{\sum_{h=k}^{n} \binom{n}{h} \left[\sum_{i=\max(0,k-t)}^{h} \binom{h}{i} \right]}{\left[\sum_{i=k}^{n} \binom{n}{i} \right]^2} \tag{6.8}$$

Hence, this new approach affects the biclique efficiency by a factor

$$b_{n,k} = \frac{\sum_{h=k}^{n} \binom{n}{h} \left[\sum_{i=\max(0,k-t)}^{h} \binom{h}{i} \right]}{\left[\sum_{i=k}^{n} \binom{n}{i} \right]^2} \frac{2^{2n}}{3^n} \tag{6.9}$$

Results The complexity of one iteration of the `repeat` loop in Algorithm 5.5 is the sum of:

- Biclique construction
- Forward and backward computations
- Matching procedure
- Biclique property verification (negligible)

Time complexities of biclique construction, forward and backward computations are affected by a factor $c_{n,k}$, since they are linear with respect to $|\mathcal{M}_1|$ and $|\mathcal{M}_2|$. The complexity of the matching procedure will be affected by a factor $c_{n,k}^2$, since it is linear with respect to $|\mathcal{M}_1| \times |\mathcal{M}_2|$. Hence, assuming that the matching does not contribute significantly to the total complexity, the cost of one iteration of the loop changes approximately by a factor $c_{n,k}$.

Furthermore, due to the loss of candidates and the increased biclique efficiency, the loop must iterate $\frac{1}{b_{n,k}c_{n,k}^2}$ times more, so the total complexity is affected by a factor

$$g_{n,k} = \frac{c_{n,k}}{b_{n,k}c_{n,k}^2} = \frac{1}{b_{n,k}c_{n,k}}$$

Hence, by combining Equations 6.7 and 6.9, the following formula is obtained:

$$g_{n,k} = \frac{3^n}{\sum_{h=k}^{n}\left[\binom{n}{h}\left(\sum_{i=\max(0,k-t)}^{h}\binom{h}{i}\right)\right]}\frac{\sum_{i=k}^{n}\binom{n}{i}}{2^n} \tag{6.10}$$

Then, by minimizing Equation 6.10 as a function of k, the best trade-off between increased biclique efficiency and reduced number of candidates is found. Optimal values are listed in Table 6.3

$\mathbf{n}$	$\mathbf{k_{opt}}$	$\mathbf{g_{n,k_{opt}}}$	$\mathbf{\log_2(g_{n,k_{opt}})}$
1	0	1.000	0.000
2	1	0.964	−0.052
3	1	0.945	−0.082
4	2	0.884	−0.178
5	3	0.862	−0.214
6	3	0.821	−0.285
7	4	0.765	−0.386
8	5	0.736	−0.442
9	5	0.704	−0.507

Table 6.3: Values of k_{opt} for several length of interaction.

6.2.2 Complexity Analysis

How the modified biclique affects the time complexity has already been computed; for what concerns memory complexity, memory is needed to store the

starting states provided by the biclique and the table containing the backward candidates, which are affected by a factor $c_{n,k}$ and $c_{n,k}^2$ respectively.

Tables 6.4 and 6.5 show the complexity of pseudo-preimage attacks and preimage attack when the biclique is optimized according to Table 6.3.

Available Targets	Time Complexity	Memory Complexity
2^0	$2^{114.6}$	$2^{19.8}$
2^1	$2^{114.2}$	$2^{18.9}$
2^2	$2^{113.9}$	$2^{19.5}$
2^3	$2^{113.5}$	$2^{20.0}$
2^4	$2^{113.2}$	$2^{19.4}$
2^5	$2^{112.8}$	$2^{20.1}$
2^6	$2^{112.5}$	$2^{20.7}$
2^7	$2^{112.1}$	$2^{20.3}$
2^8	$2^{111.7}$	$2^{21.0}$
2^9	$2^{111.3}$	$2^{21.0}$

Table 6.4: Complexity of a Multi-Target-Pseudo-Preimage Attack - Variant II

Time Complexity	Memory Complexity
$2^{121.3}$	$2^{20.7}$

Table 6.5: Preimage Attack with MTPP Approach - Variant II

Chapter 7

One-Block Preimage on MD5

This chapter describes an optimized brute force attack for one-block preimages on MD5. The attack is based on the technique presented in Section 4.4.1 and the brute force approach originates from an inefficient matching procedure; an analogous matching technique was applied to AES in [BKR11]. Section 7.1 provides a high-level description of the attack, Sections 7.2 and 7.3 focus respectively on matching and biclique. Section 7.4 provides an attack algorithm together with its complexity analysis.

7.1 Attack Outline

$\mathcal{M}_1$ is constituted by bit positions $[15-9]$ of m_{12} together with message words m_8 and m_9, which will be defined as a function of m_{12} in order to obtain a local collision (see Section 7.3). $\mathcal{M}_2$ is constituted by bit positions $[19-13]$ of m_7. This determines position and length of the chunks, biclique and matching part:

Forward Chunk Steps $13, 14, 15, \ldots, 23$ (11 in total)

Backward Chunk Steps $63, 62, 61, \ldots, 50$ (14 in total)

Biclique Steps $7, 8, 9, 10, 11, 12$ (6 in total)

Matching Steps $24, 25, 26, \ldots, 49$ (26 in total)

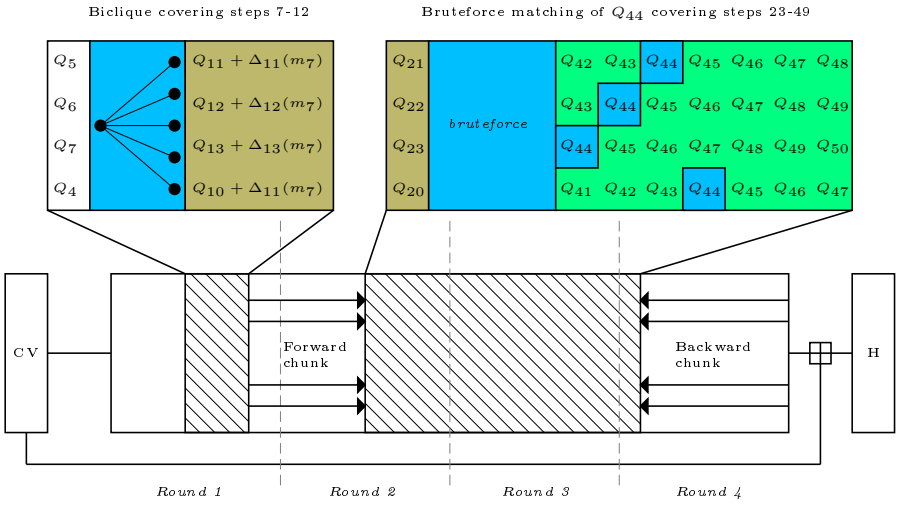


Figure 7.1: Attack Outline - Green, brown and blue background indicate computations affected by $\mathcal{M}_1$, $\mathcal{M}_2$ and both of them, respectively.

7.2 Matching

Due to the length of the matching part, the techniques described in Section 4.3 cannot provide an efficient matching procedure; however, they can be used to reduce the number of steps to be covered by the brute force approach.

Section 7.2.0.1 shows step by step how candidates are computed; in order to do so, the following notation is introduced:

- $A[x-y]$ indicates the value of bit positions $[x-y]$ of A.
- $A^{[x-y]}$ indicates that only $A[x-y]$ is known.
- $A^{[x-y]\times z}$ indicates that only $A[x-y]$ is known, and because of uncertainties on the carries there are z possible values for it.

7.2.0.1 Backward Candidates

Backward candidate consists of a partially known state, namely: $Q_{44}[12-0]$. During the computation, even though m_7 is mostly fixed, only its least significant part is used.

Step 49

$$Q_{46} = (Q_{50} - Q_{49})_{>>>10} - \Phi_{49}(Q_{49}, Q_{48}, Q_{47}) - m_7^{[12-0]} - k_{49} =$$
$$= [(Q_{50} - Q_{49})_{>>>10} - \Phi_{49}(Q_{49}, Q_{48}, Q_{47}) - m_7 - k_{49}]^{[12-0]} =$$

Step 48

$$Q_{45} = (Q_{49} - Q_{48})_{>>>6} - \Phi_{48}(Q_{48}, Q_{47}, Q_{46}^{[12-0]}) - m_0 - k_{48} =$$
$$= [(Q_{49} - Q_{48})_{>>>6} - \Phi_{48}(Q_{48}, Q_{47}, Q_{46}) - m_0 - k_{48}]^{[12-0]}$$

Step 47

$$Q_{44} = (Q_{48} - Q_{47})_{>>>23} - \Phi_{47}(Q_{47}, Q_{46}^{[12-0]}, Q_{45}^{[12-0]}) - m_2 - k_{47}$$
$$= [(Q_{48} - Q_{47})_{>>>23} - \Phi_{47}(Q_{47}, Q_{46}, Q_{45}) - m_2 - k_{47}]^{[12-0]}$$

Complexity Analysis Three compression steps have to be computed without any ambiguity on the carries, the complexity is then

$$C_{fc} = \frac{3}{64} \tag{7.1}$$

compression function evaluations and 1 candidate is generated.

7.2.1 Matching Procedure

The matching procedure has to be executed for every M_1 and M_2, bringing the total complexity extremely close to naive brute force.

Algorithm 7.1 Matching Procedure

Input:
A couple (M_1, M_2)
State $(Q_{20}, Q_{23}, Q_{22}, Q_{21})$ correspondent to M_2
State $(Q_{47}, Q_{50}, Q_{49}, Q_{48})$ correspondent to M_1
Candidate $Q_{44}[12-0]$ correspondent to M_1

Output:
`true` in case of full match, `false` otherwise.

Description:
1: */* If any of the checks fail, the algorithm will stop and return* `false` **/*
2: Obtain Q_{44} by computing steps 24 to 43
3: Check for a partial match on the backward candidate $Q_{44}[12-0]$
4: Compute steps 44 to 49
5: Check for further matching on full state $(Q_{47}, Q_{50}, Q_{49}, Q_{48})$
6: **return** `true`

Complexity Analysis A matching on Q_{44} will happen with probability 2^{-13}, the expected running time is then:

$$C_{match} = \frac{20}{64}(1 - 2^{-13}) + \frac{26}{64}2^{-13} \tag{7.2}$$

compression function evaluations.

7.3 Biclique

This section provides a construction algorithm for the biclique and a high level description of the differential trails. A complete proof of their soundness and an accurate computation of the biclique efficiency is provided in Appendix B.

7.3.1 Trails Description

Differential trails are defined with respect to modular addition, hence, rotations and boolean functions are a source of non-linearities. These latter will be handled by properly initializing states and messages so that:

- Rotations can be considered as multiplications or divisions.
- Boolean functions absorb differences.
- Forward and backward trails affect different portions of the states.

Figure 7.2 provides a visual description of the trails.

Backward Trail As explained in Section 4.4.1, the backward trail has to from a local collision. The type of local collision used here has already been applied to MD5 in [SA08] and it requires the freedom to modify three message words $m_{\pi(i)}$, $m_{\pi(i+1)}$ and $m_{\pi(i+4)}$:

1. At step $(i+4)$ the difference is generated by $m_{\pi(i+4)}$.
2. At steps $(i+4)$, $(i+3)$ and $(i+2)$ the difference propagation is stopped by exploiting absorption properties of the boolean functions.
3. At steps $(i+1)$ and i the difference propagation is stopped by exploiting absorption properties of the boolean functions and message words $m_{\pi(i+1)}$ and $m_{\pi(i)}$.

Forward Trail Other than non interfering with the backward trail, the forward trail does not have to fulfil any particular requirement.

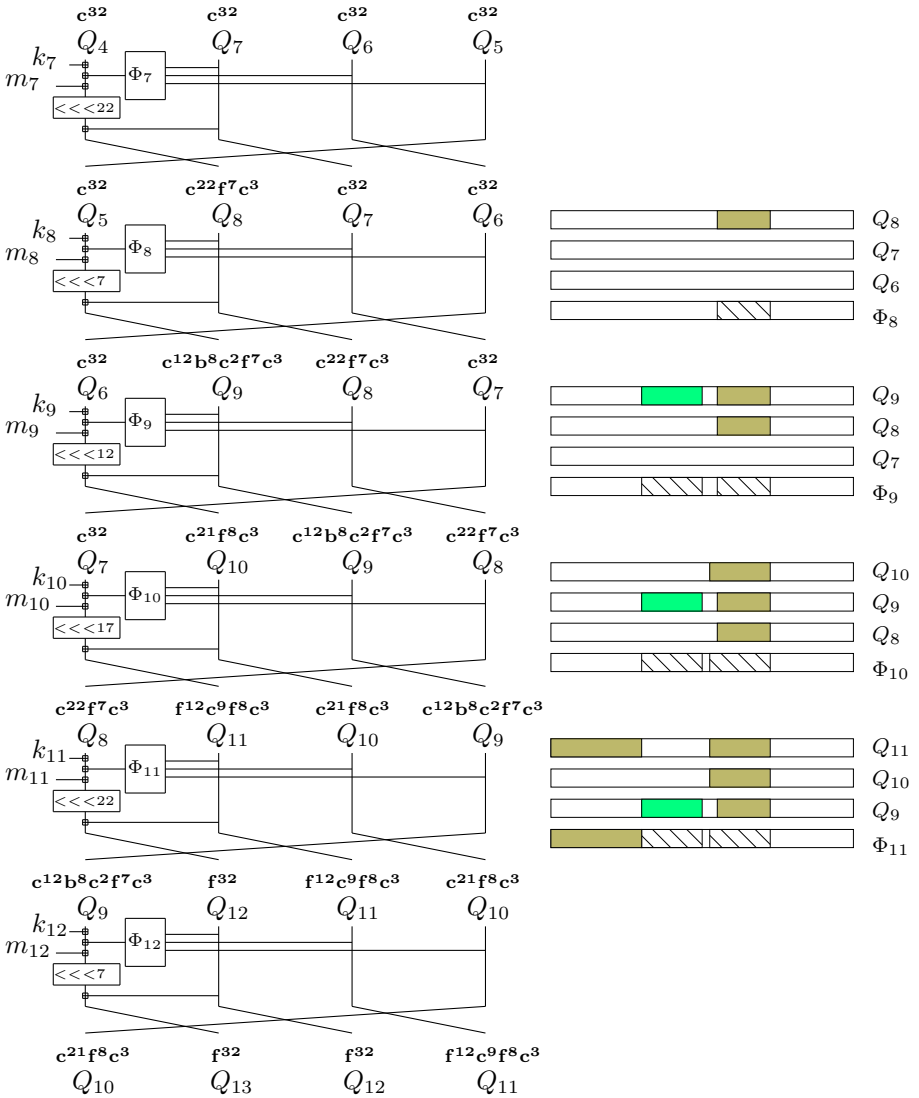


Figure 7.2: Biclique for Steps 7-12 - Above each state its configuration is indicated. Notations like $f^x c^y b^z$ mean that x bits are possibly affected by the forward trail, followed by y constant bits and z bits possibly affected by the backward trail. On the right side, the Φ functions and their inputs: Colours white, green, brown, blue and oblique lines indicate respectively constant parts, influence of backward trail, influence of forward trail, influence of both and absorption of differences.

7.3.2 Construction Algorithm

The procedure used to construct the biclique, presented in Algorithm 7.2, can be divided into three parts:

Message and State Initialization [Lines 2-12]
The starting state (Q_4, Q_7, Q_6, Q_5) and message words m_7, m_9 and m_{10} are initialized so that the differential trails behave as expected.

Forward Trail [First Loop]
For every possible difference on m_7, the correspondent starting state $(Q_{10}, Q_{13}, Q_{12}, Q_{11})$ is found by computing steps 7 to 12.

Backward Trail [Second Loop]
Nothing to do: Thanks to the local collision, the starting state is the same regardless of the value of m_{12}.

Algorithm 7.2 Biclique construction

Input:
None
Output:
Arrays of starting states $Q[]$ and $P[]$

Description:
1: /* *Any non-initialized variable can be freely chosen* */
2: $Q_8[9-3] \leftarrow$ `0x0`
3: $Q_9[9-3] \leftarrow$ `0x0`
4: $Q_9[19] \leftarrow$ `0x1`
5: $Q_9[10] \leftarrow Q_8[10]$
6: $Q_{10}[10] \leftarrow$ `0x1`
7: $Q_{10}[19-12] \leftarrow$ `0x0`
8: $Q_{11}[10] \leftarrow$ `0x1`
9: $Q_{11}[19-12] \leftarrow$ `0xff`
10: Choose m_{10} so that:
$Q_7[9-3] =$ `0x7f`
$Q_7[19-12] = Q_7[19-12]$
$(Q_7 + \Phi_{10}(Q_{10}, Q_9, Q_8) + m_{10} + k_{10})[31] = 0$
11: Choose m_9 so that $Q_6[9-3] =$ `0x7f`
12: Choose m_7 so that $(Q_4 + \Phi_7(Q_7, Q_6, Q_5) + m_7 + k_7)[31] = 0$
13: $Q[0] \leftarrow (Q_4, Q_7, Q_6, Q_5)$
14: **for** $(i = 0; i < 2^7; i++)$ **do**
15: $\Delta_F^{m_7} \leftarrow 2^{13} i$
16: *CurrentState* $\leftarrow Q[0]$
17: Compute Steps 14 to 12
18: $P[i] \leftarrow$ *CurrentState*
19: **end for**
20: **for** $(i = 0; i < 2^7; i++)$ **do**
21: $Q[i] \leftarrow Q[0]$
22: **end for**
23: **return** $(Q[], P[])$

Complexity Analysis The biclique consists of 6 steps, which have to be computed only for the forward trail since the backward trail constitutes a local collision. The time complexity is then

$$C_{biclique} = \frac{6}{64} 2^7 \tag{7.3}$$

compression function evaluations. For what concern memory complexity, again because of the local collision, only one list of 2^7 internal states each has to be

stored, for a total of 2^9 32-bit memory words.

7.4 Attack Algorithm

The procedure used to attack `MD5`, presented in Algorithm 7.4, can be divided into the following parts:

MitM Initialization [Lines 2-4]
Before each iteration of the `repeat` loop, the message block is reinitialized and a biclique is built.

Backward Computations [Lines 6-9]
For every m_{12}, message words m_9 and m_8 are chosen so that the local collision is achieved. Backward candidates are then computed and stored.

Forward Computations [Lines 12-14]
For every m_7, forward candidates are computed.

Candidates Testing [Line 15]
Determines which couples of candidates fully match.

Algorithm 7.3 One-Block Preimage

Input:
A target hash H
Output:
A one-block preimage M for H

Description:

1: **repeat**
2: Randomly initialize M_3 but assure that:
padding is satisfied
$D^1_{M_3}(Q) = IV$
3: Build biclique
4: $C \leftarrow H \ominus IV$
5: **for** $(i = 0; i < 2^7; i++)$ **do**
6: $\Delta_B^{m_{12}} \leftarrow 2^{12}i$; $\Delta_B^{m_9} \leftarrow i$; $\Delta_B^{m_8} \leftarrow -2^5 i$
7: $v_2' \leftarrow D^3_{M_1,M_3}(C)$
8: Generate candidates v_2 from v_2'
9: Store (v_2, v_2', M_1) in a table
10: **end for**
11: **for all** $(i = 0; i < 2^7; i++)$ **do**
12: $\Delta_F^{m_7} \leftarrow 2^3 i$
13: $P \leftarrow$ starting state corresponding to M_2
14: $v_1' \leftarrow E^2_{M_2,M_3}(P)$
15: Run matching procedure
16: **end for**
17: **until** a preimage is found
18: **return** any $(M1, M_2, M_3)$ for which there was a full match

7.4.1 Complexity Analysis

Both time complexity and memory complexity are analysed, the former is measured in compression functions evaluations and the latter in 32-bit words. When not explicitly specified, the term complexity always refers to time complexity.

Biclique Construction [Line 3]
As showed in Section 7.3.2, a biclique can be built in time

$$C_{biclique} = \frac{6}{64} 2^7$$

and requires 2^9 memory.

Backward Computations [Lines 6-9]
For all 2^7 possible choices of M_1, 14 compression steps are computed and 1 candidate is generated, for total a complexity of:

$$2^7 C_b = 2^7 \left(\frac{14}{64} + C_{bc} \right) = 2^7 \frac{17}{64} \tag{7.4}$$

Furthermore, a candidate needs to be stored at each iteration, together with a full state and the corresponding M_1. This requires $6 \times 2^7 = 3 \times 2^8$ memory.

Forward Computations [Lines 12-14]
For all 2^7 possible choices of M_2, 11 compression steps are computed:

$$2^7 C_f = 2^7 \frac{11}{64} \tag{7.5}$$

No memory is needed.

Matching Procedure [Line 15]
For all 2^{14} possible choices of (M_1, M_2) the matching procedure is performed:

$$2^{14} C_{match} = 2^{14} \left(\frac{20}{64} + \frac{6}{64} 2^{-13} \right) = \frac{20}{64} 2^{14} + \frac{6}{64} 2^1 \tag{7.6}$$

No memory is needed.

Total Complexity The total complexity is dominated by the brute force matching, giving:

$$\begin{aligned} C_{loop} &= C_{biclique} + 2^7 (C_b + C_f) + 2^7 C_{match} = \\ &= \tfrac{6}{64} 2^7 + 2^7 \tfrac{17}{64} + 2^7 \tfrac{11}{64} + \tfrac{20}{64} 2^{14} + \tfrac{6}{64} 2^1 \approx \tfrac{20}{64} 2^{14} \end{aligned} \tag{7.7}$$

During one execution of the `repeat` loop, 2^{14} couples of candidates are checked for matching. Hence, In order to obtain a full match the loop need to be iterated 2^{114} times, for a total complexity of:

$$C_{pi} = 2^{114} C_{loop} = \frac{20}{64} 2^{128} \approx 2^{126.4} \tag{7.8}$$

For what concerns the total memory complexity, the only steps requiring memory are biclique construction and backward candidates generation, for total of

$$2^9 + 3 \times 2^8 = 2^{10.3}$$

Time Complexity	Memory Complexity
$2^{126.4}$	$2^{10.3}$

Table 7.1: One-Block Preimage Attack with Optimized Brute Force.

Chapter 8

Attacks Comparison and Conclusions

In the second part of this thesis several attacks have been presented, this chapter divides them into three categories: pseudo-preimage, preimage and one-block preimage. In each category the attacks are compared with the best known results for `MD5`, namely: [SA09] for pseudo-preimages and preimages and [AS08] for one-block preimages. As usual, time and memory complexities are measured in compression functions evaluations and 32-bit memory words, respectively.

8.1 Pseudo-Preimages

Three pseudo-preimage attacks were presented in this thesis, all of them are based on the same attack:

- Section 5.5: Standard variation (bigraph).
- Section 6.1: Variation I (biclique).
- Section 6.2: Variation II (optimized bigraph).

Table 8.1 compares these attacks with the result of Sasaki and Aoki in [SA09].

Attack	Time Complexity	Memory Complexity
Section 5.5	$2^{115.1}$	$2^{21.3}$
Section 6.1	$2^{115.7}$	$2^{16.6}$
Section 6.2	$2^{114.6}$	$2^{19.8}$
[SA09]	$2^{116.9}$	$2^{48.4}$

Table 8.1: Comparison of Pseudo-Preimage Attacks

Thanks to the reduced biclique dimension, both variants of the attack presented in Chapter 5 have a lower memory complexity; however, the optimized biclique allows Variant II to achieve a better time complexity as well.

MTPP

The three variants of the attack can exploit multiple targets, Figures 8.1 and 8.2 compare their performances in different situations.

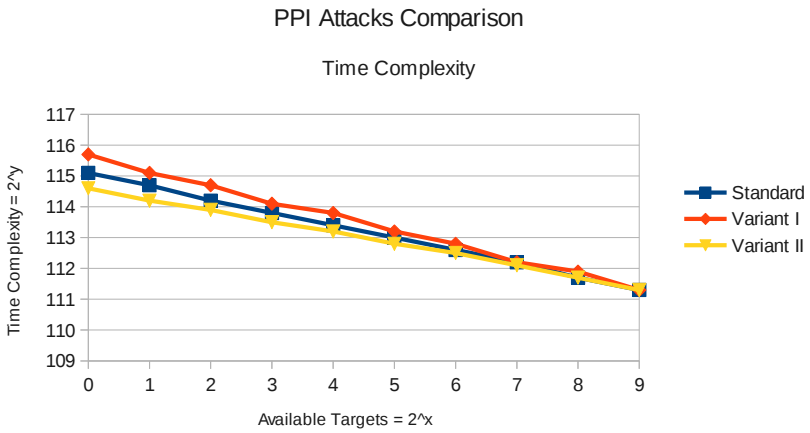


Figure 8.1: MTPP Time Complexity

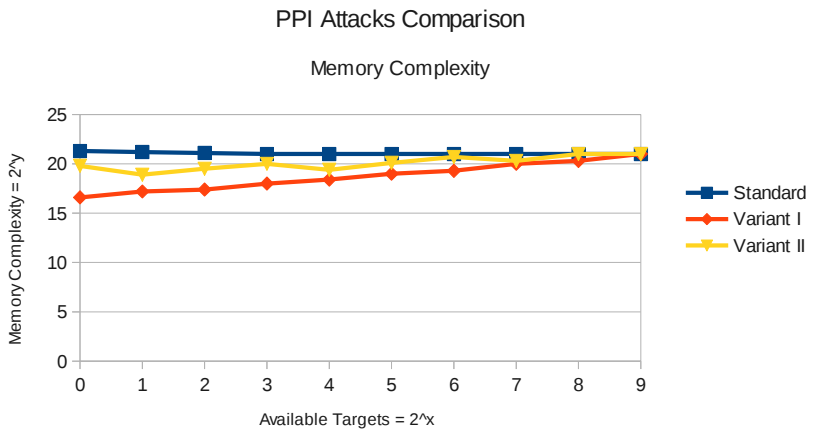


Figure 8.2: MTPP Memory Complexity

8.2 Preimages

All pseudo-preimage attacks compared in the previous section can be converted to preimage attacks using the standard MitM conversion oulined in Algorithm 4.1, whereas the three presented in this thesis can exploit MTTP techniques and Algorithm 4.4 as well. Table 8.2, compares the performances of the different attacks and different conversion methods.

Attack	Time Complexity	Memory Complexity
Section 5.5 [MitM]	$2^{122.6}$	$2^{21.3}$
Section 6.1[MitM]	$2^{122.9}$	$2^{16.6}$
Section 6.2 [MitM]	$2^{122.3}$	$2^{19.8}$
[SA09] [MitM]	$2^{123.4}$	$2^{48.4}$
Section 5.5 [MTPP]	$2^{121.4}$	$2^{21.3}$
Section 6.1[MTPP]	$2^{121.5}$	$2^{20.0}$
Section 6.2[MTPP]	$2^{121.3}$	$2^{20.7}$

Table 8.2: Comparison of Preimage Attacks

For what concerns MitM conversion, all attacks maintain the same characteristics of the original pseudo-preimage attacks; when using MTPP, however, the complexities are really similar.

The reason for this is the way a layered hash tree is built: 2^k pseudo-preimages are computed when 2^k targets are available for $k = 0, 1, \ldots, K-1$, hence the average number of available targets while computing a pseudo-preimage is

$$\frac{1}{2^K} \sum_{k=0}^{K-1} 2^k 2^k = \frac{1}{2^K} \frac{4^K - 1}{3} \approx 2^{K-1.6}$$

This implies that most of the attacks are performed when many targets are already available, which is precisely the condition when all variants have similar performances.

8.3 One-Block Preimages

This is the last attack presented in this thesis and it will be compared to the optimized brute force in [AS08].

Attack	Time Complexity	Memory Complexity
Section 7.4	$2^{126.4}$	$2^{10.3}$
[AS08]	$2^{127.0}$	negligible

Table 8.3: Comparison of One-Block Preimage Attacks

8.4 Conclusions

The only known preimage attack on full `MD5` was [SA09], whose practical applications are limited by the following issues:

- Time complexity too high.
- Memory complexity too high to allow for an efficient implementation of the attack.
- The length of the preimage cannot be chosen.

The attack presented in Section 5.5 and its variants solve the problem of the memory complexity and slightly improve the time complexity. The reduced memory requirement is due to the use of bicliques, whereas their extension to bigraphs and application in the context of MTPP techniques allowed for modest improvements on the time complexity. However, two issues still remain:

Time Complexity If the attack performance is assumed to be bounded by the birthday paradox, $|\mathcal{M}_1| = |\mathcal{M}_2| = 2^{17}$ implies that the time complexity cannot be lower than $\approx 2^{111}$, still too high be of practical use. On the other hand, further increasing the size of $\mathcal{M}_1$ and $\mathcal{M}_2$ would lead to problems on the matching which cannot be addressed by the techniques presented in this thesis; finding differential trails would be harder as well.

Preimage Length Message word m_{14} cannot be freely chosen, so the attacker cannot decide the length of the preimage. On the other end, other configurations of the chunks will have worse performances due to the number of steps to be covered by either the matching procedure or the biclique.

For what concerns one-block preimages, the only known attack on full `MD5` was the optimized brutefoce in [AS08]. The attack presented in this thesis is still an optimization over bruteforce, the time complexity is slightly lower whereas the memory complexity is higher but still feasible for an efficient implementation. However, none of the attacks can be of practical use due to their high time complexity.

Appendix A

Biclique

A.1 Biclique construction

Here a detailed description of the biclique covering steps 14-17 is given. Each differential trail is first analysed independently, then it is either shown that they do not interfere with each other or computed exactly for what couples of differences they do interfere.

A.1.1 Notation

- `F` stands for forward trail, related to bits $[31-15]$ of m_{14}.
- `B1` stands for backward trail one, related to bits $[31-24]$ of m_6.
- `B2` stands for backward trail two, related to bits $[8-0]$ of m_6.
- `BT` stands for backward trails, related to bits $[31-24]$ and $[8-0]$ of m_6.
- Δ_T represents the difference introduced in the trail T, which could be F,B,$B1$ or $B2$.
- Δ_T^n represents the difference at state Q_n when a difference Δ_T is introduced.

- $\Delta_T^{\Phi_n}$ represents the difference at the output of Φ_n when a difference Δ_T is introduced.
- The value of state n when no differences are applied is indicated with Q_n, when those differences are considered, the notations $Q_n + \Delta_T^n$ or Q_n^T are used instead.

A.1.2 States initialization

States $Q_{11}, Q_{12}, Q_{13}, Q_{14}$ and message words m_{14}, m_{15}, m_1, m_6 can be freely chosen provided the following equations be verified:

$$Q_{13}[17-0] = 00000000111111111 \tag{A.1}$$

$$Q_{14}[17-0] = 00000000111111111 \tag{A.2}$$

$$Q_{12} = -(\Phi_{15}(Q_{14}, Q_{14}, Q_{13}) + m_{15} + k_{15}) \tag{A.3}$$

$$Q_{11} = -(\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + m_{14} + k_{14}) \tag{A.4}$$

Equations A.4 together with A.3 gives:

$$\begin{aligned} Q_{12} &= -(\Phi_{15}(Q_{14}, Q_{14}, Q_{13}) + m_{15} + k_{15}) = \\ &= -(\Phi_{15}(Q_{15}, Q_{14}, Q_{13}) + m_{15} + k_{15}) \end{aligned} \tag{A.5}$$

Hence

$$Q_{14} = Q_{15} = Q_{16} \tag{A.6}$$

A.1.3 Forward trail

A.1.3.1 Step 14

$$\begin{aligned} Q_{15}^F &= Q_{14} + \left[Q_{11} + \Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + m_{14} + 2^{15}\Delta_F + k_{14}\right]_{<<<17} = \\ &= Q_{14} + \left[2^{15}\Delta_F\right]_{>>>15} = \end{aligned}$$

$$= Q_{14} + \Delta_F$$

Where the first equation is the MD5 step, the second equation is true thanks to (A.4) and the last one is true since $\Delta_F \in \{0, 1, 2, \cdots, 2^{17} - 1\}$ and hence does not generate any carry previous to the shift. So we have:

$$\Delta_F^{15} = \Delta_F \tag{A.7}$$

Furthermore, thanks to (A.2) only $Q_{15}[17, 0]$ is affected by the difference.

A.1.3.2 Step 15

$$Q_{16}^F = Q_{15} + \Delta_F^{15} + \left[Q_{12} + \Phi_{15}(Q_{15}^F, Q_{14}, Q_{13}) + m_{15} + k_{15}\right]_{<<<22} =$$
$$= Q_{15} + \Delta_F^{15} + \left[Q_{12} + \Phi_{15}(Q_{15}, Q_{14}, Q_{13}) + m_{15} + k_{15}\right]_{<<<22} =$$
$$= Q_{15} + \Delta_F^{15} + [0]_{<<<22}$$

Where the first equation is the MD5 step, the second equation is true thanks to (A.1),(A.2) and the absorption properties of Φ_{15}. The last one is true thanks to (A.3) and (A.6). So we have:

$$\Delta_F^{16} = \Delta_F^{15} \tag{A.8}$$

Furthermore $Q_{16}^F = Q_{15}^F$.

A.1.3.3 Step 16

$$Q_{17}^F = Q_{16} + \Delta_F^{16} + \left[Q_{13} + \Phi_{16}(Q_{16} + \Delta_F^{16}, Q_{15} + \Delta_F^{15}, Q_{14}) + m_1 + k_{16}\right]_{<<<5} =$$
$$= Q_{16} + \Delta_F^{16} + \left[Q_{13} + \Delta_F^{16} + \Phi_{16}(Q_{16}, Q_{15}, Q_{14}) + m_1 + k_{16}\right]_{<<<5}$$

Where the first equation is the MD5 step and the second equation is true thanks to the the fact that $Q_{15}^F = Q_{16}^F$ and the absorption properties of Φ_{16}. Now, depending on whether or not Δ_F^{16} generates a carry previous to the shift and how far it propagates, we have:

$$\Delta_F^{17} = \begin{cases} \Delta_F^{16} + 2^5\Delta_F^{16} & \text{carry no further than bit 26} \\ \Delta_F^{16} + 2^5\Delta_F^{16} + 1 & \text{carry between bits 27-31} \\ \Delta_F^{16} + 2^5\Delta_F^{16} + 1 - 2^5 & \text{carry past bit 31} \end{cases} \tag{A.9}$$

A.1.3.4 Step 17

$$Q_{18}^F = Q_{17}^F + \left[Q_{14} + \Phi_{17}(Q_{17}^F, Q_{16}^F, Q_{15}^F) + m_6 + k_{17}\right]_{<<<9} =$$
$$= Q_{17} + \Delta_F^{17} + \left[Q_{14} + \Phi_{17}(Q_{17}, Q_{16}, Q_{15}) + \Delta_F^{\Phi_{17}} + m_6 + k_{17}\right]_{<<<9}$$

Where the first equation is the MD5 step and the second equation is true by definition. However, in order to compute $\Delta_F^{\Phi_{17}}$ it is necessary to know the value of the states Q_{17}^F, Q_{16}^F and Q_{15}^F. Like in the previous step, knowing whether a carry will be generated and how far it will propagate, it is possible to write a formula to predict the difference on Q_{18}

A.1.4 Backward Trail 1

A.1.4.1 Step 17

$$Q_{14}^{B1} = [Q_{18} - Q_{17}]_{>>>9} - \Phi_{17}(Q_{17}, Q_{16}, Q_{15}) - (m_6 + 2^{24}\Delta_{B1}) - k_{17}$$

Where the first equation is the MD5 inverse step and $\Delta_{B1} \in \{0, 1, 2, \cdots, 2^8 - 1\}$. So clearly we have:

$$\Delta_{B1}^{14} = -2^{24}\Delta_{B1} \tag{A.10}$$

Furthermore only $Q_{14}[31, 24]$ is affected by the difference.

A.1.4.2 Step 16

$$Q_{13}^{B1} = [Q_{17} - Q_{16}]_{>>>5} - \Phi_{16}(Q_{16}, Q_{15}, Q_{14}^{B1}) - m_1 - k_{16} =$$
$$= [Q_{17} - Q_{16}]_{>>>5} - \Phi_{16}(Q_{16}, Q_{15}, Q_{14}) - m_1 - k_{16}$$

Where the first equation is the MD5 inverse step and the second one is true thanks to the fact that $Q_{15} = Q_{16}$ and the absorption properties of Φ_{16}. Hence:

$$\Delta_{B1}^{13} = 0 \tag{A.11}$$

A.1.4.3 Step 15

$$Q_{12}^{B1} = [Q_{16} - Q_{15}]_{>>>22} - \Phi_{15}(Q_{15}, Q_{14}^{B1}, Q_{13}) - m_{15} - k_{15} =$$
$$= [Q_{16} - Q_{15}]_{>>>22} - (\Phi_{15}(Q_{15}, Q_{14}, Q_{13}) + \Delta_{B1}^{\Phi_{15}}) - m_{15} - k_{15}$$

Where the first equation is the MD5 inverse step and the second equation is true by definition. However, in order to compute $\Delta_{B1}^{\Phi_{15}}$ it is necessary to know the value of $Q_{15}[31,24]$, $Q_{14}^{B1}[31,24]$ and $Q_{13}[31,24]$. So we have:

$$\Delta_{B1}^{12} = -\Delta_{B1}^{\Phi_{15}} \tag{A.12}$$

Furthermore only $Q_{12}[31,24]$ is affected by the difference.

A.1.4.4 Step 14

$$\begin{aligned}
Q_{11}^{B1} &= \left[Q_{15} - Q_{14}^{B1}\right]_{>>>17} - \Phi_{14}(Q_{14}^{B1}, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} = \\
&= [Q_{15} - (Q_{14} + \Delta_{B1}^{14})]_{>>>17} - \Phi_{14}(Q_{14}^{B1}, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} = \\
&= [-\Delta_{B1}^{14}]_{>>>17} - \Phi_{14}(Q_{14}^{B1}, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} = \\
&= [2^{24}\Delta_{B1}]_{>>>17} - \Phi_{14}(Q_{14}^{B1}, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} = \\
&= 2^{7}\Delta_{B1} - \Phi_{14}(Q_{14}^{B1}, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} = \\
&= 2^{7}\Delta_{B1} - (\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}}) - m_{14} - k_{14} = \\
&= \left[Q_{15} - (Q_{14} + \Delta_{B1}^{14})\right]_{>>>17} - (\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}}) - m_{14} - k_{14} =
\end{aligned}$$

Where the first equation is the MD5 inverse step, the second equation is true by definition, the third one is true because $Q_{15} = Q_{14}$, the fourth one is true thanks to (A.10), the fifth one is true since no carry is generated previous to the shift and the sixth one is true by definition. However, in order to compute $\Delta_{B1}^{\Phi_{14}}$ it is necessary to know the value of $Q_{14}^{B1}[31,24]$, $Q_{13}[31,24]$ and $Q_{12}^{B1}[31,24]$. Hence:

$$\Delta_{B1}^{11} = 2^{7}\Delta_{B1} - \Delta_{B1}^{\Phi_{14}} \tag{A.13}$$

A.1.5 Backward trail 2

A.1.5.1 Step 17

$$Q_{14}^{B2} = [Q_{18} - Q_{17}]_{>>>9} - \Phi_{17}(Q_{17}, Q_{16}, Q_{15}) - (m_6 + \Delta_{B2}) - k_{17}$$

Where the first equation is the MD5 inverse step and $\Delta_{B2} \in \{0, 1, 2, \cdots, 2^9 - 1\}$. So clearly we have:

$$\Delta_{B2}^{14} = -\Delta_{B2} \tag{A.14}$$

Furthermore, thanks to (A.2) the difference will affect only $Q_{14}[8,0]$.

A.1.5.2 Step 16

$$Q_{13}^{B2} = [Q_{17} - Q_{16}]_{>>>5} - \Phi_{16}(Q_{16}, Q_{15}, Q_{14}^{B2}) - m_1 - k_{16} =$$
$$= [Q_{17} - Q_{16}]_{>>>5} - \Phi_{16}(Q_{16}, Q_{15}, Q_{14}) - m_1 - k_{16}$$

Where the first equation is the MD5 inverse step and the second one is true thanks to the fact that $Q_{15} = Q_{16}$ and the absorption properties of Φ_{16}. So:

$$\Delta_{B2}^{13} = 0 \tag{A.15}$$

A.1.5.3 Step 15

$$Q_{12}^{B2} = [Q_{16} - Q_{15}]_{>>>22} - \Phi_{15}(Q_{15}, Q_{14}^{B2}, Q_{13}) - m_{15} - k_{15} =$$
$$= [Q_{16} - Q_{15}]_{>>>22} - \Phi_{15}(Q_{15}, Q_{14}, Q_{13}) - m_{15} - k_{15}$$

Where the first equation is the MD5 inverse step and the second one is true assuming that the difference is absorbed. So we have:

$$\Delta_{B2}^{12} = 0 \tag{A.16}$$

A.1.5.4 Step 14

$$Q_{11}^{B2} = \left[Q_{15} - Q_{14}^{B2}\right]_{<<<15} - \Phi_{14}(Q_{14}^{B2}, Q_{13}, Q_{12}) - m_{14} - k_{14} =$$
$$= \left[Q_{15} - (Q_{14} + \Delta_{14}^{B2})\right]_{<<<15} - \Phi_{14}(Q_{14}^{B2}, Q_{13}, Q_{12}) - m_{14} - k_{14} =$$
$$= \left[-\Delta_{14}^{B2}\right]_{<<<15} - \Phi_{14}(Q_{14}^{B2}, Q_{13}, Q_{12}) - m_{14} - k_{14} =$$
$$= [\Delta_{B2}]_{<<<15} - \Phi_{14}(Q_{14}^{B2}, Q_{13}, Q_{12}) - m_{14} - k_{14} =$$
$$= 2^{15}\Delta_{B2} - \Phi_{14}(Q_{14}^{B2}, Q_{13}, Q_{12}) - m_{14} - k_{15} =$$
$$= 2^{15}\Delta_{B2} - (\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B2}^{\Phi_{14}}) - m_{14} - k_{14}$$

Where the first equation is the MD5 inverse step, the second equation is true by definition, the third one is true because $Q_{15} = Q_{14}$, the fourth one is true thanks to (A.14), the fifth one is true since no carry past position 16 is generated previous to the shift and the sixth one is true by definition. However, in order to compute $\Delta_{B2}^{\Phi_{14}}$ it is necessary to know the value of $Q_{14}^{B2}[8,0]$, $Q_{13}[8,0]$ and $Q_{12}[8,0]$. Hence:

$$\Delta_{B2}^{11} = 2^{15}\Delta_{B2} - \Delta_{B2}^{\Phi_{14}} \tag{A.17}$$

A.1.6 Trails interaction

Here each step is analysed when both trails are active.

A.1.6.1 Step 14

Backward trails

$$Q_{11}^B = \left[Q_{15}^F - Q_{14}^B\right]_{<<<15} - \Phi_{14}(Q_{14}^B, Q_{13}^B, Q_{12}^B) - m_{14} - k_{14} =$$
$$= \left[Q_{15} + \Delta_{15}^F - (Q_{14} + \Delta_{14}^B)\right]_{<<<15} - \Phi_{14}(Q_{14}^B, Q_{13}^B, Q_{12}^B) - m_{14} - k_{14} =$$
$$= \left[Q_{15} + \Delta_{15}^F - (Q_{14} + \Delta_{14}^B)\right]_{<<<15} - \Phi_{14}(Q_{14}^B, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} =$$
$$= \left[\Delta_{15}^F - \Delta_{14}^B\right]_{<<<15} - \Phi_{14}(Q_{14}^B, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14} =$$
$$= \left[\Delta_F + 2^{24}\Delta_{B1} + \Delta_{B2}\right]_{<<<15} - \Phi_{14}(Q_{14}^B, Q_{13}, Q_{12}^{B1}) - m_{14} - k_{14}$$

Where the first equation is the `MD5` inverse step, the second equation is true by definition, the third one is true since Q_{13} is not affected by any trail and Q_{12} is affected only by `B1`, the fourth one is true because $Q_{15} = Q_{14}$, the fifth one is true thanks to (A.14), (A.10) and (A.7).

Now, the following facts:

- $Q_{14}^B[8,0]$, $Q_{13}[8,0]$ and $Q_{12}^{B1}[8,0]$ are independent of `B1` and `F`
- $Q_{14}^B[31,24]$, $Q_{13}[31,24]$ and $Q_{12}^{B1}[31,24]$ are independent of `B2` and `F`

allow us to compute $\Delta_{B1}^{\Phi_{14}}$ and $\Delta_{B2}^{\Phi_{14}}$ independently of the other trails. So we can write:

$$Q_{11}^B = \left[\Delta_F + 2^{24}\Delta_{B1} + \Delta_{B2}\right]_{<<<15} - (\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}}) - m_{14} - k_{15} =$$
$$= [\Delta_F + \Delta_{B2}]_{<<<15} + 2^7\Delta_{B1} - (\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}}) - m_{14} - k_{15}$$

where the last equation is true since $\Delta_F + \Delta_{B2} < 2^{18}$ and $2^{24}\Delta_{B2} < 2^{32} - 2^{24}$, so there are no carries past bit 31. Now, if we assume that $\Delta_F + \Delta_{B2} < 2^{17}$ there cannot be a carry past bit 16, so we can write:

$$Q_{11}^B = 2^{15}\Delta_F + 2^{15}\Delta_{B2} + 2^7\Delta_{B1} - (\Phi_{14}(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}}) - m_{14} - k_{15}$$

To sum up, Δ_{11}^{B1} is always correct, whereas for Δ_{11}^{B2} to be correct we must have $\Delta_F + \Delta_{B2} < 2^{17}$.

Forward trail

$$Q_{15}^F = Q_{14}^B + \left[Q_{11}^B + \Phi(Q_{14}^B, Q_{13}^B, Q_{12}^B) + (m_{14} + 2^{15}\Delta_F) + k_{14}\right]_{>>>15}$$

Exactly like before, we can write $\Phi(Q_{14}^B, Q_{13}^B, Q_{12}^B)$ as $\Phi(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}}$ obtaining:

$$Q_{15}^F = Q_{14}^B + \left[Q_{11}^B + \Phi(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}} + (m_{14} + 2^{15}\Delta_F) + k_{14}\right]_{>>>15} =$$

$$= Q_{14}^B + \left[Q_{11} + \Delta_{11}^B + \Phi(Q_{14}, Q_{13}, Q_{12}) + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}} + (m_{14} + 2^{15}\Delta_F) + k_{14}\right]_{>>>15} =$$

$$= Q_{14}^B + \left[\Delta_{11}^B + \Delta_{B1}^{\Phi_{14}} + \Delta_{B2}^{\Phi_{14}} + 2^{15}\Delta_F\right]_{>>>15} =$$

$$= Q_{14}^B + \left[2^7\Delta_{B1} + 2^{15}\Delta_{B2} + 2^{15}\Delta_F\right]_{>>>15} =$$

Where the first equation is the `MD5` step, the second equation is true by definition, the third one is true thanks to (A.4) and the fourth one is obtained using equations (A.10) and (A.14). Now, for what concerns $2^7\Delta_{B1} + 2^{15}\Delta_{B2} + 2^{15}\Delta_F$, the first term affects only bits 14-7 whereas the last two affects bits 31-15, thus they do not interact with carries. Assuming now that $(2^{15}\Delta_{B2} + 2^{15}\Delta_F) < 2^{32}$ we can write:

$$Q_{15}^F = Q_{14}^B + \left[2^7\Delta_{B1}\right]_{>>>15} + \Delta_{B2} + \Delta_F$$

To sum up, Δ_{15}^F is correct whenever $\Delta_{B2} + \Delta_F < 2^{17}$

A.1.6.2 Step 15

Backward Trail 1

$$Q_{12}^{B1} = \left[Q_{16} - Q_{15}^F\right]_{>>>22} - \Phi_{15}(Q_{15}^F, Q_{14}^B, Q_{13}) - m_{15} - k_{15} =$$

$$= \left[Q_{16} - Q_{15}^F\right]_{>>>22} - (\Phi_{15}(Q_{15}^F, Q_{14}^{B2}, Q_{13}) + \Delta_{\Phi_{15}}^{B1} - m_{15}) - k_{15}$$

Where the first equation is the `MD5` inverse step and the second one is true since $Q_{14}^B[31, 24]$ is affected exclusively by `BT1` and $Q_{15}^F[31, 24]$ and $Q_{13}[31, 24]$ are constant. Hence $\Delta_{\Phi_{15}}^{B1}$ can be computed independently of the other trails.

Backward Trail 2 & Forward Trail

$$\Phi_{15}(Q_{15}^F, Q_{14}^{B2}, Q_{13})[17, 0] = \Phi_{15}(Q_{15}, Q_{14}, Q_{13})[17, 0]$$

If the equation above is verified, then trails do not interact and the predicted differences are correct. Since $Q_{14}[17-9]$ and $Q_{13}[17-9]$ are not affected by any trail and are equal to each other, we always have $\Phi(Q_{15}^F, Q_{14}^{B2}, Q_{13})[17,9] = \Phi(Q_{15}, Q_{14}, Q_{13})[17,9]$. For the remaining bits, as explained in Section 5.3.2, given 2^9 possible configuration for $Q_{15}^F[8,0]$ and 2^9 possible configurations for $Q_{14}^{B2}[8,0]$ we have absorption for

$$\sum_{i=0}^{9} \binom{n}{i} 2^i = 3^9$$

couples of configurations. This number, however, does not consider that some couples might fail because of step 14. The question is then: among all the couples of differences that would be absorbed, how many do not verify the conditions necessary at step 14?

Observation 1 *For a couple* (Δ_F, Δ_B) *to fail at step* 14 *we must have* $\Delta_{B2} + \Delta_F \geq 2^{17}$ *and since* $\Delta_{B2} < 2^9$*, it must be* $\Delta_F[16,9] = 11111111$.

Observation 2 *When a couple* (Δ_F, Δ_B) *fail at step* 14*, an unwanted carry is generated at position* 17 *of* Q_{15}^F*, on the other hand, this implies that* $Q_{15}^F[8,0]$ *is always correctly predicted.*

Observation 3 $Q_{15}^F[8,0] = (\Delta_F - 1) \mod 2^9$.

Observation 4 $Q_{14}^{B2}[8,0] = \overline{\Delta_{B2}[8,0]}$.

Now let us assume that the first one in the binary representation of Δ_F is at position 3:

$$\Delta_F[8,0] = b_8 b_7 b_6 b_5 b_4 1000$$

and let

$$\Delta_{B2}[8,0] = c_8 c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0$$

Because of Observation 3 we have $Q_{15}^F[8,0] = b_8 b_7 b_6 b_5 b_4 0111$ and Observation 4 tells us that for the difference to be absorbed it must be:

$$b_i = 1 \Rightarrow c_i = 0$$

$$c_0 = c_1 = c_2 = 0$$

on the other hand, for the carry to generate and propagate and it must be

$$c_3 = 1$$

$$b_i = 0 \Rightarrow c_i = 1$$

A generalization of this example shows that for every $\Delta_F > (2^{17} - 2^9)$ there exist only one Δ_{B2} capable of absorbing the difference at step 15 and generating a carry at step 14, this implies that the answer to the previous question is $2^9 - 1$, and the total number of valid couples of differences is:

$$(2^8 3^9 - (2^9 - 1))2^8$$

The last thing to check is that $Q_{16}^F = Q_{15}^F$:

$$Q_{16}^F = Q_{15}^F + \left[Q_{12}^{B1} + \Phi(Q_{15}^F, Q_{14}^B, Q_{13}) + m_{15} + k_{15}\right]_{<<<22} =$$

$$= Q_{15}^F + \left[Q_{12}^{B1} + \Phi(Q_{15}, Q_{14}^{B1}, Q_{13}) + m_{15} + k_{15}\right]_{<<<22} =$$

$$= Q_{15}^F + \left[Q_{12} + \Delta_{B1}^{12} + \Phi(Q_{15}, Q_{14}, Q_{13}) + \Delta_{B1}^{\Phi_{15}} + m_{15} + k_{15}\right]_{<<<22} =$$

$$= Q_{15}^F + [0]_{<<<22}$$

Where the first equation is the `MD5` step, the second equation is true assuming that absorption is successful, the third step is true by definition and the fourth step is true thanks to equations(A.3) and (A.16).

A.1.6.3 Step 16

Backward & Forward Trails

$$Q_{13}^B = \left[Q_{17}^F - Q_{16}^F\right]_{>>>5} - \Phi_{16}(Q_{16}^F, Q_{15}^F, Q_{14}^B) - m_1 - k_{16}$$

$$Q_{17}^F = Q_{16}^F + \left[Q_{13} + \Phi_{16}(Q_{16}^F, Q_{15}^F, Q_{14}^B) + m_1 + k_{16}\right]_{<<<5}$$

Since $Q_{15}^F = Q_{16}^F$ differences are absorbed by Φ_{16}:

$$Q_{13}^B = \left[Q_{17}^F - Q_{16}^F\right]_{>>>5} - \Phi_{16}(Q_{16}^F, Q_{15}^F, Q_{14}) - m_1 - k_{16}$$

$$Q_{17}^F = Q_{16}^F + \left[Q_{13} + \Phi_{16}(Q_{16}^F, Q_{15}^F, Q_{14}) + m_1 + k_{16}\right]_{<<<5}$$

hence there is no interaction between trails.

A.1.6.4 Step 17

Backward trails The difference is directly introduced by m_6.

Forward trail

$$Q_{18} = Q_{17}^F + \left[Q_{14}^B + \Phi(Q_{17}^F, Q_{16}^F, Q_{15}^F) + (m_6 + 2^{24}\Delta_{B1} + \Delta_{B2}) + k_{17}\right]_{<<<9} =$$

$$= Q_{17}^F + \left[Q_{14} + \Phi(Q_{17}^F, Q_{16}^F, Q_{15}^F) + m_6 + k_{17}\right]_{<<<9}$$

Where the first equation is the `MD5` step and the second one is true thanks to equations A.10 and A.14. So the way differences propagate is independent of the backward trails.

Appendix B

Biclique with Local Collision

B.1 Biclique construction

Here a detailed description of the biclique covering steps 7-12 is given. Each differential trail is first analysed independently, then it is shown that they do not interfere with each other.

B.1.1 Notation

- `F` stands for forward trail, related to free bits $19 - 13$ of m_7.
- `B` stands for backward trails, related to free bits $15 - 9$ of m_{12}.
- Δ_T represents the difference introduced in the trail T, which could be F,B,$B1$ or $B2$.
- $\Delta_T^{m_i}$ represents the difference in message word m_i when a difference Δ_T is introduced.
- Δ_T^n represents the difference at state Q_n when a difference Δ_T is introduced.
- $\Delta_T^{\Phi_n}$ represents the difference at the output of Φ_n when a difference Δ_T is introduced.

- The value of state n when no differences are applied is indicated with Q_n, when those differences are considered, the notations $Q_n + \Delta_T^n$ or Q_n^T are used instead.

B.1.2 States initialization

States $Q_{11}, Q_{12}, Q_{13}, Q_{14}$ and message words m_{14}, m_{15}, m_1, m_6 can be freely chosen provided the following equations be verified:

$$Q_6[9-3] = 1111111 \tag{B.1}$$

$$Q_7[10-3] = 11111111 \tag{B.2}$$

$$Q_8[9-3] = 0000000 \tag{B.3}$$

$$Q_9[9-3] = 0000000 \tag{B.4}$$

$$Q_9[19] = 1 \tag{B.5}$$

$$Q_9[10] = Q_8[10] \tag{B.6}$$

$$Q_8[12-9] = Q_7[12-9] \tag{B.7}$$

$$Q_{10}[10] = 0 \tag{B.8}$$

$$Q_{10}[19-12] = 00000000 \tag{B.9}$$

$$Q_{11}[7] = 0 \tag{B.10}$$

$$Q_{11}[19-12] = 11111111 \tag{B.11}$$

$$(Q_4 + \Phi_7(Q_7, Q_6, Q_5) + m_7 + k_7)[31] = 0 \tag{B.12}$$

$$(Q_7 + \Phi_{10}(Q_{10}, Q_9, Q_8) + m_{10} + k_{10})[31] = 0 \tag{B.13}$$

$$(Q_9 - Q_8)[31] = 0 \tag{B.14}$$

$$(Q_{10} - Q_9)[31] = 0 \tag{B.15}$$

B.1.3 Forward trail

B.1.3.1 Step 7

$$\begin{aligned} Q_8^F &= Q_7 + \left[Q_4 + \Phi_7(Q_7, Q_6, Q_5) + m_7 + 2^{13}\Delta_F + k_7\right]_{<<<22} = \\ &= Q_7 + [Q_4 + \Phi_7(Q_7, Q_6, Q_5) + m_7 + k_7]_{>>>10} + 2^3\Delta_F \end{aligned}$$

Where the first equation is the `MD5` step and $\Delta_F \in \{0, 1, 2, \cdots, 2^7 - 1\}$, the second equation is true thanks to (B.12), so any carry generated by $2^{13}\Delta_F$ cannot propagate past bit position 31 previous to the shift. Hence:

$$\Delta_F^8 = 2^3\Delta_F \tag{B.16}$$

Furthermore, thanks to (B.3) only $Q_8[9-3]$ is affected by the difference.

B.1.3.2 Step 8

$$\begin{aligned} Q_9^F &= Q_8^F + \left[Q_5 + \Phi_8(Q_8^F, Q_7, Q_6) + m_8 + k_8\right]_{<<<7} = \\ &= Q_8^F + [Q_5 + \Phi_8(Q_8, Q_7, Q_6) + m_8 + k_8]_{<<<7} = \\ &= Q_8 + \Delta_F^8 + [Q_5 + \Phi_8(Q_8, Q_7, Q_6) + m_8 + k_8]_{<<<7} \end{aligned}$$

Where the first equation is the MD5 step, the second equation is true thanks to (B.1),(B.2) and the absorption properties of Φ_8. The last one is true by definition. So we have:

$$\Delta_F^9 = \Delta_F^8 = 2^3\Delta_F \tag{B.17}$$

Furthermore, thanks to (B.4) and (B.6) only $Q_9[9-3]$ is affected by the difference and $Q_8^F[10-3] = Q_9^F[10-3]$.

B.1.3.3 Step 9

$$\begin{aligned} Q_{10}^F &= Q_9^F + \left[Q_6 + \Phi_9(Q_9^F, Q_8^F, Q_7) + m_9 + k_9\right]_{<<<12} = \\ &= Q_9^F + \left[Q_6 + \Phi_9(Q_9, Q_8, Q_7) + m_9 + k_9\right]_{<<<12} = \\ &= Q_9 + \Delta_F^9 + \left[Q_6 + \Phi_9(Q_9, Q_8, Q_7) + m_9 + k_9\right]_{<<<12} \end{aligned}$$

Where the first equation is the MD5 step, the second equation is true thanks to the the fact that $Q_8^F[10-3] = Q_9^F[10-3]$, (B.2) and the absorption properties of Φ_9. The third equation is true by deifnition, hence

$$\Delta_F^{10} = \Delta_F^9 = 2^3\Delta_F \tag{B.18}$$

Furthermore, thanks to (B.8) only $Q_{10}[10,3]$ is affected by the difference.

B.1.3.4 Step 10

$$\begin{aligned} Q_{11}^F &= Q_{10}^F + \left[Q_7 + \Phi_{10}(Q_{10}^F, Q_9^F, Q_8^F) + m_{10} + k_{10}\right]_{<<<17} = \\ &= Q_{10} + \Delta_F^{10} + \left[Q_7 + \Phi_{10}(Q_{10}^F, Q_9^F, Q_8^F) + m_{10} + k_{10}\right]_{<<<17} = \\ &= Q_{10} + \Delta_F^{10} + \left[Q_7 + \Phi_{10}(Q_{10}, Q_9, Q_8) + 2^3\Delta_F + m_{10} + k_{10}\right]_{<<<17} = \\ &= Q_{10} + \Delta_F^{10} + 2^{20}\Delta_F + \left[Q_7 + \Phi_{10}(Q_{10}, Q_9, Q_8) + m_{10} + k_{10}\right]_{<<<17} = \end{aligned}$$

Where the first equation is the MD5 step, the second one is true by definition, the third one is true thanks to the fact that $Q_9^F[10-3] = Q_8^F[10-3]$, (B.17) and the absorption properties of Φ_{10} and the fourth one is true thanks to (B.13), so any carry generated by $2^3\Delta_F$ cannot propagate past position 31 previous to the shift. Hence:

$$\Delta_F^{11} = 2^3\Delta_F + 2^{20}\Delta_F \tag{B.19}$$

Furthermore, thanks to (B.10) $Q_{11}[19, 11]$ is not affected by the difference.

B.1.3.5 Step 11

$$\begin{aligned}
Q_{12}^F &= Q_{11}^F + \left[Q_8^F + \Phi_{11}(Q_{11}^F, Q_{10}^F, Q_9^F) + m_{11} + k_{11}\right]_{<<<22} = \\
&= Q_{11} + \Delta_F^{11} + \left[Q_8^F + \Phi_{11}(Q_{11}^F, Q_{10}^F, Q_9^F) + m_{11} + k_{11}\right]_{<<<22} = \\
&= Q_{11} + \Delta_F^{11} + \left[Q_8^F + \Phi_{11}(Q_{11}, Q_{10}, Q_9) + \Delta_F^{\Phi_{11}} + m_{11} + k_{11}\right]_{<<<22} = \\
&= Q_{11} + \Delta_F^{11} + \left[Q_8 + 2^3\Delta_F + \Phi_{11}(Q_{11}, Q_{10}, Q_9) + \Delta_F^{\Phi_{11}} + m_{11} + k_{11}\right]_{<<<22}
\end{aligned}$$

Where the first equation is the MD5 step and the second three are true by definition. Knowing whether a carry will be generated by $2^3\Delta_F + \Delta_F^{\Phi_{11}}$ and how far it will propagate before the shift, it is possible to write a formula to predict the difference on Q_{12}

B.1.3.6 Step 12

$$\begin{aligned}
Q_{13}^F &= Q_{12}^F + \left[Q_9^F + \Phi_{12}(Q_{12}^F, Q_{11}^F, Q_{10}^F) + m_{12} + k_{12}\right]_{<<<7} = \\
&= Q_{12} + \Delta_F^{12} + \left[Q_9^F + \Phi_{12}(Q_{12}^F, Q_{11}^F, Q_{10}^F) + m_{12} + k_{12}\right]_{<<<7} = \\
&= Q_{12} + \Delta_F^{12} + \left[Q_9^F + \Phi_{12}(Q_{12}, Q_{11}, Q_{10}) + \Delta_F^{\Phi_{12}} + m_{12} + k_{12}\right]_{<<<7} = \\
&= Q_{12} + \Delta_F^{12} + \left[Q_9 + 2^3\Delta_F + \Phi_{12}(Q_{11}, Q_{10}, Q_9) + \Delta_F^{\Phi_{12}} + m_{12} + k_{12}\right]_{<<<7}
\end{aligned}$$

Where the first equation is the MD5 step and the second three are true by definition. Knowing whether a carry will be generated by $2^3\Delta_F + \Delta_F^{\Phi_{12}}$ and how far it will propagate before the shift, it is possible to write a formula to predict the difference on Q_{13}

B.1.4 Backward Trail

B.1.4.1 Step 12

$$Q_9^B = [Q_{13} - Q_{12}]_{>>>7} - \Phi_{12}(Q_{12}, Q_{11}, Q_{10}) - (m_{12} + 2^{12}\Delta_B) - k_{12}$$

Where the first equation is the MD5 inverse step and $\Delta_B \in \{0, 1, 2, \cdots, 2^7 - 1\}$. Hence:

$$\Delta_B^9 = -2^{12}\Delta_B \tag{B.20}$$

Furthermore, thanks to (B.5) only $Q_9[19, 12]$ is affected by the difference.

B.1.4.2 Step 11

$$\begin{aligned} Q_8^B &= [Q_{12} - Q_{11}]_{>>>22} - \Phi_{11}(Q_{11}, Q_{10}, Q_9^B) - m_{11} - k_{11} = \\ &= [Q_{12} - Q_{11}]_{>>>22} - \Phi_{11}(Q_{11}, Q_{10}, Q_9) - m_{11} - k_{11} \end{aligned}$$

Where the first equation is the MD5 inverse step and the second one is true thanks to (B.11) and the absorption properties of Φ_{11}. Hence:

$$\Delta_B^8 = 0 \tag{B.21}$$

B.1.4.3 Step 10

$$\begin{aligned} Q_7^B &= [Q_{11} - Q_{10}]_{>>>17} - \Phi_{10}(Q_{10}, Q_9^B, Q_8) - m_{10} - k_{10} = \\ &= [Q_{11} - Q_{10}]_{>>>17} - \Phi_{10}(Q_{10}, Q_9, Q_8) - m_{10} - k_{10} \end{aligned}$$

Where the first equation is the MD5 inverse step and the second one is true thanks to (B.9) and the absorption properties of Φ_{10}. Hence:

$$\Delta_B^7 = 0 \tag{B.22}$$

B.1.4.4 Step 9

$$\begin{aligned} Q_6^B &= \left[Q_{10} - Q_9^B\right]_{>>>12} - \Phi_9(Q_9^B, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 = \\ &= \left[Q_{10} - Q_9^B\right]_{>>>12} - \Phi_9(Q_9, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 = \\ &= \left[Q_{10} - (Q_9 + \Delta_B^9)\right]_{>>>12} - \Phi_9(Q_9, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 = \end{aligned}$$

$$= \left[Q_{10} - Q_9 + 2^{12}\Delta_B\right]_{>>>12} - \Phi_9(Q_9, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 =$$
$$= [Q_{10} - Q_9]_{>>>12} + \Delta_B - \Phi_9(Q_9, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9$$

Where the first equation is the MD5 step, the second equation is true thanks to (B.7) and the absorption properties of Φ_9, third and fourth ones are true by definition and the fifth one is true thanks to (B.15), so any carry generated by $2^{12}\Delta_B$ cannot propagate past bit 31 before the shift. Hence, by picking

$$\Delta_B^{m_9} = \Delta_B \tag{B.23}$$

no difference will affect Q_6

$$\Delta_B^6 = 0 \tag{B.24}$$

B.1.4.5 Step 8

$$Q_5^B = \left[Q_9^B - Q_8\right]_{>>>7} - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 =$$
$$= \left[Q_9 + \Delta_B^9 - Q_8\right]_{>>>7} - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 =$$
$$= \left[Q_9 - 2^{12}\Delta_B - Q_8\right]_{>>>7} - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 =$$
$$= [Q_9 - Q_8]_{>>>7} - 2^5\Delta_B - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 =$$

Where the first equation is the MD5 step, second and third ones are true by definition and the fourth one is true thanks to (B.14), so any carry generated by $2^{12}\Delta_B$ cannot propagate past bit 31 before the shift. Hence, by picking

$$\Delta_B^{m_8} = -2^5\Delta_B \tag{B.25}$$

no difference will affect Q_5

$$\Delta_B^5 = 0 \tag{B.26}$$

B.1.4.6 Step 7

$$Q_4 = [Q_8 - Q_7]_{>>>22} - \Phi_7(Q_7, Q_6, Q_5) - m_7 - k_7$$

$$\Delta_B^4 = 0 \tag{B.27}$$

B.1.5 Trails interaction

Here each step is analysed when both trails are active.

B.1.5.1 Step 7

$$Q_8^F = Q_7 + \left[Q_4 + \Phi_7(Q_7, Q_6, Q_5) + m_7 + 2^{13}\Delta_F + k_7\right]_{<<<22}$$

Only the forward trail influences this step.

B.1.5.2 Step 8

Forward Trail

$$\begin{aligned} Q_9^{F,B} &= Q_8^F + \left[Q_5 + \Phi_8(Q_8^F, Q_7, Q_6) + (m_8 + \Delta_B^{m_8}) + k_8\right]_{<<<7} = \\ &= Q_8^F + \left[Q_5 + \Phi_8(Q_8, Q_7, Q_6) + (m_8 + \Delta_B^{m_8}) + k_8\right]_{<<<7} = \\ &= Q_8 + \Delta_F^8 + \left[Q_5 + \Phi_8(Q_8, Q_7, Q_6) + (m_8 + \Delta_B^{m_8}) + k_8\right]_{<<<7} \end{aligned}$$

Where the first equation is the `MD5` step, the second equation is true thanks to (B.1),(B.2) and the absorption properties of Φ_8. The last one is true by definition. Hence the forward trail is not affected by the backward trail.

Backward Trail

$$\begin{aligned} Q_5^B &= \left[Q_9^{F,B} - Q_8^F\right]_{>>>7} - \Phi_8(Q_8^F, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 = \\ &= \left[Q_9^{F,B} - Q_8^F\right]_{>>>7} - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 = \\ &= \left[(Q_9 + \Delta_F^9 + \Delta_B^9) - (Q_8 + \Delta_F^8)\right]_{>>>7} - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 = \\ &= \left[Q_9 + \Delta_B^9 - Q_8\right]_{>>>7} - \Phi_8(Q_8, Q_7, Q_6) - (m_8 + \Delta_B^{m_8}) - k_8 \end{aligned}$$

Where the first equation is the `MD5` step, the second equation is true thanks to (B.1),(B.2) and the absorption properties of Φ_8, the third one is true by definition, and the last one is true thanks to (B.17). Hence the backward trail is not affected by the forward trail.

B.1.5.3 Step 9

Forward Trail

$$Q_{10}^F = Q_9^{F,B} + \left[Q_6 + \Phi_9(Q_9^{F,B}, Q_8^F, Q_7) + (m_9 + \Delta_B^{m_9}) + k_9\right]_{<<<12} =$$
$$= Q_9^{F,B} + \left[Q_6 + \Phi_9(Q_9^B, Q_8, Q_7) + (m_9 + \Delta_B^{m_9})\right]_{<<<12} =$$
$$= Q_9^B + \Delta_F^9 + \left[Q_6 + \Phi_9(Q_9^B, Q_8, Q_7) + (m_9 + \Delta_B^{m_9})\right]_{<<<12}$$

Where the first equation is the MD5 step and the second equation is true thanks to the following facts:

- Forward and backward trails affects different parts of Q_9.
- $Q_9^{F,B}[10-3] = Q_9^F[9-3] = Q_8^{F,B}[9-3]$
- Absorption properties of Φ_9 and (B.2).

The third step is true by definition.

Backward Trail

$$Q_6^B = \left[Q_{10}^F - Q_9^{B,F}\right]_{>>>12} - \Phi_9(Q_9^{B,F}, Q_8^F, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 =$$
$$= \left[Q_{10}^F - Q_9^{B,F}\right]_{>>>12} - \Phi_9(Q_9^B, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 =$$
$$= \left[Q_{10}^F - Q_9^{B,F}\right]_{>>>12} - \Phi_9(Q_9, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9 =$$
$$= \left[Q_{10} - Q_9^B\right]_{>>>12} - \Phi_9(Q_9, Q_8, Q_7) - (m_9 + \Delta_B^{m_9}) - k_9$$

Where the first equation is the MD5 step, the second equation is true thanks to the same reasoning made for the forward trail, the third equation is true thanks to B.7 and the absorption properties of Φ_9 and the last one is true since $\Delta_{10}^F = \Delta_9^F$.

B.1.5.4 Step 10

$$Q_{11}^F = Q_{10}^F + \Big[Q_7 + \Phi_{10}(Q_{10}^F, Q_9^{F,B}, Q_8^F) + m_{10} + k_{10}\Big]_{<<<17} =$$
$$= Q_{10}^F + \big[Q_7 + \Phi_{10}(Q_{10}^F, Q_9^F, Q_8^F) + m_{10} + k_{10}\big]_{<<<17}$$

Where the first equation is the `MD5` inverse step and the second one is true thanks to (B.9) and the cross absorption properties of of Φ_{10}. Hence there is no interaction.

B.1.5.5 Step 11

$$Q_{12}^F = Q_{11}^F + \Big[Q_8 + \Phi_{11}(Q_{11}^F, Q_{10}^F, Q_9^{F,B}) + m_{11} + k_{11}\Big]_{<<<22} =$$
$$= Q_{11}^F + \big[Q_8 + \Phi_{11}(Q_{11}^F, Q_{10}^F, Q_9^F) + m_{11} + k_{11}\big]_{<<<22}$$

Where the first equation is the `MD5` inverse step and the second one is true thanks to (B.11) and the cross absorption properties of of Φ_{11}. Hence there is no interaction.

B.1.5.6 Step 12

$$Q_{13}^F = Q_{12}^F + \Big[Q_9^{F,B} + \Phi_{12}(Q_{12}^F, Q_{11}^F, Q_{10}^F) + m_{12} + 2^{12}\Delta_B + k_{12}\Big]_{<<<22} =$$
$$= Q_{12}^F + \big[Q_9^F + \Delta_B^9 + \Phi_{12}(Q_{12}^F, Q_{11}^F, Q_{10}^F) + m_{12} + 2^{12}\Delta_B + k_{12}\big]_{<<<22} =$$
$$= Q_{12}^F + \big[Q_9^F + \Phi_{12}(Q_{12}^F, Q_{11}^F, Q_{10}^F) + m_{12} + k_{12}\big]_{<<<7}$$

Where the first equation is the `MD5` step, the second one is true by definition and the third one thanks to (B.20). Hence there is no trail interaction.

Appendix C

MD5 Parameters

	IV_{-3}	IV_0	IV_{-1}	IV_{-3}
IV:	0x67452301	0xefcdab89	0x98badcfe	0x10325476

Table C.1: MD5 Initialization Vector

Round 1	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	$\pi(i)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Round 2	Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	$\pi(i)$	1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12
Round 3	Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	$\pi(i)$	5	8	11	14	1	4	7	10	13	0	3	6	9	12	15	2
Round 4	Step	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	$\pi(i)$	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

Table C.2: MD5 Message Expansion

Round 1	Step	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	s_i	7	12	17	22	7	12	17	22	7	12	17	22	7	12	17	22
Round 2	Step	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	s_i	5	9	14	20	5	9	14	20	5	9	14	20	5	9	14	20
Round 3	Step	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	s_i	4	11	16	23	4	11	16	23	4	11	16	23	4	11	16	23
Round 4	Step	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	s_i	6	10	15	21	6	10	15	21	6	10	15	21	6	10	15	21

Table C.3: MD5 Rotation Indices

Round 1	$\Phi_i(X,Y,Z) = \texttt{IF}(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
Round 2	$\Phi_i(X,Y,Z) = \texttt{IF3}(X,Y,Z) = (Z \wedge X) \vee (\neg Z \wedge Y)$
Round 3	$\Phi_i(X,Y,Z) = \texttt{XOR}(X,Y,Z) = X \oplus Y \oplus Z$
Round 4	$\Phi_i(X,Y,Z) = \texttt{ONX}(X,Y,Z) = Y \oplus (X \vee \neg Z)$

Table C.4: MD5 Boolean Functions

$k_0 = \texttt{0xd76aa478}$	$k_1 = \texttt{0xe8c7b756}$	$k_2 = \texttt{0x242070db}$	$k_3 = \texttt{0xc1bdceee}$
$k_4 = \texttt{0xf57c0faf}$	$k_5 = \texttt{0x4787c62a}$	$k_6 = \texttt{0xa8304613}$	$k_7 = \texttt{0xfd469501}$
$k_8 = \texttt{0x698098d8}$	$k_9 = \texttt{0x8b44f7af}$	$k_{10} = \texttt{0xffff5bb1}$	$k_{11} = \texttt{0x895cd7be}$
$k_{12} = \texttt{0x6b901122}$	$k_{13} = \texttt{0xfd987193}$	$k_{14} = \texttt{0xa679438e}$	$k_{15} = \texttt{0x49b40821}$
$k_{16} = \texttt{0xf61e2562}$	$k_{17} = \texttt{0xc040b340}$	$k_{18} = \texttt{0x265e5a51}$	$k_{19} = \texttt{0xe9b6c7aa}$
$k_{20} = \texttt{0xd62f105d}$	$k_{21} = \texttt{0x02441453}$	$k_{22} = \texttt{0xd8a1e681}$	$k_{23} = \texttt{0xe7d3fbc8}$
$k_{24} = \texttt{0x21e1cde6}$	$k_{25} = \texttt{0xc33707d6}$	$k_{26} = \texttt{0xf4d50d87}$	$k_{27} = \texttt{0x455a14ed}$
$k_{28} = \texttt{0xa9e3e905}$	$k_{29} = \texttt{0xfcefa3f8}$	$k_{30} = \texttt{0x676f02d9}$	$k_{31} = \texttt{0x8d2a4c8a}$
$k_{32} = \texttt{0xfffa3942}$	$k_{33} = \texttt{0x8771f681}$	$k_{34} = \texttt{0x6d9d6122}$	$k_{35} = \texttt{0xfde5380c}$
$k_{36} = \texttt{0xa4beea44}$	$k_{37} = \texttt{0x4bdecfa9}$	$k_{38} = \texttt{0xf6bb4b60}$	$k_{39} = \texttt{0xbebfbc70}$
$k_{40} = \texttt{0x289b7ec6}$	$k_{41} = \texttt{0xeaa127fa}$	$k_{42} = \texttt{0xd4ef3085}$	$k_{43} = \texttt{0x04881d05}$
$k_{44} = \texttt{0xd9d4d039}$	$k_{45} = \texttt{0xe6db99e5}$	$k_{46} = \texttt{0x1fa27cf8}$	$k_{47} = \texttt{0xc4ac5665}$
$k_{48} = \texttt{0xf4292244}$	$k_{49} = \texttt{0x432aff97}$	$k_{50} = \texttt{0xab9423a7}$	$k_{51} = \texttt{0xfc93a039}$
$k_{52} = \texttt{0x655b59c3}$	$k_{53} = \texttt{0x8f0ccc92}$	$k_{54} = \texttt{0xffeff47d}$	$k_{55} = \texttt{0x85845dd1}$
$k_{56} = \texttt{0x6fa87e4f}$	$k_{57} = \texttt{0xfe2ce6e0}$	$k_{58} = \texttt{0xa3014314}$	$k_{59} = \texttt{0x4e0811a1}$
$k_{60} = \texttt{0xf7537e82}$	$k_{61} = \texttt{0xbd3af235}$	$k_{62} = \texttt{0x2ad7d2bb}$	$k_{63} = \texttt{0xeb86d391}$

Table C.5: MD5 Step Constants

Bibliography

[Abe10] Masayuki Abe, editor. *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*. Springer, 2010.

[AFS96] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A fast provably secure cryptographic hash function. In *Proceedings of the 2 nd Conference on Object-Oriented Technology and Systems (COOTS'96), Usenix Association*, pages 9–7, 1996.

[AKS09] Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors. *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*. Springer, 2009.

[AS08] Kazumaro Aoki and Yu Sasaki. Preimage attacks on one-block MD4, 63-step MD5 and more. In Avanzi et al. [AKS09], pages 103–119.

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 344–371. Springer, 2011.

[CE85] David Chaum and Jan-Hendrik Evertse. Crytanalysis of des with a reduced number of rounds: Sequences of linear factors in block ciphers. In Hugh C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 1985.

[CJ98] Florent Chabaud and Antoine Joux. Differential collisions in sha-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[CLS06] Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an efficient and provable collision-resistant hash function. In Vaudenay [Vau06], pages 165–182.

[CP88] David Chaum and Wyn L. Price, editors. *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*. Springer, 1988.

[Cra05] Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[Dam87] Ivan Damgård. Collision free hash functions and public key signature schemes. In Chaum and Price [CP88], pages 203–216.

[Dam89] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.

[dBB91] Bert den Boer and Antoon Bosselaers. An attack on the last two rounds of MD4. In Feigenbaum [Fei92], pages 194–203.

[dBB93] Bert den Boer and Antoon Bosselaers. Collisions for the compressin function of MD5. In Helleseth [Hel94], pages 293–304.

[DH77] W. Diffie and M. E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, June 1977.

[Dia05] Eduardo Diaz. Exploiting MD5 collisions in C#. `http://www.codeproject.com/Articles/11643/Exploiting-MD5-collisions-in-C`, 2005.

[DL05] Magnus Daum and Stefan Lucks. The poisoned message attack. `https://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/`, 2005.

[Dob96] Hans Dobbertin. Cryptanalysis of MD5 compress. *Lecture Notes in Computer Science*, 1039:53–69, 1996.

[Fei92] Joan Feigenbaum, editor. *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*. Springer, 1992.

[GB01] Shafi Goldwasser and Mihir Bellare. Lecture notes on cryptography, 2001.

[Gib91] J.K. Gibson. Discrete logarithm hash function that is collision free and one way. *Computers and Digital Techniques, IEE Proceedings E*, 138(6):407 – 410, nov 1991.

[Gir87] Marc Girault. Hash-functions using modulo-n operations. In Chaum and Price [CP88], pages 217–226.

[GLRW10] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle preimage attacks: First results on full Tiger, and improved results on MD4 and SHA-2. In Abe [Abe10], pages 56–75.

[Hel94] Tor Helleseth, editor. *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*. Springer, 1994.

[Jou09] Antoine Joux, editor. *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*. Springer, 2009.

[Kli06] Vlastimil Klima. Tunnels in hash functions: MD5 collisions within a minute. *IACR Cryptology ePrint Archive*, 2006:105, 2006.

[KRS11] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: Attacks on Skein-512 and the SHA-2 family. *IACR Cryptology ePrint Archive*, 2011:286, 2011.

[KS05] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Cramer [Cra05], pages 474–490.

[Leu08] Gaëtan Leurent. MD4 is Not One-Way. In Nyberg [Nyb08], pages 412–428.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Nyberg [Nyb08], pages 54–72.

[LWdW05] Arjen K. Lenstra, Xiaoyun Wang, and Benne de Weger. Colliding x.509 certificates. *IACR Cryptology ePrint Archive*, 2005:67, 2005.

[MH81] Ralph C. Merkle and Martin E. Hellman. On the security of multiple encryption. *Commun. ACM*, 24(7):465–467, 1981.

[MR07] Florian Mendel and Vincent Rijmen. Weaknesses in the has-v compression function. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 335–345. Springer, 2007.

[MvOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Nyb08] Kaisa Nyberg, editor. *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*. Springer, 2008.

[Rob94] Matt Robshaw. On pseudo-collisions in MD5. Technical report, RSA Laboratories, July 1994. Technical Report TR-102, version 1.1.

[SA08] Yu Sasaki and Kazumaro Aoki. Preimage attacks on step-reduced md5. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2008.

[SA09] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Joux [Jou09], pages 134–152.

[SLdW07] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne Weger. Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, pages 55–69, Berlin, Heidelberg, 2009. Springer-Verlag.

[Vau06] Serge Vaudenay, editor. *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*. Springer, 2006.

[WFLY04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. *IACR Cryptology ePrint Archive*, 2004:199, 2004.

[WRG+11] Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling. Improved meet-in-the-middle cryptanalysis of KTANTAN. In *Proceedings of the 16th Australasian conference on Information security and privacy*, ACISP'11, pages 433–438, Berlin, Heidelberg, 2011. Springer-Verlag.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Cramer [Cra05], pages 19–35.