Reconstructing the Cryptanalytic Attack behind the Flame Malware

MSc Thesis *(Afstudeerscriptie)*

written by

Fillinger, Maximilian Johannes
(born March 22nd, 1988 in Wuppertal, Germany)

under the supervision of **Dr. Marc Stevens** and **Dr. Christian Schaffner**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
September 30, 2013	Dr. Alexandru Baltag
	Dr. Marc Stevens
	Dr. Christian Schaffner
	Dr. Benne de Weger
	Prof. Dr. Ronald de Wolf

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

Flame was an advanced malware, used for espionage, which infected computers running a Microsoft Windows operating system. Once a computer in a local network was infected, Flame could spread to the other computers in the network via Windows Update, disguised as a security patch from Microsoft. Windows Update relies on digital signatures to ensure that updates originate from Microsoft. Using an attack on the cryptographic hash function `MD5`, the attackers forged a Microsoft signature for a certificate which allowed them to produce signed updates accepted by Windows Update. Using a technique from [25] dubbed *counter-cryptanalysis*, it was found that the certificate was generated by a *chosen-prefix collision attack* on `MD5`, i.e., an attack that extends two prefixes P and P' with suffixes S and S' such that $P\|S$ and $P'\|S'$ have the same hash value: a *collision.* Although the attack seems to be based on the same principles as published collision attacks, such as the attack by Wang et al. from [29] and the attack by Stevens et al. from [26], it has not been published before.

The hash function `MD5` splits its input into 512-bit blocks. `MD5` processes these blocks with a so-called *compression function* while updating an *intermediate hash value.* The intermediate hash value after the whole input has been processed is the output of `MD5`. The Flame chosen-prefix collision attack, like other attacks of this type, consists of the following steps: It begins with a so-called Birthday Search which extends the chosen prefixes such that the difference in the intermediate hash values has a specific form. Then, a series of *near-collision blocks* is constructed such that after these blocks have been processed, the intermediate hash value difference is 0. Thus, a collision is achieved. The construction of these blocks is based on *differential paths*, systems of equations that specify ways in which input differences in the compression function can propagate. In chosen-prefix attacks, it is necessary to *algorithmically* generate such differential paths.

The goal of this thesis is to work towards a reconstruction of the collision attack that generated the Flame authors' certificate, based on the differential paths of the near-collision blocks in the certificate, which were extracted by Stevens. Our main contribution is an attempt at reconstructing the possible *end-segments* of the differential paths used in the attack. These end-segments allow us to infer the cost of constructing the near-collision blocks. Also, we can infer what intermediate hash value differences the attackers looked for in the Birthday Search. This allows us to give an estimate of the Birthday Search cost as well. We prove that, assuming our reconstruction is correct, the expected cost of the attack was equivalent to *at least* $2^{46.6}$ calls to the `MD5`-compression function. We also show that, using the right parameters, this cost can be achieved. In comparison, the attack by Stevens et al. has an expected cost of $2^{44.55}$ for suitably chosen parameters when we require that, as in the Flame attack, only four near-collision blocks can be used. We argue that it is very likely that the expected cost of the Flame attack exceeds our lower bound. A likely reason for this higher cost is that the attackers optimized the attack for speed on massively parallel architectures, not for theoretical cost. The Birthday Search is more cost-effective to parallelize than the construction of the near-collision blocks and it is possible to *decrease* the cost of constructing near-collision blocks by *increasing* the Birthday Search cost.

Furthermore, we analyzed the *connection step* of the Flame attack: The most expensive part of the differential path construction algorithm is the *connection* of an *upper* and a *lower* differential path. Not all pairs of upper and lower paths can be connected. Compared to the attack by Stevens et al., the number of pairs that have to be examined increases by a factor of 2^{13}.

So-called *tunnels* are an important technique for increasing the speed of the near-collision block construction algorithm. The speed-up that a tunnel offers is determined by its *strength.* As Stevens shows in [25], tunnels were used in the collision attack, but their strengths were not maximized. As an explanation, he proposes that the attackers used tunnels when they were available, but did not actively try to increase their strength. We show that this explanation does not account for the observed tunnel strengths and offer an alternative hypothesis. Unfortunately, this hypothesis is less straightforward and impossible to test conclusively with only one output sample of the attack.

Contents

Chapter 1

Introduction

1.1 Outline

This thesis is about *collision attacks* on the *cryptographic hash function* `MD5`. In particular, we attempt to reconstruct details about a collision attack that enabled spreading a spyware called *Flame* within local networks via Windows Update. Although similar attacks have already been published in the scientific literature, the variant that the Flame authors used was not known previously. In Chapter 1, we first explain what cryptographic hash functions are, what purpose they serve in cryptography, what it means to attack such a function and what practical goals such an attack might achieve. We then give a definition of `MD5` and explain *differential cryptanalysis* which has been the most successful approach for attacking `MD5` and similar hash functions.

In Chapter 2, we describe two attacks on `MD5` from the scientific literature which serve as a starting point for understanding the Flame attack. In Chapter 3, we give some background information about Flame and attempt to reconstruct details about the Flame collision attack.

1.2 Hash Functions in Cryptography

1.2.1 Cryptography

Originally, *cryptography* could be described as the art of inventing and breaking *ciphers*, i.e., methods to encode messages in a way that is supposed to make them hard to decode for everyone except the intended recipient. During the second half of the 20th century, considerable changes took place. Cryptography was transformed from a practical art to a science with strong ties to mathematics and computer science and its scope widened considerably. Nevertheless, the most basic example of a cryptographic task is still secret communication. Suppose that one person, let us call her Alice, wants to send another person, say Bob, a message.[1] The information in the message is sensitive, so Alice and Bob do not want an eavesdropper to read the message. Using a secret piece of information, e.g., a randomly selected string of bits, called a *key*, it is possible for Alice to "scramble" the message in such a way that only someone who knows the key can "unscramble" it again. Thus, if Alice and Bob have agreed on a key, say in a private meeting, they can use public communication

[1]In actual applications, one or both of Alice and Bob may not be a person at all, but, for example, an organization or a computer program.

channels to send private messages. The scenario where the communicating parties share a common key is dealt with in *private-key* or *symmetric* cryptography.

However, with cheap worldwide communication available, two people who want or need to communicate securely will often not be able to establish a key beforehand. *Public-key cryptography*, which was introduced by Whitfield Diffie and Martin Hellman in their 1976 paper [4], helps to overcome this problem. In the public-key setting, Bob has both a *public key* and a *secret key.* As the names imply, the public key is meant to be available to anyone who wants to communicate with Bob while the secret key must not be known to anyone else. The public key allows anyone to encrypt a message so that only Bob can decrypt it, using his secret key. In contrast to the private-key setting, it is impossible to achieve unconditional secrecy. We have to make assumptions about the computing power of adversaries and about the hardness of certain mathematical problems, such as the factoring or discrete logarithm problem.

But there is more to secure communication than just secrecy. Verifying the *authenticity* of messages is important as well. For example, a bank needs to make sure that a transfer from an account was indeed requested by the bank account holder, not by a scammer impersonating him. Also, the bank needs to make sure that the recipient and the amount of money were not altered en route. Therefore, cryptography also encompasses the study of methods to verify the authenticity of messages. Cryptographic primitives for message authentication are called *message authentication codes* (MAC) in the private-key setting and *digital signatures* in the public-key setting. Cryptographic hash functions, which will be described in the next section, are important building blocks for these tools.

Other fields of research in modern cryptography include *zero-knowledge proofs*, *secure multiparty computation* and *quantum cryptography.* Zero-knowledge proofs are protocols by which a *prover* can demonstrate (beyond reasonable doubt) to a *verifier* that he has some piece of information without disclosing that information. Secure multiparty computation deals with protocols by which a group of people who each hold some data can jointly compute a function on their data without disclosing it to the others. These protocols should also allow the honest parties to detect when some other people in the group do not follow the protocol. Quantum cryptography deals with ways to use *quantum effects* for cryptographic purposes. In short, modern cryptography can be described as the study of protocols for communication and computation that are *secure* even in the presence of *adversaries* where the exact definition of security and the assumptions about the adversaries' capabilities may vary depending on the research area and intended application.

There are multiple approaches in cryptography. Each has different strengths and weaknesses and each has its use for practice. The *information-theoretic* approach demands the strongest security guarantees: We do not assume any limits on the computing power of the adversary. This branch of cryptography was founded by Claude Shannon, the inventor of information theory. However, information-theoretic cryptography faces some severe limitations. In particular, it can be proved that it is impossible for public-key schemes to meet its security demands. Furthermore, for (private-key) encryption schemes, the key must have at least as many bits of entropy as the message and each key can only be used once; otherwise, there is no information-theoretic security anymore. An example of an information-theoretically secure encryption scheme is the *one-time pad* which was developed in 1882 by Frank Miller and reinvented and patented by Gilbert Vernam in 1917. Thus, it *predates* modern cryptography, although it was not proven to be secure at the time. The one-time pad encrypts an n-bit message m under an n-bit key k as $C(k, m) = m \oplus k$, the bitwise exclusive-or of m and k. The encrypted message $C(k, m)$ is decrypted by computing $C(k, m) \oplus k = m \oplus k \oplus k = m$.

If the key k is selected uniformly at random then the ciphertext $C(k, m)$ becomes a uniformly distributed random variable that is statistically independent of the message. Thus, an adversary who does not have any information about k can gain no information about the content of message m observing $C(k, m)$. However, an adversary can still learn that *some* message was sent and also how many bits were sent.

The *complexity-theoretic* approach divides algorithms into *efficient* and *non-efficient* ones where the class of efficient algorithms is usually taken to be the class of algorithms whose worst-case running time is polynomial in the input length. In the complexity-theoretic model, we only consider efficient algorithms as relevant, i.e., we require that the cryptographic algorithms are efficient and we only view efficient attacks against them as a threat. Cryptographic algorithms in this model have a security parameter n and we allow them running time polynomial in n. An attack on a scheme must be able to break the scheme with non-negligible probability[2] in time polynomial in n. Even though this approach to cryptography is less demanding than the information-theoretic approach, there are no schemes for which an *unconditional* security proof in the sense of complexity-theoretic cryptography has been found.[3] However, cryptographic schemes in this approach are proven secure under *computational hardness assumptions*, i.e., under the assumption that there exists no efficient algorithm to solve a given computational problem. We say that a scheme reduces to some computational problem if the existence of an efficient attack on the scheme implies the existence of an efficient algorithm that solves the problem. If the problem is a well-studied one for which no efficient algorithms have been found, this gives us confidence that the cryptographic scheme based on it is secure. A disadvantage of this approach is that it is asymptotic and thus a proof of security in this model does not necessarily give us any information about how we have to choose the parameter in order to achieve a desired level of security.

The *system-based* approach is concerned with the cost in time and money required to implement the best known attack on a cryptographic scheme. This can give us relatively precise estimates on the real-world security of our schemes, but progress in computer hardware and cryptanalysis can quickly make them obsolete. Cryptographic schemes designed with this approach in mind are designed to be fast and to avoid known attacks but are often not based on any precise computational assumptions. However, even schemes that reduce to some computational problem need the system-based approach in order to find out how the security parameter should be chosen.

The hash function `MD5` adheres to the system-based approach and hence the complexity of attacks on it has to be analyzed in the system-based context too. To make our complexity estimates independent of the speed of the computer that executes the attack, we specify the complexity of attacks in terms of the cost of the `MD5` compression function (see Section 1.3). To give an example, we might say that the complexity of some attack is equivalent to 2^{16} evaluations of the compression function.

1.2.2 Cryptographic Hash Functions

The purpose of a *cryptographic hash function* is to provide a fixed-length "fingerprint" for files of arbitrary length. More formally, we can define them as follows. Let $\{0,1\}^*$ be the set of all finite bit-strings. Let $\mathcal{K}$ be some finite set. We call a function $H : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^l$ a *keyed*

[2]A function f is called negligible if it is asymptotically smaller than every inverse polynomial, i.e., if $f(n) = O(n^{-c})$ for every positive integer c.

[3]Except for schemes that already are information-theoretically secure.

function and for $k \in \mathcal{K}$, we write $h_k : \{0,1\}^* \to \{0,1\}^l, x \mapsto H(k,x)$. Both l and $\mathcal{K}$ may depend on a security parameter n. For a cryptographic hash function, we require that H can be computed efficiently relative to the input size and the security parameter n and that the following properties hold (according to [20]).

- **Pre-image Resistance**: For $k \in \mathcal{K}$ and $h \in \text{Range}(h_k)$ selected uniformly at random, it is infeasible to find x with $h_k(x) = h$ when given k and h. There are two stronger variants of this property. A keyed function is **everywhere pre-image resistant** if it is infeasible for an adversary to find input x with $h_k(x) = h$ when h is chosen by the adversary and, after that, k is selected uniformly at random. It is called **always pre-image resistant** if it is infeasible to find x with $h_k(x) = h$ when k is chosen by the adversary and then h is selected as in the definition of pre-image resistance.

- **Second Pre-image Resistance**: Given a random $k \in \mathcal{K}$ and a random input y from a large finite subset of the domain (e.g., $\{0,1\}^n$), it is not feasible to find $x \neq y$ such that $h_k(x) = h_k(y)$. Similar to pre-image resistance, we can define stronger versions **everywhere second pre-image resistance** where the adversary may choose y and **always second pre-image resistance** where the adversary may choose k.

- **Collision Resistance**: Given a random $k \in \mathcal{K}$, it is not feasible to find distinct x and y such that $h_k(x) = h_k(y)$.

Here, "infeasible" can be understood in the sense of complexity-theoretic or system-based cryptography. In the former sense, it means that every probabilistic polynomial-time algorithm has only negligible probability to solve the described task. In the latter sense, it means that the cost to run the best known algorithm for solving this problem is prohibitively high. Obviously, the latter is a somewhat vague notion, depending on the current state of computer hardware and cryptanalysis and on what exactly one considers a prohibitively high cost.

Informally, these security definitions mean that it is practically impossible for an adversary to construct an input that has the same hash value as another input. Then, we can also expect that any two pieces of data that we compare in everyday life will not have the same hash value by chance. We can thus assume for all practical purposes that two pieces of data that have the same hash value are identical, even when they are deliberately constructed by a malicious adversary.

Usually, collision resistance is the first security property to be broken. Note that collision resistance is already broken if there is an efficient algorithm for finding *any* collision; we do not require that it produces collisions that are useful for some malicious purpose. One reason for defining it that way is that it is difficult to say whether or not a given collision attack is dangerous to any possible application of the hash function. Another reason is that any collision attack should be seen as a warning sign that more practical ones might exist too. Nevertheless, if a collision attack allows us some control over the collisions that it produces, it is more dangerous.

In practice, most hash functions are not keyed functions, but simply functions (i.e., $|\mathcal{K}| = 1$) and l is a fixed constant. This causes a problem with our definition of *collision resistance* which is known as the *foundations-of-hashing dilemma*. If the domain of a function has a higher cardinality than its range, it is certain that collisions exist. When the hash function is not keyed, then this implies that there also exists a very efficient attack: It simply outputs some values x and y that collide. However, as long as no collisions are actually *known*, such attacks only exist as mathematical objects – we can not implement them. Thus, we consider collision resistance to be broken if an efficient algorithm for generating a collision is known. This is quite vague and hard to formalize; for an attempt, see Phillip Rogaway's paper [19]. Rogaway tries to avoid this problem as follows: When we base the security of a cryptographic protocol Π on the collision-resistance of an (unkeyed) hash function h, we would like to claim that the existence of an efficient adversary breaking Π implies the existence of an efficient algorithm for finding collisions in h. However, such a collision-finding algorithm *always* exists, at least in the mathematical sense. Rogaway modifies this reduction as follows: When we base the security of Π on the collision-resistance of h, we should give an *explicit reduction* that transforms an adversary for Π into an equally efficient collision attack on h. That way, when an attack on Π is known, a collision attack on h is known too. Conversely, as long as there are no known collision attacks on h, there can be no known attack on Π. Rogaway calls this paradigm the *human-ignorance* or *explicit-reduction* approach. Note that Rogaway does not attempt to give a formal definition of collision-resistance for unkeyed hash functions. Rather, he proposes a formal definition of what it means to base the security of a protocol on the collision resistance of a hash function, while still using the intuitive definition of collision resistance.

Since the attacks that we consider in this thesis have been actually carried out, the informal definition is acceptable for our purposes.

Let us look at some applications for hash functions. Cryptographic hash functions can be used to quickly compare data and verify data integrity. Consider the case that two servers store copies of some program, together with the corresponding hash value. Alice downloads the program from one server, but she is concerned that her copy might be corrupted, e.g., due to an error on the server or due to an adversary that got access to the server and replaced the program with malware. She could download another copy from the second server and compare both copies. If they match, she concludes that the copy is not corrupted: An adversary would have had to break into both servers to achieve this and that the exact same error happens on both servers is unlikely.[4] Depending on the amount of data and the speed of the connection to the servers, this procedure might take a long time. With a cryptographic hash function, she could instead compute the hash of her copy and then download the hash from the second server and compare the hashes. This would be much faster and – if the function indeed satisfies the requirements for a cryptographic hash function – just as good.

However, adversaries now have more ways to attack than before. Instead of trying to break into both servers, they could try to attack the hash function instead. Suppose that an adversary somehow creates a corrupted copy that has the same hash value as the correct program. Then, they could fool the procedure described in the previous paragraph. To avoid this attack, we want cryptographic hash functions to be second pre-image resistant.

1.2.3 Digital Signatures and the Hash-And-Sign Paradigm

Digital signatures is another area where hash functions are applied. In digital communication, it is relatively easy to impersonate someone else. E-mail headers can be forged to display a false sender address and an ordinary signature appended to a mail can simply be copied. Without a solution for this problem, online banking and online purchases would be completely insecure. However, *digital signatures* can overcome this difficulty. Suppose that Alice wants to allow other people to verify that some message m was indeed written by her. A digital signature scheme consists of three

[4]Note that if the two copies differ, she does not know which one is correct but only that at least one of them was altered. If copies are stored on three or more servers, this problem can be addressed too.

(randomized) algorithms, `KeyGen`, `Sign` and `Verify`. Using `KeyGen`, Alice generates a pair of bit strings (sk, pk). Here, sk stands for "secret key", so called because it must not be known to anyone but Alice, and pk stands for "public key"; as the name implies, it is meant to be available to anyone. Alice might publish pk on her homepage or in some public listing, similar to a phone book. But note that we have a chicken-and-egg problem here: How can we verify that this public key indeed belongs to Alice and not to someone who is impersonating her? This problem will be discussed in Section 1.2.4, so we will not concern ourselves with solutions here. To sign the message m, she computes $m' = \texttt{Sign}(sk, m)$ and appends m' to m. Computing $\texttt{Verify}(pk, m, m')$ allows to verify whether m' is indeed a valid signature for m generated with key sk. Of course, we require that it is infeasible to create a valid signature without the secret key. We will not give a formal security definition here since it is not our main concern in this thesis. A thorough and formal treatment of digital signatures can be found, for example, in [8, Chapter 12].

Signing large messages in a straightforward way using, for example, the RSA signature scheme is a slow process and the signatures are as long as the messages themselves. A way around this problem is the *hash-and-sign* paradigm. Instead of directly signing m, Alice computes a hash $h(m)$ and signs it instead. Given that both the hash function and the signature scheme are secure, this method is secure as well. Intuitively, it seems that an attacker would have to overcome either the signature scheme or the hash function. If we assume both to be secure, it follows that this is not feasible. For a mathematical proof of this statement and for more information about the hash-and-sign paradigm, we refer again to [8]. However, as in our example in the previous section, a weak hash function opens up another way for an adversary to break the scheme.

1.2.4 Collision Resistance and Chosen-Prefix Attacks

Given our previous examples, it is clear why second pre-image resistance is an important property. We now show why we want a cryptographic hash function to satisfy the stronger requirement of collision resistance. The crucial difference between second pre-image resistance and collision resistance is that in an attack on the second pre-image resistance, the adversary is given y and has to construct $x \neq y$ with $h(x) = h(y)$ while in an attack on the collision resistance, the adversary can construct both x and y simultaneously.

To see why attacks on the collision resistance are relevant, recall the problem of verifying that a public key indeed belongs to Alice. One way this can be solved is by a *certification authority* (CA) that is universally trusted. When Alice generates her public key, she fills in a certificate that includes fields for her name, her public key and some further fields (e.g., validity period) and sends it to the CA. The CA verifies Alice's identity and then signs the certificate. Alice can now put the signed certificate on her homepage or a public listing. If someone else, say Bob, wants to check whether the public key pk indeed belongs to Alice, he can use the public key of the CA (which he trusts) to check whether the signature on the certificate is valid. In practice, the *root certificates* of about 50 CAs are distributed with most web browsers. Thus, these CAs are implicitly (and often unknowingly) trusted by almost every internet user.

If the CA uses the hash-and-sign approach for its signatures and the hash function does not offer collision resistance, an adversary, let us call him Charles, might be able to generate two certificates C_1 and C_2 with the same hash value where the first one contains his real name and the second one contains Alice's name. Charles could then ask the CA to sign certificate C_1. If the name and other identifying information in that certificate is correct, the CA would have no reason to refuse.

However, since $h(C_1) = h(C_2)$, the signature on C_1 is also valid for C_2.[5] Thus, Charles could now impersonate Alice. In general, whenever someone signs a message that is partly or completely determined by someone else, collision attacks become a concern. As another example, consider the case that Alice digitally signs a contract written by Charles, using the hash-and-sign method. With a collision attack on the hash function, Charles might be able to craft a pair of contracts with the same hash value such that Alice would accept the first contract, but not the second. But when Alice signs the first contract, Charles can simply paste her signature to the second one.

Of course, these abuse scenarios require that Charles's collision attack gives him sufficient control over the collision that it produces. An attack that just produces some random-looking collision will not help him. In this thesis, two kinds of collision attacks are described that allow the attacker some control over the result. The first kind is *identical-prefix attacks* which take input P and produce distinct suffixes S and S' such that $P\|S$ and $P\|S'$ collide. The second, more dangerous, kind is *chosen-prefix attacks* which take input (P, P') and produce suffixes (S, S') such that $P\|S$ and $P'\|S'$ collide. A chosen-prefix attack could be useful for Charles's attempt to fool the CA: The certificates might contain some comment field at the end that is not read by the CA. He could make one certificate with his name and one certificate with Alice's name and hide away the suffixes in the comment fields. For practical examples of such applications of chosen-prefix attacks, see Sections 3.1 and 1.4.4.

1.3 The Hash Function MD5

1.3.1 The Merkle-Damgård Construction

The *Merkle-Damgård construction* is a general method for designing cryptographic hash functions. It was developed independently by Ivan Damgård and Ralph Merkle in [1] and [15]. It allows to construct a hash function on domain $\{0,1\}^*$ from a *compression function* $\texttt{Compress} : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ in such a way that a collision attack on the hash function implies a collision attack on the compression function. Thus, the collision resistance of the hash function reduces to the collision resistance of the compression function which is easier to analyze. Given a compression function and an initial value $IV \in \{0,1\}^n$, we construct a hash function as follows: Given input $M \in \{0,1\}^*$, we first append some *unambiguous* padding[6] to M so that its length is a multiple of k; let M_i denote the ith k-bit block of the padded input (with i starting from 0). Then, we compute a sequence of *intermediate hash values* (IHV): We set $IHV_0 = IV, IHV_1 = \texttt{Compress}(IHV_0, M_0), IHV_2 = \texttt{Compress}(IHV_1, M_1), \ldots$ until we reach the end of M. The final IHV is the output of the function. See Figure 1.1 for a visual representation of the Merkle-Damgård construction. A proof that shows how a collision attack on the hash function is transformed to a collision attack on the compression function can be found, for example, in [8, Section 4.6.4].

[5] In practice, CAs fill in fields that contain the validity period and a serial number of the certificate. This makes the task of the adversary more difficult but it remains feasible if the content of the fields is sufficiently predictable and multiple attempts are possible in a short time.

[6] By unambiguous padding, we mean that the padded versions of two distinct messages remain distinct. For example, simply padding the input with zeros allows for trivial collision attacks: Let M be an arbitrary message of length l with $k \nmid l$ and let i be the least integer with $l < i \cdot k$. Then, for any $j \leq i \cdot k - l$, the padded versions of M and $M\|0^j$ will be identical. Hence, these messages have the same hash value. One way to create unambiguous padding is to first append a '1' and then '0's and/or to use the length of the message in the padding, like MD5 does.

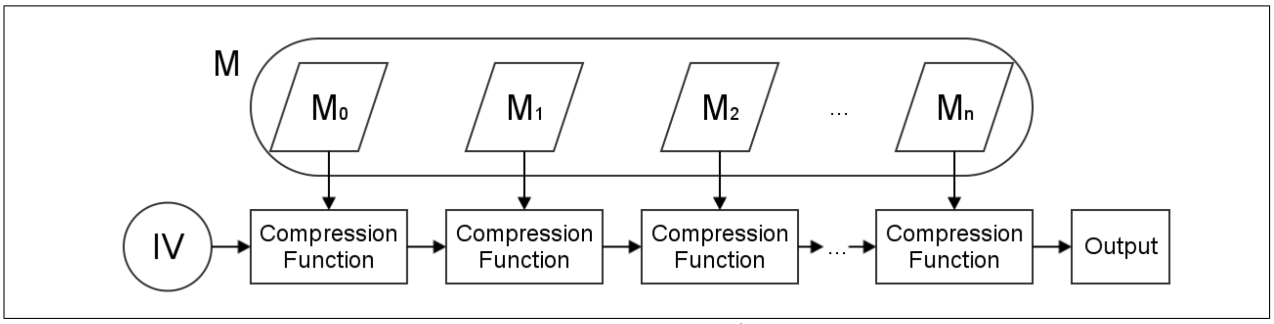

Figure 1.1: The Merkle-Damgård construction

1.3.2 Outline of MD5

The hash function `MD5` was designed by Ronald Rivest in 1991 and published 1992 in [18]. It takes as input a bit string M of arbitrary length and outputs a string consisting of 128 bits. The function follows the Merkle-Damgård-construction, based on a compression function which we will call `MD5Compress`. This function will be described in detail later on. It takes as input a 512-bit message block and an intermediate hash value $IHV_{in} = (a, b, c, d)$ of four 32-bit words and returns as output a tuple IHV_{out} of the same kind. A 32-bit word is a sequence $X = (X[i])_{i=0}^{31}$ of bits. It represents the number $\sum_{i=0}^{31} X[i] \cdot 2^i$ in $\mathbb{Z}_{2^{32}} = \mathbb{Z}/2^{32}\mathbb{Z}$ and we identify the word X with that number. Thus, if X is a 32-bit word, it can occur in arithmetic operations modulo 2^{32}, in bitwise Boolean functions and in bit-rotation functions. For 32-bit words X and Y, we use $X + Y$ and $X \cdot Y$ to denote addition and multiplication modulo 2^{32} and $\wedge$, $\vee$, $\oplus$ and $\overline{\cdot}$ (an overline) to denote the bitwise Boolean and, or, exclusive-or and negation, respectively.

There are two main standards for representing 32-bit words. The most straightforward way is to store a 32-bit word X as the bit string $X[31]X[30] \ldots X[0]$, i.e., ordering the bits from most to least significant. This standard, which is called *big endian*, is how we write 32-bit words in this thesis. A less straightforward standard is *little endian* which is, for example, used in the x86 CPU architecture. The little endian representation can be derived from the big endian representation by reversing the order of the *bytes*, i.e., 8-bit blocks. That is, X is stored as

$$X[7]X[6] \ldots X[0]X[15]X[14] \ldots X[8] \ldots X[31]X[30] \ldots X[24].$$

We will also sometimes write numbers or bit-strings in the hexadecimal system with digits `0`, `1`, . . . , `9`, `a`, `b`, . . . , `f`. We will write hexadecimal numbers with 16 as a subscript. In bit-strings, a hexadecimal digit denotes the 4-bit representation of the digit, sorted from most to least significant.

The computation of `MD5Compress` involves arithmetic modulo 2^{32}, bit-wise Boolean operations and bit-rotation. The intention behind combining all these different kinds of operations is to create complex dependencies between each input and output bit which cannot be easily solved in one unified calculus. Viewing `MD5Compress` as a black box for now, `MD5` looks as follows:

1. First, M is padded by appending a '1' and as many '0's as are necessary to make the length equivalent to $448 \mod 512$. Then, the length of the original message M is appended as a 64-bit little-endian integer. After padding, M has bit length $512 \cdot N$ for some integer N.

2. For $i = 0, \ldots, N - 1$, we let M_i denote the ith 512-bit block of M. We let

$$IV = IHV_0 = (\texttt{67452301}_{16}, \texttt{efcdab89}_{16}, \texttt{98badcfe}_{16}, \texttt{10325476}_{16})$$

which is the bit-string `0123456789abcdeffedcba9876543210`$_{16}$ split up into four 32-bit blocks which are then read as 32-bit words using the little endian standard.[7]

3. For $i = 1, \dots, N$, we let $IHV_i = \texttt{MD5Compress}(IHV_{i-1}, M_{i-1})$.

4. The output of `MD5` is the concatenation of the elements of IHV_N converted back from their little-endian representation.

1.3.3 The Compression Function

Let us now have a look at the inner workings of the compression function. We let B denote the 512-bit message block and $IHV_{in} = (a, b, c, d)$ the tuple that forms the input of `MD5Compress`. We partition B in 16 consecutive 32-bit strings $m_0, \dots, m_{15}$ which we call the *message words*. The compression function is computed in 64 steps numbered 0 to 63; each step updates a and then rotates (a, b, c, d) one place to the right, i.e., d becomes the new a, a becomes the new b and so on. We let $a_0 = a, b_0 = b, c_0 = c, d_0 = d$ and successively compute $(a_1, b_1, c_1, d_1), \dots, (a_{64}, b_{64}, c_{64}, d_{64})$. For $t = 0, \dots, 63$, we have an addition constant AC_t, a rotation constant RC_t, a bitwise Boolean function f_t on three 32-bit words and a 32-bit word $W_t \in \{m_0, \dots, m_{15}\}$. The t-th step function $\texttt{Step}_t$ on input (A, B, C, D) is computed as follows:

1. Compute $F_t = f_t(B, C, D)$ and $T_t = A + F_t + AC_t + W_t$.

2. Set $A' = B + RL(T_t, RC_t)$ where $RL(T_t, n)$ denotes the rotation of T_t n bits to the left. More precisely, we define $RL(T_t, n)$ to be the 32-bit word with $RL(T_t, n)[i] = T_t[i + n \bmod 32]$ for $i = 0, \dots, 31$.

3. Output (D, A', B, C).

For each $t = 0, ..., 63$, we let $(a_{t+1}, b_{t+1}, c_{t+1}, d_{t+1}) = \texttt{Step}_t(a_t, b_t, c_t, d_t)$. The output of `MD5Compress` is obtained by adding IHV_{in} to $(a_{64}, b_{64}, c_{64}, d_{64})$. That is, we let

$$IHV_{out} = (a_0 + a_{64}, b_0 + b_{64}, c_0 + c_{64}, d_0 + d_{64}).$$

To complete our description, it now suffices to list the addition and rotation constants, the Boolean functions and the words denoted by W_t.

The addition constants are $AC_t = \lfloor 2^{32} \cdot |\sin(t+1)| \rfloor$. The rotation constants are given by

$$(RC_t, RC_{t+1}, RC_{t+2}, RC_{t+3}) = \begin{cases} (7, 12, 17, 22) & \text{for } t = 0, 4, 8, 12 \\ (5, 9, 14, 20) & \text{for } t = 16, 20, 24, 28 \\ (4, 11, 16, 23) & \text{for } t = 32, 36, 40, 44 \\ (6, 10, 15, 21) & \text{for } t = 48, 52, 56, 60 \end{cases}$$

and the Boolean functions are

$$f_t(X, Y, Z) = \begin{cases} (X \wedge Y) \oplus (\overline{X} \wedge Z) & \text{for } 0 \le t \le 15 \\ (Z \wedge X) \oplus (\overline{Z} \wedge Y) & \text{for } 16 \le t \le 31 \\ X \oplus Y \oplus Z & \text{for } 32 \le t \le 47 \\ Y \oplus (X \vee \overline{Z}) & \text{for } 48 \le t \le 63 \end{cases}$$

[7]The choice of the initial value is essentially arbitrary. This particular value was chosen to make it obvious that there is nothing special about it, i.e., it is a so-called "nothing-up-my-sleeve number". The choice of the addition constants in `MD5Compress` is motivated similarly.

The words W_t are defined by

$$W_t = \begin{cases} m_t & \text{for } 0 \leq t \leq 15 \\ m_{(1+5t) \mod 16} & \text{for } 16 \leq t \leq 31 \\ m_{(5+3t) \mod 16} & \text{for } 32 \leq r \leq 47 \\ m_{(7t) \mod 16} & \text{for } 48 \leq t \leq 63 \end{cases}$$

We can give an equivalent formulation of `MD5Compress` as follows:

1. Let $Q_{-3} = a$, $Q_{-2} = d$, $Q_{-1} = c$ and $Q_0 = b$.
2. For $t = 0, \ldots, 63$, compute

$$\begin{aligned} F_t &= f_t(Q_t, Q_{t-1}, Q_{t-2}) \\ T_t &= F_t + Q_{t-3} + AC_t + W_t \\ R_t &= RL(T_t, RC_t) \\ Q_{t+1} &= Q_t + R_t \end{aligned}$$

3. Output $IHV_{out} = (Q_{61} + a, Q_{64} + b, Q_{63} + c, Q_{62} + d)$.

The Q_t are called *working states*. They, and the other intermediate variables in the algorithm above play an important role in the attacks on `MD5`.

1.3.4 Inverting the MD5 Compression Function

Pre-image resistance requires that it is infeasible to find for a given intermediate hash value IHV_{out} an intermediate hash value IHV_{in} together with a message block B such that $IHV_{out} =$ `MD5Compress`(IHV_{in}, B). There are still no practical attacks against the pre-image resistance of `MD5`, but it is possible to invert `MD5Compress` in a different sense that is important not for pre-image attacks but for collision attacks. For fixed IHV_{in}, there is an easily computable one-to-one correspondence between the message block B and the working states $Q_1, \ldots, Q_{16}$ in the computation of `MD5Compress`(IHV_{in}, B). This correspondence allows us to *choose* the outcomes of the first 16 steps of a computation of `MD5Compress`$(IHV_{in}, \cdot\,)$ and then obtain a message block B that makes these outcomes happen.

Proposition 1.1. *For given $IHV = (Q_{-3}, Q_0, Q_{-1}, Q_{-2})$ and working states $Q_1, \ldots, Q_{16}$, there is exactly one message block $B = m_0 \| \ldots \| m_{15}$ such that these Q_t are the first working states in the computation of* `MD5Compress` *on the input (IHV, B). It is given by the following formula:*

$$m_t = RL(Q_{t+1} - Q_t, 32 - RC_t) - f_t(Q_t, Q_{t-1}, Q_{t-2}) - Q_{t-3} - AC_t \text{ for } t = 0, \ldots, 15.$$

Proof. For $t < 16$, we have $W_t = m_t$. By definition of T_t,

$$W_t = T_t - Q_{t-3} - F_t - AC_t$$

and F_t is defined as $f_t(Q_t, Q_{t-1}, Q_{t-2})$. We have $R_t = RL(T_t, RC_t)$. Since R_t and T_t are 32-bit words, this holds if and only if $T_t = RL(R_t, 32 - RC_t)$. Because $Q_{t+1} = Q_t + R_t$, we have $R_t = Q_{t+1} - Q_t$. Taking all this together gives the formula. □

1.4 Cryptanalysis of MD5

1.4.1 Types of Collision Attacks

In this section, we describe different types of attacks on compression functions and hash functions derived from them using the Merkle-Damgård construction. Let `Compress` be some compression function. Technically, any two distinct inputs (IHV, B) and (IHV', B') to `Compress` with `Compress`$(IHV, B) =$ `Compress`(IHV', B') are a *collision* of the compression function. However, when $IHV \neq IHV'$, the inputs are usually called a *pseudo-collision* while the name *collision* is reserved for colliding inputs with $IHV = IHV'$.

An algorithm that outputs a (pseudo-)collision of `Compress` is called a *(pseudo-)collision* attack. While (pseudo-)collision attacks on `Compress` in general do not lead to collision attacks on a hash function constructed from that compression function, they can undermine it in two ways. Firstly, if there are known (pseudo-)collision attacks on `Compress`, then `Compress` is not collision resistant and we cannot invoke the guarantee of the Merkle-Damgård construction to argue for the collision-resistance of the derived hash function. Secondly, some (pseudo-)collision attacks could play a part in collision attacks on the hash function.

A collision attack on a hash function h is an algorithm that outputs some messages M and M' with $h(M) = h(M')$. However, some types of collision attacks give the attacker some control over the generated collision. In this thesis, we are interested in two types of collision attacks.

Definition 1.2 (Identical-prefix attack)**.** *Let h be a cryptographic hash function. We say that an algorithm is an* identical-prefix collision attack *on h if, given some bit-string P (the prefix) as input, the algorithm outputs $P\|S$ and $P\|S'$ such that $S \neq S'$ and $h(P\|S) = h(P\|S')$. That is, it produces colliding messages with an* identical prefix *chosen by the attacker.*

Definition 1.3 (Chosen-prefix attack)**.** *Let h be a cryptographic hash function. An algorithm is a* chosen-prefix collision attack *if, given input bit-strings P and P', it outputs $P\|S$ and $P'\|S'$ such that $h(P\|S) = h(P'\|S')$. That is, it extends two prefixes* chosen *by the attacker in such a way that a collision results.*

1.4.2 Generic Collision Attacks by Birthday Search

Before looking at more sophisticated collision attacks on `MD5`, let us establish a baseline against which these attacks can be compared. The attack that we are about to describe is generic in the sense that it works on *any* hash function. If n is the number of output bits, the attack requires an expected number of $O(2^{n/2})$ evaluations of the hash function or, if the hash function follows the Merkle-Damgård construction, of the compression function. Thus, when designing a hash function, one must make sure that the number of output bits is large enough to make this attack infeasible.

Birthday Search is a method to search for a collision in a function f where the running time of the search can be estimated using an analogue of the *Birthday Theorem*. Intuitively, it seems unlikely that two people in a small group have the same birthday, but assuming that birthdays are uniformly distributed throughout the year[8] and disregarding leap-years, we already have a chance of over 50% in a group of 23 people. This fact is called the *Birthday Paradox* and it holds because the number of *pairs* grows quadratically with the number of people in the group. More generally,

[8]This is not actually true, but the actual distribution makes a birthday collision even more likely.

if we select elements uniformly at random from a finite set A, we can expect to select roughly $\sqrt{\pi \cdot |A|/2}$ elements until the same element is selected twice. Thus, we can find a collision of a (pseudo-)random function $f : \{0,1\}^k \to \{0,1\}^n$ evaluating f an expected number of $O(2^{n/2})$ times: We first select a random input x_0 and store $(x_0, f(x_0))$ in a list. Then, for $i = 1, 2, 3, \ldots$, we select a random input x_i and check whether the value of $f(x_i)$ is already in our list. If yes, we have found a collision (assuming that x_i does not already occur in the list). Otherwise, we append $(x_i, f(x_i))$ to the list and continue. A problem that this algorithm faces in practice is that maintaining the list consumes a lot of memory. Since Birthday Search also is a building block for the attack by Stevens et al. and also for the Flame attack, we describe a variant of the Birthday Search method developed by van Oorschot and Wiener in [28] which avoids this problem in Section 2.2.3. We will also give a proof for the expected running time in that section.

For MD5, this means that there is an attack that requires an expected number of $\sqrt{\pi} \cdot 2^{63.5} \approx 2^{64.3}$ evaluations of MD5Compress. To find a collision, we can simply perform a Birthday Search on the function $f = \texttt{MD5Compress}(IV, \ \cdot\ )$. This is easily adapted to an identical-prefix attack: We can replace IV with the intermediate hash value after MD5 has processed the prefix. A chosen-prefix attack is possible as well, although its complexity is a bit higher. Appending some padding if necessary, we can assume without loss of generality that the two chosen prefixes P and P' have the same length and that their length is a multiple of 512. Let IHV and IHV' be the intermediate hash values after MD5 has processed P and P', respectively. We define the function $f : \{0,1\}^{512} \to \{0,1\}^{128}$ by

$$f(x) = \begin{cases} \texttt{MD5Compress}(IHV, x) & \text{if } x \text{ is even} \\ \texttt{MD5Compress}\,(IHV', x) & \text{if } x \text{ is odd} \end{cases}$$

We can expect to find a collision of f after $2^{64.3}$ evaluations. But not every collision of f gives us a valid output for a chosen-prefix attack. We get a valid output if and only if one of the colliding inputs is even and the other is odd. We call such collisions *useful*. Assuming that f is pseudo-random, the probability that a collision is useful is $1/2$. In Section 2.2.3, we will see that if p is the probability that a collision is useful, then the expected cost of finding a useful collision is $\sqrt{1/p}$ times higher than the cost of finding an arbitrary collision. Thus, in a chosen-prefix collision attack on MD5 by Birthday Search we can expect to compute MD5Compress $\sqrt{2} \cdot 2^{64.3} = 2^{64.8}$ times.

1.4.3 A Short History of MD5-Cryptanalysis

A weakness in MD5Compress was found already in 1992. In [3], den Boer and Bosselaers present a pseudo-collision attack on MD5Compress that, given an arbitrary $IHV = (a, b, c, d)$, efficiently finds a message block B such that for

$$IHV' = \left(a + 2^{31} \bmod 2^{32}, b + 2^{31} \bmod 2^{32}, c + 2^{31} \bmod 2^{32}, d + 2^{31} \bmod 2^{32}\right)$$

we have $\texttt{MD5Compress}(IHV, B) = \texttt{MD5Compress}(IHV', B)$. A collision attack on MD5Compress was found in 1996 by Dobbertin and presented in [5] and [6]. Because of this attack, Dobbertin advised against the continued use of MD5 in digital signature schemes.

The first collision in MD5 was found by Wang, Feng, Lai and Yu and presented 2004 in the CRYPTO rump session. They generated the collision with an identical-prefix attack, published in their 2005 paper [29]. It has an expected running time equivalent to 2^{39} computations of

`MD5Compress` and takes ca. 15 minutes on an IBM P690. In [7], Hawkes, Paddon and Rose attempted to reconstruct the attack before Wang et al. published it. We will give a short overview of the attack in Section 2.1. On the first look, it might seem impossible to generate any meaningful collision with their attack. To generate each of the two colliding messages, the prefix P is extended with two random-looking 512-bit blocks. One might think that we should always be able to detect random strings that appear inside otherwise meaningful data. However, we must not forget that randomness can be meaningful too, for example as cryptographic keys. Exploiting this fact, Lenstra, Wang and de Weger created two X.509 certificates[9] that are identical except that they contain different public keys (see [13] and [12]). It is also possible to create a pair of programs that have the same hash value but behave very differently by making the program flow dependent on which collision blocks are in the program. As an example, Daum and Lucks created two Postscript[10] files that display very different messages but have the same `MD5`-hash (see [2]). When either of these files is opened in a text editor, the Postscript code can be read directly. Then, *both* messages are revealed and a string of random characters can be seen, indicating that something is strange about that document. However, the documents resulting from the code look innocuous and give no cause to inspect the code.

After the breakthrough by Wang et al., many improved identical-prefix attacks have been found. In [10], Klima describes a modification of the original attack that finds collisions on an ordinary notebook PC in about eight hours; an attack by Yajima and Shimoyama in [31] also takes several hours on a PC. In 2006, Klima discovered an attack that finds collisions in under a minute on a PC, using a technique that he calls *tunnels*, published in [11]. Tunnels are also used in later identical- and chosen-prefix attacks. In [23], Stevens published techniques to speed up the attack by controlling the bit-rotations. An identical-prefix attack by Stevens et al., published in [24, Section 6.4] can find collisions with a theoretical cost of 2^{16} calls to `MD5Compress`. This is currently the lowest cost among published attacks, but in [30], Xie and Feng claim, to have found message block differences that, under a certain cost estimation function, might lead to an attack with a cost of 2^{10}. An attack that they publish in that paper has an expected theoretical cost of $2^{20.96}$ and takes only seconds on a PC.

The first *chosen-prefix* attack on `MD5` was found in 2007 by Stevens, Lenstra and de Weger and published in [26]. It has a cost equivalent to approximately 2^{50} `MD5Compress`-calls. The paper discusses several abuse scenarios for the attack. They show that colliding certificates with different distinguished names can be generated, with the collision-causing blocks again hidden inside the public keys. Also, chosen-prefix attacks allow to create colliding executables or documents that do not rely on using the collision blocks for program flow and thus look less suspicious. The collision-causing blocks can be hidden at the end of the program or inside images in various document formats. In their 2009 paper [27] Stevens, Sotirov, Appelbaum, Lenstra, Molnar, Osvik and de Weger published an improved attack with a theoretical cost of $2^{39.1}$. They used this attack to establish a rogue certification authority. The CA was deliberately made harmless by setting the expiration date of its certificate in the past, but setting the validity period in the present would not have been any more difficult. We describe the collision attack in Section 2.2 of this thesis.

In 2012, the malware *Flame* was discovered and it was found that it could spread via Windows Update. To make this possible, the attackers had to obtain a certificate with code-signing rights,

[9]X.509 is a widespread standard for certificates in the context of public-key cryptography.

[10]Postscript is a programming language for instructing printers and document viewer software to print or display documents.

signed by Microsoft. This certificate was obtained with the help of a chosen-prefix collision attack. A preliminary analysis of the attack was done by Stevens and published in [25]. In Chapter 3, we give more background about Flame and attempt to reconstruct the collision attack.

All these attacks and applications indicate that MD5 should not be relied on when collision attacks are a concern. Hash functions from the SHA-2 family are a safer alternative.

1.4.4 Differential Cryptanalysis

The most successful approach for finding collisions in MD5 and similar hash functions is *differential cryptanalysis* and all the identical- and chosen-prefix attacks mentioned in the previous subsection utilize it. In this approach, it is analysed how differences in the message block and intermediate hash value propagate through the intermediate variables in the computation of the compression function. There are three ways of representing these differences. The first is the bitwise XOR and the second one is the *arithmetic* difference modulo 2^{32}, also called the *modular* difference. The third way is the *binary signed digit representation* (BSDR). What makes the BSDR useful is that, as we will see in Section 1.4.5, it encodes information about both bitwise differences and the modular difference.

Let us begin with an outline of how attacks on MD5 in the framework of differential cryptanalysis operate. Suppose that we have two messages M and M' of length $k \cdot 512$ for some integer k. Let M_i and M_i' be the ith 512-bit block of M and M', respectively, and let $IHV_0 = IHV_0' = IV$,

$$IHV_1 = \texttt{MD5Compress}(IHV_0, M_0), \ldots, IHV_k = \texttt{MD5Compress}(IHV_{k-1}, M_{k-1})$$

and

$$IHV_1' = \texttt{MD5Compress}(IHV_0', M_0'), \ldots, IHV_k' = \texttt{MD5Compress}(IHV_{k-1}', M_{k-1}').$$

We call

$$\delta IHV_k = IHV_k' - IHV_k = (a', b', c', d') - (a, b, c, d) = (a' - a, b' - b', c' - c, d' - d)$$

the intermediate hash value differential. Our goal is to construct 512-bit message blocks M_k, M_k' that cause a certain target value for δIHV_{k+1}. In the cases we are interested in, this target value for δIHV_{k+1} will be part of some collision attack. We construct these blocks with the help of *differential paths.* A differential path can be viewed as a set of differential equations for the working states and other intermediate variables in the computation of MD5Compress. We say that a pair of MD5Compress-inputs $(IHV, B), (IHV', B')$ *solves* a differential path if the differences between the variables in the computations of MD5Compress(IHV', B') and MD5Compress(IHV, B) are the same as the differentials given in the path.

If we have a differential path such that any solutions of the path give rise to our target δIHV_{k+1} and if we have an efficient algorithm to find M_k, M_k' such that $(M_k, IHV_k), (M_k', IHV_k')$ solves the differential path, then we can construct the required message blocks. A more formal definition of differential paths will be given in Section 1.4.6. As an example, let us look at a very brief outline of the identical-prefix collision attack by Wang et al. from [29]. We start from $M = M'$ since the attack is an identical-prefix attack and thus we have $\delta IHV_k = 0$. Wang et al. designed a differential path that can take us from $\delta IHV_k = 0$ to $\delta IHV_{k+1} = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25})$ and another differential path to take us from this value of δIHV_{k+1} back to $\delta IHV_{k+2} = 0$. Using their first differential path, their identical-prefix attack finds M_k, M_k' such that IHV_{k+1} and IHV_{k+1}'

instantiate the differential δIHV_{k+1} given above. When we append M_k and M'_k to M and M' respectively, we introduce a *difference* in the two messages. To complete the identical-prefix attack, we then use the second differential path to compute M_{k+1} and M'_{k+1} such that the difference between IHV_{k+2} and IHV'_{k+2} is 0 again. When we append these new blocks to M and M', the two messages have the same hash value and we have generated an `MD5`-collision.

The algorithms for solving differential paths used by Wang et al. and by Stevens et al. work roughly as follows: Given input IHV_k and IHV'_k and a differential path, we select values for $Q_1, \dots, Q_{16}$ and $Q'_1, \dots, Q'_{16}$ that are compatible with the differential path. From these working states, we derive message blocks M_k and M'_k using Proposition 1.1. We then compute `MD5Compress`(IHV'_k, M'_k) and `MD5Compress`(IHV_k, M_k) and check if these inputs solve the differential path. If yes, we have found appropriate message blocks. If not, we try different values for $Q_1, \dots, Q_{16}$ and $Q'_1, \dots, Q'_{16}$.

This algorithm might seem inefficient: We just take care of the first 16 working states and, as far as the remaining 48 working states are concerned, we just hope for the best. For *most* differential paths it is indeed inefficient, but it is possible to design differential paths such that once the working states $Q_1, \dots, Q_{16}$ and $Q'_1, \dots, Q'_{16}$ are fixed in accordance with the differential path, there is a relatively high probability that the remaining working states will also match the differential path. The key to constructing such differential paths are so-called *trivial differential steps.* Trivial differential steps are special sequences of differentials that occur with a relatively high probability or even with certainty when certain preconditions are met. Appropriately designed differential paths make collision attacks on `MD5` possible that are fast enough to be carried out in practice.

1.4.5 Binary Signed Digit Representation

A BSDR Z is a sequence $Z = (Z[i])_{i=0}^{31}$ in $\{-1, 0, 1\}$. We say that Z is a BSDR of

$$\sigma(Z) = \sum_{i=0}^{31} Z[i] \cdot 2^i \in \mathbb{Z}_{2^{32}}.$$

We let $w(Z)$ be the *weight* of Z, i.e., the number of signed digits that are non-zero. A number $X \in \mathbb{Z}_{2^{32}}$ has no unique BSDR, but there is a normal form: We say that a BSDR is in *non-adjacent form* (NAF), if $Z[i] \neq 0$ only holds if $Z[i-1] = Z[i+i] = 0$. This NAF is still not unique since $2^{31} \equiv -2^{31} \mod 2^{32}$, but we can make it unique by requiring $Z[31] \in \{0, 1\}$. This normal form can be computed as

$$\mathrm{NAF}(X)[i] = (X+Y)[i] - Y[i] \text{ for } Y = 0X[31]X[30]\dots X[1].$$

The NAF of X has minimal weight among the BSDRs of X. We denote the modular difference between two numbers $X, X' \in \mathbb{Z}_{2^{32}}$ as $\delta X = X' - X \mod 2^{32}$ and we let $\Delta X = (X'[i] - X[i])_{i=0}^{31}$ be the BSDR differential. Then ΔX is a BSDR of δX, so ΔX encodes information about the modular differential. Also, if $\Delta X[i] = 0$, we know that $X[i] = X'[i]$, if $\Delta X[i] = 1$, then $X[i] = 0$ and $X'[i] = 1$ and if $\Delta X[i] = -1$, we have $X[i] = 1$ and $X'[i] = 0$.

A convenient shorthand notation for BSDRs with many zeros is to give a list of the non-zero digits and write the (-1)-digits with an overline. Thus, we write the BSDR $(Z[i])_i$ with $Z[i] = 1$ for $i = 1, 17, 21$ and $Z[i] = -1$ for $i = 2, 13, 15$ and $Z[i] = 0$ otherwise as $(21, 17, \overline{15}, \overline{13}, \overline{2}, 1)$.

For a 32-bit word X, we call $w(\mathrm{NAF}(X))$ the *NAF-weight* of X. This is a more appropriate measure for the weight of an arithmetic differential than the ordinary Hamming-weight since the

NAF-weight of δX is a lower bound on the number of bit positions where X and X' with $X' - X = \delta X$ differ. The Hamming-weight might be much larger: Suppose that $\delta X = X' - X = -1 \equiv \sum_{i=0}^{32} 2^i \mod 2^{32}$. Then δX has a Hamming-weight of 32, but X and X' might only differ in a single bit which is reflected in the NAF-weight of 1. Low-weight BSDR-differentials are more likely than higher-weight ones in the sense that if we fix some δX, choose X at random and let $X' = X + \delta X$, lower-weight ΔX occur with higher probability: The probability that the BSDR-differential is ΔX is $2^{-w(\Delta X)}$.

1.4.6 Differential Paths

As said previously, a differential path is a system of differential equations on the working states and intermediate variables of `MD5Compress`. Let

$$(IHV, m_0 \| \dots \| m_{15}) \text{ and } (IHV', m'_0 \| \dots \| m'_{15})$$

be two inputs to `MD5Compress`. We denote the working states and other intermediate variables of the computation on the first input as Q_t, F_t, W_t, etc. as in Section 1.3.3. Their counterparts in the computation on the second input are denoted by Q'_t, F'_t, W'_t, etc. For A, A' a pair of such variables, we define the *arithmetic differential* $\delta A = A' - A \mod 2^{32}$ and the *BSDR differential* ΔA given by $\Delta A[i] = A'[i] - A[i]$.

Definition 1.4 (Differential Path)**.** *A* partial differential path *for steps $t = t_0, t_0+1, \dots, t_1$ consists of the following information:*

- *ΔQ_t for $t = t_0 - 2, \dots, t_1$, δQ_{t_0-3} and δQ_{t_1+1}.*
- *ΔF_t for $t = t_0, \dots, t_1$.*
- *$\delta m_0, \dots, \delta m_{15}$ from which $\delta W_0, \dots, \delta W_{63}$ can be derived.*
- *δT_t and δR_t for $t = t_0, \dots, t_1$.*

We say that `MD5Compress`*-inputs (IHV, B) and (IHV', B') (or the computations of* `MD5Compress` *on these inputs)* solve *steps $t'_0, \dots, t'_1$ of the differential path if $B' - B = (\delta m_0, \dots, \delta m_{15})$ and for $t = t'_0, \dots, t'_1$, the working states and intermediate variables of the computations on (IHV, B) and (IHV', B') solve the differentials given by the path for $t = t'_0, \dots, t'_1$ and, additionally, the working state differentials $\delta Q_{t'_0-3}$, $\Delta Q_{t'_0-2}$, $\Delta Q_{t'_0-1}$ and $\delta Q_{t'_1+1}$. They are said to* solve *the path* up to step *t'_1 if they solve steps $t_0, \dots, t'_1$ and they are said to simply* solve *it if they solve steps $t_0, \dots, t_1$.*

A partial differential path is called valid *if there is a pair of inputs for* `MD5Compress` *that solves it. A partial differential path is called a* full *differential path if $t_0 = 0$ and $t_1 = 63$.*

The reason that the BSDR is given for the differences in Q_t is that they occur in arithmetic and in Boolean operations, so we are interested in both their arithmetic differences and their bitwise differences. For the F_t, BSDRs are given because we want to use *bitconditions* to fix the differentials δF_t. But using bitconditions, it is not possible to fix δF_t without fixing ΔF_t. All variables other than the Q_t only occur in arithmetic operations, except for the bitwise rotations. The rotations are taken care of probabilistically: Given a difference δT_i, there are (at most) four possible values for the arithmetic difference in the rotated value R_i. We can easily calculate these differences and with what probability they happen and choose a high probability rotation for δR_i (see Lemma 1.6). Thus, we only need the arithmetic differences for these variables.

Let us summarize in what ways an input pair that solves a valid differential path up to step $t-1$ can deviate from the path in step t. Suppose we have a valid differential path and inputs (IHV, B) and (IHV', B') that solve the differential path up to step $t-1$. Thus, the value for δQ_t agrees with the differential path. But the differential path specifies a value for ΔQ_t and, unless $\delta Q_t = 0$, there are multiple ΔQ_t that are compatible with δQ_t. Thus, it is possible that the ΔQ_t of our computation deviates from the differential path. The likelihood of this event depends on the weight of ΔQ_t. For an illustration, suppose that we have $\delta Q_t = 2^k$ for some $k < 31$ and that Q_t is uniformly distributed. If $Q_t[k] = 0$, we have $Q'_t[k] = (Q_t + \delta Q_t)[k] = 1$ and $Q'_t[i] = Q_t[i]$ for all $i \neq k$. Hence, $\Delta Q_t = (k)$. The probability of $Q_t[k] = 0$ is $1/2$. If $Q_t[k+1] = 0$ and $Q_t[k] = 1$, which happens with probability $1/4$, we have $\Delta Q_t = (k+1, \overline{k})$. With probability $1/8$, we have $Q_t[k+2] = 0$, $Q_t[k+1] = 1$ and $Q_t[k] = 1$ so that $\Delta Q_t = (k+2, \overline{k+1}, \overline{k})$, and so on. In differential paths for collision attacks, it is therefore generally preferable to have low-weight values for ΔQ_t. The NAF is optimal, but in the attack by Stevens et al. it is necessary to generate a large number of differential paths which is done by varying the values for ΔQ_t.

Suppose now that our computation agrees with the value for ΔQ_t from the differential path. This does not guarantee that it agrees with ΔF_t. Using *bitconditions*, which we will introduce in Section 1.4.8, we can easily see whether Q_{t-2}, Q_{t-1} and Q_t are compatible with ΔF_t just by inspecting them individually. Now assume that our inputs agree with ΔF_t. If our differential path is valid, the value for δT_t in our computation agrees with the path since $\delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t$. In δR_t, our inputs can again deviate from the path since R_t is a rotation of T_t and a given arithmetic input difference in the rotation can result in several different output differences. If our inputs agree with δT_t but not with δR_t from the path, we say that an *incorrect rotation* happened at step t. We will show in Section 1.4.7 that this problem can be managed probabilistically. If our inputs agree with δR_t from the path, it follows that they must also agree with δQ_{t+1} since $\delta Q_{t+1} = \delta Q_t + \delta R_t$. Thus, the rotation was the last obstacle for our inputs to solve one more step of the differential path.

1.4.7 Rotations of Arithmetic Differentials

Let us now summarize the possible effects of rotations on arithmetic differentials. Let X and X' be 32-bit words and $\delta X = X' - X$. One might expect that $RL(X', n) - RL(X, n) = RL(\delta X, n)$, but this equation does not always hold which is due to possible carries when adding δX to X. This equation might fail in *three* different ways, as we will prove shortly:

- The carries introduced by adding δX to X overflow the $(32-n)$-bit boundary. Then, $RL(X', n) - RL(X, n) = RL(\delta X, n) + 1$.
- The carries introduced by adding δX to X overflow the 32-bit boundary. Then, $RL(X', n) - RL(X, n) = RL(\delta X, n) - 2^n$.
- The carries overflow both boundaries. Then, $RL(X', n) - RL(X, n) = RL(\delta X, n) - 2^n + 1$.

Before giving a proof, let us illustrate the first two cases with examples. The third case is essentially just a combination of the first two cases. Suppose we have $\delta X = 2^{25}$ and $n = 5$. We start with a value for X where no boundary is overflown. Suppose that

$$\begin{aligned} X &= \texttt{11000}\|\texttt{000}\ldots\texttt{0} \\ \Rightarrow X' &= \texttt{11000}\|\texttt{010}\ldots\texttt{0} \end{aligned}$$

where the '$\|$' marks the $(32-n)$-bit boundary. Rotating these words by n swaps the part before the '$\|$' with the part after the '$\|$'. Thus,

$$\begin{aligned} RL(X,n) &= \texttt{000}\ldots\texttt{0}\|\texttt{11000} \\ RL(X',n) &= \texttt{010}\ldots\texttt{0}\|\texttt{11000} \end{aligned}$$

and the difference is $RL(X',n) - RL(X,n) = 2^{30} = RL(\delta X, n)$, as expected. Now consider the following value for X.

$$\begin{aligned} X &= \texttt{11000}\|\texttt{110}\ldots\texttt{0} \\ \Rightarrow X' &= \texttt{11001}\|\texttt{000}\ldots\texttt{0} \end{aligned}$$

The rotated words are

$$\begin{aligned} RL(X,n) &= \texttt{110}\ldots\texttt{0}\|\texttt{11000} \\ RL(X',n) &= \texttt{000}\ldots\texttt{0}\|\texttt{11001} \end{aligned}$$

which gives us $RL(X',n) - RL(X,n) = -2^{31} - 2^{30} + 1 \equiv 2^{30} + 1 \mod 2^{32}$. To illustrate the second case, consider $\delta X = 2^{28}$ and

$$\begin{aligned} X &= \texttt{11110}\|\texttt{110}\ldots\texttt{0} \\ \Rightarrow X' &= \texttt{00000}\|\texttt{110}\ldots\texttt{0} \end{aligned}$$

where the '$\|$' again indicates the $(32-n)$-bit boundary. The rotated words are

$$\begin{aligned} RL(X,n) &= \texttt{110}\ldots\texttt{0}\|\texttt{11110} \\ RL(X',n) &= \texttt{110}\ldots\texttt{0}\|\texttt{00000} \end{aligned}$$

which gives $RL(X',n) - RL(X,n) = -2^4 - 2^3 - 2^2 - 2 = -2^5 + 2$ and we have $RL(\delta X, n) = 2^{28+5 \bmod 32} = 2$.

Let us now proceed to the proof.

Definition 1.5. *Let $\delta X \in \mathbb{Z}_{2^{32}}$ be an arithmetic differential and $n \in \{1,\ldots,31\}$. We define the set of* rotated differentials *as*

$$dRL(\delta X, n) = \{RL\,(X + \delta X, n) - RL(X,n) \mid X \in \mathbb{Z}_{2^{32}}\}\,.$$

We can easily compute the elements of $dRL(\delta X, n)$ and with what probability they occur, as the following lemma shows.

Lemma 1.6 ([24, Lemma 5.4])**.** *For every $\delta X \in \mathbb{Z}_{2^{32}}$ and every $n \in \{1,\ldots,31\}$, the set $dRL(\delta X,n)$ contains at most four elements $D_1,\ldots,D_4$. These elements and the probabilities*

$$p_i = Pr_{X\in\mathbb{Z}_{2^{32}}}\left[RL\left(X + \delta X \bmod 2^{32}, n\right) - RL(X,n) = D_i\right]$$

can be computed as follows. Let $X_{low} = \sum_{i=0}^{31-n} \delta X[i] \cdot 2^i$ *and* $X_{high} = \sum_{i=32-n}^{31} \delta X[i] \cdot 2^i$*. Then, the* D_i *and* p_i *are given by the formulas*

i	D_i	p_i
1	$RL(\delta X, n)$	$2^{-64+n} \cdot (2^{32-n} - X_{low}) \cdot (2^{32} - X_{high})$
2	$RL(\delta X, n) - 2^n$	$2^{-64+n} \cdot (2^{32-n} - X_{low}) \cdot X_{high}$
3	$RL(\delta X, n) + 1$	$2^{-64+n} \cdot X_{low} \cdot (2^{32} - X_{high} - 2^{32-n})$
4	$RL(\delta X, n) - 2^n + 1$	$2^{-64+n} \cdot X_{low} \cdot (X_{high} + 2^{32-n})$

Proof. For a BSDR differential ΔX, we can easily determine the BSDR differential after the rotation: This is simply $(\Delta X_{i+n \bmod 32})_{i=0}^{31}$. Two BSDRs ΔX_1 and ΔX_2 of δX give rise to the same rotated differential if and only if

$$\sum_{i=0}^{31-n} \Delta X_1[i] \cdot 2^i = \sum_{i=0}^{31-n} \Delta X_2[i] \cdot 2^i \text{ and } \sum_{i=32-n}^{31} \Delta X_1[i] \cdot 2^i = \sum_{i=32-n}^{31} \Delta X_2[i] \cdot 2^i.$$

We define a *partition* of δX as a pair of integers (α, β) such that $|\alpha| < 2^{31-n}$, $|\beta| < 2^{31}$, $2^{31-n}|\beta$ and $\alpha + \beta \equiv \delta X \mod 2^{32}$. For all $k_1, \dots, k_{31} \in \{-1, 0, 1\}$ such that

$$\alpha = \sum_{i=0}^{31-n} k_i \cdot 2^i \text{ and } \beta = \sum_{i=32-n}^{31} k_i \cdot 2^i,$$

$(k_i)_{i=0}^{31}$ is a BSDR of δX since $\alpha + \beta \equiv \delta X \mod 2^{32}$. Conversely, every BSDR of δX gives rise to a partition of δX and two BSDRs result in the same partition (α, β) if and only if they result in the same rotated differential

$$RL((\alpha, \beta), n) = 2^n \cdot \alpha + 2^{-32+n} \cdot \beta.$$

Let us determine the possible partitions. We define the function $\phi : \mathbb{Z}_{32} \to \{0, \dots 2^{32} - 1\}$ as the function that maps elements of $\mathbb{Z}_{32}$ to their unique representative in the set $\{0, \dots, 2^{32} - 1\}$. For fixed α, it is clear that, since $\alpha + \beta \equiv \delta X \mod 2^{32}$, we must have $\beta \equiv \delta X - \alpha \mod 2^{32}$. Due to the restriction that $|\beta| < 2^{32}$, there are at most two possible values for β: The first one is

$$\beta_0 = (\phi(\delta X) - \alpha) \bmod 2^{32}$$

and the second one is

$$\beta_1 = \beta_0 - 2^{32}$$

which is only available if $\beta_0 \neq 0$.

Since 2^{32-n} divides β, it follows that $\delta X - \alpha \bmod 2^{32}$ is divided by 2^{32-n}. Which values for α with $|\alpha| < 2^{32-n}$ achieve this? We must have

$$\delta X - \alpha \equiv 0 \mod 2^{32-n} \Leftrightarrow \alpha \equiv \delta X \mod 2^{32-n}$$

and $|\alpha| < 2^{32-n}$. There are at most two values for α that satisfy these constraints, namely $\alpha_0 = X_{low}$ and, if $X_{low} \neq 0$, also $\alpha_1 = X_{low} - 2^{32-n}$.

In total, we thus have at most four possibilities:

$$\begin{aligned}\alpha = X_{low}, \beta &= X_{high}\\ \alpha = X_{low}, \beta &= X_{high} - 2^{32}\\ \alpha = X_{low} - 2^{32-n}, \beta &= (X_{high} + 2^{32-n}) \bmod 2^{32}\\ \alpha = X_{low} - 2^{32-n}, \beta &= (X_{high} + 2^{32-n}) \bmod 2^{32} - 2^{32}\end{aligned}$$

These correspond to the rotated differentials $D_1 = RL(\delta X, n)$, $D_2 = RL(\delta X, n) - 2^n$, $D_3 = RL(\delta X, n) + 1$ and $D_4 = RL(\delta X, n) - 2^n + 1$, respectively.

To compute the probabilities of these rotations, let

$$p_{(\alpha,\beta)} = \Pr_X\left[RL((\alpha,\beta),n) = RL(X + \delta X, n) - RL(X,n)\right]$$

and count the $X \in \mathbb{Z}_{2^{32}}$ such that the BSDR $((X+\delta X)[i] - X[i])_i$ induces partition (α, β). This occurs if and only if

$$\alpha = \sum_{i=0}^{31-n} ((X+\alpha+\beta)[i] - X[i]) \cdot 2^i \text{ and } \beta = \sum_{i=32-n}^{32} ((X+\alpha+\beta)[i] - X[i]) \cdot 2^i.$$

Since $2^{32-n}|\beta$, the first equation simplifies to

$$\alpha = \sum_{i=0}^{31-n} ((X+\alpha)[i] - X[i]) \cdot 2^i$$

which implies that $(X+\alpha)[i] = X[i]$ for $i > 31-n$, so the second equation simplifies to

$$\beta = \sum_{i=32-n}^{31} ((X+\beta)[i] - X[i]) \cdot 2^i.$$

These equations hold if and only if

$$0 \le \alpha + \sum_{i=0}^{31-n} X[i] \cdot 2^i < 2^{32-n} \text{ and } 0 \le \beta + \sum_{i=32-n}^{31} X[i] \cdot 2^i < 2^{32}.$$

There are $2^{32-n} - |\alpha|$ possible choices for the first $32-n$ bits of X such that the first equation is satisfied and $2^n - |\beta| \cdot 2^{-32+n}$ possible choices for the remaining bits such that the second equation is satisfied. This gives us

$$\begin{aligned}p_{(\alpha,\beta)} = \frac{2^{32-n} - |\alpha|}{2^{32-n}} \cdot \frac{2^n - |\beta| \cdot 2^{-32+n}}{2^n} &= \frac{2^{32-n} - |\alpha|}{2^{32-n}} \cdot \frac{2^{32} - |\beta|}{2^{32}}\\ &= 2^{-64+n}\left(2^{32-n} - |\alpha|\right)\left(2^{32} - |\beta|\right)\end{aligned}$$

and inserting the four possible values for α and β given before into this formula gives us the formulas for the probabilities of the different rotations. □

1.4.8 Bitconditions

To help in finding message blocks that solve a given path, Wang et al. gave "sufficient conditions" or bitconditions on the bits in the working states. It turned out that these conditions are not actually sufficient: A pair of inputs might satisfy the conditions up to some step but fail to solve the next step of the path, due to possible incorrect rotations. Nevertheless, with high probability the correct rotations happen. We can also use bitconditions to specify a differential path itself. Below, we give tables with symbols for the various bitconditions and their meanings. There are two kinds: *Differential bitconditions* that determine the values of the ΔQ_t and *Boolean function bitconditions* that determine the ΔF_t. The differential bitconditions are summarized in Table 1.1 and the Boolean function bitconditions in Table 1.2.

We write bitconditions on a pair of working states Q_t, Q'_t as a tuple of 31 bitconditions $\mathfrak{q}_t$ where the ith element $\mathfrak{q}_t[i]$ is the condition on the ith bits of Q_t and Q'_t, counted from least to most significant. When listing bitconditions, we write the tuples $\mathfrak{q}_t$ from right to left, i.e., as $\mathfrak{q}_t[31] \ldots \mathfrak{q}_t[0]$, to reflect the significance of the bits.

Symbol	**Condition**	$\Delta Q_t[i]$
`.`	$Q_t[i] = Q'_t[i]$	0
`+`	$Q_t[i] = 0 \wedge Q'_t[i] = 1$	$+1$
`-`	$Q_t[i] = 1 \wedge Q'_t[i] = 0$	-1

Table 1.1: Differential bitconditions

Symbol	**Condition**	**Type**	**Direction**
`.`	$Q_t[i] = Q'_t[i]$	direct	
`0`	$Q_t[i] = Q'_t[i] = 0$	direct	
`1`	$Q_t[i] = Q'_t[i] = 1$	direct	
`^`	$Q_t[i] = Q'_t[i] = Q_{t-1}[i]$	indirect	backwards
`v`	$Q_t[i] = Q'_t[i] = Q_{t+1}[i]$	indirect	forwards
`!`	$Q_t[i] = Q'_t[i] = \overline{Q_{t-1}[i]}$	indirect	backwards
`y`	$Q_t[i] = Q'_t[i] = \overline{Q_{t+1}[i]}$	indirect	forwards
`m`	$Q_t[i] = Q'_t[i] = Q_{t-2}[i]$	indirect	backwards
`w`	$Q_t[i] = Q'_t[i] = Q_{t+2}[i]$	indirect	forwards
`#`	$Q_t[i] = Q'_t[i] = \overline{Q_{t-2}[i]}$	indirect	backwards
`h`	$Q_t[i] = Q'_t[i] = \overline{Q_{t+2}[i]}$	indirect	forwards
`?`	$Q_t[i] = Q'_t[i] \wedge (Q_t[i] = 1 \vee Q_{t-2}[i] = 0)$	indirect	backwards
`q`	$Q_t[i] = Q'_t[i] \wedge (Q_t[i] = 1 \vee Q_{t+2}[i] = 0)$	indirect	forwards

Table 1.2: Boolean function bitconditions

Wang et al. used bitconditions as a tool to check whether computations of `MD5Compress` follow a given differential path. It is easy to see how the differential bitconditions enforce certain values for ΔQ_t: If a pair of working states Q_t, Q'_t satisfies $\mathfrak{q}_t$, then we have

$$\Delta Q_t[i] = Q'_t[i] - Q_t[i] = \begin{cases} 1 & \text{if } \mathfrak{q}_t[i] = \texttt{+} \\ -1 & \text{if } \mathfrak{q}_t[i] = \texttt{-} \\ 0 & \text{otherwise} \end{cases}$$

However, in Section 1.4.9, we will see how bitconditions can also be used as *replacements* for differential paths: All valid differential paths (and some invalid ones) can be completely specified by bitconditions. For every BSDR ΔQ_t, the tuple of differential bitconditions $\mathfrak{q}_t$ specified by

$$\mathfrak{q}_t[i] = \begin{cases} \texttt{+} & \text{if } \Delta Q_t[i] = 1 \\ \texttt{-} & \text{if } \Delta Q_t[i] = -1 \\ \texttt{.} & \text{if } \Delta Q_t[i] = 0 \end{cases} \tag{1.1}$$

is satisfied by working states Q_t, Q'_t if and only if they solve the differential ΔQ_t. Given bitconditions $\mathfrak{q}_t$ and $\mathfrak{q}_{t+1}$ on Q_t, Q'_t and Q_{t+1}, Q'_{t+1} respectively, the values of δQ_t and δQ_{t+1} are fixed. Then, the only value for δR_t that can occur in a valid differential path is $\delta R_t = \delta Q_{t+1} - \delta Q_t$. However, ΔF_t remains a source of ambiguity, as the following example shows:

Suppose we have bitconditions $(\texttt{-}, \texttt{-}, \texttt{.})$ on $Q_t[i], Q_{t-1}[i], Q_{t-2}[i]$ for some $t \leq 15$. Then we have $Q_t[i] = Q_{t-1}[i] = 1$, $Q'_t[i] = Q'_{t-1}[i] = 0$ and $Q_{t-2}[i] = Q'_{t-2}[i] \in \{0, 1\}$. It follows that

$$\Delta F_t[i] = \begin{cases} ((0 \wedge 0) \oplus (\overline{0} \wedge 0)) - ((1 \wedge 1) \oplus (\overline{1} \wedge 0)) = 0 - 1 = -1 & \text{if } Q_{t-2}[i] = Q'_{t-2}[i] = 0 \\ ((0 \wedge 0) \oplus (\overline{0} \wedge 1)) - ((1 \wedge 1) \oplus (\overline{1} \wedge 1)) = 1 - 1 = 0 & \text{if } Q_{t-2}[i] = Q'_{t-2}[i] = 1 \end{cases}$$

so $\Delta F_t[i]$ is undetermined. We can resolve this ambiguity by replacing '`.`'-bitconditions with Boolean function bitconditions. In this case, replacing the `.` by `0` or `1` resolves the ambiguity.

1.4.9 Representing Differential Paths with Bitconditions

In this subsection, we will prove the following theorem which states that every valid differential path can be completely specified in terms of bitconditions. The proof is constructive in the sense that we give an algorithm to extract appropriate bitconditions from the path. Many of the concepts we introduce for this proof will reappear later when we describe the chosen-prefix collision-attack by Stevens et al.

Theorem 1.7 (Bitconditions as Differential Paths)**.** *For each valid differential path on steps $t_0, \ldots, t_1$, there is a sequence of bitconditions $\mathfrak{q}_{t_0-2}, \ldots, \mathfrak{q}_{t_1}$ such that inputs (IHV, B) and (IHV', B') solve the differential path if and only if*

- *Q_{t_0-3} and Q'_{t_0-3} satisfy δQ_{t_0-3} from the differential path.*
- *Q_{t_1+1} and Q'_{t_1+1} satisfy δQ_{t_1+1} from the differential path.*
- *$Q_{t_0-2}, \ldots, Q_{t_1}$ and $Q'_{t_0-2}, \ldots, Q'_{t_1}$ satisfy the bitconditions.*
- *$R'_t - R_t = \delta R_t$ for every $t = t_0, \ldots, t_1$.*

Such bitconditions can be found by an efficient algorithm.

Remark 1.8. *The bitconditions from this theorem together with δQ_{t_0-3}, δQ_{t_1+1} and $\delta m_0, \ldots, \delta m_{15}$ are a complete representation of the differential path. Therefore, we identify the bitconditions together with these arithmetic differentials with the differential path itself. We can also encode δQ_{t_0-3} and δQ_{t_1+1} as bitconditions by computing bitconditions from their NAFs using Equation (1.1).*

Computing the bitconditions is accomplished by first translating all ΔQ_t from the differential path into bitconditions using Equation (1.1) and then replacing '`.`'-bitconditions by Boolean function bitconditions so that the ΔF_t from the path are enforced. To prove that this is possible in a way that does not *exclude* any sequence of (Q_t, Q'_t) which satisfy the path, we need the following definitions and a lemma.

Definition 1.9 (Forward and backward bitconditions)**.** *We say that a bitcondition on a bit in working state Q_t is* forward *if it involves a bit of Q_{t+1} or Q_{t+2}. We call a bitcondition backward if it involves Q_{t-1} or Q_{t-2}. If a bitcondition is neither forward nor backward we say that it is* direct*. The forward bitconditions are `v`, `y`, `w`, `h` and `q`; the backward bitconditions are `^`, `!`, `m`, `#` and `?`.*

Definition 1.10. *For a tuple $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ of bitconditions such that $\mathfrak{a}$ is not forward, $\mathfrak{b}$ is at most one step forward or backward and $\mathfrak{c}$ is not backward, we define the set $U_{\mathfrak{abc}}$ as the set of tuples*

$$\left(x, x', y, y', z, z'\right) = \left(Q_t[b], Q'_t[b], Q_{t-1}[b], Q'_{t-1}[b], Q_{t-2}[b], Q'_{t-2}[b]\right)$$

that satisfy the bitconditions $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$.

Let $\mathcal{L}$ be the set of triples $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ of bitconditions as described above with $U_{\mathfrak{abc}} \neq \emptyset$. We call the elements of $\mathcal{L}$ the local *triples of bitconditions. A triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{L}$* strengthens *another triple $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ if $U_{\mathfrak{def}} \subseteq U_{\mathfrak{abc}}$.*

We define

$$V_{t,\mathfrak{abc}} = \left\{ f_t(x', y', z') - f_t\left(x, y, z\right) \mid \left(x, x', y, y', z, z'\right) \in U_{\mathfrak{abc}} \right\} \subseteq \{-1, 0, 1\}$$

as the set of values for $\Delta F_t[b]$ that are consistent with $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$.

A triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{L}$ is called a solution *for step t, if it fixes ΔF_t, i.e., if $|V_{t,\mathfrak{def}}| = 1$. We let $\mathcal{S}_t$ be the set of solutions for step t. For $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ and $g \in V_{t,\mathfrak{abc}}$, we let $\mathcal{S}_{t,\mathfrak{abc},g}$ be the set of solutions for step t that strenghten $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ and that have $V_{t,\mathfrak{def}} = \{g\}$. Formally, this can be expressed as*

$$\mathcal{S}_{t,\mathfrak{abc},g} = \{(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{S}_t \mid U_{\mathfrak{def}} \subseteq U_{\mathfrak{abc}} \text{ and } V_{t,\mathfrak{def}} = \{g\}\}\,.$$

For any $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ and any $g \in V_{t,\mathfrak{abc}}$, it holds that $\mathcal{S}_{t,\mathfrak{abc},g} \neq \emptyset$ because for any $(x, x', y, y', z, z') \in U_{\mathfrak{abc}}$ with $f_t\left(x', y', z'\right) - f_t\left(x, y, z\right) = g$, we can simply replace `.`-bitconditions by `0` or `1` so that only (x, x', y, y', z, z') satisfies the resulting bitconditions. However, doing this will in most cases exclude many sequences of working states that satisfy the differential path. The bitconditions are designed in such a way that there are $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{S}_{t,\mathfrak{abc},g}$ that pick out *exactly* those elements of $U_{\mathfrak{abc}}$ which give the correct value for $\Delta F_t[b]$.

Lemma 1.11. *For every $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ and every $g \in V_{t,\mathfrak{abc}}$, there are bitconditions $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{S}_{t,\mathfrak{abc},g}$ such that the indirect bitconditions in $\mathfrak{d}, \mathfrak{e}, \mathfrak{f}$ are all forward (all backward) and*

$$U_{\mathfrak{def}} = \left\{ \left(x, x', y, y', z, z'\right) \in U_{\mathfrak{abc}} \mid f_t\left(x', y', z'\right) - f_t\left(x, y, z\right) = g \right\} \tag{1.2}$$

The proof of this lemma is by "brute force". We only give one example here. Let us find $(\mathfrak{d},\mathfrak{e},\mathfrak{f})$ for the Boolean function F of the first round, $g=0$ and all $(\mathfrak{a},\mathfrak{b},\mathfrak{c})\in\mathcal{L}$ consisting only of differential bitconditions. We have

$$\begin{aligned}\Delta F_t = F\left(x',y',z'\right)-F(x,y,z)=0 \Leftrightarrow &\left(x=0\wedge x'=0\wedge z=z'\right)\\ &\vee\left(x=0\wedge x'=1\wedge z=y'\right)\\ &\vee\left(x=1\wedge x'=0\wedge y=z'\right)\\ &\vee\left(x=1\wedge x'=1\wedge y=y'\right)\end{aligned}$$

Suppose first that $\mathfrak{a}=\texttt{.}$, so that $x=x'$. Then, $\mathfrak{b}=\texttt{.}$ or $\mathfrak{c}=\texttt{.}$, for otherwise $\Delta F_t=0$ is not consistent with these bitconditions. If $\mathfrak{b}=\mathfrak{c}=\texttt{.}$, we do not need to add any further bitconditions to fix ΔF_t. If $\mathfrak{b}\neq\texttt{.}$, we have $y\neq y'$ and thus, we have $\Delta F_t=0$ if and only if $x=x'=0$. Thus, we set $(\mathfrak{d},\mathfrak{e},\mathfrak{f})=(\texttt{0},\mathfrak{b},\mathfrak{c})$. If $\mathfrak{c}\neq\texttt{.}$, we have $z\neq z'$, so we have $\Delta F_t=0$ if and only if $x=x'=1$. Thus, we choose $\mathfrak{d}=1$, $\mathfrak{e}=\mathfrak{b}$ and $\mathfrak{f}=\mathfrak{c}$.

Now assume that $\mathfrak{a}=\texttt{+}$, so that $x=0$ and $x'=1$. Then, $\Delta F_t=0$ if and only if $z=y'$. If $\mathfrak{b}=\mathfrak{c}\in\{\texttt{+},\texttt{-}\}$, this condition can not be fullfilled. Suppose that $\mathfrak{b}=\texttt{+}$. If we also have $\mathfrak{c}=\texttt{-}$, we can not impose any additional conditions, so we have $(\mathfrak{d},\mathfrak{e},\mathfrak{f})=(\mathfrak{a},\mathfrak{b},\mathfrak{c})$. Otherwise, we must have $\mathfrak{c}=\texttt{.}$ and we must set $(\mathfrak{d},\mathfrak{e},\mathfrak{f})=(\mathfrak{a},\mathfrak{b},\texttt{1})$ to satisfy $z=y'$. The case $\mathfrak{b}=\texttt{-}$ can be handled analogously. If $\mathfrak{b}=\texttt{.}$ and $\mathfrak{c}=\texttt{+}$, we have to set $(\mathfrak{d},\mathfrak{e},\mathfrak{f})=(\mathfrak{a},\texttt{0},\mathfrak{c})$ and if $\mathfrak{c}=\texttt{-}$, we set $(\mathfrak{d},\mathfrak{e},\mathfrak{f})=(\mathfrak{a},\texttt{1},\mathfrak{c})$. If $\mathfrak{b}=\mathfrak{c}=\texttt{.}$, we must have $y=y'=z=z'$. To enforce this, we need indirect bitconditions. We set

$$(\mathfrak{d},\mathfrak{e},\mathfrak{f})=\begin{cases}(\texttt{+},\texttt{.},\texttt{v}) & \text{if we want only forward bitconditions}\\ (\texttt{+},\texttt{\^{}},\texttt{.}) & \text{if we want only backward bitconditions}\end{cases}$$

This makes sure that $y=y'$, $z=z'$ and $y=z$. Conversely, every tuple $(0,1,y,y',z,z')$ where $y=y'=z=z'$ satisfies these bitconditions and thus $U_{\mathfrak{def}}=U_{\mathfrak{abc}}$. The case $\mathfrak{a}=\texttt{-}$ is treated analogously to $\mathfrak{a}=\texttt{+}$.

If there are multiple $(\mathfrak{d},\mathfrak{e},\mathfrak{f})$ that fullfil Condition (1.2), we prefer direct bitconditions over indirect ones and among the indirect ones, we prefer the "closer" conditions `^v!y` that only reach one position backward/forward over `mw#h?q` that reach two positions backward/forward. We denote the choice of $\mathfrak{d},\mathfrak{e},\mathfrak{f}$ with only forward bitconditions by $FC(t,\mathfrak{abc},g)$ and the choice with backward bitconditions by $BC(t,\mathfrak{abc},g)$. With these functions, which can be implemented using exhaustive search or lookup tables, we are ready to prove the theorem.

Proof (Theorem 1.7). For every step $t\in\{t_0,\dots,t_1\}$, we let $\mathfrak{q}_t$ be the differential bitconditions corresponding to ΔQ_t according to Equation (1.1). Then, a sequence of working states will satisfy the bitconditions if and only if it solves the ΔQ_t from the differential path. For $t=t_0,\dots,t_1$ and $i=0,\dots,31$, we replace bitconditions $(\mathfrak{q}_t[i],\mathfrak{q}_{t-1}[i],\mathfrak{q}_{t-2}[i])$ by $FC(t,\mathfrak{q}_t[i]\mathfrak{q}_{t-1}[i]\mathfrak{q}_{t-2}[i],\Delta F_t[i])$.

We now need to show two things: That FC is indeed defined on every input it receives in this process and that our differential path is satisfied by precisely those working states that occur for inputs which solve the differential path. To this end, we prove the following.

Claim. *For each $t\in\{t_0,\dots,t_1\}$, the bitcondition triple $(\mathfrak{q}_t[i],\mathfrak{q}_{t-1}[i],\mathfrak{q}_{t-2}[i])$ before replacement is local for every i and after the replacement has been carried out, the working state sequences that satisfy the bitconditions up to step t are exactly those that solve $\delta Q_{t_0-3},\Delta Q_{t_0-2},\dots,\Delta Q_t$ and $\Delta F_{t_0},\dots,\Delta F_t$ from the differential path.*

This statement implies Theorem 1.7 because if the computations on (IHV, B) and (IHV', B') satisfy the ΔQ_t and ΔF_t in the differential path and B, B' satisfy the message block differences, then they must satisfy the δT_t of the differential path. The δR_t must be satisfied because $\delta R_t = \sigma(\Delta Q_{t+1}) - \sigma(\Delta Q_t)$.

We prove the claim by induction. For $t = t_0$, the bitcondition triple $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ is local since initially, all bitconditions are direct. Working states $Q_{t_0-3}, \ldots, Q_{t_0}$, $Q'_{t_0-3}, \ldots, Q'_{t_0}$ satisfy the bitconditions up to step t_0 *before replacement* if and only if their differentials agree with those of the differential path. Since the differential path is valid, there must be some working states that satisfy the bitconditions and solve ΔF_{t_0} from the differential path. Thus, $FC(t_0, \mathfrak{q}_{t_0}[i]\mathfrak{q}_{t_0-1}[i]\mathfrak{q}_{t_0-2}[i], \Delta F_t[i])$ is defined. *After replacement*, the working states that satisfy the bitconditions up to step t_0 are exactly those that satisfied them before replacement *and* have the same ΔF_t as the differential path because of Lemma 1.11.

Suppose that our claim holds up to step $t-1$ for some $t > t_0$. We show that this implies that it is also true for step t. The triple of bitconditions $(\mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i], \mathfrak{q}_{t-3}[i])$ after step $t-1$ has been performed must be local and all bitconditions in it must be forward by definition of the function FC. It follows that $\mathfrak{q}_{t-2}[i]$ is direct or one step forward and that $\mathfrak{q}_{t-1}[i]$ is direct. Since $\mathfrak{q}_t[i]$ has not been replaced yet, it is direct, too. This shows that $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ is local. By induction hypothesis, $Q_{t_0-2}, \ldots, Q_t$ and $Q'_{t_0-2}, \ldots, Q'_t$ satisfy the bitconditions up to step t if and only if they solve $\delta Q_{t_0-3}, \Delta Q_{t_0-2}, \ldots, \Delta Q_t$ and $\Delta F_{t_0}, \ldots, \Delta F_{t-1}$ from the differential path. Since the differential path is valid, $FC(t_0, \mathfrak{q}_{t_0}[i]\mathfrak{q}_{t_0-1}[i]\mathfrak{q}_{t_0-2}[i], \Delta F_t[i])$ must be defined and Lemma 1.11 ensures that sequences of working states satisfy the new bitconditions if and only if they solve $\delta Q_{t_0-3}, \Delta Q_{t_0-2}, \ldots, \Delta Q_t$ and $\Delta F_{t_0}, \ldots, \Delta F_t$ from the differential path. $\square$

Remark 1.12. *Instead of replacing the bitconditions in ascending order using FC we can also replace them in descending order using BC. This results in different but equivalent bitconditions. In Section 2.2, we show how to use FC and BC to extend (bitcondition representations of) differential paths forward and backward.*

When the differential path is given by bitconditions, it makes sense to redefine the notion of *partially* solving the differential path.

Definition 1.13. *Suppose that we have a differential path for steps $t_0, \ldots, t_1$ specified by bitconditions. We say that a pair of inputs* solves *steps $t'_0, \ldots, t'_1$ for $t_0 \le t'_0 \le t'_1 \le t_1$ if the working states Q_i, Q'_i for $i = t'_0 - 2, \ldots, t'_1$ satisfy their respective bitconditions and the values for $Q_{t'_0-3}, Q'_{t'_0-3}$ and $Q_{t'_1+1}, Q'_{t'_1+1}$ agree with the values for $\delta Q_{t'_0-3}$ and $\delta Q_{t'_1+1}$ as derived from the path. A pair of inputs* solves *the path up to step t if it solves steps $t_0, \ldots, t$ of the path.*

This new definition differs from Definition 1.4 because the bitconditions $\mathfrak{q}_t$ for $t < t_1$ depend on ΔF_{t+1} and ΔF_{t+2} but in the old definition, ΔF_{t+1} and ΔF_{t+2} play no role in deciding whether a differential path is solved up to step t.

1.4.10 A Basic Algorithm for Solving Differential Paths

A simple algorithm for solving differential paths, given by bitconditions, works as follows: Suppose we are given a differential path and intermediate hash values

$$IHV = (Q_{-3}, Q_0, Q_{-1}, Q_{-2}), IHV' = \left(Q'_{-3}, Q'_0, Q'_{-1}, Q_{-2}\right)$$

that agree with $\mathfrak{q}_{-3}, \dots, \mathfrak{q}_0$ of the differential path. Our goal is to find message blocks B and B' such that (IHV, B) and (IHV', B') solves the differential path.

1. Without loss of generality, suppose the differential path is given by direct and *backward* bitconditions. Randomly select $Q_1, \dots, Q_{16}$ that agree with the bitconditions.

2. Compute $m_0, \dots, m_{15}$ from the selected working states and then compute $Q_{17}, \dots, Q_{64}$. Check if these agree with their respective bitconditions.

3. If they do, let $B = m_0 \| \dots \| m_{15}$, $B' = m_0 + \delta m_0 \| \dots \| m_{15} + \delta m_{15}$, $IHV = (Q_{-3}, Q_0, Q_{-1}, Q_{-2})$ and $IHV' = (Q_{-3} + \delta Q_{-3}, Q_0 + \delta Q_0, Q_{-1} + \delta Q_{-1}, Q_{-2} + \delta Q_{-2})$ and output (IHV, B) and (IHV', B') as an input pair that follows the differential path. If not, start over at step 1.

This algorithm works because for any ΔQ_t, if Q_t agrees with ΔQ_t in the sense that

- $Q_t[i] = 0$ if $\Delta Q_t[i] = 1$, and
- $Q_t[i] = 1$ if $\Delta Q_t[i] = -1$

then for $Q'_t = Q_t + \delta Q_t$, the differential ΔQ_t is solved by Q_t and Q'_t. Thus, if the Q_t agree with the bitconditions then the $Q'_t = Q_t + \delta Q_t$ (where the δQ_t are read off from the bitconditions) also agree with the bitconditions. Since all bitconditions except '`.`' put some constraints on the Q_t, it is generally advantageous for differential paths to have as many '`.`'-bitconditions as possible.[11]

This algorithm only serves as an illustration; it leaves a lot of room for improvement. We can also use so-called *tunnels* or *message modification* to speed up the computation. We will see a more sophisticated algorithm in Section 2.2.9. For an algorithm like that to be effective, we need long sequences of so-called *trivial* differential steps which we will discuss in Section 1.4.11.

In the case of Wang et al.'s identical prefix collision attack, the differential paths were constructed "by hand". This was possible because for identical prefix attacks, the difference in the input intermediate hash value is known. In a chosen-prefix attack, this is not possible because it needs to accommodate a large number of input differences to be useful. The paths need to be constructed algorithmically *during* the attack. We will describe the algorithmic construction for the attack by Stevens et al. in Section 2.2.

1.4.11 Trivial Differential Steps

Trivial differential steps are working state differences that, under certain conditions, replicate themselves in the next step. For `MD5`, such sequences are caused by four occurrences of $\delta Q_t = 0$ in a row or four occurrences of $\delta Q_t = 2^{31}$ in a row. If $\delta Q_t = 0$ occurs four times in a row, the difference $\delta Q_t = 0$ will repeat in all further steps until we hit $\delta W_t \neq 0$. The case where $\delta Q_t = 2^{31}$ occurs four times in a row is somewhat less smooth, but there still is a good probability that the next step has $\delta Q_t = 2^{31}$ as well. The reason why these trivial steps exist is that, for $\delta Q_t \in \{0, 2^{31}\}$, the number of possible values for ΔQ_t and ΔF_t is very small and, in most cases, we have $\delta T_t = 0$ which has only one rotation, $\delta R_t = 0$.

Theorem 1.14 (Trivial differential steps)**.** *The following statements about working state differences in* `MD5Compress` *hold:*

[11] Technically, a '`.`'-bitcondition on bit i of working state t imposes the constraint that $Q_t[i] = Q'_t[i]$, but since we first compute the Q_t and then the appropriate Q'_t, this does not *act* as a constraint in our algorithm.

1. *Suppose that, for some t, $\delta Q_{t-3}, \delta Q_{t-2}, \delta Q_{t-1}, \delta Q_t = 0$ and $\delta W_t = 0$. Then the next working state difference will be $\delta Q_{t+1} = 0$. It follows that if four consecutive working state differences are 0, the working state differences that follow them will be 0 too until a non-zero message word difference δW_i occurs.*
2. *Suppose that for some t with $16 \leq t < 32$, it holds that $\delta Q_{t-3}, \ldots, \delta Q_t = 2^{31}$ and either*
   - *$\Delta Q_t = \Delta Q_{t-1}$ and $\delta W_t = 0$, or*
   - *$\Delta Q_t \neq \Delta Q_{t-1}$ and $\delta W_t = 2^{31}$.*

   Then, the next working state difference will be $\delta Q_{t+1} = 2^{31}$.
3. *Suppose that for t with $32 \leq t < 48$, we have $\delta Q_{t-3}, \delta Q_{t-2}, \delta Q_{t-1}, \delta Q_t = 2^{31}$ and $\delta W_t = 0$. Then, the next working state difference will be $\delta Q_{t+1} = 2^{31}$.*
4. *Suppose that for some t with $48 \leq t < 64$, we have $\delta Q_{t-3}, \ldots, \delta Q_t = 2^{31}$ and either*
   - *$\Delta Q_t = \Delta Q_{t-2}$ and $\delta W_t = 0$, or*
   - *$\Delta Q_t \neq \Delta Q_{t-2}$ and $\delta W_t = 2^{31}$.*

   Then, the next arithmetic working state difference will be $\delta Q_{t+1} = 2^{31}$.

Proof. Item 1 is easy to see: The bitwise Boolean function at step t is evaluated on Q_t, Q_{t-1}, Q_{t-2} and Q'_t, Q'_{t-1}, Q'_{t-2}. Thus, $\delta Q_t = \delta Q_{t-1} = \delta Q_{t-2} = 0$ implies that $F_t = F'_t$, i.e., $\delta F_t = 0$. If $\delta W_t = \delta Q_{t-3} = 0$, then also $\delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t = 0$. The only possible rotation of $\delta T_t = 0$ is $\delta R_t = 0$. Therefore, we get $\delta Q_{t+1} = \delta Q_t + \delta R_t = 0$ for the next working state as long as the previous three working states and the message blocks W_t, W'_t have difference 0.

To prove the remaining statements, we will show again that $\delta T_t = 0$. We will use the fact that there are only two BSDRs for 2^{31}, namely (31) and $(\overline{31})$. This implies that the values of the Boolean functions evaluated on Q_t, Q_{t-1}, Q_{t-2} and on Q'_t, Q'_{t-1}, Q'_{t-2} can differ only in the most significant bit when $\delta Q_{t-2}, \ldots, \delta Q_t = 2^{31}$. Let us now find necessary and sufficient conditions for the most significant bits to be different. Let X, Y and Z denote the most significant bits of Q_t, Q_{t-1} and Q_{t-2}. Since $\Delta Q_{t-2}, \ldots, \Delta Q_t \in \{(31), (\overline{31})\}$, the most significant bits of Q'_t, Q'_{t-1} and Q'_{t-2} are $\overline{X}$, $\overline{Y}$ and $\overline{Z}$, respectively.

For $t \in \{16, \ldots, 31\}$, we then have

$$\Delta F_t[31] = (\overline{Z} \wedge \overline{X}) \oplus (Z \wedge \overline{Y}) - (Z \wedge X) \oplus (\overline{Z} \wedge Y)$$

which is non-zero if and only if $Y = X$. Since the only BSDRs for 2^{31} are (31) and $(\overline{31})$, the most significant bits of Q_t and Q_{t-1} are identical if and only if we have $\Delta Q_t = \Delta Q_{t-1}$. Thus, we have

$$\delta F_t = \begin{cases} 2^{31} & \text{if } \Delta Q_t = \Delta Q_{t-1} \\ 0 & \text{otherwise} \end{cases}$$

This shows that statement 2 holds: If $\Delta Q_t = \Delta Q_{t-1}$ and $\delta W_t = 0$, then $\delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t = 2 \cdot 2^{31} \equiv 0$. If $\Delta Q_t \neq \Delta Q_{t-1}$ and $\delta W_t = 2^{31}$, then we again have $\delta T_t = 2 \cdot 2^{31} \equiv 0$. Thus, $\delta T_t = 0$ holds in either case. As in the proof of item 1., if follows that $\delta Q_{t+1} = \delta Q_t$, but in this case, this means that $\delta Q_{t+1} = 2^{31}$.

For $t \in \{32, \dots, 47\}$, and X, Y and Z as before, we have

$$\Delta F_t[31] = \overline{X} \oplus \overline{Y} \oplus \overline{Z} - X \oplus Y \oplus Z$$

and since $\overline{A \oplus B} = \overline{A} \oplus B = A \oplus \overline{B}$, we have

$$\overline{X} \oplus \overline{Y} \oplus \overline{Z} = \overline{\overline{\overline{X \oplus Y \oplus Z}}} = \overline{X \oplus Y \oplus Z}.$$

It follows that $\Delta F_t[31]$ is always non-zero and hence, $\delta F_t = 2^{31}$. Together with $\delta Q_{t-3} = 2^{31}$ and $\delta W_t = 0$, this gives us $\delta T_t = 0$ again and thus, statement 3 holds.

For $t \in \{48, \dots, 63\}$ and X, Y, Z as before, we have

$$\begin{aligned}\Delta F_t[31] = \overline{Y} \oplus (\overline{X} \vee Z) - Y \oplus (X \vee \overline{Z}) &= \overline{Y \oplus (\overline{X} \vee Z)} - Y \oplus (X \vee \overline{Z}) \\ &= Y \oplus \overline{\overline{X} \vee Z} - Y \oplus (X \vee \overline{Z})\end{aligned}$$

and thus, $\Delta F_t[31]$ is non-zero if and only if

$$(X \vee \overline{Z}) = (\overline{X} \vee Z) \Leftrightarrow X = Z.$$

Analogously to the proof of statement 2, we can show that $\delta T_t = 0$ if either $\Delta Q_t = \Delta Q_{t-2}$ and $\delta W_t = 0$ or $\Delta Q_t \neq \Delta Q_{t-2}$ and $\delta W_t = 2^{31}$. This proves item 4. □

Remark 1.15. *It is possible to prove a statement similar to 2 – 4 for $t \in \{0, \dots, 15\}$, but this is not useful since we* choose *the working states on these steps when we solve a differential path.*

We can express Theorem 1.14 as follows in terms of bitconditions:

Corollary 1.16. *Suppose we have a valid differential path given by message word differentials $\delta m_0, \dots, \delta m_{15}$ and bitconditions $\mathfrak{q}_{t_0}, \dots, \mathfrak{q}_{t_1}$. The following statements hold:*

1. *Suppose that for some t, t', we have*

   $$\mathfrak{q}_{t-3}, \dots, \mathfrak{q}_{t'+1} = \texttt{........ ....}$$

   and $\delta W_t, \dots, \delta W_{t'} = 0$. Then, all input pairs that solve the differential path up to step $t-1$ also solve it up to step t'.

2. *Suppose that for some t with $16 \leq t < 32$, we have*

   $$\mathfrak{q}_{t-3}, \dots, \mathfrak{q}_{t+1} = \texttt{X....... ....}$$

   where `X` can be either `+` or `-` (`X` does not have to stand for the same bitcondition in each $\mathfrak{q}$). Suppose further that $\mathfrak{q}_{t-1}[31] = \mathfrak{q}_t[31]$ and $\delta W_t = 0$ or $\mathfrak{q}_{t-1}[31] \neq \mathfrak{q}_t[31]$ and $\delta W_t = 2^{31}$. Then, all input pairs that satisfy the differential path up to step $t-1$ and, additionally, ΔQ_t also satisfy the differential path up to step t.

3. *Suppose that for some t with $32 \leq t < 48$, we have bitconditions $\mathfrak{q}_{t-3}, \dots, \mathfrak{q}_t$ as in part 2 and $\delta W_t = 0$. Then, all input pairs that satisfy the differential path up to step $t-1$ and also satisfy ΔQ_t solve the path up to step t.*

4. *Suppose that, for t with $48 \leq t < 64$, we have $\mathfrak{q}_{t-3}, \ldots, \mathfrak{q}_t$ as in step 2 and either $\mathfrak{q}_t[31] = \mathfrak{q}_{t-2}[31]$ and $\delta W_t = 0$ or $\mathfrak{q}_t[31] \neq \mathfrak{q}_{t-2}[31]$ and $\delta W_t = 2^{31}$. If an input pair satisfies the path up to step $t-1$ and ΔQ_t, then it solves the path up to step t.*

The fact that a useful differential path must have such trivial differential steps can be exploited to *detect* that a message was constructed by a collision attack even when only one message of the colliding pair is known. Marc Stevens describes such counter-measures in [24, Chapter 8]. These techniques were also used in retrieving the differential path of the Flame collision attack. We will say more about this topic in Section 2.3. Concrete examples of full differential paths, expressed as bitconditions, can be found in Appendix A which lists the differential paths of the Flame collision attack. These differential paths follow the design principles we mentioned in this section: A large number of bitconditions is concentrated on the first 16 working states. They then thin out and are followed by sequences of trivial differential steps so that the differential path can be solved quickly. The final four working states are the “payload” of the differential path and cause a target δIHV to occur.

Note that by displaying the differential paths in terms of bitconditions, we do *not* mean to imply that the attackers used differential paths and bitconditions in the exact same way as we defined them here, although the near-collision blocks were definitely crafted using differential paths of some form. We do not know *how* exactly the near-collision blocks were constructed, but we attempt to reconstruct details of their attack in Chapter 3.

Chapter 2

Attacks on MD5

2.1 The Identical-Prefix Attack by Wang et al.

This section gives a rough outline of the identical prefix attack by Wang et al. in [29]. The input for the attack is a pair of messages M and M' such that, for some k, they both have length $512 \cdot N$ and such that the computations of `MD5` on M and M' have the same intermediate hash value IHV_k. The easiest way to satisfy this condition is to take $M = M'$. The attack algorithm appends 512-bit blocks B_0 and B_1 to M and different blocks B_0' and B_1' to M' such that $\texttt{MD5}(M \| B_0 \| B_1) = \texttt{MD5}(M' \| B_0' \| B_1')$. That is, the algorithm produces a collision of `MD5` where the colliding messages have identical (or already colliding) prefixes, hence the name. Due to the incremental nature of `MD5`, the resulting messages can then be extended with identical suffixes or further instances of the collision attack while maintaining the collision.

The attack is based on two "hand-crafted" differential paths. The path associated with the blocks B_0, B_0' starts with $\delta IHV_k = IHV_k' - IHV_k = (0, 0, 0, 0)$ and ends with

$$\delta IHV_{k+1} = (\delta a, \delta b, \delta c, \delta d) = (2^{31}, 2^{31} + 2^{25}, 2^{31} + 2^{25}, 2^{31} + 2^{25})$$

which is produced by the following differentials in $\delta B_0 = (\delta m_0, \ldots, \delta m_{15})$:

$$\delta m_4 = 2^{31}, \delta m_{11} = 2^{15}, \delta m_{14} = 2^{31} \text{ and } \delta m_i = 0 \text{ for all other } m_i.$$

The path associated with B_1, B_1' begins with δIHV_{k+1} as above and ends with $\delta IHV_{k+2} = (0, 0, 0, 0)$ which is brought about by negating the message block differences from before, i.e.,

$$\delta m_4 = 2^{31}, \delta m_{11} = -2^{15}, \delta m_{14} = 2^{31} \text{ and } \delta m_i = 0 \text{ for all other } m_i.$$

We use an algorithm similar to the one in Section 1.4.9 to find inputs that solve the differential paths and generate the blocks B_0, B_1, B_0', B_1' that cause the messages to collide. Two tricks speed up the algorithm for solving the paths: The first one is *message modification.* If we have a solution of the path up to a certain step, we can use message modification to generate a large number of solutions up to that step. We can then check for each of these solutions whether it solvers the further steps. This is similar to the *tunnels* discussed in Section 2.2.8. The second trick is *early abort.* As soon as we see that a condition is not satisfied and that no message modification is possible to solve this problem, we abort the attempt and start over. On average, the algorithm by Wang et al. can generate the collision blocks with cost equivalent to roughly 2^{39} `MD5Compress`-calls.

2.2 The Chosen-Prefix Attack by Stevens et al.

2.2.1 Outline of the Attack

We begin with a high-level overview of the chosen-prefix attack by Stevens et al. as described in [24]. The input is a pair of prefixes P and P' that we wish to extend with suffixes S and S' such that $\texttt{MD5}(P\|S) = \texttt{MD5}(P'\|S')$. As a first step, we pad the prefixes so that for some integer n, their lengths are both equal to $512 \cdot n - 64 - k$ where $0 \leq k \leq 32$ is a parameter of the algorithm. From now on, we assume without loss of generality that P and P' both already have the appropriate length.

The next step is a *Birthday Search* to find strings of $64 + k$ "Birthday Bits" S_b, S'_b for each message such that the intermediate hash value after processing $P\|S_b$ and $P'\|S'_b$ belongs to a certain *set* of intermediate hash values that we are prepared to eliminate. We then construct sequences of *near-collision blocks* that successively reduce the NAF-weight, i.e., the weight of the non-adjacent forms, of the intermediate hash value differentials until there is no difference left. Constructing each near-collision blocks requires the following steps.

- Construct a full differential path from two partial differential paths: The *upper path* which is dictated by the previous intermediate hash value and the *lower path* which causes a certain change in the intermediate hash value differential. This is achieved by algorithms to
  - extend the upper path forward,
  - the lower path backward, and
  - combine the partial differential paths into one when they meet.
- Solve the differential path to obtain a message block that achieves the target intermediate hash value.

The lower paths were "hand-crafted" by Stevens et al. and consist of a long middle segment that contain many trivial differential steps as described in Section 1.4.11 and an end-segment which causes the required changes in the intermediate hash value.

In the next section, we describe the set of intermediate hash value differentials that the near-collision blocks can eliminate, what changes an individual near-collision block can make, and the *elimination strategy*, i.e., the process of selecting the lower paths for the near-collision blocks. We then describe the Birthday Search, the construction of the full differential paths and the algorithm for solving them.

2.2.2 Elimination Strategy

Our goal in the Birthday Search is to find bit-strings S_b and S'_b of length $64 + k$ such that, for IHV_n the intermediate hash value after processing $P\|S_b$ and IHV'_n the intermediate hash value after processing $P'\|S'_b$, the arithmetic differential $\delta IHV_n = IHV'_n - IHV_n$ is of the form

$$\delta IHV_n = (\delta a, \delta b, \delta c, \delta d) = (0, \delta b, \delta c, \delta c)$$

where δb and δc are such that $\delta b - \delta c \equiv 0 \mod 2^k$. This condition implies that, on average, the NAFs of δb and δc differ in $(32 - k)/3$ positions. In this section, we explain how these differences can be eliminated.

Let us first give some more details about the end-segments of the differential paths from which we construct the near-collision blocks. For each constant $w < 32$, each r with $0 \leq r < 32$ and each tuple $(s_0, \ldots, s_{w'}) \in \{-1, 0, 1\}^{w'+1}$ where $w' = \min(w, 31 - r)$, there is a lower differential path with $\delta Q_{61} = 0$, $\delta Q_{62} = \delta Q_{63} = \pm 2^r$ and $\delta Q_{64} = 2^r + \sum_{\lambda=0} s_\lambda \cdot 2^{r+21+\lambda \bmod 32}$. We explain below how these values are brought about. The following table displays the differentials in the various intermediate variables.

i	δQ_i	δF_i	δW_i	δT_i	RC_i	δR_i
61	0	0	$\pm 2^{r-10 \bmod 32}$	$\pm 2^{r-10 \bmod 32}$	10	$\pm 2^r$
62	$\pm 2^r$	0	0	0	15	0
63	$\pm 2^r$	$\sum_{\lambda=0}^{w'} s_\lambda \cdot 2^{r+\lambda}$	0	δF_i	21	$\sum_{\lambda=0}^{w'} s_\lambda \cdot 2^{r+21+\lambda \bmod 32}$
64	$\pm 2^r + \delta R_{63}$					

These values are brought about as follows: The differential paths have message block differentials $\delta m_{11} = \pm 2^{r-10 \bmod 32}$ and $\delta m_i = 0$ for $i \neq 11$. Step 61 is the last in a sequence of trivial differential steps with $\delta Q_i = 0$. We have $\delta F_{61} = 0$, since the previous three steps are trivial with $\delta Q_i = 0$. Thus, $\delta T_{61} = \delta W_{61} = \delta m_{11} = 2^{r-10 \bmod 32}$. The rotation constant is $RC_{61} = 10$. Provided that the right rotation happens, we have $\delta R_{61} = \delta Q_{62} = \pm 2^r$.[12] Then, a likely value for ΔQ_{62} is (r) or $(\overline{r})$, respectively. If $Q_{60}[r] = Q'_{60}[r] = 0$, we have $\delta F_{62} = 0$. Also, $\delta W_t = 0$. It follows that $\delta Q_{63} = \delta Q_{62}$. It is then possible to have $\Delta Q_{63}[r], \ldots, \Delta Q_{63}[r + w'] \neq 0$ which allows for $\Delta F_{63}[r + \lambda] = s_\lambda$. After rotation with $RC_{63} = 21$, we may get the value for δQ_{64} given above. Below, we give an example of a differential path end-segment with $w = 4$, $\delta Q_{62}, \delta Q_{63} = 2^{11}$ and $\delta Q_{64} = 2^{11} - 2^4 + 2^2$:

61	`........ ........ 10101... ........`
62	`........ ........ 1.0.+... ........`
63	`........ ........ +----... ........`
64	`........ ........ ....+... ...-.+..`

If we have a pair of near-collision blocks based on a differential path with these values for $\delta Q_{61}, \ldots, \delta Q_{64}$ and if $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$ is the intermediate hash value *before* this pair of near-collision blocks, then the intermediate hash value *after* the new pair of near-collision blocks will be

$$\left(\delta a, \delta b \pm 2^p + \sum_{\lambda=0}^{w'} s_\lambda 2^{r+21+\lambda \bmod 32}, \delta c \pm 2^r, \delta d \pm 2^r \right).$$

We use such near-collision blocks to successively eliminate δc and δd and as much of δb as possible. The remainder of δb is eliminated by introducing a difference in δc and δd and removing it again while making use of the s_i to eliminate differences in δb.

Let us now make the strategy for eliminating the IHV-difference more precise. Let $\delta IHV = (0, \delta b, \delta c, \delta c)$ be the intermediate hash value after the Birthday Search. Fix some parameter $w < 32$. Let $(x_i)_{i=0}^{31}$ be the NAF of $\delta b - \delta c$ and $(y_i)_{i=0}^{31}$ the NAF of $\delta c = \delta d$. In phase 1 of the elimination process, we want to reduce δc to 0. Until $\delta c = 0$, repeat the following process:

[12] If we make the idealizing assumption that T_{61} is random, there is a probability of 1/2 that $T_{61}[r] = 0$. In that case, adding δT_{61} to T_{61} causes no carries, so we have $RL(T'_{61}, RC_{61}) - RL(T_{61}, RC_{61}) = RL(\delta T_{61}, RC_{61})$. Thus, the correct rotation happens with a probability of *at least* 1/2.

1. Pick some $r \in \{0, \ldots, 31\}$ such that $y_r \neq 0$.

2. Let $w' = \min(w, 31 - r)$.

3. For $\lambda = 0, \ldots, w'$, let $s_\lambda = -x_{r+21+\lambda \bmod 32}$.

4. Construct a pair of near-collision blocks from a differential path with $\delta Q_{61} = 0$, $\delta Q_{62} = \delta Q_{63} = -y_r \cdot 2^r$ and $\delta Q_{64} = -y_r \cdot 2^r - \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^{r+21+\lambda \bmod 32}$.

5. Append the constructed message block pair to the initial messages and update δIHV to the intermediate hash value after the new near-collision blocks.

Each iteration reduces the NAF-weight of δc by one. When δc has reached 0, we can write $\delta IHV = (0, \delta b, 0, 0)$. We now proceed with the second phase of the elimination strategy. Let $(l_i)_{i=0}^{31}$ be the NAF of δb. Until $\delta b = 0$, repeat the following steps:

1. Pick $i \in \{0, \ldots, 31\}$ such that $l_{i-21 \bmod 32} \neq 0$ and $r = i - 21 \mod 32$ is minimal. Let $w' = \min(w, 31 - r)$.

2. For $\lambda = 0, \ldots, w'$, let $s_\lambda = -l_{r+21+\lambda \bmod 32} = -l_{i+\lambda \bmod 32}$.

3. Construct a pair of near-collision blocks from a differential path with $\delta Q_{61} = 0$, $\delta Q_{62} = \delta Q_{63} = 2^r$ and $\delta Q_{64} = 2^r + \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^{r+21+\lambda \bmod 32} = 2^r + \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^{i+\lambda \bmod 32}$.

4. Append the resulting pair of message blocks.

5. Construct a pair of near-collision blocks with $\delta Q_{61} = 0$ and $\delta Q_{62} = \delta Q_{63} = \delta Q_{64} = -2^r$ and append this pair. Update δIHV accordingly.

Each iteration reduces the NAF-weight of δb by *at least* one. The other words in δIHV remain 0; the contributions of the first and second pair to these differentials cancel each other. Once δb reaches 0, the collision attack is complete. The second phase is more expensive than the first since we need to construct *two* pairs of near-collision blocks per iteration of the loop. A high value of w reduces the average number of iterations in phase 2, but the differential paths require more non-constant bitconditions (i.e., bitconditions other than '`.`') when w is higher. Thus, the parameter w enables trade-offs between the cost of constructing an *individual* message block and the expected number of message blocks we need to construct.

In the next section, we discuss how to find the Birthday Bits that produce a δIHV to which we can apply the elimination strategy. After that, we show how to construct full differential paths and how to solve them.

2.2.3 Birthday Search

Framework

As mentioned in Section 2.2.1, we set aside $64 + k$ bits for a Birthday Search to find message blocks that induce a difference in the intermediate hash values of a particular form. However, simply generating random Birthday Bits until a suitable pair of intermediate hash values is found is not practical due to the memory requirements. To avoid having to store too many values and in order to make use of parallelism, the framework by van Oorschot and Wiener in [28] is used. Below, we will give a description of that framework and apply it to our problem.

Suppose we have a deterministic function f on some search space $\mathcal{S}$ with $|\mathcal{S}| = n$ and we want to find a collision of f, i.e., distinct elements $x, y \in \mathcal{S}$ such that $f(x) = f(y)$. If we make the additional assumption that f is pseudo-random – which should be the case for a compression function of a cryptographic hash function – we can make the idealizing assumption that for x selected at random, the sequence $x, f(x), f(f(x)), \ldots$ is a random walk. Since it is impractical to store the whole random walk, we choose some subset D of $\mathcal{S}$ whose elements have some easily recognizable property (e.g., a certain bit pattern) and we end a random walk once it enters D. We call the elements of this set the *distinguished points*. For several random $x \in \mathcal{S}$ we compute $f(x), f(f(x)), \ldots$ until we reach an element of D, storing only the starting points x, the end points $d_x \in D$ and the lengths l_x of the walks.

Let $\theta = |D|/n$ be the fraction of distinguished points. Let X be the random variable that gives the length of a random walk. Then X is distributed geometrically with mean $1/\theta$, i.e., $\Pr[X = k] = \theta(1-\theta)^{k-1}$ under the idealizing assumption that the walks are truly random. This holds because a random walk of length k would have to first pick $k-1$ elements outside of D, which happens with probability $(1-\theta)^{k-1}$ and then pick one element inside D which happens with probability θ.

When two walks intersect, a collision has occurred and since f is deterministic, they will both end at the same distinguished point d_x. Thus, we can detect that a collision has occurred by comparing the values of d_x. As we will see, the more distinguished points we have, the less time our computation will take. However, we will also need to store more data when there are many distinguished points. Thus, the parameter θ allows us to make a time-memory trade-off.

We can then use the stored information about the two walks to recompute the collision. We can use multiple processors in parallel to compute sequences $x_i, f(x_i), \ldots$ for multiple randomly chosen $x_1, \ldots, x_m$. The final results of the walks, x_i, l_{x_i}, d_{x_i} are all stored in a common list; once a processor is finished, it selects a new random x to start another walk.

One problem that might occur with this approach is that a walk might never reach D because it loops outside of D. When this happens, the processor that computes the walk would stop contributing to our Birthday Search. This problem can be solved by abandoning walks that are too long. The probability that a random walk goes on for $20/\theta$ steps is circa e^{-20}, so we will not loose much by abandoning walks longer than $20/\theta$. Another problem is that $d_x = d_y$ also occurs when x lies on the walk of y or the other way around. This unlikely case is called a "Robin Hood"[13] and does not yield a collision of f.

When we have found distinct $x, y \in \mathcal{S}$ such that $d_x = d_y$, we know that $f^{l_x}(x) = d_x = d_y = f^{l_y}(y)$. Since we have stored the starting points x and y of these walks and also their lengths, we can search for $k_x < l_x$ and $k_y < l_y$ such that $f^{k_x}(x) \neq f^{k_y}(y)$, but $f^{k_x+1}(x) = f^{k_y+1}(y)$. If $l_x \leq l_y$, we set $k_x = l_y - l_x$ and $k_y = 0$. Then, we increase k_x and k_y until $f^{k_x+1}(x) = f^{k_y+1}(y)$. If $l_y < l_x$ we do the same with $k_x = 0$ and $k_y = l_x - l_y$. Assuming that no "Robin Hood" occurred, $f^{k_x}(x)$ and $f^{k_y}(y)$ is a collision of f. This computation can be split up among two processors.

We now calculate the number of expected steps until a collision occurs. Under our assumption that $x, f(x), f(f(x)), \ldots$ are random walks, it suffices to calculate the expected number of elements that we need to select at random from $\mathcal{S}$ until the same element is selected twice. Let Y be the

[13]In archery, a shot where the arrow hits the end of another arrow sticking in the target is called a "Robin Hood". Here, the two random walks play the role of the arrows.

random variable that counts the number of elements selected uniformly at random from $\mathcal{S}$ until the same element is selected twice. By [28, Appendix A],

$$\Pr[Y > k] = \left(1 - \frac{1}{n}\right)\left(1 - \frac{2}{n}\right) \cdot \dots \cdot \left(1 - \frac{k-1}{n}\right) \approx e^{-k^2/(2n)}$$

and

$$\begin{aligned} E[Y] = \sum_{k=0}^{\infty} k \cdot \Pr[Y = k] = \sum_{k=1}^{\infty} k \cdot (\Pr[Y > k-1] - \Pr[Y > k]) &= \sum_{k=0}^{\infty} \Pr[Y > k] \\ &\approx \int_0^{\infty} e^{-k^2/(2n)} \,\mathrm{d}k \\ &= \sqrt{\pi n/2} \end{aligned}$$

Thus we expect to compute roughly $\sqrt{\pi n/2}$ steps until a collision has occurred. If we distribute this work on m processors, it requires $\sqrt{\pi n/2}/m$ steps on each. Once a collision has occurred in some walk, it takes an expected number of $1/\theta$ steps until the walk reaches a distinguished point and it is detected.

In some applications of Birthday Searches, not every collision is useful. Unfortunately, our application is one of them. Let us therefore consider the case that a random collision of f is useful with some probability p. How many times can we expect to compute f until a useful collision is found? To simplify the analysis, we assume that for every $F \in \mathrm{Range}(f)$, there is at most one pair (x, y) that collides in F. Thus, we can label the collision value F itself as useful or useless and every collision value is useful with probability p. This simplification is justified by the fact that it is relatively unlikely that the same value occurs three times in a random walk.[14]

We model our search for a useful collision as follows: Let $\mathcal{S}'$ be the set of useful collision values in $\mathcal{S}$, so $|\mathcal{S}'| = n \cdot p$. At each step, with probability p we select a random element of $\mathcal{S}'$ and with probability $1-p$, we do nothing which simulates selecting a useless value. When we have selected the same element of $\mathcal{S}'$ twice, we stop. We want to calculate the expectation of the number of steps until we stop. By the analysis for the case where every collision is useful it follows that we can expect to select $\sqrt{\pi np/2}$ elements from $\mathcal{S}'$ until we find a repetition. We expect to make $1/p$ steps in order to select one element of $\mathcal{S}'$. Thus, the expected number of steps is $\sqrt{\pi np/2}/p = \sqrt{\pi np/(2p^2)} = \sqrt{\pi n/(2p)}$, an increase by a factor $\sqrt{1/p}$ compared to the case where every collision is useful.

By this analysis, we need to compute f an expected number of

$$\mathcal{C}_{tr} = \sqrt{\frac{\pi n}{2p}}$$

times until a useful collision occurs. Whenever a collision occurs (useful or not), we can expect to compute f another $2/\theta$ times until it is detected.[15] Then, we expect to compute f $2/\theta$ times to

[14]For $F \in \mathcal{S}$, let $X_{F,k}$ be the random variable counting the number of times that F occurs in a random walk $x_1, \dots, x_k \in \mathcal{S}$. Then, the distribution of $X_{F,k}$ is the binomial distribution $B(k, 1/n)$ and the Chernoff bound gives us $\Pr[X_{F,k} \geq 3] \leq k \cdot (n-1)/(4n^2) \approx k/(4n)$.

[15]The fact that both the expected length and the expected length after a collision occurred are $1/\theta$ seems paradoxical, but this is no actual paradox since longer walks are more likely to contain collisions.

recompute the collision and check whether it is useful. Therefore,

$$\mathcal{C}_{coll} = \frac{4}{\theta}$$

is the expected number of steps to detect and recompute a collision after it has occurred. We can expect to go through this process $1/p$ times. Thus, the overall complexity is[16]

$$\mathcal{C}_{tr} + \frac{\mathcal{C}_{coll}}{p} = \sqrt{\frac{\pi n}{2p}} + \frac{4}{\theta p}$$

Application

For the purpose of searching Birthday Bits S_b, S_b' to bring about the differences in the intermediate hash values that we require, we let $\mathcal{S} = \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^k}$. For B and B' the last $512 - 64 - k$ bits of P and P' respectively and for IHV and IHV' the intermediate hash values before these blocks, we let

$$f(x,y,z) = \left(a, c-d, b-c \bmod 2^k\right)$$
$$\text{where } (a,b,c,d) = \begin{cases} \texttt{MD5Compress}\,(IHV, B\|x\|y\|z) & \text{if } x \equiv 0 \mod 2 \\ \texttt{MD5Compress}\,(IHV', B'\|x\|y\|z) & \text{if } x \equiv 1 \mod 2 \end{cases}$$

However, as said before, not every collision we find is useful. We need one sequence of Birthday Bits to append to P and one to append to P', so we want to find a collision where one colliding element has $x \equiv 0 \bmod 2$ and the other one has $x \equiv 1 \bmod 2$. Furthermore, we want the differences δc and δb to be such that they can be removed with r collision blocks with parameter w. The probability that a collision satisfies all these requirements is denoted as $p_{r,k,w}$. Thus, we need to evaluate f an expected number of

$$C_{tr}(r,k,w) = \sqrt{\frac{\pi \cdot |\mathcal{S}|}{2 \cdot p_{r,k,w}}}$$

times until a collision occurs.

Suppose that we want to allow the algorithm to use an expected number M of memory bytes. The data for a walk can be stored in 28 bytes ($3 \cdot 32 = 96$ bits for the start and end point each; 32 bits for the length). Thus, if we want to allow an expected number of M bytes, we need to make sure that the expected number of walks that are generated until a useful collision is found is (at most) $M/28$. Since $1/\theta$ is the expected length of a walk, the expected number of walks that have to be generated until a useful collision is found is $\mathcal{C}_{tr} \cdot \theta$. That means, we must choose θ such that $\mathcal{C}_{tr} \cdot \theta = M/28$, i.e., $\theta = M/(28\mathcal{C}_{tr})$. Then, the cost for detecting and finding a collision after it has occurred is

$$\mathcal{C}_{coll}(r,k,w,M) = \frac{4 \cdot 28 \cdot \mathcal{C}_{tr}(r,k,w)}{M} = \frac{112 \cdot \mathcal{C}_{tr}(r,k,w)}{M}$$

[16]This formula differs somewhat from the one given by van Oorschot and Wiener in [28] because they calculate the time requirement with multiple processors instead of the complexity and because they seem to assume that we can decide whether a collision is useful without recomputing it.

evaluations of f. The total complexity is

$$\mathcal{C}(r,k,w,M) = \mathcal{C}_{tr}(r,k,w) + \frac{\mathcal{C}_{coll}(r,k,w,M)}{p_{r,k,w}} = \left(1 + \frac{112}{M \cdot p_{r,k,w}}\right) \cdot \mathcal{C}_{tr}(r,k,w)$$
$$= \left(1 + \frac{112}{M \cdot p_{r,k,w}}\right) \cdot \sqrt{\frac{\pi \cdot |\mathcal{S}|}{2 \cdot p_{r,k,w}}}$$

The value of $p_{r,k,w}$ is hard to compute exactly. Empirical estimates for various parameter settings are given by Stevens in [24, Table 6-6 and Appendix D]. For example, setting $k = 0$, $w = 3$ and $r = 6$ gives $p_{r,k,w} \approx 2^{-18.14}$. Then,

$$\mathcal{C}_{tr}(6,0,3) = \sqrt{\frac{\pi \cdot 2^{64}}{2^{-17.14}}} = \sqrt{\pi \cdot 2^{81.14}} \approx 2^{41.40}.$$

When we use enough memory, we can bring the actual cost of the collision search arbitrarily close to this value. Suppose that we allow our algorithm to have an expected memory use of 20 GB $\approx 2^{34.22}$ bytes.[17] Then, the Birthday Search has a total complexity of about

$$\left(1 + \frac{112}{2^{34.22} \cdot 2^{-18.14}}\right) \cdot 2^{41.40} = \left(1 + 112 \cdot 2^{-16.08}\right) \cdot 2^{41.40} \approx 2^{0.00} \cdot 2^{41.40} = 2^{41.40}.$$

Birthday Search for a Single Block Collision

There also exists a family of differential paths that can eliminate differences from some set $\mathcal{D}$ with $|\mathcal{D}| = 2^{23.3}$ of intermediate hash value differentials δIHV that have the property that $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$ is of the form $\delta a = -2^5$, $\delta d = -2^5 + 2^{25}$ and $\delta c \equiv -2^5 \mod 2^{20}$. Thus, we can generate a collision with only one pair of collision blocks if we can find an appropriate δIHV using Birthday Search. To do this, we use search space $\mathcal{S} = \{0,1\}^{84}$. Let B, B' be the last pair of message blocks of our prefix and IHV and IHV' the intermediate hash values after these blocks have been processed. Define a function $f : \mathcal{S} \to \mathcal{S}$ by

$$f(x) = \begin{cases} \phi\left(\texttt{MD5Compress}(IHV, B\|x) + \widehat{\delta IHV}\right) & \text{if } \tau(x) = 0 \\ \phi\left(\texttt{MD5Compress}\left(IHV', B'\|x\right)\right) & \text{if } \tau(x) = 1 \end{cases}$$

where $\tau : \mathcal{S} \to \{0,1\}$ is some balanced selector function, $\widehat{\delta IHV} = (-2^5, 0, -2^5, -2^5 + 2^{25})$ and $\phi(a,b,c,d) = a\|d\|c \bmod 2^{20}$. When we find some collision x, y with $\tau(x) = 0$ and $\tau(y) = 1$, then it follows that, for $(a,b,c,d) = \texttt{MD5Compress}(IHV, B\|x)$ and $(a',b',c',d') = \texttt{MD5Compress}(IHV', B'\|y)$, we have $\delta a = a' - a = -2^5$, $\delta d = -2^5 + 2^{25}$ and $\delta c \equiv -2^5 \mod 2^{20}$.

There are $2^{128-84} = 2^{44}$ pairs of intermediate hash values that instantiate this difference. Since $|\mathcal{D}| = 2^{23.30}$, the probability that a collision x, y with $\tau(x) \neq \tau(y)$ is useful is $2^{23.3}/2^{44} = 2^{-20.7}$. Since τ is balanced, the probability that any given collision is useful is $p = 2^{-20.70}/2 = 2^{-21.70}$. Plugging p and $|\mathcal{S}|$ in our formula for $\mathcal{C}_{tr}$, we get

$$\mathcal{C}_{tr} = \sqrt{\frac{\pi \cdot 2^{84}}{2 \cdot 2^{-21.70}}} = \sqrt{\pi \cdot 2^{104.70}} \approx 2^{53.18}$$

[17]In practice, it is of course necessary to make more than just the expected amount of memory available.

and allowing the algorithm an expected memory use of 140 GB $\approx 2^{37.03}$ bytes, we get a total complexity of

$$\left(1 + \frac{112}{2^{37.03} \cdot 2^{-21.70}}\right) \cdot 2^{53.18} \approx 2^{53.18}.$$

2.2.4 Extending Partial Differential Paths

When constructing a differential path for the chosen prefix attack, we start with message block differentials $\delta m_0, \dots, \delta m_{15}$ for the path that we want to build which fixes $\delta W_0, \dots, \delta W_{63}$ and with given intermediate hash values. This fixes $Q_{-3}, \dots, Q_0$ and $Q'_{-3}, \dots, Q'_0$, so we already have $\Delta Q_{-3}, \dots, \Delta Q_0$ and corresponding bitconditions $\mathfrak{q}_{-3}, \dots, \mathfrak{q}_0$ where each $\mathfrak{q}_t[i]$ is either `+`, `-`, `0` or `1`. Thus, we have the beginning of a differential path. We also have a "hand-crafted" ending of a differential path that leads to the difference in the output intermediate hash values that we want to achieve. We choose some step $\hat{t}$ at which we want to connect these two partial paths. A good choice is $\hat{t} = 11$, since this will help us in making use of the "tunnels" described in Section 2.2.8. As described below in detail, we extend our initial segment of a path forward until we have $\mathfrak{q}_{-2}, \dots, \mathfrak{q}_t$ and $\delta Q_{-3}, \delta Q_{t+1}$. We also extend the other end of the part backward until we have $\mathfrak{q}_{t+3}, \dots, \mathfrak{q}_{64}$ and $\delta Q_{t+2}, \delta Q_{64}$. Then we select additional bitconditions so that these partial paths fit together. We call the differential path that extends forward from the input intermediate hash value the *upper path* and the path that extends backwards from the desired end segment the *lower path*.

Given a valid partial differential path that fixes the values for ΔF_t, we want to be able to algorithmically extend it forward and backward such that it remains valid and for each step t, only one value for ΔF_t is compatible with the path.

2.2.5 Extending a Path Forward

A partial differential path is extended forward by the following method. Suppose that for some step t, we are given δQ_{t-3}, δQ_t, bitconditions $\mathfrak{q}_{t-2}, \mathfrak{q}_{t-1}$, and possibly $\mathfrak{q}_t$. Extending the path forward one step means doing the following:

- Select bitconditions $\mathfrak{q}_t$ on working state Q_t that enforce δQ_t.
- Select a Boolean function differential ΔF_t and, if necessary, replace '`.`'-bitconditions by Boolean function bitconditions to enforce it.
- Compute δT_t and the value for δQ_{t+1} that is based on the most likely rotation of it.

Suppose further that for every $i \in \{0, \dots, 31\}$, the following invariant holds:

$$\mathfrak{q}_t[i] \text{ and } \mathfrak{q}_{t-1}[i] \text{ are both direct and if } \mathfrak{q}_{t-2}[i] \text{ is indirect, it only involves } Q_{t-1}[i]. \tag{2.1}$$

This is the case for $t = 0$ (since initially, $\mathfrak{q}_{-3}, \dots, \mathfrak{q}_0$ are all direct) and we will see that the algorithm carries this condition over to the next step. The algorithm works as follows:

1. For step $t = 0$, $\mathfrak{q}_0$ is already given. In the later steps, we select some BSDR $(Z[i])_{i=0}^{31}$ of δQ_t with low weight. A possible choice is the non-adjacent form, but we can vary the choice of the BSDR in order to create a large number of paths which is necessary since not every pair of upper and lower paths can be connected.

2. When we have chosen the BSDR, we let

$$\mathfrak{q}_t[i] = \begin{cases} \texttt{-} & \text{if } Z[i] = -1 \\ \texttt{.} & \text{if } Z[i] = 0 \\ \texttt{+} & \text{if } Z[i] = 1 \end{cases}$$

   so that $\Delta Q_t[i] = Z[i]$. The lower the weight of $(Z[i])_{i=0}^{31}$ is, the more freedom we have in the choice of Q_t when solving the path.

3. To determine ΔF_t, we do the following for $i = 0, \ldots 31$.

   (a) We let $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$. Since $\mathfrak{q}_{t-2}[i]$ involves at most $Q_{t-1}[i]$, $\mathfrak{q}_{t-1}[i]$ is direct and $\mathfrak{q}_t[i]$ is a differential bitcondition, we have $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$, the set of *local* bitconditions as defined in Section 1.4.9.

   (b) If $V_{t,\mathfrak{abc}}$, as defined in Section 1.4.9, is a singleton, $\Delta F_t[i]$ is already determined by $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ and we proceed to the next i.

   (c) Otherwise, we select an arbitrary $g \in V_{t,\mathfrak{abc},t}$ and let $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) = FC(t, \mathfrak{abc}, g) \in \mathcal{L}$. We then replace $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2})$ by $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f})$. Similar to the choice of the BSDR in step 1, we can vary g to generate a large number of differential paths.

4. Since the message block differences $\delta m_0, \ldots, \delta m_{15}$ are already given, we now can compute $\delta T_t = \delta Q_{t-3} + \delta W_t + \delta F_t$ and select the most likely $\delta R_t \in dRL(\delta T_t, RC_t)$.

5. We compute $\delta Q_{t+1} = \delta Q_t + \delta R_t$.

After that, we have extended our path one step forward. It remains to show that the invariant (2.1) holds for step $t+1$ if it held for step t. The bitconditions $\mathfrak{q}_{t+1}$ are not given, so they do not need to concern us – we choose a tuple of differential (and hence direct) bitconditions in step 1 of the algorithm. For every $i = 0, \ldots, 31$, $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ is in $\mathcal{L}$. Thus, we know that if $\mathfrak{q}_{t-1}[i]$ is indirect, it may only involve $Q_{t-2}[i]$ or $Q_t[i]$. If $\mathfrak{q}_{t-1}$ was replaced during the algorithm it must be direct or forward since the replacement is done using the function FC. If it was not replaced, then it is direct because we assumed that (2.1) holds for step t. Thus, if $\mathfrak{q}_{t-1}[i]$ is indirect, it only involves $Q_t[i]$. Likewise, $\mathfrak{q}_t[i]$ must be either direct or forward. But since $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ is local, it can not be forward and thus it must be direct.

2.2.6 Extending a Path Backward

Suppose that for some step t, we are given $\mathfrak{q}_t, \mathfrak{q}_{t-1}$ and the differences δQ_{t+1} and δQ_{t-2}. We want to extend the differential path backwards by giving bitconditions $\mathfrak{q}_{t-2}$, changing '`.`'-bitconditions in $\mathfrak{q}_t, \mathfrak{q}_{t-1}$ to Boolean function bitconditions if necessary and giving a modular differential δQ_{t-3}. We assume that the conditions in $\mathfrak{q}_{t-1}$ are direct and that the indirect conditions in $\mathfrak{q}_t$ only involve Q_{t-1}. This condition is carried over to the next step $t-1$.

The algorithm is similar to the one for forward extension. We first choose a low-weight BSDR for δQ_{t-2}, e.g., NAF(δQ_{t-2}), and select $\mathfrak{q}_{t-2}$ accordingly. Then we replace for every $i = 0, \ldots, 31$ the triple $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ by BC$(t, \mathfrak{q}_t[i]\mathfrak{q}_{t-1}[i]\mathfrak{q}_{t-2}[i], g)$ for some $g \in V_{t,\mathfrak{q}_t[i]\mathfrak{q}_{t-1}[i]\mathfrak{q}_{t-2}[i]}$

if $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ do not fix $\Delta F_t[i]$ yet. For $\delta R_t = \delta Q_{t+1} - \delta Q_t$, we choose a high-probability[18] $\delta T_t \in dRL(\delta R_t, 32 - RC_t)$. This allows us to compute $\delta Q_{t-3} = \delta T_t - \delta F_t - \delta W_t$ and we have finished our step backwards.

2.2.7 Connecting Upper and Lower Paths

To merge a upper and lower differential path at step t, we extend the upper path forward up to step t and the lower part backward up to $t+5$. From the forward extension, we get $\mathfrak{q}_{-2}, \ldots, \mathfrak{q}_t$ and δQ_{-3} and δQ_{t+1}. From the backwards extension, we get $\mathfrak{q}_{t+3}, \ldots, \mathfrak{q}_{63}$ and differentials δQ_{t+2} and δQ_{64}. Since the bitconditions are computed with the forward and backward extension algorithms, we can assume that all bitconditions in $\mathfrak{q}_t$ are direct, that the indirect bitconditions in $\mathfrak{q}_{t-1}$ only involve Q_t, that the bit conditions in $\mathfrak{q}_{t+3}$ are direct and that all indirect bit conditions in $\mathfrak{q}_{t+4}$ only involve Q_{t+3}. Our goal is now to find bitconditions that are compatible with our previous bitconditions and that fix ΔQ_{t+1}, ΔQ_{t+2}, ΔF_{t+1}, ΔF_{t+2}, ΔF_{t+3} and ΔF_{t+4} so that a valid complete differential path results. To this end, we use an algorithm that computes these BSDRs in a bitwise manner. Not all combinations of upper and lower paths can be connected. Therefore, it is necessary to generate a large number of upper and lower paths by varying the ΔQ_t and ΔF_t in the forward and backward extension.

For the connection algorithm, we first find suitable values for $\delta F_{t+1}, \ldots, \delta F_{t+4}$:

For $i = t+1, \ldots, t+4$,

1. Compute $\delta R_i = \delta Q_{i+1} - \delta Q_i$.
2. Choose a high-probability rotation difference δT_i from $dRL(\delta R_i, 32 - RC_i)$.
3. Let $\delta F_i = \delta T_i - \delta Q_{i-3} - \delta W_i$.

For $i = 1, \ldots, 32$, let $\mathcal{U}_i$ be the set of tuples $(q_1, q_2, f_1, f_2, f_3, f_4)$ where the q_j and f_k are 32-bit words with $q_j \equiv f_k \equiv 0 \mod 2^i$ such that for $l = 0, \ldots, i-1$, there exist bitconditions $\mathfrak{q}_{t-1}[l], \ldots, \mathfrak{q}_{t+4}[l]$ compatible with our previous bitconditions which determine $\Delta Q_t[l]$, $\Delta Q_{t+1}[l]$ and $\Delta F_t[l], \ldots, \Delta F_{t+4}[l]$ such that

$$\delta Q_{t+j} = q_j + \sum_{l=0}^{i-1} 2^l \cdot \Delta Q_{t+j}[l] \text{ for } j = 1, 2$$

$$\delta F_{t+k} = f_k + \sum_{l=0}^{i-1} 2^l \cdot \Delta F_{t+k}[l] \text{ for } k = 1, 2, 3, 4$$

Clearly, $\mathcal{U}_0 = \{(\delta Q_{t+1}, \delta Q_{t+2}, \delta F_{t+1}, \delta F_{t+2}, \delta F_{t+3}, \delta F_{t+4})\}$ and $\mathcal{U}_{32}$ is either empty or contains only $(0,0,0,0,0,0)$. If it is non-empty, it follows that there are bitconditions that enforce the required values for the ΔQ_i and ΔF_i. Below, we give an algorithm for computing $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$. If we reach a $\mathcal{U}_i = \emptyset$ then there are no bitconditions for connecting the two partial differential paths. We

[18]The probabilities are computed according to Lemma 1.6. This assumes that T_t is distributed uniformly at random. In general, this is not true, especially when there are many bitconditions (see Appendix A). Nevertheless, choosing the highest theoretical probability here helps us to avoid rotations that are particularly bad – or impossible – in practice.

have to select a new combination of upper and lower path and start over. Otherwise, there exist bitconditions that ensure the required differences.

Given $\mathcal{U}_i$ for $0 \leq i \leq 31$, we compute $\mathcal{U}_{i+1}$ as follows.

Algorithm 2.1: Compute $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$

```
Initialize $\mathcal{U}_{i+1}$ as $\emptyset$;
$\mathfrak{a} = \mathfrak{q}_{t+4}[i]$, $\mathfrak{b} = \mathfrak{q}_{t+3}[i]$, $\mathfrak{e} = \mathfrak{q}_t[i]$ and $\mathfrak{f} = \mathfrak{q}_{t-1}[i]$;
for $(q_1, q_2, f_1, f_2, f_3, f_4) \in \mathcal{U}_i$ do
    for $\mathfrak{c} \in \begin{cases} \{\texttt{.}\} & \text{if } q_1[i] = 0 \\ \{\texttt{+},\texttt{-}\} & \text{if } q_1[i] = 1 \end{cases}$ and $\mathfrak{d} \in \begin{cases} \{\texttt{.}\} & \text{if } q_2[i] = 0 \\ \{\texttt{+},\texttt{-}\} & \text{if } q_2[i] = 1 \end{cases}$ do
        Let $q_1' = 0, 1$ or $-1$ if $\mathfrak{c} = \texttt{.}, \texttt{+}$ or $\texttt{-}$, respectively;
        Let $q_2' = 0, 1$ or $-1$ if $\mathfrak{d} = \texttt{.}, \texttt{+}$ or $\texttt{-}$, respectively;
        for $f_1' \in \{-f_1[i], f_1[i]\} \cap V_{t+1,\mathfrak{def}}$ do
            Let $(\mathfrak{d}', \mathfrak{e}', \mathfrak{f}') = \mathrm{FC}(t+1, \mathfrak{def}, f_1')$;
            for $f_2' \in \{-f_2[i], f_2[i]\} \cap V_{t+2,\mathfrak{cd}'\mathfrak{e}'}$ do
                Let $(\mathfrak{c}', \mathfrak{d}'', \mathfrak{e}'') = \mathrm{FC}(t+2, \mathfrak{cd}'\mathfrak{e}', f_2')$;
                for $f_3' \in \{-f_3[i], f_3[i]\} \cap V_{t+3,\mathfrak{bc}'\mathfrak{d}''}$ do
                    Let $(\mathfrak{b}', \mathfrak{c}'', \mathfrak{d}''') = \mathrm{FC}(t+3, \mathfrak{bc}'\mathfrak{d}'', f_3')$;
                    for $f_4' \in \{-f_4[i], f_4[i]\} \cap V_{t+4,\mathfrak{ab}'\mathfrak{c}''}$ do
                        Let $(\mathfrak{a}', \mathfrak{b}'', \mathfrak{c}''') = \mathrm{FC}(t+4, \mathfrak{ab}'\mathfrak{c}'', f_4')$;
                        insert the element
                            $\big(q_1 - q_1' \cdot 2^i, q_2 - q_2' \cdot 2^i, f_1 - f_1' \cdot 2^i, f_2 - f_2' \cdot 2^i, f_3 - f_3' \cdot 2^i, f_4 - f_4' \cdot 2^i\big)$
                        into $\mathcal{U}_{i+1}$;
```

Running this algorithm for $i = 0, \ldots, 31$ allows us to determine whether $\mathcal{U}_{32} \neq \emptyset$. But if $\mathcal{U}_{32} \neq \emptyset$, we also want to find the bitconditions that fix $\Delta Q_t, \Delta Q_{t+1}$ and $\Delta F_t, \ldots, \Delta F_{t+4}$. After we have found that $\mathcal{U}_{32} \neq \emptyset$, we run the algorithm again, but this time we store with each element we insert into $\mathcal{U}_{i+1}$ the tuples $(\mathfrak{a}', \mathfrak{b}'', \mathfrak{c}''', \mathfrak{d}''', \mathfrak{e}'', \mathfrak{f}')$ of bitconditions and a pointer to the previous element that lead us to insert it. Backtracking from $(0, \ldots, 0) \in \mathcal{U}_{32}$, we find bitconditions for every bit that make the differences $\Delta Q_{t+1}, \Delta Q_{t+2}$ and $\Delta F_{t+1}, \ldots, \Delta F_{t+4}$ happen.

2.2.8 Tunnels

The algorithm for solving differential paths uses *tunnels* to find collisions more quickly. Recall our basic algorithm from Section 1.4.10. We want to find a pair of message blocks that solves a given differential path. We randomly select working states $Q_1, \ldots, Q_{16}$ that satisfy the beginning of our differential path. These fix the message block $m_0, \ldots, m_{15}$ and the remaining working states $Q_{17}, \ldots, Q_{64}$. We check if the remainder of our differential path is satisfied. If yes, our next block is $m_0 \| \ldots \| m_{15}$. If not, we have to select new working states. However, using *tunnels*, we often do not have to start over from scratch. A tunnel $\mathcal{T}$ specifies some Q_k, a list of working states $Q_{i_1}, \ldots, Q_{i_m}$, a list of message block words $m_{j_1}, \ldots, m_{j_n}$ and some bitconditions such that for every $b = 0, \ldots, 31$, if the bitconditions of the tunnel are satisfied for b, we can change the bit $Q_k[b]$ without affecting any $Q_{k'}$ or $m_{k'}$ not listed by the tunnel. Typically, the list of affected working states is of the form $Q_l, \dots, Q_{64}$ for some $l > 16$. Tunnels in `MD5` have been described by Vlastimil Klima in [11].

We use eight tunnels, labeled $\mathcal{T}_1, \dots, \mathcal{T}_8$, which are described in Table 2.1. To see an example of how tunnels can help us and why they work, let us look at $\mathcal{T}_8$. Tunnel $\mathcal{T}_8$ states the following: Given that $Q_{10}[b] = 0$ and $Q_{11}[b] = 1$, we can change $Q_9[b]$ and only affect working states $Q_{25}, \dots, Q_{64}$ and message words m_8, m_9 and m_{12}.

To see why $\mathcal{T}_8$ is useful, suppose that we have selected working states $Q_1, \dots, Q_{16}$ that satisfy the beginning of our differential path. We compute further working states $Q_{17}, \dots, Q_k$ for some $k \geq 24$ which satisfy the path as well. Suppose that the next working state Q_{k+1} does not satisfy our differential path. We then need to choose new $Q_1, \dots, Q_{16}$. If we just choose them at random, we have to recompute all the message words and all the working states and our previous computation is completely wasted. However, for every index b such that $Q_{10}[b] = 0$, $Q_{11}[b] = 1$, $\mathfrak{q}_9[b] = \texttt{.}$ and $Q_9[b]$ is not referenced by any indirect bitcondition, we may flip $Q_9[b]$ and still reuse $Q_1, \dots, Q_{24}$ and all message words except m_8, m_9 and m_{12}. The number k of indices b for which this is possible is called the *strength* of $\mathcal{T}_8$. If we have $Q_1, \dots, Q_{24}$ that agree with our differential path and $\mathcal{T}_8$ has strength k, we can easily generate 2^k different sequences of working states that all satisfy our differential path up to step 24.

We now show why $\mathcal{T}_8$ works. Let us first see which parts of the message block $m_0, \dots, m_{15}$ are affected. Clearly, $m_0, \dots, m_7$ remain unchanged. The word W_8, i.e., m_8 must be changed so that $Q_9[b]$ is flipped. To ensure that Q_{10} remains unchanged, m_9 must be altered. However, for Q_{11} no change is made: The change in $Q_9[b]$ can affect Q_{11} only through $F_{10}[b]$. But

$$F_{10}[b] = (Q_{10}[b] \wedge Q_9[b]) \oplus (\overline{Q_{10}[b]} \wedge Q_8[b]) = Q_8[b]$$

since $Q_{10}[b] = 0$ is a condition of the tunnel. The working state Q_{12} does not require a change in the message block for the same reason. We have

$$F_{11}[b] = (Q_{11}[b] \wedge Q_{10}[b]) \oplus (\overline{Q_{11}[b]} \wedge Q_9[b]) = Q_{10}[b]$$

since the conditions for the tunnel require that $Q_{11}[b] = 1$. To ensure that Q_{13} remains unchanged, we need to alter m_{12} because the change in Q_9 affects T_{12}. Since the computation of each working state Q_{t+1} only uses working states $Q_{t-3}, \dots, Q_t$, the change in $Q_9[b]$ does not *directly* affect any other working states. However, the changes in m_8, m_9 and m_{12} will affect later working states. We calculate the earliest $t > 16$ such that the computation of Q_t involves one of these words. We have $W_t = m_{1+5t \bmod 16}$ for $t \in \{16, \dots, 31\}$. The least integer $t > 16$ such that $1 + 5t \bmod 16 \in \{8, 9, 12\}$ is $t = 24$ and thus none of these words are reused before we compute Q_{25}. This shows that if the bitconditions of the tunnel hold for b, we can flip the bit $Q_9[b]$ without having to recompute any working state before Q_{25} and any message word except for m_8, m_9 and m_{12}.

Table 2.1 gives a complete description of the tunnels. We call $Q_k[b]$ *free* if $\mathfrak{q}_k[b] = \texttt{.}$ and $Q_k[b]$ is not referenced by any indirect bitcondition. When using tunnels to satisfy a differential path, we may only change bits that are free. The additional bitconditions only apply to the Q_i, not to the Q'_i. For example, the condition $Q_t[b] = 0$ is also satisfied when we have $\Delta Q_t[b] = 1$. We say that a working state bit is *active* for a tunnel $\mathcal{T}_i$ when it is free and the conditions of the tunnel are met for that bit. The expression $Q_3[b] = Q_{14}[b]$ in the "Change bit" column for $\mathcal{T}_3$ means that the bits $Q_3[b]$ and $Q_{14}[b]$ can be changed but have to be equal *before and after* the change for the tunnel to work. Similarly, $Q_8[b] = RR(Q_{12}, 22)[b]$ for $\mathcal{T}_6$ means that the bits $Q_8[b]$ and $Q_{12}[b + 22 \bmod 32]$ can be changed, but only if they have the same value before and after the change.

Tunnel	Change bit	Additional bitconditions	Affected states	Affected message words
$\mathcal{T}_1$	$Q_4[b]$	$Q_5[b] = Q_6[b] = 1$	$Q_{21}, \dots, Q_{64}$	m_3, m_4, m_5, m_7
$\mathcal{T}_2$	$Q_5[b]$	$Q_6[b] = 0$	$Q_{21}, \dots, Q_{64}$	m_4, m_5, m_7, m_8
$\mathcal{T}_3$	$Q_3[b] = Q_{14}[b]$	$Q_{15}[b] = Q_{16}[b]$	$Q_{21}, \dots, Q_{64}$	$m_2, \dots, m_6, m_{13}, m_{14}, m_{15}$
$\mathcal{T}_4$	$Q_9[b]$	$Q_{10}[b] = Q_{11}[b] = 1$	$Q_{22}, \dots, Q_{64}$	m_8, m_9, m_{10}, m_{12}
$\mathcal{T}_5$	$Q_{10}[b]$	$Q_{11}[b] = 0$	$Q_{22}, \dots, Q_{64}$	$m_9, m_{10}, m_{12}, m_{13}$
$\mathcal{T}_6$	$Q_8[b] = RR(Q_{12}, 22)[b]$	$Q_{10}[b] = 1$	$Q_{23}, \dots, Q_{64}$	$m_7, m_8, m_9, m_{12}, \dots, m_{15}$
$\mathcal{T}_7$	$Q_4[b]$	$Q_5[b] = 0, Q_6[b] = 1$	$Q_{24}, \dots, Q_{64}$	m_3, m_4, m_7
$\mathcal{T}_8$	$Q_9[b]$	$Q_{10}[b] = 0, Q_{11}[b] = 1$	$Q_{25}, \dots, Q_{64}$	m_8, m_9, m_{12}

Table 2.1: Tunnels of `MD5Compress`

Let us now, as more difficult examples, verify that $\mathcal{T}_3$ and $\mathcal{T}_6$ work. Suppose the additional bitconditions for $\mathcal{T}_3$ are satisfied for some b, that we have $Q_3[b] = Q_{14}[b]$ and that we flip these bits. We let $a = -2^b$ if we flip them from 0 to 1 and we let $a = 2^b$ if we flip from 1 to 0 so that our new $Q_3[b]$ is $Q_3[b] - a$ and likewise for Q_{14}. In order to flip the bit $Q_3[b]$, we have to change m_2. We then change $m_3, \dots, m_6$ so that no changes occur in the working states $Q_4, \dots, Q_7$. For our further analysis, the change we have to make in m_6 is important: Our change in $Q[3]$ can affect $Q[7]$ only through the variable $T_6 = Q_3 + F_6 + m_6 + RC_6$. To ensure that T_6 remains unchanged, we just add a to m_6. From now on, our change in Q_3 can have no direct influence on working states. Similarly, we change m_{13} to flip $Q_{14}[b]$ and then change m_{14} and m_{15} to avoid changes in Q_{15} and Q_{16}. Our change in $Q_{14}[b]$ can affect Q_{17} only through the Boolean function, but it holds that

$$F_{16}[b] = (Q_{14}[b] \wedge Q_{16}[b]) \oplus (\overline{Q_{14}[b]} \wedge Q_{15}[b])$$

which is independent of $Q_{14}[b]$ under the condition that $Q_{15}[b] = Q_{16}[b]$. The final working state that Q_{14} can affect is Q_{18}. But we have $T_{17} = Q_{14} + F_{17} + AC_{17} + W_{17}$ with $W_{17} = m_6$, so there is no change in T_{17}: The changes in Q_{14} and in m_6 cancel each other. The first time that any of the changed messages are reused after that is step 20.

Suppose the additional bitconditions for $\mathcal{T}_6$ are satisfied, $Q_8[b] = RR(Q_{12}, 22)[b]$ and that we flip these bits. If we flip from 0 to 1, let $z = 0$. Otherwise, let $z = 1$. Clearly, m_7 must be changed to bring about the bit flip in $Q_8[b]$. This bit flip changes the value of Q_8 to $\tilde{Q}_8 = Q_8 + (-1)^z \cdot 2^b$. Next, m_8 and m_9 are changed so that no changes occur in Q_9 and Q_{10}. Our bitcondition $Q_{10}[b] = 1$ prevents that the change in $Q_8[b]$ affects Q_{11} through the Boolean function as follows. We have

$$F_{10}[b] = (Q_{10}[b] \wedge Q_9[b]) \oplus (\overline{Q_{10}[b]} \wedge \tilde{Q}_8[b]) = Q_9[b]$$

since $Q_{10}[b] = 1$. The new value for Q_{12} is $\tilde{Q}_{12} = Q_{12} + (-1)^z \cdot 2^{b+22 \bmod 32}$ which one can see as follows: For $\tilde{T}_{11} = W_{11} + AC_{11} + F_{11} + \tilde{Q}_8$, we have

$$\begin{aligned} \tilde{Q}_{12} &= Q_{11} + RL(\tilde{T}_{11}, 22) \\ &= Q_{11} + RL(T_{11}, 22) + (-1)^z \cdot 2^{b+22 \bmod 32} \\ &= Q_{12} + (-1)^z \cdot 2^{b+22 \bmod 32}. \end{aligned}$$

Given our assumption that $Q_8[b]$ and $RR(Q_{12}, 22)[b]$ are equal before we make our change, this change means that the bit $RR(Q_{12}, 22)[b]$ is flipped. We then alter the message words $m_{12}, \dots, m_{15}$

to make sure that the change in Q_{12} does not affect the working states $Q_{13}, \ldots, Q_{16}$. From then on, the changes in the working states do not have any direct effects, so it remains to check when the altered message words are reused. The first step at which this happens is step 24 which reuses the word m_{15}.

2.2.9 Solving Differential Paths

Algorithm 2.2 shows in detail how we can solve a full valid differential path with the help of tunnels. The key differences to the simplified version described at the beginning of this subsection are as follows: We do not compute the whole message block right after choosing the working states $Q_1, \ldots, Q_{16}$. Instead, we compute the individual words in the block as they are needed. This allows us most of the time to change tunnel bits and only recompute *one* message word. Also, we do not check whether the bitconditions $\mathfrak{q}_{25}, \ldots, \mathfrak{q}_{64}$ are actually met because most of them are trivial bitconditions with $\delta Q_i = 0$ and because the bitconditions do not need to be solved *exactly* to yield a useful output. Thus, we only check if $\delta Q_{61}, \ldots, \delta Q_{64}$ have the values that we want. We use $\mathfrak{q}_1, \ldots, \mathfrak{q}_{16}$ to select $Q_1, \ldots, Q_{16}$. With the help of tunnels $\mathcal{T}_1, \ldots, \mathcal{T}_7$, we modify our working states so that they also solve the steps given by the bitconditions $\mathfrak{q}_{17}, \ldots, \mathfrak{q}_{24}$. We can then vary the bits of tunnel $\mathcal{T}_8$ to generate many sequences of working states that all satisfy $\mathfrak{q}_1, \ldots, \mathfrak{q}_{24}$. For all these working state sequences, we check whether they cause the right δIHV. We also do not check whether the rotations of the first 16 steps are correct because they change frequently due to our use of tunnels. It is possible to bring the rotation probabilities very close to 1 by adding bitconditions. This is described in more detail in [24, Section 6.3].

2.2.10 Cost of the Attack

For suitable parameters, the chosen-prefix attack has a total cost equivalent to $2^{39.1}$ `MD5Compress`-calls, using 8 near-collision blocks. A collision with only 3 blocks can be achieved with a cost of 2^{49} (see [24, Section 6.5.3]). The single-block chosen-prefix collision has a total cost of $2^{53.2}$ (see [24, Section 6.5.4]).

Algorithm 2.2: Solving a differential path

```
Convert all forward bitconditions in the path to backward bitconditions using $BC$.;
For every tunnel, check if its conditions can be met and if yes, add these bitconditions to the differential path.
Replace the '.'-bitconditions on the change bits by 0;
/* this restriction will only be imposed on the first set of working states we choose */
while no output has been produced do
    Select $Q_1, Q_2, Q_{13}, \dots, Q_{16}$ that meet bitconditions $\mathfrak{q}_1, \mathfrak{q}_2, \mathfrak{q}_{13}, \dots, \mathfrak{q}_{16}$;
    Compute $W_{16} = m_1$ and $Q_{17}$;
    if $Q_{17}$ does not satisfy $\mathfrak{q}_{17}$ or the correct rotation does not occur at step 16 then
        go to the next loop iteration;
    Store the set $\mathcal{Z}$ of all pairs $(\tilde{Q}_1, \tilde{Q}_2)$ of working states that satisfy $\mathfrak{q}_1$ and $\mathfrak{q}_2$, result in the same message
    word $m_1$ as $Q_1, Q_2$, and have the same bits as $Q_1$ and $Q_2$ in the places that are relevant to $\mathfrak{q}_3$;
    for all $Q_3, \dots, Q_7$ that satisfy $\mathfrak{q}_3, \dots \mathfrak{q}_7$ do
        Compute $W_{17} = m_6$ and $Q_{18}$;
        if $Q_{18}$ does not satisfy $\mathfrak{q}_{18}$ or the wrong rotation occurs at step 17 then
            go to the next loop iteration
        for all $Q_8, \dots, Q_{12}$ meeting $\mathfrak{q}_8, \dots, \mathfrak{q}_{12}$ do
            Compute $W_{18} = m_{11}$ and $Q_{19}$;
            if $Q_{19}$ does not meet its bitconditions or the rotation was incorrect then
                go to the next loop iteration;
            for every pair $(Q_1, Q_2) \in \mathcal{Z}$ do
                Compute $W_{19} = m_0$ and $Q_{20}$;
                if $Q_{20}$ does not meet $\mathfrak{q}_{20}$ or an incorrect rotation occurs at step 19 then
                    go to the next loop iteration;
                for every available bit of tunnels $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ do
                    Compute $W_{20} = m_5$ and $Q_{21}$;
                    if $Q_{21}$ does not meet its bitconditions or an incorrect rotation happened then
                        go to the next loop iteration;
                    for every available bit of tunnels $\mathcal{T}_3$ and $\mathcal{T}_4$ do
                        Compute $W_{21} = m_{10}$ and $Q_{22}$;
                        if $Q_{22}$ does not meet $\mathfrak{q}_{22}$ or an incorrect rotation happens then
                            go to the next loop iteration;
                        for every available bit of tunnel $\mathcal{T}_6$ do
                            Compute $W_{22} = m_{15}$ and $Q_{23}$. if $Q_{23}$ does not meet its bitconditions or an
                            incorrect rotation happens then
                                go to the next loop iteration;
                            for every available bit of tunnel $\mathcal{T}_7$ do
                                Compute $W_{23} = m_4$ and $Q_{24}$;
                                if $Q_{24}$ does not meet $\mathfrak{q}_{24}$ or an incorrect rotation happens then
                                    go to the next loop iteration;
                                for every available bit of $\mathcal{T}_8$ do
                                    Compute $W_{24} = m_9$ and $Q_{25}$;
                                    if $Q_{25}$ does not meet $\mathfrak{q}_{25}$ or an incorrect rotation happens then
                                        go to the next loop iteration
                                    Compute the message block words that have not been computed yet,
                                    the working states $Q_{26}, \dots, Q_{64}$ and $Q'_1, \dots, Q'_{64}$;
                                    if for $t = 61, \dots, 64$, the differences $\delta Q_t = Q'_t - Q_t$ agree with
                                    bitconditions $\mathfrak{q}_t$, then
                                        return $B = m_0 \| m_1 \| \dots \| m_{15}$ and
                                        $B' = m_0 + \delta m_0 \| m_1 + \delta m_1 \| \dots \| m_{15} + \delta m_{15}$
```

2.3 Counter-Cryptanalysis: Detecting Collision Attacks

2.3.1 Counter-Cryptanalysis

When a hash function is broken, ideally, its users would replace it with a more secure one. However, in many cases, this will require a lot of time and effort so that the hash function will have to remain in use for a long time after the decision to switch was made. For example, a certification authority might switch to another signature scheme after a collision attack appears, but the certificates that it has signed using the old hash function will still be valid. Tracking down all the certificates and asking their owners to replace them would not be practical. This also shows that it is sensible to start replacing cryptographic hash functions as soon as the first weaknesses start to appear – even if they are not considered practically relevant yet. In [24, Chapter 8], techniques are introduced to detect whether a given message was constructed via a collision attack on `MD5` or `SHA1` based on differential cryptanalysis without knowing the precise attack algorithm and the counterpart of the message. These techniques also allow to extract the differential paths of the colliding messages. This approach is called *counter-cryptanalysis* by Stevens in [25] since it uses cryptanalytic techniques to detect cryptanalytic attacks. We will describe the detection algorithm for `MD5` here, since it was used to extract the differential paths for the Flame collision attack from the rogue certificate.

2.3.2 Collision Detection in MD5

The collision detection algorithm is based on the following assumptions about the near-collision blocks.

- The differential path contains *trivial steps*, i.e., a sequence of at least four working states that all have arithmetic differential 0 or all have the differential 2^{31}.
- The set of message block differences that allow for feasible collision attacks is known and not too large.

Thus, the algorithm might be bypassed by collision attacks that use a completely different methodology or a message block difference that was not considered useful for collision attacks before. However, since there has been done a lot of research into cryptanalysis of `MD5`, both of these possibilities seem unlikely. Moreover, the algorithm can easily be updated when more message block differences that allow a collision are discovered. In this respect, the algorithm has some similarity to anti-virus software that regularly updates its malware definitions. The collision detection should not be viewed as a *fix* to `MD5` but as a transitional safety measure until `MD5` can be replaced by a more secure hash function.

The collision detection algorithm is based on an algorithm for detecting whether the kth 512-bit block M_k of message M is the *final* near-collision block of a collision attack. If that is the case, the intermediate hash value differential after M_k is processed should be 0. By testing each block of the message, we can determine whether the message was constructed using a collision attack. This can be done in tandem with computing the hash of a message: To check if M_k is the final near-collision block, we need the intermediate hash value IHV_k and we compute IHV_{k+1}. Thus, since $IHV_0 = IV$ is fixed, we can check if M_0 is the final near-collision block and obtain IHV_1 in the process. Then we can check if M_1 is the final near-collision block and obtain IHV_2 in the process, and so on until we reach the end of the message. The last IHV that we obtain is the output of `MD5`.

Let $\mathcal{M}$ be the set of tuples $(\delta m_0, \ldots, \delta m_{15}, i, \delta Q_i)$ for which it is suspected that there might be a feasible differential path based on message block differences $\delta m_0, \ldots, \delta m_{15}$ with a sequence of four trivial steps with difference δQ_i starting at step i. Algorithm 2.3 allows to check whether the kth 512-bit block M_k in message M is the final near-collision block. It works as follows: We first compute $IHV_{k+1} = \texttt{MD5Compress}(IHV_k, M_k)$, storing the working states $Q_{-3}, \ldots, Q_{64}$. Every element of $\mathcal{M}$ allows us to compute a candidate M'_k for the kth block of the colliding message and four working states $Q'_i, \ldots, Q'_{i+3}$. From these working states, we can compute the working states $Q'_{i-1}, \ldots, Q_{-3}$ by computing the steps of `MD5Compress` backwards and $Q'_{i+4}, \ldots, Q'_{64}$ by computing them forwards. From these working states, we obtain IHV'_{k+1} and check if $IHV_{k+1} = IHV'_{k+1}$. If yes, we conclude that M was constructed with a collision attack.

Algorithm 2.3: Detect final near-collision block

```
Compute $IHV_{k+1} = \texttt{MD5Compress}(IHV_k, M_k)$ and store the working states $Q_{-3}, \ldots, Q_{64}$;
for $(\delta m_0, \ldots, \delta m_{15}, i, \delta Q_i) \in \mathcal{M}$ do
    Let $M'_k = m_0 + \delta m_0 \| \ldots \| m_{15} + \delta m_{15}$;
    Compute working states $Q'_i = Q_i + \delta Q_i, \ldots, Q'_{i+3} = Q_{i+3} + \delta Q_i$;
    /* Compute MD5Compress backwards */
    for $t = i - 1, \ldots, -3$ do
        Let $R'_{t+3} = Q'_{t+4} - Q'_{t+3}$;
        Let $T'_{t+3} = RR(R', RC_{t+3})$;
        Let $F'_{t+3} = f_{t+3}(Q'_{t+3}, Q'_{t+2}, Q'_{t+1})$;
        Let $Q'_t = T'_{t+3} - F'_{t+3} - W'_{t+3} - AC_{t+3}$;
    Compute $Q'_{i+4}, \ldots, Q'_{64}$ using the step-function of MD5Compress;
    Determine $IHV'_k$ from $Q'_{-3}, \ldots Q'_0$ and $IHV'_{k+1}$ from $IHV'_k$ and $Q'_{61}, \ldots Q'_{64}$;
    if $IHV_{k+1} = IHV'_{k+1}$ then
        return "Collision detected"
return "No collision detected", $IHV_{k+1}$
```

If the message block M_k was indeed constructed using a differential path whose characteristics are contained in $\mathcal{M}$, the algorithm will detect that. The cost of computing this algorithm on every block of the message while computing $\texttt{MD5}(M)$ is roughly $|\mathcal{M}| + 1$ times the cost of computing $\texttt{MD5}(M)$. Whether this can be improved by early abort is currently investigated. The probability of a false positive is *lower bounded* by $|\mathcal{M}| \cdot 2^{-128}$. Most known differential paths for near-collision blocks have a large number of non-constant bitconditions, so it is improbable that a block M_k will match such a differential path by accident. Since `MD5` is designed to be pseudo-random, we can assume that, as long as M_k was not constructed in a collision attack, the values for IHV'_{k+1} that we compute are random. If *all* differential paths were like that, the probability for a false positive would be around $|\mathcal{M}| \cdot 2^{-128}$. However, there is at least one relevant differential path which has very few bitconditions, namely the one used in the *pseudo-collision* attack by den Boer and Bosselaers in [3]. This path has $\delta Q_i = 2^{31}$ for $i = -3, \ldots, 64$, no message block differences and no Boolean function bitconditions. Thus, it eliminates a difference of $\delta IHV = (2^{31}, 2^{31}, 2^{31}, 2^{31})$. When the collision detection algorithm suggests that this path has been used, we apply the collision detection algorithm on the previous block, checking for $\delta IHV = (2^{31}, 2^{31}, 2^{31}, 2^{31})$ rather than $\delta IHV = (0, 0, 0, 0)$.

Let us now see how this detection algorithm can be used as a *forensic tool* to extract the differential paths of *all* near-collision blocks. Suppose we have found that for some tuple in $\mathcal{M}$, $IHV_{k+1} = IHV'_{k+1}$. In the detection algorithm, we computed $Q_{-3}, \ldots, Q_{64}$ and $Q'_{-3}, \ldots, Q'_{64}$ from which we can determine the differential path for the final near-collision block. We also computed IHV'_k in the algorithm. Now we repeat the detection algorithm on message block M_{k-1} except that instead of checking for $IHV_k = IHV'_k$ we check whether the $\widehat{IHV'}_k$ that we compute agrees with the value for IHV'_k that we already have. Iterating this procedure for $k-2, k-3, \ldots,$ we find the working states for all near-collision blocks and hence we can reconstruct the differential paths. When we reach a block M_l for which no element of $\mathcal{M}$ yields the correct $\widehat{IHV'}$, we conclude that M_{l+1} was the first near-collision block. The data on which Section 3 is based was derived by Marc Stevens using this algorithm.

Chapter 3

The Collision Attack for Flame

3.1 The Flame Collision Attack

3.1.1 About Flame

Flame is a highly advanced malware, used for espionage, that attacked computers running a Microsoft Windows operating system. Among other things, one of its mechanisms for spreading within local computer networks is noteworthy. Disguising itself as a security update for Windows, it spread locally via Windows Update, even though Windows Update requires that the updates it installs carry a valid digital signature from a certificate that chains to the Microsoft Root Authority certificate.[19] To circumvent this security measure, the authors of Flame used a previously unknown chosen-prefix collision attack on the hash function `MD5` to obtain a certificate that allowed them to sign code in the name of Microsoft. The collision attack is *not* part of Flame itself but had to be carried out once in order to allow the malware to abuse Windows Update. The aim of this section is to give some background about Flame and to give references to detailed descriptions. After that, we go into some more details about the collision attack.

According to the technical report [21] by CrySyS Lab at the Budapest University of Technology and Economics and the blog post [9] by Kaspersky Labs, Flame gathers information such as keyboard inputs, screen content and, if available, input from microphones or cameras. It consists of various modules that can be downloaded after the initial infection. Its size, when fully deployed, is over 20 MB which is unusually large for malware. Flame could spread within networks using Windows Update and also infect flash drives to cross "air gaps". As described in [22], when Flame infected a computer in a network, it used WPAD (Web Proxy Auto-Discovery Protocol) to register itself as a proxy for the domain `update.windows.com` and served a fake security update to the other computers that installed Flame. Flame was not spread wildly – only a small number of computers were infected, most of them in the Middle East. It was first found by an Iranian Computer Emergency Response Team (see [14]) in May 2012. According to [9], it was active since at least 2010, but the CrySyS Lab report states that a file named `WAVESUP3.DRV`, which is the name of a dynamic link library used in Flame, was found as early as 2007 by the computer security enterprise Webroot. This might mean that Flame or a preliminary version thereof was already active at that time. The vast majority of infections occurred in Iran. There seems to be no clear pattern for the targets of

[19] A certificate C chains to a certificate D if and only if C is signed by D or it is signed by a certificate C' that chains to D.

Flame. Among the targets were government-related organizations, private companies, educational institutions and specific individuals. The Washington Post article [16] states that Flame, like the similar malware Stuxnet, was developed jointly by the USA and Israel.

3.1.2 The Rogue Certificate

The signature on the rogue certificate that was used to sign the fake security update was obtained from a certificate for Microsoft's Terminal Services Licensing Server. Terminal Services (now called Remote Desktop Services) allows a user to log in remotely via a network to another computer running Windows. This process requires a license. In businesses where many users need to access some server, a Terminal Services Licensing Server can be used as a license management system. Upon activation, a Licensing Server generates a public and secret key and automatically requests an X.509 certificate from Microsoft. These certificates chained to the Microsoft Root Authority certificate and could therefore be used to sign code in the name of Microsoft which was presumably not the intention of the developers. This chaining could have allowed attackers to exploit Windows Update for malicious purposes without any collision attack at all. Luckily, the certificates contain an extension which is marked critical that could not be "understood" by Windows Update. On Windows Vista and Windows 7, when a critical certificate extension is not recognized, the certificate is rejected so that, inadvertently, a simple attack on Windows Update was prevented.[20] After the rogue certificate was discovered in June 2012, Microsoft revoked the certificate chain for Terminal Services (see the blog post [17]).

During the analysis of Flame, the rogue certificate was uncovered and sent to Marc Stevens via private communication. Using methods described in [24, Chapter 8] and [25], he verified that it was constructed by a chosen-prefix collision attack and reconstructed the underlying *differential paths* of the collision attack. The legitimate counterpart of the rogue certificate is supposedly lost.

We will now give a description of the two colliding certificates, according to [22]. A graphical representation from [22] follows in Figure 3.1. A certificate for a Terminal Services Licensing Server consists of the following parts:

1. A serial number, composed of the number of milliseconds since boot, a constant CA index of 8 bits and a sequential 32-bit number
2. The validity period of the certificate
3. The *Common Name (CN)* "Terminal Services LS"
4. A 4072-bit RSA public key
5. The certificate extensions, including the one that prevents code signing for Windows Update
6. An `MD5`-based signature by the Certification Authority

The rogue certificate, on the other hand, has the following parts:

1. A serial number

[20]This does not hold for Windows XP which simply ignores extensions that it does not recognize, even if they are marked critical. If the Flame authors had been content with attacking Windows XP, they would not have needed the collision attack.

2. The validity period

3. The Common Name “MS”

4. A 2048-bit RSA public key

5. A certificate extension called “issuerUniqueID” which is not used by Microsoft and ignored in Windows

6. An MD5-based signature

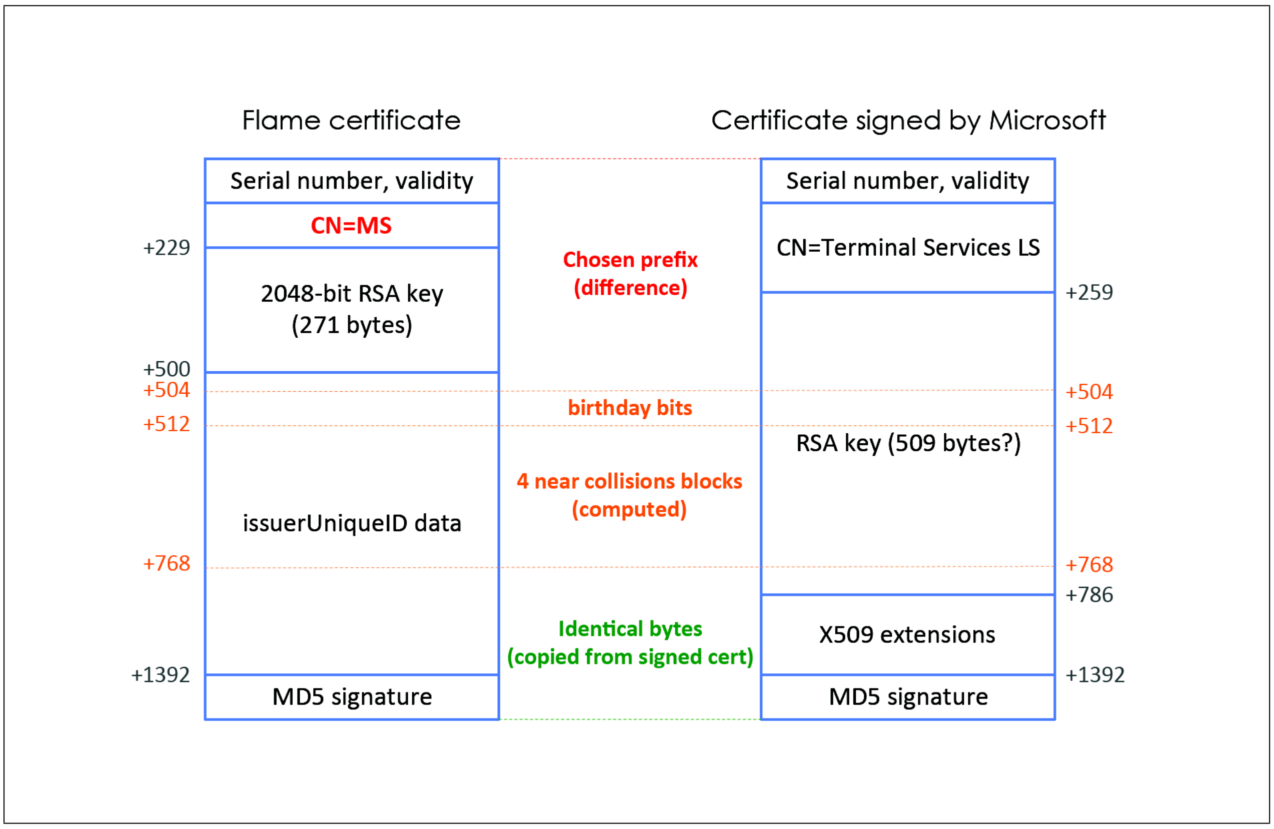

Figure 3.1: A schematic representation from [22] of the colliding certificates constructed for Flame.

The colliding certificates were constructed as follows: On the side of the legitimate certificate, the serial number, validity period, common name and the beginning of the RSA key form the chosen prefix; on the side of the rogue certificate, it is the serial number, validity period, a full RSA public key and the beginning of the “issuerUniqueID”-field. Since the serial number and validity period in the legitimate certificate are filled in by the CA, the attackers had to *guess* these values. On the side of the legitimate certificate, the collision blocks are contained in the field for the RSA key – this is a good place to hide them since a cryptographic key is *supposed* to look random. Due to the incremental nature of MD5, it is possible to append identical suffixes to the legitimate and the rogue certificate after the collision has been achieved. The attackers then appended the required certificate extensions to both certificates. In the rogue certificate, they are hidden away in the “issuerUniqueID”-field, so that the critical extension does not take effect.

Guessing the sequential number in the serial number is relatively easy: The attackers could have submitted several legitimate certificates and observed the differences in the sequential numbers to estimate how many certificates the CA will sign in a given period of time. Furthermore, they could have submitted additional requests just before the targeted request time to advance the sequential number to the guessed value. The validity period depends only on the second in which the certificate is received by the CA. But since the serial number also contains the milliseconds since the boot of the signing computer, the CA had to receive the certificate in the exactly right millisecond for the attackers to succeed. Therefore, with near certainty the attackers had to make a large number of attempts.

It is worth noting that this attack could have been stopped at several different levels: Of course, a more secure hash-function could have been used. But also some non-cryptographic measures could have prevented the attack. Instead of accepting signatures from *every* certificate with code signing rights that chains to the Root Authority, Windows Update could have used a public key infrastructure (PKI) of its own that is isolated from other PKIs. The certificates from the Terminal Services chain could have been given an extension so that all certificates in the chain are prevented from code-signing. The serial numbers of the certificate could have included random bits to make guessing the right serial number for the chosen-prefix attack prohibitively hard.

3.2 Hypotheses

The differential paths that were reconstructed by Stevens (see Appendix A) clearly show that the certificates described in Section 3.1.2 were crafted using a collision attack, consisting of a Birthday Search and a sequence of near-collision blocks to eliminate the resulting δIHV. The message block differences are identical to the differences used by Wang et al. However, the differential paths do not match any previously published attacks on `MD5`. In the remainder of this chapter, we try to reconstruct details about this collision attack, in particular about the differential path construction, the *family* of lower differential paths that were used, the cost of the Birthday Search and the cost of the message block construction. We show that – under our stated assumptions – the expected cost of the attack was equivalent to at least $2^{46.6}$ calls to `MD5Compress`. We use several different methods to achieve our results: inspection of the differential paths, calculations (under idealizing assumptions based on the pseudo-randomness of `MD5`), and empirical estimates. In the remainder of this section, we state our hypotheses about the collision attack.

The Flame collision attack is overall similar to the attack by Stevens et al., in that it starts with a Birthday Search to find a suitable value for δIHV, followed by a series of near-collision blocks. Similar to the attack by Stevens et al., the differential paths seem to be constructed from an upper and lower part with the upper part obtained from the input δIHV and the lower part designed to provide a certain output δIHV. However, the process of constructing full differential paths seems to be different.

Hypothesis 3.1. *The upper parts of the differential paths are constructed in a* brute force *way instead of a forward-extension algorithm.*

Hypothesis 3.2. *The connection of the upper and lower parts is done by brute force. The upper and lower paths are connected at working states Q_5, Q_6, Q_7 and Q_8.*

We will argue for these conclusions in Section 3.3.1. In Section 3.3.2, we give an estimate of the complexity of connecting the upper and lower parts, in comparison to the method used by Stevens et al.

It seems that the four blocks can be grouped into two *pairs* of blocks: The first and second block form a pair, and the third and fourth. In each of the pairs, the first block uses the message block differences of the first near-collision block in the identical-prefix attack by Wang et al. and the second block uses the difference of the second near-collision block in that attack.

In Section 3.4, we investigate to what extent the attackers used tunnels to solve differential paths. It appears that tunnel $\mathcal{T}_8$ was used, but not to the maximal extent that was possible given the differential paths. However, we were unable to determine how exactly this tunnel was used. We discuss several possible methods how they *could* have used the tunnel and show why these methods do not explain the observed tunnel strengths.

To determine the complexity of the Birthday Search and of the message block construction algorithm, we describe a family of possible end-segments of the differential path for each of the four near-collision blocks. We believe that the elimination strategy is as follows:

Hypothesis 3.3. *The differentials δa and δd in δIHV are constant. δb and δc consist of variable and constant parts. The purpose of the first pair of near-collision blocks is to eliminate δc (except for a constant term) from the input $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$, while making some mostly random changes to δb in the process. The purpose of the second pair is to eliminate δb, including the random changes from the previous pair.*

We compute the complexity of the Birthday Search and the complexity of the algorithm for generating near-collision blocks on the basis of our reconstruction of the end-segments. Our main result can be summarized as follows:

Theorem 3.4. *Under plausible assumptions (the hypotheses in this section and the assumptions stated in Section 3.5.1), the Flame collision attack has an expected cost equivalent to at least $2^{46.6}$ calls to the function* `MD5Compress`.

We will prove this result in Section 3.6.3 and also show how the parameters in our reconstruction of the attack must be chosen to achieve this lower bound. However, while this parameter choice is consistent with the observed differential paths, it seems likely that the Flame attack used different parameters. The expected full cost of the attack seems to be higher; a likely explanation is that the attack was optimized for *speed* on massively parallel architectures[21], not for theoretical cost.

3.3 Analysis of the Differential Paths

3.3.1 Some Features of the Near-collision blocks

In this section, we list several observations about the Flame near-collision blocks that are relevant to our analysis.

Observation 3.5. *The first and third near-collision blocks of the Flame collision attack use the message block differences from the first differential path of Wang et al.'s identical-prefix attack, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = 2^{15}$ and $\delta m_i = 0$ for $i \neq 4, 11, 14$. The second and fourth block use the differences from the second differential path of the identical prefix attack, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = -2^{15}$ and $\delta m_i = 0$ for $i \neq 4, 11, 14$.*

[21]e.g., graphic processing units (GPUs).

Observation 3.6. *For steps $i = 25, \dots, 32$, all four near-collision blocks have the trivial working state differences $\delta Q_i = 0$. For steps $i = 35, \dots, 59$, they have the trivial working state differences $\delta Q_i = 2^{31}$, like the differential paths used in the attack by Wang et al.*

Observation 3.7. *The working state differences ΔQ_6 are maximal in all four near-collision blocks, i.e., for every $i = 0, \dots, 31$, we have $\Delta Q_6[i] \neq 0$. For $t = 6, \dots, 32$, the first and third differential path have the same value for ΔQ_t. Likewise, the second and fourth paths have the same value for ΔQ_i with only one exception: In the second block, we have $\Delta Q_{22}[31] = -1$ and in the fourth path, we have $\Delta Q_{22}[31] = 1$. However, this still gives us the same value for δQ_{22}. In contrast, the values of δQ_i for $i < 6$ are different in all four paths.*

Observation 3.8. *In the first and third near-collision block, the values of $\Delta F_9, \dots, \Delta F_{35}$ are identical. In the second and fourth block, the values for ΔF_{11}, ΔF_{12}, ΔF_{13}, $\Delta F_{15}, \dots, \Delta F_{22}$ and $\Delta F_{24}, \dots, \Delta F_{35}$ are equal. Also, δF_{14} and δF_{23} are equal in the second and fourth block, and the values for ΔF_{23} only differ in bit position 31 where the sign does not matter modulo 2^{32}.*

Proof. In the first and third differential paths, all the bitconditions on working states $Q_9, \dots, Q_{32}$ are identical. This implies that $\Delta F_{11}, \dots, \Delta F_{32}$ are identical for the first and third block. To see that the values for ΔF_9 and ΔF_{10} are identical, first recall that $\mathfrak{q}_i[j], \mathfrak{q}_{i-1}[j], \mathfrak{q}_{i-2}[j] \notin \{\texttt{+}, \texttt{-}\}$ implies $\Delta F_i[j] = 0$. Thus, it remains to inspect the positions where one of the relevant bitconditions is + or -. At all these places, the bitconditions in the first and third path are the same. Finally, the bitconditions $\mathfrak{q}_{35}[31]\mathfrak{q}_{34}[31]\mathfrak{q}_{33}[31]$ in those two paths cause the same $\Delta F_{35}[31]$.

In the second and fourth path, the bitconditions on Q_9, Q_{10}, Q_{11}, $Q_{14}, \dots, Q_{21}$ and $Q_{23}, \dots, Q_{31}$ are identical. This shows that the values for ΔF_{11}, $\Delta F_{16}, \dots, \Delta F_{21}$ and $\Delta F_{25}, \dots, \Delta F_{31}$ are identical in those two blocks. Also, ΔF_{12} and ΔF_{13} have the same values in both blocks because there are no '+'- or '-'-bitconditions involved. The values for ΔF_{14} are different, but the values for δF_{14} are still the same in the two blocks. The values for ΔF_{15} are the same. The bitconditions on steps $20, 21, 22$ still produce the same ΔF_{22}. The values for ΔF_{23} are different but due to $2^{31} \equiv -2^{31} \mod 2^{32}$, we have the same values for δF_{23}. The values for ΔF_{24} are equal again. □

Observation 3.9. *The probabilities for the correct rotations of δT_t for $11 \leq t \leq 61$ in all four blocks, as given by the formulas in Lemma 1.6, are optimal, i.e., the rotations in the observed path have the highest probability among all alternatives. The conditional estimates are quite similar to the computed probabilities at these steps.*

However, on steps with many bitconditions, the formulas for the probabilities in Lemma 1.6 are less meaningful and the first 8 steps of the Flame differential paths have more bitconditions than the later steps. The conditional estimates and the computed probabilities may differ drastically, as can be seen, for example, in the second and third block at step 4 which has a computed probability of roughly 0.1 while the conditional estimate is 1.0. Thus, we should also take a closer look at the conditional estimates:

Observation 3.10. *The following table summarizes the* estimated conditional probabilities *for the rotations in steps 0 to 10 of all four near-collision blocks. For the specified ranges for the probability p, it lists for each near-collision blocks the steps where the rotation probability falls in the given range, followed by the total number of such steps.*

Probability p	1st block	2nd block	3rd block	4th block	Sum
$0.05 < p < 0.06$			8	8	2
$0.10 \leq p < 0.15$	10		10		2
$0.15 \leq p < 0.20$	$2, 8$				2
$0.25 \leq p < 0.50$		$1, 2$	2		3
$0.50 \leq p < 0.75$	$1, 9$	$5, 8, 9$	9	$1, 6, 7, 9$	10
$0.75 \leq p < 1.00$		$0, 3, 10$	1	$2, 10$	6
$p = 1.00$	$0, 3, \ldots, 7$	$4, 6, 7$	$0, 3, \ldots, 7$	$0, 3, 4, 5$	19

Thus, we can see that the probabilities are mostly rather large.

These observations do not match the attack by Stevens et al. The observations support Hypothesis 3.1 as follows. They show that the four blocks all have a common structure: Up to and including step 5, the differences δQ_t vary among all four blocks. Then, there is a maximal difference in step 6. After that, the values for ΔQ_t and ΔF_t are mostly identical in the first and third and in the second and fourth blocks, leading up to long sequences of trivial steps. The final five steps again differ greatly among all four blocks. We thus conclude that, similar to the attack by Stevens et al., a lower part based on the input IHVs and an upper part were generated separately and then combined.

The conclusion that the upper differential paths are generated by "brute force" while the lower paths are not is supported by Observations 3.9 and 3.10. It is noteworthy that in all the steps from 11 to 61, all four blocks use the highest-probability rotation of δT_t as δR_t. Of course, in the trivial steps, there is only one possible rotation. However, the non-trivial steps 11 up to 25 use the highest probabilities while the steps before sometimes use rotations with smaller probability. As already said, the formulas for the probabilities in Lemma 1.6 are quite inaccurate when there are many bitconditions. The conditional rotation probabilities are mostly quite high, but a few steps have low rotation probabilities. This indicates a brute force approach: For random working states $Q_1, \ldots, Q_{10}$ and $Q'_1, \ldots, Q'_{10}$, we would expect to see *mostly* high-probability rotations, but also some low-probability ones. When extending a differential path deliberately, on the other hand, one would choose high-probability rotations.

In the following subsection, we take a closer look at the steps in the differential paths where we believe that the connection between the upper and lower part occurred. Our findings corroborate the second part of Hypothesis 3.2. We believe that the connection takes place over working states Q_5, Q_6, Q_7 and Q_8 because of Observations 3.7 and 3.8: ΔQ_6, ΔQ_7 and ΔQ_8 appear to belong to the lower differential paths, but ΔF_7 and ΔF_8 vary in all four differential paths, so they do *not* belong to the lower paths. This makes it seem likely that ΔF_7 and ΔF_8 are varied to achieve the appropriate values for ΔQ_7, ΔQ_8 and ΔQ_9. To achieve maximal control over ΔF_7, we need to include steps Q_5 and Q_6 in the connection step. Having maximal differences in ΔQ_6 aids in having many possible alternatives for δF_6 available.

3.3.2 Connection Steps

For analyzing the connection step, we experimented with the algorithm from Section 2.2.7 on the Flame differential paths. Using the C++ library `libhashutil5` from the HashClash project[22], we

[22] https://code.google.com/p/hashclash

first removed all Boolean function bitconditions from the Flame differential paths and added them back in for steps 1 to 4 and steps 9 to 64. We reimplemented the connection algorithm from Section 2.2.7 so that it allows to choose if the bitconditions on Q_6 should be taken from the input lower differential path or if multiple bitconditions that are compatible with δQ_6 should be tried, as in the connection algorithm by Stevens et al. First, we used the values for the δF_i from the original differential paths and compared in how many ways the upper and lower paths could be connected when the bitconditions for Q_6 are taken from the original path and when multiple bitconditions for the working state Q_6 are tried. The results are summarized in Table 3.1.

	Q_6 fixed	Q_6 not fixed
Block 8	2	18
Block 9	2	30
Block 10	8	12
Block 11	12	96

Table 3.1: Number of different possible full differential paths based on the upper and lower paths of the Flame near-collision blocks. The values for the δF_i are taken from the differential paths.

Next, we checked what happens if we compute the δF_i according to Section 2.2.7 instead of taking them from the differential paths. Choosing the δF_i based on the δT_i with the highest probabilities did not allow to connect the upper and lower paths except for block 10.

We then proceeded to check which of the possible choices for the δF_i allow to connect the upper and lower paths and what their corresponding probabilities are, taking the bitconditions on Q_6 from the differential paths. Our results are summarized in Table 3.2.

In blocks 9, 10 and 11, the rotation probabilities are optimal in the sense that it is impossible to choose δF_i with higher rotation probabilities and still connect the upper and lower paths. In the eighth block, δF_5, δF_6 and δF_7 are optimal while δF_8 has a very low probability. This supports the first part of Hypothesis 3.2. If we select random working states for the connection step, we would expect that most but not all of the rotations that occur have high probabilities, but if the connection was done in a more systematic way, we would expect the rotation probabilities to be optimized.

	Choices for δF_is	Optimal probabilities	Actual probabilities
Block 8	4	$\delta T_5 : .2292, \delta T_6 : .4248, \delta T_7 : .8380, \delta T_8 : .4375$	$\delta T_8 : .0625$
Block 9	32	$\delta T_5 : .3989, \delta T_6 : .4580, \delta T_7 : .9607, \delta T_8 : .4683$	Optimal
Block 10	4	$\delta T_5 : .7759, \delta T_6 : .5137, \delta T_7 : .8380, \delta T_8 : .4375$	Optimal
Block 11	8	$\delta T_5 : .7489, \delta T_6 : .5181, \delta T_7 : .9607, \delta T_8 : .0317$	Optimal

Table 3.2: The number of possible choices for the δF_i in the connection step and the optimal and observed rotation probabilities for each near-collision block.

3.3.3 Estimating the Success Probability

In order to get an estimate on the number of upper parts that the authors of Flame needed to try until they found one that would connect to the lower path, we experimentally determined what impact fixing the Q_6 bitconditions has on the connection algorithm by Stevens et al. Memory limitations did not allow us to use the Flame lower differential paths for this since we could not generate enough upper paths.

We used `libhashutil5` to generate 2 million upper paths up to step 4 starting from the intermediate hash value differences before the first near-collision block. We note that these differential paths are *not* based on brute force and are designed to have relatively few bitconditions. This could make them somewhat *easier* to connect than the upper paths in the Flame collision attack. For estimating the success probability when Q_6 is determined by the algorithm, we generated 100000 lower paths down to step 9 from step 18 of the Flame lower path for block 8 and counted how many pairs would connect without Q_6 fixed. For estimating the success probability when Q_6 is fixed, we generated roughly 100000 "Flame-like" lower paths as follows. First, we generated 200000 lower paths using `libhashutil5` and fixed the bitcondition in Q_6 so that they induce the maximal difference in the working states by executing the following algorithm on every path:

1. If δQ_6 is even, throw the path away.
2. For $i = 0, \ldots, 31$, do the following steps.
3. Set the ith bitcondition on Q_6 to `+`.
4. If the ith bit of δQ_6 is 0, set the $(i-1)$th bitcondition on Q_6 to `-`.

Clearly, after this algorithm, every bitcondition on Q_6 is `+` or `-`. To see that the bitconditions induce the correct δQ_6, note that $2^k - 2^{k-1} = 2^{k-1}$. This algorithm left 100074 paths. For the case that Q_6 is not fixed, 9312 out of $2 \cdot 10^{11}$ combinations were successful which corresponds to a success probability of 2^{-24}. In the case that Q_6 is fixed, no combination was successful, so we needed a larger set of inputs to estimate the probability. We generated 10 million upper paths and 2 million lower ones. After executing the algorithm for making the differences in Q_6 maximal, roughly one million was left over. We shuffled the lower paths and the upper paths and split up the lower paths into chunks of size 100000. Processing the first few chunks resulted in 24 successful connections out of $3.44 \cdot 10^{12}$ combinations. This corresponds to a success probability of approximately 2^{-37}. Thus, if the bitconditions on Q_6 are fixed to create maximal differences, we can expect to check roughly 2^{13} times as many lower parts compared to the case where the bitconditions are selected as part of the connection algorithm.

3.4 Tunnels

3.4.1 Tunnel Strengths in the Near-collision Blocks

To estimate the complexity of the Flame collision attack, it is important to know to what extent the attackers used *tunnels*. The tunnels $\mathcal{T}_4$, $\mathcal{T}_5$ and $\mathcal{T}_8$ are the most useful in the differential paths; the others can only have very low strength.[23] Let us recall these three tunnels. If $Q_9[i]$ is free, i.e., $\mathfrak{q}_9[i] = \texttt{.}$ and $Q_9[i]$ is not referenced by any indirect bitcondition, and if we have $Q_{10}[i], Q_{11}[i] = 1$, tunnel $\mathcal{T}_4$ allows us to flip the bit $Q_9[i]$ without changing any message words or working states other than $Q_{22}, \ldots, Q_{64}$ and $m_8, \ldots, m_{10}, m_{12}$. If we have $Q_{10}[i] = 0$ and $Q_{11}[i] = 1$ instead, tunnel $\mathcal{T}_8$ allows us to flip $Q_9[i]$ without changing anything besides $Q_{25}, \ldots, Q_{64}$ and m_8, m_9, m_{12}. If $Q_{10}[i]$

[23]While we can not exclude with certainty that the attackers discovered previously unknown tunnels, the amount of research into the cryptanalysis of `MD5` makes it seem unlikely that more tunnels exist.

is free and $Q_{11}[i] = 0$, we can use tunnel $\mathcal{T}_5$ to flip $Q_{10}[b]$ without changing anything except for $Q_{22}, \dots, Q_{64}$ and $m_9, m_{10}, m_{12}, m_{13}$.

The Flame authors used tunnel $\mathcal{T}_8$, but not to the maximal extent that is possible given the differential paths. This was found by Stevens by inspecting the working states Q_9, Q_{10} and Q_{11} derived from the blocks of the rogue certificate. It is unclear whether tunnels $\mathcal{T}_4$ and $\mathcal{T}_5$ were used. *How* exactly $\mathcal{T}_8$ was used is unclear as well.

Observation 3.11 ([25, Section 3.3])**.** *In the near-collision blocks of the rogue certificate, the strength of the tunnel $\mathcal{T}_8$, in comparison to the maximal strength that is possible in the differential path and the average strength is described in the table below. For the average strength, we assume that in a solution of the differential path, '.'-bitconditions are with equal probability solved by a 0 or a 1. Thus, it is one quarter of the maximal strength.*

Near-collision Block	Observed strength	Maximal strength	Average strength
1	7	17	4.25
2	13	18	4.5
3	10	17	4.25
4	9	18	4.5

Furthermore, we summarize the tunnel strengths of $\mathcal{T}_4$ and $\mathcal{T}_5$:

Observation 3.12. *The strength of tunnel $\mathcal{T}_4$ is given in the following table. We again assume that .-bitconditions are solved by* 0 *or* 1 *with equal probability. In the first and third block, the* 24*th bit of Q_9 is always active for $\mathcal{T}_4$ since the Boolean function bitconditions force $Q_{10}[24], Q_{11}[24] = 1$. Furthermore, in the second and fourth block, we have $\mathfrak{q}_{11}[18] =$ ^ and $\mathfrak{q}_{10}[18] =$., so $Q_9[18]$ is active for $\mathcal{T}_4$ with probability* $1/2$*. In all other positions i that could be active for $\mathcal{T}_4$, we have $\mathfrak{q}_{10}[i], \mathfrak{q}_{11}[i] =$., so these are active with probability* $1/4$.

Near-collision Block	Observed strength	Maximal strength	Average strength
1	4	19	5.5
2	3	20	5.25
3	2	19	5.5
4	3	20	5.25

The strength of $\mathcal{T}_5$ is summarized below. In the second and fourth block, the bitconditions force $Q_{11}[29] = 0$ and thus, $Q_{10}[29]$ is always active for $\mathcal{T}_5$. In all other positions i that could be active for $\mathcal{T}_5$, we have $\mathfrak{q}_{11}[i] =$., so they are active with probability $1/2$.

Near-collision Block	Observed strength	Maximal strength	Average strength
1	8	19	9.5
2	5	20	10.5
3	7	19	9.5
4	10	20	10.5

It is clear that the tunnel $\mathcal{T}_8$ has been used since the observed tunnel strengths are much larger than the average. The smaller-than-average tunnel strength for $\mathcal{T}_4$ and $\mathcal{T}_5$ can be explained by the fact that these three tunnels have conflicting preconditions; we can not conclude that $\mathcal{T}_4$ and $\mathcal{T}_5$ were not used. In our estimate of the complexity, we assume the strengths of these three tunnels to be the average over all four blocks and that all of these three tunnels were used. That is, we assume that tunnel $\mathcal{T}_4$ has strength 3, $\mathcal{T}_5$ has strength 7.5 and $\mathcal{T}_8$ has strength 9.75.

It is unclear how these tunnel strengths were brought about since they are neither average nor maximised. Reconstructing the method would be interesting and could lead to more precise complexity estimates. However, we were not able to find a definitive explanation for the observed tunnel strengths. In the next section, we will describe some attempts at explaining the observed tunnel strengths.

3.4.2 Explanation Attempts

An explanation for Observation 3.11 proposed in [25] is that the Flame authors did not actively try to maximize the tunnel strength but used tunnels in their message block construction algorithm to the extent that they were available. In that case, the algorithm is more likely to output a solution with a higher-than-average tunnel strength. If we have a partial solution up to step 24 and $\mathcal{T}_8$ has strength k in that partial solution, it gives rise to 2^k candidates for a complete solution if we use tunnel $\mathcal{T}_8$. If we do not use tunnels at all, every partial solution has the same chance of being extended to a full solution, independent of the tunnel strength. However, the expected tunnel strength of this method is still too low to explain the observed tunnel strength, as we show below.

Let us estimate the expected tunnel strength of $\mathcal{T}_8$ that would result if the Flame authors indeed used tunnels when they were available, but did nothing to improve tunnel strengths. Let m be the maximal tunnel strength for $\mathcal{T}_8$ that is possible with the differential path. Let the random variable **strength** be the tunnel strength of $\mathcal{T}_8$ in a randomly selected solution for the first 24 steps of the differential path. Let **solve** be the event that a random solution for the first 24 steps gives rise to a solution for the whole path. We assume that $\Pr[\textbf{solve} \mid \textbf{strength} = k] \approx 2^k \cdot p$ for some p independent of k. Furthermore, we assume that

$$\Pr[\textbf{strength} = k] = \binom{m}{k} \cdot \left(\frac{1}{4}\right)^k \cdot \left(\frac{3}{4}\right)^{m-k}.$$

We have $\Pr[\textbf{solve}] = \sum_{i=0}^{m} \Pr[\textbf{strength} = i] \cdot \Pr[\textbf{solve} \mid \textbf{strength} = i]$ and it follows that

$$\Pr[\textbf{solve}] \approx \sum_{i=0}^{m} \binom{m}{i} \cdot \left(\frac{1}{4}\right)^i \cdot \left(\frac{3}{4}\right)^{m-i} \cdot 2^i p.$$

Using Bayes' Theorem, we conclude that a solution to the full path has strength k with probability

$$\begin{aligned}\Pr[\textbf{strength} = k \mid \textbf{solve}] &= \frac{\Pr[\textbf{strength} = k]}{\Pr[\textbf{solve}]} \cdot \Pr[\textbf{solve} \mid \textbf{strength} = k] \\ &\approx \frac{\Pr[\textbf{strength} = k]}{\sum_{i=0}^{m} \Pr[\textbf{strength} = i] \cdot 2^{i-k}}.\end{aligned}$$

The expected value of the tunnel strength of a solution for $m = 17$ is 6.8 and for $m = 18$ it is 7.2. The standard deviation, given by

$$\sigma = \sqrt{\mathbb{E}[\textbf{strength}^2] - (\mathbb{E}[\textbf{strength}])^2}$$

is 2.0 or 2.1, respectively. Although our sample size is very limited, this result makes the above explanation seem unlikely since the observed tunnel strength is always larger than the expected value and, in the case of the second and third block, exceeds the expected value by 2.7 or 1.6 standard deviations, respectively.

However, inspection of the working states in the computation of `MD5` on the rogue certificate revealed that in the first and third block, only seven bits in working states Q_{10} are '1', and in the second and fourth, only 11 bits are '1'. Possibly, the attackers selected the working states Q_{10} like that to improve the chance of bits in Q_9 being active for $\mathcal{T}_8$. Let α be the probability that a bit which could be active for tunnel $\mathcal{T}_8$ is indeed active, so that

$$\Pr[\textbf{strength} = k] = \binom{m}{k} \cdot \alpha^k \cdot (1-\alpha)^{m-k}.$$

We searched for values of α such that the observed values are close to the expectation (in terms of multiples of the standard deviation). To be more precise, we searched for a value of α that minimizes

$$f(\alpha) = \max_{i \in \{1,\ldots,4\}} (|(E_i - s_i)/\sigma_i|)$$

where E_i is the expected tunnel strength in the ith near-collision block, σ_i is the standard deviation of the tunnel strength in the ith block and s_i is the observed tunnel strength. By choosing $\alpha = 0.40$, we can minimize $f(\alpha)$. For this value of α, all observed tunnel strengths are within 1.3 standard deviations. We summarize the results below:

Block	Expectation	Standard deviation (σ)	Observed strength	Distance from expectation
1	9.7	2.0	7	$1.3 \cdot \sigma$
2	10.3	2.1	13	$1.3 \cdot \sigma$
3	9.7	2.0	10	$0.2 \cdot \sigma$
4	10.3	2.1	9	$0.6 \cdot \sigma$

It is also possible that the first and third block have a different probability than the second and fourth. Let α_1 be the probability for the first and third block and α_2 the probability for the second and fourth. We searched for α_1 and α_2 that minimize

$$f_1(\alpha_1) = \max(|(E_1 - s_1)/\sigma_1|, |(E_3 - s_3)/\sigma_3|) \text{ and}$$
$$f_2(\alpha_2) = \max(|(E_2 - s_2)/\sigma_2|, |(E_4 - s_4)/\sigma_4|)$$

For α_1, the value 0.33 minimizes $f_1(\alpha_1)$; the observed strengths in the first and third block are within 0.8 standard deviations from the expected value. For α_2, the value 0.44 minimizes $f_2(\alpha_2)$. With this probability, the observed strengths in the second and fourth block are within 1.0 standard deviations from the expectation. The results of the calculation are summarized below:

Block	Expectation	Standard deviation (σ)	Observed strength	Distance from expectation
1	8.4	2.1	7	$0.7 \cdot \sigma$
2	11.0	2.1	13	$1.0 \cdot \sigma$
3	8.4	2.1	10	$0.8 \cdot \sigma$
4	11.0	2.1	9	$1.0 \cdot \sigma$

Thus, we make the following tentative conjecture which is unfortunately impossible to test conclusively with only one output sample of the attack.

Conjecture 3.13. *In the first and third block, a bit of Q_9 which* could *be active for tunnel $\mathcal{T}_8$ is active with probability* 0.33*. In the second and fourth block, the corresponding probability is* 0.44*.*

Another possible explanation, which is also difficult to test, is that the attackers set some tunnel bits to be active by imposing further bitconditions in the differential paths, but they did not maximise the tunnel strength because they also added some other bitconditions to bring the rotation probabilities close to 1. But in that case, it would seem likely that the same tunnel bits are active in the first and third and the second and fourth block which is not the case. In each block, different tunnel bits are active.

3.5 Differential Path Family

3.5.1 Overview

In this section, we attempt to reconstruct the *family* of differential paths that was used for the Flame collision attack. The difference between the intermediate hash values at the beginning of the first near-collision block for the rogue certificate is $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$ with

$$\begin{aligned}
\delta a &= -2^{5} \\
\delta b &= +2^{30} - 2^{21} - 2^{19} - 2^{17} + 2^{12} - 2^{2} \\
\delta c &= -2^{27} - 2^{20} + 2^{14} + 2^{12} - 2^{5} \\
\delta d &= +2^{9} - 2^{5}
\end{aligned}$$

But for a chosen-prefix attack, the attackers must have been prepared to eliminate any δIHV from some large set. First, a Birthday Search was used to find a value for δIHV that could be eliminated; then, the near-collision blocks were constructed. By reconstructing the family of differential paths and the strategy used for eliminating δIHV, we can get information about the complexity of solving the differential paths. Furthermore, we find out what differences the attackers could eliminate which allows us to give an estimate of the Birthday Search complexity. For our reconstruction attempt, we make the following assumptions:

- The attack uses no more than *four* near-collision blocks. This assumption is justified because the space where the collision blocks were hidden was strictly limited (see 3.1.2 and [22]) so that the collision had to be achieved in four blocks.
- Each differential path for the first (second, third, fourth) near-collision block in the family is similar to the *observed* differential path for the first (second, ...) near-collision block. It is possible that the attackers had also prepared some differential paths that are very different from the observed paths, but obviously we could not say anything about these paths. Let us now make the similarities we assume more precise: We assume that, for each near-collision block in the family, the bitconditions on $Q_{60}, \ldots, Q_{64}$ are the same except for:
  - Boolean function bitconditions in steps 61 to 63.

  - Carries (e.g., stretching `+` to `+---`)
  - Rotations
  - Interchanging '`+`' and '`-`' in $\mathfrak{q}_t[31]$ for $t = 62, 63$.

- δa and δd are fixed constants; $\delta a = -2^5$ and $\delta d = 2^9 - 2^5$. That is, in the Birthday Search, the attackers looked for exactly these differences in δa and δd. We make this assumption because these steps have an extremely low NAF-weight.

In our reconstruction attempt, we describe a family of end segments for near-collision blocks by differential paths. We do *not* claim that the Flame authors described differential paths in the way they are specified here themselves. But we do not know their actual methods and we need some "language" to represent this family of end segments. Also note that our reconstruction remains somewhat speculative since we only have *one* output of the attack.

Before beginning with the reconstruction, let us examine the δIHVs after each near-collision block. After the first block, we have

$$\begin{aligned}
\delta a &= 2^{31} - 2^5 \\
\delta b &= -2^{30} + 2^{25} - 2^{22} + 2^{20} + 2^{17} + 2^9 - 2^2 \\
\delta c &= 2^{31} - 2^{27} + 2^{25} - 2^{20} + 2^9 - 2^5 \\
\delta d &= 2^{31} + 2^{25} + 2^9 - 2^5
\end{aligned}$$

After the second block,

$$\begin{aligned}
\delta a &= 0 \\
\delta b &= -2^{30} - 2^{24} - 2^{20} + 2^{17} - 2^{14} + 2^5 + 2^0 \\
\delta c &= -2^{24} \\
\delta d &= 0
\end{aligned}$$

After the third block, the differences are

$$\begin{aligned}
\delta a &= 2^{31} \\
\delta b &= 2^{25} + 2^{14} + 2^9 + 2^5 - 2^3 + 2^0 \\
\delta c &= 2^{31} + 2^{25} \\
\delta d &= 2^{31}
\end{aligned}$$

and after the fourth block, $\delta a, \delta b, \delta c, \delta d = 0$. These IHVs again show a symmetry between the first pair and the second pair of near-collision blocks. After the first pair, δa and δd are 0 and δc only has a NAF-weight of 1. The NAF-weight of δb increases to 7; before the first near-collision block, it was 6. This leads us to believe that the purpose of the first pair is to eliminate δc except for a constant term. The second pair then eliminates δb and the remaining constant terms.

3.5.2 Reconstructing the End Segments

We begin with an outline of what we assume to be the elimination strategy. The differences in δc are eliminated by the first two blocks, but a difference of -2^{24} is introduced which is then eliminated in the final two blocks. For δb, matters are more complicated. We assume that the changes to δb in the first two blocks are *mostly* random and that the elimination of differences in δb really starts at the third near-collision block. This assumption is justified as follows: Most of the bits in δb that can be affected by the first two blocks can also be affected by the second pair of differential paths. Thus,

- most *deliberate* changes to δb that are possible with the first and second block can just as well be done in the third and fourth, and
- most *random* changes to δb that are possible in the first two near-collision blocks can be *undone* in the third and fourth.

It is therefore for the most part unnecessary to control δQ_{64} in the first two blocks which allows for faster message block construction.

The differential paths eliminate the differences in the IHVs by varying Boolean function bitconditions. In Table 3.3, we summarize several important tuples of bitconditions and the corresponding differences $\Delta F[i]$. These bitconditions are taken from [24, Table C-4]. The most important observation from the table is the following:

Observation 3.14. *If* $\mathfrak{q}_i[j]\mathfrak{q}_{i-1}[j]\mathfrak{q}_{i-2}[j]$ = `+..` *or* `-..` *then we can achieve* any $\Delta F_i[j] \in \{-1, 1, 0\}$ *by replacing the '.'s with appropriate Boolean function bitconditions.*

$\mathfrak{q}_i[j]\mathfrak{q}_{i-1}[j]\mathfrak{q}_{i-2}[j]$	$\Delta F_i[j]$	$\mathfrak{q}_i[j]\mathfrak{q}_{i-1}[j]\mathfrak{q}_{i-2}[j]$	$\Delta F_i[j]$
`+.0`	0	`+++`	+1
`+01`	+1	`++-`	0
`+11`	−1	`+-+`	+1
`-.0`	0	`+--`	0
`-01`	−1	`-++`	0
`-11`	+1	`-+-`	−1
`0+1`	+1	`--+`	0
`?+.`	−1	`---`	+1
`0-1`	−1	`+-0`	+1
`?-.`	+1	`+-1`	0
`-+0`	−1	`-+1`	0
`++0`	−1	`++1`	0
`--0`	+1	`--1`	0

Table 3.3: Several tuples of bitconditions and their associated values for ΔF with respect to the Boolean function over rounds 48-64 of `MD5Compress`.

We now begin with our reconstruction attempt. For each block, we give both a template of a differential path end segment and a description of how it is used to eliminate differences. In Section 3.5.3, we argue that the family we describe is the one used for the Flame attack, based on our assumptions. The end segment templates are split into a *constant* part and a *variable* part. For the variable parts, we use the following additional symbols:

- `D`: A variable differential bitcondition, i.e., $\mathtt{q}_t[i] \in \{\mathtt{.}, \mathtt{+}, \mathtt{-}\}$.
- `B`: A variable Boolean function bitcondition. Precisely which bitconditions are relevant at a given position varies, but in general, we have $\mathtt{q}_t[i] \in \{\mathtt{.}, \mathtt{0}, \mathtt{1}, \mathtt{?}\}$.
- `X`: A non-constant bitcondition, i.e., $\mathtt{q}_t[i] \in \{\mathtt{+}, \mathtt{-}\}$.
- `*`: A (for now) irrelevant differential bitcondition.
- `A`: The same differential as above, $\mathtt{q}_t[i] = \mathtt{q}_{t-1}[i]$.

Note that, since we ultimately only care about the δIHV that a near-collision block causes, the paths of the observed near-collision blocks may differ somewhat from the templates. Adjacent differential bitconditions can in some cases be stretched or shortened by carries: For example, `+` and `+---` cause the same arithmetic difference. Consider the bitconditions $\mathtt{q}_{63}[30]\mathtt{q}_{62}[30]\mathtt{q}_{61}[30] =$ `+BB` in the template for the first near-collision block (Table 3.4). The purpose of the '`B`'s is to make $\Delta F_{63}[30]$ variable. But if we need $\Delta F_{63}[30] = 0$, we might as well change these three bitconditions to `...` and $\mathtt{q}_{63}[29]$ to `+`, as it is the case in the observed first near-collision block. Since we have only one output of the collision attack, we do not know how many carries the attackers would have maximally relied on.

In our templates, we are generous with carries, but in testing the performance, we will use the maximal number of carries as a parameter that might be reduced. These parameters allow us to trade off Birthday Search complexity versus the complexity of solving the path. For example, if we choose not to use the carries in step 62 of the first near-collision block from position 25 to 30, but only to 29, we will not be able to affect position 13 in Q_{63}, so we must require that $c[13] = c'[13]$ in the Birthday Search. Thus, one problem with estimating the Birthday Search complexity of the Flame attack is that we do not know whether the attackers *decided* not to rely on that carry or whether it just so happened that they did not need it for the δIHV that their Birthday Search produced; similar problems arise in the following blocks. Hence, we will have to estimate the complexity for several parameter choices. We can, however, obtain a lower bound on the cost of the Birthday Search and of the message block construction. We can also find a parameter setting that allows for an optimal trade-off between these two costs.

In the first and third block, the carries begin close to position 31. Since they cannot proceed past that point, we include all the carries up to that position in our templates. In the other two blocks, the carries start far away from that position. Since we estimate the complexity of solving the differential paths with Monte Carlo-simulations, we limit the number of carries to what we were able to simulate in reasonable time. In all cases, the maximal number of carries in our templates is larger than the number observed in the observed differential paths. We calculate the Birthday Search complexity under the assumption that all the carries in the template are relied on. It is easy to compute the Birthday Search complexity for a lower number of carries: For each carry that is dropped, the complexity increases by a factor of $2^{0.5}$. In Section 3.6.2, we use Monte Carlo-simulations to estimate the complexity of solving the differential paths. There, we parametrize the complexity in terms of the carries that are relied on.

Also, some of the Boolean function bitconditions adjacent to variable bitconditions might change in the observed near-collision blocks. The first near-collision block can again serve as an example.

In the template, we have $\mathtt{q}_{62}[26]\mathtt{q}_{61}[26]\mathtt{q}_{60}[26] = \mathtt{-11}$ so that we have $\Delta F_{62}[26] = 1$ and hence $\Delta Q_{63}[9] = 1$. But we could also achieve the correct δQ_{63} by putting $\Delta F_{62}[26] = \Delta F_{62}[27] = -1$ and $\Delta Q_{63}[28] = 1$, as it happens in the observed near-collision block.

Below each end segment, we write the change that this block is supposed to make to δc and δb where we color the variable part of this change in blue and underline them. In the first two blocks, we do not write down the possibilities for the random changes to δb. For $i = 0, \ldots, 31$, we let B_i and C_i be variables that may have values in $\{-1, 1, 0\}$. Their values are determined by the variable bitconditions. Note also that when we talk about *solving* the end segments, we are more lax than in other places: We require only the δQ_i to be correct.

Block 1 (Table 3.4)

Steps	Bitconditions
59	`+....... ........ ........ ........`
60	`+BBBB1B. ........ ........ ........`
61	`+BBBB1B. ........ ........ ........`
62	`X+-----. ........ ........ ........`
63	`X.....+. ........ .DDDDD+D ........`
64	`***...+. ...***** ***AAA+A ....****`

$$\delta c : +2^{31} + 2^{25} + 2^{9} + \underline{C_{14} \cdot 2^{14} + \sum_{i=8}^{8+w_1} C_i \cdot 2^i}$$

$$\delta b : +2^{25} + 2^{9} + \sum_{i=8}^{\min(w_1,12)} C_i \cdot 2^i$$

where w_1 is the number of carries from position 25 in step 62 that we rely on and the $C_i \in \{0, 1, -1\}$ are variables.

Table 3.4: End segment of the first near-collision block.

The purpose of this block is to eliminate the differences in Δc at positions $8, \ldots, 8 + w_1$ for a parameter $w_1 \in \{1, \ldots, 5\}$. In the static part, we introduce $+2^{31} + 2^{25} + 2^9$ to δc. These static differences are eliminated by the static parts of the following blocks. We have $\delta Q_{61} = 2^{31}$ and $\delta Q_{62} = 2^{31} + 2^{25}$. The term 2^{25} is introduced via $\delta W_{61} = \delta m_{14} = 2^{15}$ which is rotated by $RC_{61} = 10$. At the next step, we aim for a difference of $\delta Q_{63} = 2^{31} + 2^{25} + \sum_{i=8}^{8+w_1} C_i \cdot 2^i + 2^9$ where the C_i are variable. This difference can be achieved as follows: $2^{31} + 2^{25}$ comes from δQ_{62}. The term $\sum_{i=0}^{8+w_1} C_i \cdot 2^i + 2^9$ is produced by δF_{62} and the rotation of δT_{62} as follows. Since we have the summand 2^{25} in δQ_{62}, we can have $\Delta Q_{62}[25], \ldots, \Delta Q_{62}[30] \neq 0$ and since ΔQ_{61} and ΔQ_{60} are 0 at these positions, we can get *any* value for ΔF_{62} at these positions, as Observation 3.14 shows. Finally, if we have $\Delta Q_{62}[31] = \Delta Q_{60}[31]$, we get $\Delta F_{62}[31] = \pm 1$ and otherwise 0. We thus have $\delta T_{62} = \delta Q_{59} + \delta W_{62} + \delta F_{62} = 2^{31} + \sum_{i=25}^{31} \Delta F_{62}[i] \cdot 2^i$. The rotation with constant $RC_{62} = 15$ then might give us the desired δQ_{63}. The static term $+2^9$ is added to eliminate the -2^9 that will be introduced in the next block. For the most part, we do not aim for any specific difference in δQ_{64}.

The static $+2^{25}$ and $+2^9$ will be eliminated as they are for δc. All differences introduced at the positions marked with $*$ can be eliminated later on.

However, a few differential paths that conform to this template are unsolvable. This is due to the constant term 2^9 in δQ_{63}. Consider the case where $C_9, \dots, C_{8+w_1} = 1$ and $C_8 \geq 0$. Then, we have $2^9 + \sum_{i=8}^{8+w_1} C_i \cdot 2^i \geq 2^{w+1}$ and thus, we need one more carry in step 62 to solve the path. But if $w_1 = 5$, $C_9, \dots, C_{14} = 1$ and $C_8 \geq 0$, the path is impossible to solve. There are several ways around this: We can drop the variable term $C_9 \cdot 2^9$ or we just do not use $w = 5$. But in both of these cases, we lose one bit of freedom in the Birthday Search. We might instead check during the Birthday Search whether the result will force us to take $C_9 = \cdots = C_{14} = 1$ and, if yes, search for a new result. This increases the Birthday Search complexity somewhat, but not as much as losing one bit of freedom.

Furthermore, we might want to reduce the number of irrelevant bitconditions at $\mathfrak{q}_{64}[0], \dots, \mathfrak{q}_{64}[3]$ because a higher number forces a higher parameter value in the fourth near-collision block to eliminate the random differences. For $i = 0, \dots, 4$, we can set $\mathfrak{q}_{61}[i - 21 \bmod 2^{32}] = \texttt{0}$ to ensure that $\Delta Q_{64}[i] = 0$. Let v_1 be the maximal number ≤ 4 such that we allow $\Delta Q_{64}[v_1] \neq 0$. We will estimate the solution complexity depending on this parameter.

In the observed near-collision block, we have three carries from position 25 in ΔQ_{63}. Assuming that the template was followed strictly, the parameter choice in the attack must have been $w_1 \geq 4$ since 4 is the smallest value for w_1 with which we can achieve the correct δQ_{63}. In the observed path, we have $\delta Q_{63} = 2^{31} + 2^{25} - 2^{14} - 2^{12} + 2^9$. Thus, we need $C_8, \dots, C_{11} = 0$. To achieve the term -2^{12}, we need at least C_{12} to be non-zero. But depending on how exactly the Birthday Search was handled, the attackers might have relied on only 2 carries since $-2^{12} + 2^9 = -2^{11} - 2^{10} - 2^9$. The ΔQ_{64} of the observed path does not conform to our template, but only the underlying δQ_{64} is relevant and it can be represented by a $\widehat{\Delta Q_{64}}$ which does conform to the template. The parameter v_1 in the Flame collision attack is at least 1.

Block 2 (Table 3.5)

We have $\delta Q_{61} = 2^{31} + 2^5$. The 2^5 comes from $\delta T_{60} = 2^{31}$ and rotation constant $RC_{60} = 6$. In the next step, $\delta Q_{62} = 2^{31} - 2^{25} - 2^9 + 2^5$. Here, -2^{25} is introduced via $\delta m_{14} = -2^{15}$ and -2^9 comes from $\delta T_{61} = 2^{31}$ and rotation constant $RC_{61} = 10$. This allows us to have $\Delta Q_{62}[i] \neq 0$ for $i \geq 5$. Potentially, having carries up to position 25 could provide useful variations in δQ_{63}. It is, however, very unlikely that that many carries occur. In our template, we have three carries from position 5 and six from position 9 which is the maximal value for which we did Monte Carlo-simulations. Next, we have $\delta Q_{63} = 2^{31} + \sum_{i=20}^{24+w_2} C_i \cdot 2^i - 2^{26} + 2^{25} - 2^9 + 2^5$. The constant terms $-2^{26} + 2^{24}$ are derived from combining the term -2^9 in δF_{62} (rotated to -2^{24}) and -2^{25} in δQ_{62}. We can choose $C_{20}, \dots, C_{24+w_2}$ from the set $\{-1, 1, 0\}$ using the variable Boolean function bitconditions at positions $5, \dots, 9 + w_2$. For δQ_{64}, most bitconditions are either constant, irrelevant or inherited from Q_{63}. We have some choice over positions 20, 27, 28 and 29. Namely, $\pm 2^{20}$ might be added due to different rotations and we can adjust the Boolean function bitconditions at positions 6, 7 and 8 to manipulate positions 27, 28 and 29. This might require introducing additional carries to the differential bitconditions $\mathfrak{q}_{63}[5]$ and $\mathfrak{q}_{63}[6]$. Also, for $i = 20, \dots 23$, if $\mathfrak{q}_{63}[i] \neq$., we require that $\mathfrak{q}_{61}[i] = \texttt{0}$. The purpose of this is to avoid any differences in positions 10, 11 and 12 since we are unable to make any corrections at these positions in the later blocks. The positions marked with a $*$ can be corrected in later blocks. We again parametrize the number of irrelevant bitconditions

Steps	Bitconditions
59	`+....... ........ ........ ..0.....`
60	`+....... ........ BBBBBB1B BBB.....`
61	`-....... 0000.... BBBBBB1B BB+.....`
62	`+.....-. ........ -+++++++ ---.....`
63	`XDDDD-D+ DDDD.... ......-B B+-.....`
64	`**DDD-A+ AAAD**** ***...-. ..+*****`

$$\delta c : +2^{31} - 2^{26} + 2^{24} - 2^9 + 2^5 + \underline{\sum_{i=20}^{24+w_2} C_i \cdot 2^i}$$

$$\delta b : -2^{26} + 2^{24} - 2^9 + 2^5 + \sum_{i=20}^{\min(24+w_2,28)} C_i \cdot 2^i + \underline{\sum_{i=27}^{29} B_i \cdot 2^i + B_{20} \cdot 2^{20}}$$

where w_2 is the number of carries from position 9 in step 62.

Table 3.5: End segment of the second near-collision block.

starting at position 0. We let v_2 be the maximal number ≤ 4 such that we allow $\Delta Q_{64}[v_2] \neq 0$.

Strictly following the template, the δQ_{63} of the observed differential path requires $w_2 \geq 3$. We have $\delta Q_{63} = 2^{31} + 2^{27} - 2^{26} + 2^{24} + 2^{20} - 2^9 + 2^5$ and thus we need $C_{24}, C_{26} = 0$ and $C_{27} \neq 0$. However, it could be possible that the attack relied on only two carries since $2^{27} - 2^{24} = 2^{26} + 2^{25} + 2^{24}$.

After these two blocks, we want to have $\delta c = -2^{24}$. That is, we want to essentially eliminate δc with the first two blocks – the -2^{24} is cancelled in the constant parts of the next two blocks. Let us now move on to eliminate δb.

Block 3 (Table 3.6)

We have $\delta Q_{59} = \delta Q_{60} = \delta Q_{61} = 2^{31}$, $\delta W_{61} = m_{11} = 2^{15}$ and $\Delta F_{61} = ()$. This results in $\delta T_{61} = 2^{31} + 2^{15}$. With rotation constant $RC_{61} = 10$, we can aim for $\delta Q_{62} = 2^{31} + 2^{25} + 2^9$. The static bitconditions give us $\delta F_{62} = 2^9$. Thus, $\delta T_{62} = 2^{31} + 2^9$. Rotating with $RC_{62} = 15$, we can get $\delta R_{62} = -2^{14} + 2^{24}$. Hence,

$$\delta Q_{63} = \delta Q_{62} + \delta R_{62} = 2^{31} + 2^{25} + 2^{24} - 2^{14} + 2^9 = 2^{31} + 2^{26} - 2^{24} - 2^{14} + 2^9.$$

We can have up to 4 carries from position 26. Let w_3 be the number of carries we rely on. In the next step, the variable Boolean function bitconditions allow us to control positions 13 to $15 + w_3$ in δQ_{64}. Due to the static bitconditions, we also get $2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9 - 2^3$.

If we have $B_{14}, \ldots, B_{15+w_3} = 1$ and $B_{13} \leq 0$, we have $-2^{14} + \sum_{i=13}^{15+w_3} B_i \cdot 2^i \leq -2^{15+w_3+1}$. To solve this, we need one more carry from position 26. In the case of $w_3 = 4$, this is not possible, but we can introduce a term 2^{20} via the rotation to eliminate this problem. For the δQ_{64} of the observed path, we need $w_3 = 4$. Note, however that in the observed near-collision block, a lesser number of carries occurred: The correct result was achieved with the help of a term 2^{20} from the rotation. The variable portion of δQ_{63} takes on the value $2^{19} + 2^{18} + 2^{17} + 2^{16} - 2^{14} = 2^{20} - 2^{16} - 2^{14}$. However, such a reduction of the number of carries can only occur for *specific* values of the B_i: It

Steps	Bitconditions
59	+....... ....
60	-.....0. ....1. ....
61	+BBBBBBB .... .1....0. ....
62	+BBBBB+B .... .0....+. ....
63	X+----B- ....-....+. ....
64	.+...+.- ....DDDD D-D...+. ....-...

$$\delta c : +2^{31} + 2^{26} - 2^{24} - 2^{14} + 2^{9}$$

$$\delta b : +2^{30} + 2^{26} - 2^{24} + 2^{9} - 2^{3} + \underline{\sum_{i=13}^{15+w_3} B_i \cdot 2^i}$$

where w_3 is the number of carries from position 26 in step 63.

Table 3.6: End segment of the third near-collision block.

is only possible when $B_j = B_{j+1} = \cdots = B_{19}$ for some $j < 19$. But the parameter w_3 must be fixed *before* the Birthday Search and we do not know beforehand which values for B_i the result of the Birthday Search will require.

Block 4 (Table 3.7)

Steps	Bitconditions
59	-....... ....
60	+.....0. ....0. ....
61	+.....1. ...BBBBB 11BBBBB. ....
62	-.....-. ...BBBBB 00BBBB-. ....
63	X.....-. ...+---- ---++++. ....
64	DD....-. ....+....-D DD-D+DDD

$$\delta c : +2^{31} + 2^{24}$$

$$\delta b : -2^{24} + 2^{14} - 2^{9} - 2^{5} + 2^{3} + \underline{\sum_{i=30}^{\min(u_4,1)} B_i \cdot 2^i + \sum_{i=0}^{u_4-2} B_i \cdot 2^i + \sum_{i=3}^{3+w_4} B_i \cdot 2^i}$$

where w_4 is the number of carries from position 14 in step 63 and u_4 is the number of carries from position 9.

Table 3.7: End segment of the fourth near-collision block.

As in the previous block, we have $\delta Q_{59} = \delta Q_{60} = \delta Q_{61} = 2^{31}$. With $\delta W_{61} = m_{11} = -2^{15}$ and $\delta F = 2^{31}$, we can get $\delta Q_{62} = 2^{31} - 2^{25} - 2^{9}$. The static bitconditions give us $\delta F_{62} = 2^{31}$ and hence we can get $\delta Q_{63} = 2^{31} - 2^{24} + 2^{14} - 2^{9}$. We can thus have $\Delta Q_{63}[9], \ldots, \Delta Q_{63}[14 + w_4] \neq 0$.

Note that carries up to position 23 could be useful. We limited the carries to position 20 since this is the maximal value for which we could perform Monte Carlo-simulations in reasonable time. The variable bitconditions and the possible +1-term from the rotation allow us to affect positions $0, \dots, 3 + w_4, 30, 31$ in ΔQ_{64}. From static bitconditions, we get $-2^{24} + 2^{14} - 2^9 - 2^5 + 2^3$. We parametrize the number of carries from position 9 and 14 separately. The parameter u_4 has to be large enough to allow us to remove the differences from the Birthday Search *and* the random differences from the first two blocks; that is, if $\max(v_1, v_2) \leq 2$, we must have $u_4 \geq \max(v_1, v_2) + 2$ and otherwise, $u_4 = 4$. We need at least two carries from position 14 for the constant terms $-2^5 + 2^3 = -2^4 - 2^3$. In the observed near-collision block, there are only two carries from position 9. The parameter u_4 used in the attack must be at least 3, even though in the end, one carry sufficed. Also, in the observed differential path, the carries occur in step 62 rather than 63, but the same value for δQ_{64} can be achieved either way.

Comparison with Stevens et al.

The differential paths used in the attack by Stevens et al., are quite different: They always target δb, δc and δd, but the differences in $\Delta Q_{62}, \dots, \Delta Q_{64}$ are very *local* in the sense that for most i, $\Delta Q_t[i] = 0$ and indices i with $\Delta Q_t[i] \neq 0$ lie in a small window of width w, were w is a parameter of the attack. Also, by using a variety of possible message block differences, they are more flexible in the δIHV that they can eliminate. The downside is that we either need a large number of differential paths – which was not an option in the Flame attack – or we need to find a low-weight δIHV in the Birthday Search. Nevertheless, it appears that the attack by Stevens et al. has a lower cost when the right parameters are used; see Section 3.6.3.

3.5.3 Arguments for the Reconstruction

We will now argue for each block how the template derives from our assumptions and the observed differential paths.

Block 1

Given our constraints, the first opportunity to deviate from the observed differential path is in step 62. We have $\Delta Q_{62} = 2^{31} + 2^{25}$. The bitconditions from position 25 have 4 carries in the observed differential path and there can be at most one more. We included this additional carry in the template. The bitconditions above $\mathfrak{q}_{62}[25], \dots, \mathfrak{q}_{62}[30]$ can then be varied to achieve different values for δF_{62} and hence for δQ_{63}. Since there are no other differential bitconditions, imposing Boolean function bitconditions anywhere else in steps 61 and 62 is useless. Varying the Boolean function bitconditions, the bitcondition $\mathfrak{q}_{62}[31]$ and the two possible rotations[24], we can get precisely those values for δQ_{63} that we give in the template (except for the rare impossible case, when $w_1 = 5$ and $C_9 = \dots C_{14} = 1$). We assume that the positions marked with a $*$ are irrelevant because we can compensate any differences in these positions in later blocks, for appropriate parameter choices.

[24]Since we have, in the terminology of Lemma 1.6, $T_{62,low} = 0$, only two rotations have non-zero probability. We have $\delta R_{62} \in \{RL(\delta T_{62}, RC_{62}), RL(\delta T_{62}, RC_{62}) - 2^{RC_{62}}\}$.

Block 2

Here, the first opportunity to deviate from the observed path is again in step 62. We have $\delta Q_{62} = 2^{31} - 2^{25} - 2^{9} + 2^{5}$. It makes no sense to use any carries from position 25 since we already used them in the previous block. Anything we could do with the help of these carries we could have done in the previous block. We have 2 carries from position 5 in the observed path. In our template, we rely on one more carry, bridging the gap to position 9. The number of carries from position 9 in the observed path is 3. We rely on at most 6 carries from position 9. The number of carries in the template is probably larger than the number of carries the Flame authors relied on. We use the number of carries from this position as a parameter when estimating the complexity. These carries allow us to achieve any of the values for δQ_{63} given in the template. Since we already handled position 14 of δc in the first block, we do not need to vary $\mathfrak{q}_{62}[31]$. In contrast to the first block, we can get an additional +1-term due to the rotation. However, we did not include this in our template since it would not be possible to get a -1-term. The positions in step 64 that are marked with a $*$ can take on differential bitconditions, but are irrelevant because we can handle them in the following blocks. We set $\mathfrak{q}_{61}[i] = \mathtt{0}$ for $i = 20, \ldots, 24$ so that no changes in δb happen at positions $9, \ldots, 12$. We *could* possibly use variable bitconditions to alter these positions, but we do not do this in the template for two reasons: First, this would *require* that we have a non-constant bitcondition at, e.g., $\mathfrak{q}_{63}[20]$. Second, the term 2^{12} appears both in the observed value for δb and δc. No other terms 2^{i} appear in the NAF of both δb and δc. This might indicate that in the Birthday Search target, δb and δc are *supposed* to be identical at this position.

Block 3

In the template, we assume that after Block 2, we have essentially eliminated δc except for the constant term -2^{24}. Thus, our templates do not allow any variable differential bitconditions in step 63. This assumption is justified because everything we could do in Block 3 to affect δc can also be done in one of the previous blocks. In step 63, we expanded the number of carries from position 26 to the maximum. We did not add any carries from position 14. Such carries occur in the observed path for Block 4, but not in Block 3 and thus, we assume that the strategy was to use these carries only in Block 4. It does not make sense to use them in two blocks. Using variable Boolean function bitconditions over positions 24 to 30 and varying $\mathfrak{q}_{63}[31]$ allows us to make the changes to δb given in the template. We assume that the bitconditions above position 13 are constant. They add the term -2^{3} to δb. In Block 4, $-2^{4} - 2^{3} = -2^{5} + 2^{3}$ will be added so that the constant term 2^{5} from Block 2 is eliminated.

Block 4

Here, our template differs from the observed path in step 62: The carry from position 9 in the observed path could be replaced with a carry from that position in the next step and achieve a larger set of possible values for δQ_{64}. Thus, we assume that the carry in step 62 is just a coincidence and that a carry in step 63 was intended instead. But as long as the correct result is achieved, it does not matter if the differential path is followed exactly. We again allow for some more carries in step 63 than there are in the additional path. We assume that there is a constant term $-2^{4} - 2^{3} = -2^{5} + 2^{3}$ to cancel the term 2^{5} from Block 2 together with 2^{3} from Block 3.

3.5.4 Birthday Search

Given the elimination strategy, we can now specify the Birthday Search target. We require that there are fixed differences in δa and δd and that for those bit positions i that we can not manipulate in our four near-collision blocks, we need $c[i] = c'[i]$ or $b[i] = b'[i]$ (modulo some constants). The Birthday Search looks for a collision of the function

$$\begin{aligned} f(x) &= (a, \tilde{b}_{10}, \ldots, \tilde{b}_{13}, \tilde{b}_{21}, \ldots, \tilde{b}_{26}, c_0, \ldots, c_7, c_{15}, \ldots, c_{19}, c_{31}, d) \\ \text{where } (a,b,c,d) &= \begin{cases} \texttt{MD5Compress}(IHV, B\|x) + \left(-2^5, 0, -2^5, 2^9 - 2^5\right) & \text{if } x \text{ is even} \\ \texttt{MD5Compress}\,(IHV', B'\|x) & \text{if } x \text{ is odd} \end{cases} \\ \text{and } \tilde{b} &= b - c \end{aligned}$$

with IHV and IHV' the intermediate hash values after processing the two chosen prefixes. Since the range of f has 88 bits, we can expect to find a collision after $\sqrt{\pi \cdot 2^{88}/2} \approx 2^{44.3}$ calls to the compression function. Not every collision is useful, of course. The probability p that a collision is useful is upper bounded by 1/2 since we require that one part of a useful collision is even and the other is odd. It is unlikely that p is much lower than 1/2 since the only other source for useless collisions appears to be the impossible differential paths for the first block. Thus, the expected number of compression function calls until a useful collision is found is

$$\sqrt{\frac{\pi \cdot 2^{88}}{2 \cdot p}} \approx \sqrt{\pi} \cdot 2^{44} \approx 2^{44.8}.$$

Even if p is much lower, $2^{44.8}$ is a lower bound on the Birthday Search cost.

As we already mentioned, we can choose not to rely on some of the carries in the templates so that the differential paths are solved more quickly. For every carry we do not rely on, we introduce another bit position where b and b' or c and c' may not differ, increasing the Birthday Search complexity by a factor of $2^{0.5}$. This allows us to trade off Birthday Search complexity against complexity in the message block construction.

The Birthday Search in the attack by Stevens et al. is different from our reconstruction of the Flame attack. In the attack by Stevens et al., we do not inspect individual bits in δIHV, but instead, we only require that $\delta a = 0$, $\delta c = \delta d$ and possibly that $\delta b - \delta c \bmod 2^k = 0$ for some parameter k. It is not important which bits in c and c' or b and b' are different; it is important whether we can eliminate all the differences using a given number r of near-collision blocks. Thus, the probability p contributes the majority of the cost. This probability and the resulting cost depend on many different parameters. If we allow $r = 4$ near-collision blocks, like in the Flame attack, $w = 5$ and allow an expected memory use of 2 GB, we can achieve a cost of $2^{44.55}$.

In contrast, the Flame Birthday Search requires *specific* positions to have no difference. This contributes the majority of the cost, while the probability p that a collision is useful is 1/2, or slightly smaller if we have to avoid the impossible differential paths in the first near-collision block.

3.6 Estimating the Cost of Constructing Message Blocks

3.6.1 A Formula for the Expected Cost

We now estimate the cost of generating a near-collision block. Since the bitconditions are concentrated on the first 16 working states and the tunnel $\mathcal{T}_8$ is used, we assume that the algorithm can be broken down into the following steps:

1. Generate a full differential path/generate a set of initial working states that connects to the lower differential path.

2. Select $Q_1, \dots, Q_{16}$ according to the path and according to tunnel requirements.

3. Try to obtain a solution up to step 24[25] with the help of tunnels $\mathcal{T}_4$ and $\mathcal{T}_5$. Return to step 2 of this algorithm and choose different Q_i if it is not possible to obtain a solution. Use early abort to reduce the cost of this step. The message words are computed as needed, like in the attack by Stevens et al.

4. Attempt to generate a solution for the whole path from our solution up to step 24 using tunnel $\mathcal{T}_8$. We use early abort to some extent. We do *not* try to solve the differential path exactly since we only care about the values of $\delta Q_{61}, \dots, \delta Q_{64}$, but we check whether $\delta Q_{35} = \cdots = \delta Q_{60} = 2^{31}$ and we abort if that is not the case.

5. Check if the values for $\delta Q_{61}, \dots, \delta Q_{64}$ are correct. If yes, we have found a solution, if not, we have to continue searching.

We will estimate the expected number of calls to the `Step`-function of `MD5Compress` that the algorithm makes. Dividing this number by 64 gives us the equivalent cost in terms of `MD5Compress`-calls. Let $\mathcal{C}_{path}$ be the cost of generating the full differential path. We do not know the exact value of $\mathcal{C}_{path}$, but we can estimate the costs that the other steps of the message block construction algorithm contribute.

In step 2, we can choose working states Q_i without computing the step-function. But we also need to verify whether the rotations in these steps are correct. Since we do not know how the states $Q_1, \dots, Q_{16}$ are selected – maybe the attackers introduced additional bitconditions to improve the rotation probabilities, as in the attack by Stevens et al. (see Section 2.2.9 and [24, Section 6.3.3]) – we assume that this cost is negligible compared to the cost of the other parts. For each t such that $16 \leq t \leq 24$, we estimate the expected cost $\mathcal{C}_t$ of generating a solution up to step t. Let p_t be the probability that a solution up to step t is a solution up to step $t+1$. For b_t the number of bitconditions on step $t+1$ and r_t the rotation probability of step $t+1$, we have $p_t = 2^{-b_t} \cdot r_t$. We let s_t be the combined strength of the tunnels that are applicable to step t, i.e., the tunnels that only change working states $Q_{t+1}, \dots, Q_{64}$. The expected *amortized* cost for generating a solution up to step t is

$$\mathcal{C}'_t = \frac{\mathcal{C}_t + 2^{s_t} - 1}{2^{s_t}}$$

which can be seen as follows: Using the tunnels, if we have *one* solution up to step t, we can generate 2^{s_t} solutions up to step t. But for each solution beyond the initial one, we have to recompute one message word.[26] The cost of this recomputation per message word is equivalent to the cost of `Step`. Thus, we can generate 2^{s_t} solutions up to step t at an expected cost of $\mathcal{C}_t + 2^{s_t} - 1$ and therefore

[25]Recall that we say that a pair of inputs solves a path up to step t if it agrees with the bitconditions $\mathfrak{q}_{-3}, \dots, \mathfrak{q}_t$ and with the δQ_{t+1} from the differential path.

[26]We have to recompute only one word because we compute the message words as they are needed. The other words that might be changed by the tunnel are not computed at this point.

the above formula for the expected amortized cost holds. If there are no applicable tunnels, we have $s_t = 0$ and $\mathcal{C}'_t = \mathcal{C}_t$. We estimate the expected cost of finding a solution up to step t as

$$\mathcal{C}_t = \begin{cases} 0 & \text{for } t = 16 \\ \frac{1}{p_{t-1}} \cdot \left(\mathcal{C}'_{t-1} + 2\right) & \text{for } 16 < t \leq 32 \\ \frac{1}{p_{t-1}} \cdot \left(\mathcal{C}'_{t-1} + 1\right) & \text{for } t > 32 \end{cases}$$

This estimate is justified as follows: We have a probability of p_{t-1} that a solution up to step $t-1$ is a solution up to step t. Thus, we expect that we need to examine $1/p_{t-1}$ solutions up to step $t-1$ until we find a solution up to step t. The expected amortized cost of generating a solution up to step $t-1$ is $\mathcal{C}'_{t-1}$. We need to compute one additional step of `MD5Compress` to determine whether a solution up to step $t-1$ is a solution up to step t. For $t \leq 32$, we need to compute one additional message word in order to do this. After step 32, we already have the whole message block. The cost of computing a message word is equivalent to one execution of `Step`.

The probabilities $p_{16}, \ldots, p_{23}$ are roughly equal for each of the four differential paths. They are given in the following table.

t	16	17	18	19	20	21	22	23
p_t	$2^{-6.0}$	$2^{-3.0}$	$2^{-2.2}$	$2^{-2.0}$	$2^{-3.0}$	$2^{-2.0}$	$2^{0.0}$	$2^{-1.0}$

Furthermore, averaging over the tunnel strengths in all four observed near-collision blocks, we have $s_{21} \approx 10.8$, $s_{24} \approx 9.8$ and $s_t = 0$ for all $t \neq 21, 24$. Thus, we have

$$\begin{aligned} \mathcal{C}_{17} &= \mathcal{C}'_{17} \approx 2^{7.0} \\ \mathcal{C}_{18} &= \mathcal{C}'_{18} \approx 2^{10.0} \\ \mathcal{C}_{19} &= \mathcal{C}'_{19} \approx 2^{12.2} \\ \mathcal{C}_{20} &= \mathcal{C}'_{20} \approx 2^{14.2} \end{aligned}$$

Then, $\mathcal{C}_{21} \approx 2^{17.2}$. The amortized cost is $\mathcal{C}'_{21} \approx 2^{6.4}$. Continuing on towards step 24, we have

$$\begin{aligned} \mathcal{C}_{22} &= \mathcal{C}'_{22} \approx 2^{8.5} \\ \mathcal{C}_{23} &= \mathcal{C}'_{23} \approx 2^{8.5} \end{aligned}$$

and $\mathcal{C}_{24} \approx 2^{9.5}$. The amortized cost is $\mathcal{C}'_{24} \approx 2^{0.9}$.

Since we have only trivial bitconditions with $\delta Q_i = 0$ from step 24 to step 33, we have $\mathcal{C}'_{33} = \mathcal{C}_{33} = \mathcal{C}'_{24} + 17 \approx 2^{4.2}$. In the next step, we have one non-constant bitcondition and a rotation probability of 1/2, so we get $\mathcal{C}_{34} = \mathcal{C}'_{34} \approx 2^{6.4}$. Then, there is a sequence of trivial steps with $\delta Q_i = 2^{31}$. From now on, we relax our definition of solving a differential path. We require only that the δQ_i are correct. By Theorem 1.14, part 3, we have $\delta Q_i = 2^{31}$ with certainty up to $i = 47$. Thus, $\mathcal{C}_{47} = \mathcal{C}'_{47} = \mathcal{C}_{34} + 13 \approx 2^{6.6}$. For the steps after that, Theorem 1.14, part 4, tells us that $\delta Q_i = 2^{31}$ implies $\delta Q_{i+1} = 2^{31}$ as long as $\Delta Q_i[31] = \Delta Q_{i-2}[31]$ and $\delta W_t = 0$ or $\Delta Q_i[31] \neq \Delta Q_{i-2}[31]$ and $\delta W_t = 2^{31}$. We assume that the condition of Theorem 1.14, part 4, holds with probability 1/2. This is justified as follows: Assuming that Q_i is selected uniformly at random and $Q'_i = Q_i + 2^{31}$, we have $\Delta Q_i[31] = 1$ or -1 with probability 1/2 each and exactly one of these two alternatives satisfies the condition of Theorem 1.14, part 4. We thus get $\mathcal{C}_{i+1} = 2 \cdot (\mathcal{C}_i + 1)$ for $47 \leq i < 59$.

That gives us $\mathcal{C}_{59} = 2^{19.6}$. If the path is solved up to step 59 (modulo the sign in $\Delta Q_i[31]$), we have $\delta Q_{57} = \cdots = Q_{60} = 2^{31}$. Let the random variable **attempts** be the number of input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ that we need to evaluate until we find an input pair that has the desired values for $\delta Q_{61}, \ldots, \delta Q_{64}$. The expected cost of finding a solution with the correct values for $\delta Q_{61}, \ldots, \delta Q_{64}$ is then $\mathbb{E}[\textbf{attempts}] \cdot (\mathcal{C}_{60} + 4) \approx \mathbb{E}[\textbf{attempts}] \cdot 2^{19.6}$. In terms of calls to `MD5Compress`, the complexity of generating a near-collision block is

$$\mathcal{C}_{block} = \mathcal{C}_{path} + \mathbb{E}[\textbf{attempts}] \cdot 2^{13.6}$$

since `MD5Compress` consists of $64 = 2^6$ steps. We will estimate $\mathbb{E}[\textbf{attempts}]$ in the next section using Monte Carlo-simulations.

3.6.2 Estimating the Expected Number of Attempts at Solving the End-Segment

In this section, we want to estimate the expected number of input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ we have to generate until an input pair with the right value for $\delta Q_{61}, \ldots, \delta Q_{64}$ is found – we will call this value the *target* of the differential path and we will call input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ *admissible* input pairs. Of course, some targets are easier than others. Therefore, we estimate the expected number of attempts required for a target sampled uniformly at random from the set of targets that a given template from Section 3.5.2 allows. We estimate $\mathbb{E}[\textbf{attempts}]$ separately for each of the four templates and several parameter values.

Let **target** be the random variable that gives the target of our differential path. We assume that for every possible target τ there is a probability p_τ such that each admissible input pair that we generate hits the target τ with probability p_τ, independent of the others. Let $\textbf{attempts}_t$ be the random variable that counts the number of admissible inputs we have to generate until target τ is solved. Then, $\textbf{attempts}_\tau$ is distributed geometrically with $\Pr[\textbf{attempts}_\tau = k] = \Pr[\textbf{attempts} = k \mid \textbf{target} = \tau] = (1 - p_\tau)^{k-1} p_\tau$ and $\mathbb{E}[\textbf{attempts}_\tau] = \mathbb{E}[\textbf{attempts} \mid \textbf{target} = t] = p_\tau^{-1}$. By the law of total probability, we can sample **attempts** by first sampling $\tau \leftarrow \textbf{target}$ and then sampling $\textbf{attempts}_\tau$.

To save time in our simulations, we do *not* generate admissible inputs as in Section 3.6. Instead, we select working states $Q_{57}, \ldots, Q_{60}$ and message words $m_0, \ldots, m_{15}$ at random and compute $Q'_{57}, \ldots, Q'_{60}$ and $m'_0, \ldots, m'_{15}$ by applying the appropriate arithmetic differentials. Proceeding like this requires the assumption that the probability for hitting the target with random $Q_{57}, \ldots, Q_{60}$ and message words is the same as hitting the target when proceeding according to the method in Section 3.6. This assumption is justified by the pseudo-randomness of `MD5`.

As a first step, we estimate the distribution of $\Delta Q_{57}[31], \ldots, \Delta Q_{60}[31]$ given that the differential path is followed up to step 30 using the following algorithm.

1. Choose random values for $Q_{31} = Q'_{31}, \ldots, Q_{34} = Q'_{34}$ and $m_0, \ldots, m_{15}$.
2. Let $m'_i = m_i + \delta m_i$ where δm_i are taken from the Flame differential paths. That is, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = \pm 2^{15}$ and $\delta m_i = 0$ for $i \notin \{4, 11, 14\}$.
3. Compute the `MD5` step function up to step 60.
4. Check if we have $\delta Q_{57}, \ldots, \delta Q_{60} = 2^{31}$. If not, return $\perp$ and exit.
5. Return $\Delta Q_{58}[31], \ldots, \Delta Q_{60}[31]$.

We ran this procedure (with $\delta m_{11} = 2^{15}$) until every possible result of step 5 occurred at least 20000 times and took the relative frequency of these results as our probability estimate. We then repeated this with $\delta m_{11} = -2^{15}$. In both cases, we observed that the distributions were roughly uniform. Table 3.8 lists our results. We say that a run was *successful* if it did not end in step 4. We represent the outputs of the algorithm as the bitconditions $\mathfrak{q}_{58}[31]\mathfrak{q}_{59}[31]\mathfrak{q}_{60}[31] \in \{\texttt{+++}, \texttt{++-}, \texttt{+-+}, \ldots, \texttt{---}\}$. The relative frequency of successes is 2^{-13} which is consistent with our estimate that each of the trivial differential steps from step 48 to 60 are successful with probability 1/2.

	$\delta m_{11} = 2^{15}$	$\delta m_{11} = -2^{15}$
Runs	1328850000	1317580000
Successful	162716	161044
+++	20255	20124
++-	20506	20055
+-+	20560	20140
+--	20458	20052
-++	20304	20154
-+-	20000	20321
--+	20307	20198
---	20326	20000

Table 3.8: Outcomes of the Monte Carlo simulation to find the distribution of $\Delta Q_{58}[31], \ldots, \Delta Q_{60}[31]$.

In the experiment for estimating $\mathbb{E}[\textbf{attempts}]$, we run Algorithm 3.1 until a fixed number `SUCCESSES` of targets is solved.

Algorithm 3.1: Monte Carlo-simulation

```
s = 0;
trials = 0;
while $s < SUCCESSES$ do
    Sample $\tau \leftarrow$ target;
    repeat
        trials = trials + 1;
        Randomly select 32-bit words $Q_{57}, \ldots, Q_{60}$ and $m_0, \ldots, m_{15}$;
        for $i = 57, \ldots, 60$ do
            $Q'_i = Q_i + 2^{31}$
        for $i = 0, \ldots, 15$ do
            $m'_i = m_i + \delta m_i$
        Compute $Q_{61}, \ldots, Q_{64}$ and $Q'_{61}, \ldots, Q'_{64}$ according to MD5Compress;
        for $i = 61, \ldots, 64$ do
            $\delta Q_i = Q'_i - Q_i$
    until $\delta Q_{61}, \ldots, \delta Q_{64}$ solve target $\tau$;
    $s = s + 1$;
return trials/SUCCESSES
```

Every iteration of the repeat-until-loop in this algorithm corresponds to a sampling of **attempts**. Thus, for large values of `SUCCESSES`, the output of this algorithm approximates $\mathbb{E}[\textbf{attempts}]$ with high probability.

Block 1

The target differences for the first near-collision block with parameters $1 \leq w_1 \leq 5$ and $-1 \leq v_1 \leq 3$ are

$$\begin{aligned}
\delta Q_{61} &= 2^{31} \\
\delta Q_{62} &= 2^{31} + 2^{25} \\
\delta Q_{63} &= 2^{31} + 2^{25} + 2^{9} + \sum_{i=8}^{8+w_1} C_i \cdot 2^i \\
\delta Q_{64} - \delta Q_{63} &= \sum_{i=29}^{31} X_i \cdot 2^i + \sum_{i=14}^{20} X_i \cdot 2^i + \sum_{i=0}^{v} X_i \cdot 2^i
\end{aligned}$$

where the C_i are selected randomly and the X_i may take on any value. That is, we sample a target by selecting the C_i uniformly at random and we say that a target is solved if there exist $X_i \in \{-1, 0, 1\}$ such that $\delta Q_{61}, \dots, \delta Q_{64}$ agree with the values above. The parameter w_1 is the number of carries from position 25 in step 62 that we rely on. We set 1 as a lower bound for w_1 since we need at least one carry to achieve the constant term 2^9 in δQ_{63}. We use v_1 to parametrize the number of irrelevant bitconditions that we allow.

We summarize the results of Algorithm 3.1 with these targets for the most important parameter values in Table 3.9. For the other parameter values, see Table B.1 in Appendix B. The column "Birthday Factor" gives the factor by which the Birthday Search complexity has to be multiplied for the given parameter choice.

Parameters	Trials	Successes	Trials/Success	Birthday factor
$w_1 = 1, v_1 = -1$	14284746	10000	$2^{10.5}$	$2^{2.0}$
$w_1 = 1, v_1 = 0$	14284746	13199	$2^{10.1}$	$2^{2.0}$
$w_1 = 1, v_1 = 1$	14284746	14182	$2^{10.0}$	$2^{2.0}$
$w_1 = 1, v_1 = 2$	14284746	15637	$2^{9.8}$	$2^{2.0}$
$w_1 = 1, v_1 = 3$	14284746	22535	$2^{9.3}$	$2^{2.0}$
$w_1 = 4, v_1 = 1$	777345731	23470	$2^{15.0}$	$2^{0.5}$
$w_1 = 4, v_1 = 2$	777345731	32195	$2^{14.6}$	$2^{0.5}$
$w_1 = 4, v_1 = 3$	777345731	42718	$2^{14.2}$	$2^{0.5}$
$w_1 = 5, v_1 = 1$	1112515535	22639	$2^{15.6}$	1
$w_1 = 5, v_1 = 2$	1112515535	30735	$2^{15.1}$	1
$w_1 = 5, v_1 = 3$	1112515535	40538	$2^{14.7}$	1

Table 3.9: Results of the Monte Carlo simulation for the end segment of the first near-collision block.

We deal with the issue of unsolvable paths mentioned in Section 3.5.2 as follows: For $w_1 = 1, \dots, 4$, we just accept that with a small probability, we have to use one more carry. For $w_1 = 5$, where we can have no additional carry, we discard targets with $C_9, \dots, C_{15} = 1$ and $C_8 \geq 0$. To achieve the δQ_{63} of the observed path, we need $w_1 \geq 4$. Also, we have $v_1 \geq 1$. Thus, the expected number of attempts until a target is solved in the attack lies between $2^{14.2}$ and $2^{15.6}$. The expected cost of constructing the first block is therefore between $2^{27.8}$ and $2^{29.2}$. Since the first template has the lowest complexity of all, it seems reasonable to maximize w_1.

Block 2

The target differences with parameters $1 \leq w_2 \leq 6$ and $-1 \leq v_2 \leq 4$ are

$$\begin{aligned}
\delta Q_{61} &= 2^{31} + 2^5 \\
\delta Q_{62} &= 2^{31} - 2^{25} - 2^9 + 2^5 \\
\delta Q_{63} &= 2^{31} - 2^{26} + 2^{24} + \sum_{i=20}^{24+w_2} C_i \cdot 2^i - 2^9 + 2^5 \\
\delta Q_{64} - \delta Q_{63} &= \sum_{i=30}^{31} X_i \cdot 2^i + \sum_{i=27}^{30} B_i \cdot 2^i + B_{20} \cdot 2^{20} + \sum_{i=13}^{19} X_i \cdot 2^i + \sum_{i=0}^{v_2} X_i \cdot 2^i
\end{aligned}$$

where the C_i and B_i are selected randomly and the X_i may take on arbitrary values. The parameter w_2 is the number of carries from position 9 in step 62 that we rely on. The most important results are given in Table 3.10. For all the results, see Table B.2 in the appendix. The second near-collision block has the highest complexity and thus, for higher values of w_2, we could only finish a small number of simulations. The attackers used a parameter $w_2 \geq 3$. This gives a lower bound of $2^{24.2}$ on the expected number of attempts and a lower bound of $2^{37.8}$ on the expected cost of constructing the second near-collision block.

Block 3

The target differences, given parameter $0 \leq w_3 \leq 4$, are

$$\begin{aligned}
\delta Q_{61} &= 2^{31} \\
\delta Q_{62} &= 2^{31} + 2^{25} + 2^9 \\
\delta Q_{63} &= 2^{31} + 2^{26} - 2^{24} - 2^{14} + 2^9 \\
\delta Q_{64} &= 2^{30} + 2^{26} - 2^{24} - 2^{14} + \sum_{i=13}^{15+w} B_i \cdot 2^i + 2^9 - 2^3
\end{aligned}$$

where the B_i are random and w_3 is the number of carries from position 26 in step 63 that we rely on. The results of the Monte Carlo-simulation are summarized in Table 3.11. With $w_3 = 0$, the expected number of attempts is $2^{18.7}$ and with $w_3 = 4$, the expected number is $2^{24.7}$. The attackers used $w_3 = 4$. Thus, the expected number of attempts is $2^{24.7}$ and the expected cost of constructing the third near-collision block is $2^{38.3}$.

Parameters	Trials	Successes	Success probability	Birthday factor
$w_2 = 0, v_2 = -1$	37689927528	10000	$2^{21.8}$	$2^{3.0}$
$w_2 = 0, v_2 = 0$	37689927528	16042	$2^{21.2}$	$2^{3.0}$
$w_2 = 0, v_2 = 1$	37689927528	17263	$2^{21.1}$	$2^{3.0}$
$w_2 = 0, v_2 = 2$	37689927528	17695	$2^{21.0}$	$2^{3.0}$
$w_2 = 0, v_2 = 3$	37689927528	17847	$2^{21.0}$	$2^{3.0}$
$w_2 = 0, v_2 = 4$	37689927528	17932	$2^{21.0}$	$2^{3.0}$
$w_2 = 3, v_2 = 0$	270004515480	2206	$2^{26.9}$	$2^{1.5}$
$w_2 = 3, v_2 = 1$	270004515480	3066	$2^{26.4}$	$2^{1.5}$
$w_2 = 3, v_2 = 2$	270004515480	3576	$2^{26.2}$	$2^{1.5}$
$w_2 = 3, v_2 = 3$	270004515480	3755	$2^{26.1}$	$2^{1.5}$
$w_2 = 3, v_2 = 4$	270004515480	3806	$2^{26.1}$	$2^{1.5}$
$w_2 = 5, v_2 = 2$	10415440989433	5008	$2^{31.0}$	$2^{0.5}$
$w_2 = 5, v_2 = 3$	10415440989433	6463	$2^{30.6}$	$2^{0.5}$
$w_2 = 5, v_2 = 4$	10415440989433	7365	$2^{30.4}$	$2^{0.5}$
$w_2 = 6, v_2 = 2$	7392736230494	565	$2^{33.6}$	1
$w_2 = 6, v_2 = 3$	7392736230494	760	$2^{33.2}$	1
$w_2 = 6, v_2 = 4$	7392736230494	990	$2^{32.8}$	1

Table 3.10: Results of the Monte Carlo simulation for the second near-collision block.

Parameters	Trials	Successes	Trials/Success	Birthday factor
$w_3 = 0$	4231476891	10000	$2^{18.7}$	$2^{2.0}$
$w_3 = 1$	16564888766	10000	$2^{20.7}$	$2^{1.5}$
$w_3 = 2$	70906580486	10000	$2^{22.8}$	$2^{1.0}$
$w_3 = 3$	243313360982	10000	$2^{24.5}$	$2^{0.5}$
$w_3 = 4$	275529380806	10000	$2^{24.7}$	1

Table 3.11: Monte Carlo-simulation results for the third near-collision block.

Block 4

Our target differences are, given parameters $0 \leq u_4 \leq 4$ and $1 \leq w_4 \leq 6$,

$$\begin{aligned}
\delta Q_{61} &= 2^{31} \\
\delta Q_{62} &= 2^{31} - 2^{25} - 2^{9} \\
\delta Q_{63} &= 2^{31} - 2^{25} + 2^{14} - 2^{9} \\
\delta Q_{64} &= \sum_{i=30}^{30+\min(u_4,1)} B_i \cdot 2^i - 2^{25} + 2^{14} - 2^9 + 2^5 - 2^3 + \sum_{i=3}^{3+w_4} B_i \cdot 2^i + \sum_{i=0}^{u_4-2} B_i \cdot 2^i
\end{aligned}$$

where the B_i are chosen randomly. The parameter u_4 is the number of carries from position 9 in step 63 that we rely on and the parameter w is the number of carries from position 14. We use a minimum of 1 for w since this is required to achieve the constant term $2^5 - 2^3$. The most important results are summarized in Table 3.12. For all the results, we refer to Table B.4 in the appendix. Note that the parameter u_4 must be large enough to eliminate the random changes to δb that are made in Blocks 1 and 2. That is, if $\max(v_1, v_2) \leq 2$, we need $u_4 \geq \max(v_1, v_2) + 2$ and otherwise, we need $u_4 = 4$. The attackers must have used $u_4 \geq 3$ and $w_4 \geq 1$. Thus, the expected number of attempts is at least $2^{24.6}$, so the expected cost of constructing the fourth near-collision block is at least $2^{38.2}$.

Parameters	Trials	Successes	Trials/Success	Birthday factor
$w_4 = 1, u_4 = 0$	14986221886	10000	$2^{20.5}$	$2^{4.5}$
$w_4 = 1, u_4 = 1$	12080362522	10000	$2^{20.2}$	$2^{4.0}$
$w_4 = 1, u_4 = 2$	40666946131	10000	$2^{22.0}$	$2^{3.5}$
$w_4 = 1, u_4 = 3$	259469948224	10000	$2^{24.6}$	$2^{3.0}$
$w_4 = 1, u_4 = 4$	350239928299	10000	$2^{25.1}$	$2^{2.5}$
$w_4 = 5, u_4 = 2$	4223606267373	10000	$2^{28.7}$	$2^{1.5}$
$w_4 = 5, u_4 = 3$	221260468157	100	$2^{31.0}$	$2^{1.0}$
$w_4 = 5, u_4 = 4$	254345380301	100	$2^{31.2}$	$2^{0.5}$
$w_4 = 6, u_4 = 2$	105022546473	100	$2^{30.0}$	$2^{1.0}$
$w_4 = 6, u_4 = 3$	1228561139591	120	$2^{33.3}$	$2^{0.5}$
$w_4 = 6, u_4 = 4$	2419872905855	120	$2^{34.2}$	1

Table 3.12: Results of the Monte Carlo-simulation for the fourth near-collision block

3.6.3 Total Cost

Let us now combine our estimates for the cost of solving the paths for different parameter choices with the Birthday Search complexity. We will calculate the following costs:

- $\mathcal{C}_{msg}$, the expected cost when we minimize the cost of constructing the message blocks.
- $\mathcal{C}_{flame}$, the expected cost when we minimize the message block construction cost while keeping the parameters consistent with the observed differential paths.
- $\mathcal{C}_{search}$, the expected cost when we minimize the Birthday Search cost.
- $\mathcal{C}_{min}$, the minimal expected cost.

Firstly, for $\mathcal{C}_{msg}$, we choose $w_1, \dots, w_4$ to be as small as possible. We have to balance the parameters v_1 and v_2 against u_4. Increasing v_1 and v_2 does not speed up the message block construction by much, so we pick $v_1, v_2 = -1$ which allows us to pick $u_4 = 1$. The combined Birthday Factor for these parameters is $2^{11.0}$. We therefore have

$$\mathcal{C}_{msg} = 4 \cdot \mathcal{C}_{path} + 2^{11.0} \cdot \frac{2^{44.3}}{\sqrt{p}} + 2^{13.6} \cdot \left(2^{10.5} + 2^{21.8} + 2^{18.7} + 2^{20.5}\right) \approx 2^{55.8}$$

where $\mathcal{C}_{path}$ is the cost of constructing a *full* differential path and p is the probability that a collision is useful. We assume that $p \approx 1/2$ and that $4 \cdot \mathcal{C}_{path}$ is insignificant compared to the other costs. Even if that is not the case, our estimate provides a lower bound.

For $\mathcal{C}_{flame}$, we must choose minimal values for the w_i that are compatible with the differential paths. That is, we must take $w_1 = 4$, $w_2 = 3$, $w_3 = 4$ and $w_4 = 1$. We have $v_1 \geq 1$ and $v_2 \geq 0$, therefore, we must have $u_4 \geq 3$. We can minimize the cost by choosing $v_1 = v_2 = u_4 = 4$. Then, we have a Birthday Factor of $2^{4.5}$. This gives us

$$\mathcal{C}_{flame} = 4 \cdot \mathcal{C}_{path} + \frac{2^{48.8}}{\sqrt{p}} + 2^{13.6} \cdot \left(2^{14.2} + 2^{26.1} + 2^{24.7} + 2^{25.1}\right) \approx 2^{49.3}$$

with the same assumptions as before. For $\mathcal{C}_{search}$, we have a Birthday Factor of 1 and

$$\mathcal{C}_{search} = 4 \cdot \mathcal{C}_{path} + \frac{2^{44.3}}{\sqrt{p}} + 2^{13.6} \cdot \left(2^{14.7} + 2^{32.8} + 2^{24.7} + 2^{34.2}\right) \approx 2^{48.4}$$

To minimize the total expected cost, we take $w_1 = 4$, $v_1 = 3$, $w_2 = 5$, $v_2 = 4$, $w_3 = 4$, $w_4 = 5$ and $u_4 = 4$. Then, we have a Birthday Factor of $2^{1.0}$ and

$$\mathcal{C}_{min} = 4 \cdot \mathcal{C}_{path} + \frac{2^{45.8}}{\sqrt{p}} + 2^{13.6} \cdot \left(2^{14.7} + 2^{30.4} + 2^{24.7} + 2^{31.2}\right) \approx 2^{46.6}$$

when we assume that $p \approx 1/2$ and that $\mathcal{C}_{path}$ is insignificant compared to the other terms. We now show that this cost is indeed minimal:

Theorem 3.15. *Given the values for $\mathbb{E}[\textbf{attempts}]$ from Section 3.6.2 and assuming that the probability p for a useful collision in the Birthday Search is $1/2$, the expected cost of the collision attack is equivalent to at least $\mathcal{C}_{min} = 2^{46.6}$ executions of* `MD5Compress`*. For suitably chosen parameters, this cost can be achieved.*

Proof. We have already given parameters which show that the second part of the theorem holds. To see that the first part holds, note that we can not increase the Birthday Factor beyond $2^{1.5}$, for otherwise, the cost of the Birthday Search already exceeds $\mathcal{C}_{min}$.

If the Birthday Factor is $2^{1.5}$, the combined cost of solving all four paths may not be larger than $2^{44.2}$ if we want to improve upon $\mathcal{C}_{min}$. But it is impossible to reduce the costs of the second and fourth block sufficiently while maintaining a Birthday Factor of $2^{1.5}$. We need $w_4 \leq 4$, but this does not leave enough room to reduce w_2 sufficiently.

For Birthday Factor $2^{1.0}$, the cost of solving the paths must be $\leq 2^{45.4}$ to achieve a cost $\leq \mathcal{C}_{min}$. We must choose $w_4 \leq 5$, or else the cost of solving the fourth block already exceeds that limit. This leaves at most one more parameter we can reduce. We have to do this reduction at the second block, setting $w_2 = 5$. But this is precisely the parameter choice that gives us $\mathcal{C}_{min}$.

For Birthday Factor $2^{0.5}$, the blocks need to be solved with a cost of $2^{45.8}$ if we want to have a total cost $\leq \mathcal{C}_{min}$. But we may only reduce *one* parameter and then, the cost of solving the second or of solving the fourth differential path will already be higher than that. For a Birthday Factor of 1, the expected cost is $\mathcal{C}_{search} > \mathcal{C}_{min}$. □

Remark 3.16. *Extending the parameters beyond the maximum for which we performed Monte Carlo-simulations will not reduce the overall cost either, since for the maximal parameter choice, the cost of solving the fourth path is already higher than $\mathcal{C}_{min}$.*

The parameters for $\mathcal{C}_{min}$ are consistent with the observed differential paths. Assuming that our reconstruction is correct, we can conclude that the expected cost of the collision attack used by the Flame authors is lower-bounded by $2^{46.6}$ calls to `MD5Compress`. It is however unlikely that they used the parameters which achieve $\mathcal{C}_{min}$, especially in the fourth block, since the minimal cost is achieved for $w_4 = 5$, but there is only one carry in the observed differential path. This might be due to several reasons:

- The attackers did not search for optimal parameters (or did not use parameters at all), since the goal was not to find out *how fast* `MD5` can be broken, but just to break it in a reasonable amount of time.

- Since the Birthday Search is more cost-effective to parallelize than the message-block construction, the attackers *intentionally* made the cost of the Birthday Search higher and the cost of solving the differential paths lower. That way, the attackers can improve the *speed* of the attack using massively parallel architectures such as graphic processing units (GPUs), even though the theoretical cost is higher.

These two reasons are not mutually exclusive.

The expected cost of the attack by Stevens et al., using four near-collision blocks, is roughly 1/4 of the lower bound of the Flame attack; its expected cost is equivalent to $2^{44.55}$ calls to `MD5Compress` (see [25, Section 3.7]). The cost of the Birthday Search is the dominating part of the cost.

3.7 Summary

To sum up, the differential paths of the Flame collision attacks appear to be constructed by joining two partial differential paths: An upper path which starts from the input-δIHV and a lower path which causes the desired output-δIHV. This is similar to the attack by Stevens et al., although ΔQ_6 is constant which makes the connection process about 2^{13} times more expensive, compared to the attack by Stevens et al. Tunnels were used in the attack, but the exact method is unclear. We show that the straightforward explanation offered by Stevens in [25] does not account for the observed tunnel strengths and offer an alternative explanation. We conjecture that in the first and third block, each eligible bit is active for $\mathcal{T}_8$ with probability 0.33 and in the second and fourth, the probability is 0.44. However, without more output samples, it is impossible to test our explanation.

We specify a family of differential path end-segments based on the observed differential paths of the Flame attack. Based on our assumptions in Section 3.5.1, we argue that for some parameter choice, these differential path end-segments were used in the Flame attack. Based on this reconstruction of the end-segments, we estimate that the expected cost of the attack is equivalent to at least $2^{46.6}$ executions of `MD5Compress`, but likely somewhat more expensive. Presumably, the goal was not to develop the theoretically cheapest attack possible, but an attack that is *fast enough* and produces only four near-collision blocks, so that they can be hidden in an RSA-key. As already mentioned, this might be due to an intentional attempt to increase the speed on massively parallel architectures. When we minimize the cost of constructing the message blocks, while keeping the parameters consistent with the observed differential paths, the expected cost becomes $2^{49.3}$ which might be closer to the expected cost of the attack that the Flame authors carried out.

Appendix A

Flame Differential Paths

Here, we show the differential paths for all four Flame near-collision blocks. The bitconditions are listed with the condition on the most significant bit to the left and the least significant one to the right. For the meaning of the bitconditions, refer to Tables 1.1 and 1.2. The rotation probabilities are given next to each line. In the first column after the bitconditions. We write "opt" if the rotation probability for δT_t that is used in the differential path is optimal, followed by that probability. Otherwise, we write "$p_a \backslash p_o$" where p_a is the actual rotation probability and p_o the optimal probability. We calculated these probabilities according to Lemma 1.6 under the idealizing assumption that T_t is a uniformly distributed random variable on the set of all 32-bit words. In the next column, we give *empirical estimates* by Marc Stevens for the *conditional* probabilities of the rotations, taking the distribution of δT_t under the condition that the working states fulfill their bitconditions into account. The calculated and estimated probabilities have the largest differences at steps with many bitconditions.

As discussed in Section 1.4.6, these differential paths have long middle segments, ranging from step 23 to 59, with mostly trivial differences. For brevity, if there is a sequence of steps where all bitconditions are identical, we do not list them all individually.

Steps	Bitconditions	Probability	Conditional estimate
-3	`........ ........ ........ ..-.....`		
-2	`00...... .1.1.01. ...1..+. ..-.10..`		
-1	`110-+..1 .1.-.00. .+.+.... ..-110..`		
0	`+-100..0 .-0+^++1 .0.+0.11 .110-+..`	opt, 0.723631	1.000000
1	`0+-++..- .-0++-+0 011-0..1 110+++..`	0.377524\0.493162	0.49707
2	`+0-0-.00 .-++00+- 0-1-+.1+ 1+-0++^.`	0.246820\0.628058	0.166016
3	`+010-000 .-+++0+1 +--.+^1+ -+-+++-.`	opt, 0.910804	1.000000
4	`-00-10+. .11-+-0+ +++11--0 -101-+0.`	0.381041\0.561097	1.000000
5	`0-+-++-^ ^0110+1- -110+0-0 -0001+1^`	0.229237\0.435275	1.000000
6	`++----+- ---+---- -----+++ ++++++++`	0.424792\0.513688	1.000000
7	`111.-111 1101011. 110-1001 +0100.00`	opt, 0.837956	1.000000
8	`00+0.111 10111101 -1101100 .1110011`	0.062526\0.444309	0.170898
9	`..0.1... .....-.. 0.10+... 0-....0.`	opt, 0.516082	0.563477
10	`..0^...1 ^....0.. 0^0-1... .1....+.`	0.054927\0.851331	0.121094
11	`..0-...1 +....-.. .+-01... .0..^.1.`	opt, 0.822020	0.899414
12	`.1-1..^+ 1....+.. .0+0.... ....+.1.`	opt, 0.710508	0.946289
13	`.0+1..-+ 1....0.. 100....1 ....0...`	opt, 0.753908	0.655273
14	`.-+...1. .....1.. 1.+....1 ....1...`	opt, 0.514114	0.578125
15	`.0+...10 ........ -.0....- ....-...`	opt, 0.992125	0.989258
16	`.1+..... .0...... ..^..... ........`	opt, 0.875000	0.887695
17	`..1..... .1....0. ^......^ ....^...`	opt, 0.999023	1.000000
18	`..0..... .+....1. ........ ........`	opt, 0.998993	0.998047
19	`........ ......-. ........ ........`	opt, 0.875000	0.864258
20	`0....... .^...... ........ ........`	opt, 1.000000	1.000000
21	`0....... ......^. ........ ........`	opt, 0.500000	0.501953
22	`-....... ........ ........ ........`	opt, 0.500000	0.517578
23	`........ ........ ........ ........`	opt, 1.000000	1.000000
24	`^....... ........ ........ ........`	opt, 1.000000	1.000000
25-33	`........ ........ ........ ........`	opt, 1.000000	1.000000
34	`!....... ........ ........ ........`	opt, 0.500000	0.507812
35-39	`+....... ........ ........ ........`	opt, 1.000000	1.000000
40	`-....... ........ ........ ........`	opt, 1.000000	1.000000
41-42	`+....... ........ ........ ........`	opt, 1.000000	1.000000
43-44	`-....... ........ ........ ........`	opt, 1.000000	1.000000
45-49	`+....... ........ ........ ........`	opt, 1.000000	1.000000
50, 52, 54, 56, 58	`-....... ........ ........ ........`	opt, 1.000000	1.000000
51, 53, 55, 57, 59	`+....... ........ ........ ........`	opt, 1.000000	1.000000
60	`+.11110. ........ ........ ........`	opt, 1.000000	1.000000
61	`+.11000. ........ .001.00. ........`	opt, 0.992188	1.000000
62	`-.+----. ........ ...0.... ........`	0.390625\0.609375	0.426758
63	`+.?0??+. ........ .--+.+-. ........`	opt, 0.867188	0.855469
64	`+......+ ++++++.. -..-.+-. .....+-.`		

Table A.1: Differential path of the first near-collision block of the Flame certificate (Block 8)

Steps	Bitconditions	Probability	Conditional estimate
-3	`+....... ........ ........ ..-.....`		
-2	`-1....+. .1.1.0.. 0....1+. .-+...0.`		
-1	`+01.-.+1 .0-+.0^. 011+---1 -++.0.10`		
0	`1-0.1.+0 ^-0+1+-1 -1011+-0 001.1^-1`	opt, 0.676736	0.749023
1	`10-.01.+ +++-0+10 --+111+- +--0-+1-`	opt, 0.552536	0.425781
2	`.01.-011 00+-++0+ 0--+.--0 ++10+0+0`	opt, 0.849149	0.492188
3	`..1.-+11 +001++^+ 01-+0110 0+1++0++`	opt, 0.623376	0.833008
4	`..-.1-11 ++1-++-+ -1111--+ ++0+-+-1`	0.100466\0.546746	1.000000
5	`^^1^+1-- 10-01011 0+10-1-+ 0-+++000`	0.398942\0.431254	0.499023
6	`+-++++++ ++++---- ------+- --+-----`	0.458008\0.518055	1.000000
7	`0010-000 01111011 1011-111 10.10010`	opt, 0.960723	1.000000
8	`00000100 1111111+ -1001111 1-010111`	opt, 0.468258	0.672852
9	`...-1... .-.....1 0..1+... .1....^.`	0.467804\0.468719	0.495117
10	`...0...0 ^0.....0 1..+0... .0....-.`	opt, 0.910110	0.895508
11	`..0+..^0 -1...^.. ...01... ......1.`	opt, 0.877075	0.807617
12	`.001..-+ 0....-.. ..01.... ......1.`	opt, 0.875061	1.000000
13	`.1-1..0- 1....0.. 1^1....1 ....1...`	opt, 0.996096	1.000000
14	`.-+...10 .....0.. 1-+....1 ....1...`	opt, 0.514206	0.586914
15	`.0+....0 ........ +01....+ ....-...`	opt, 0.992128	0.994141
16	`.^+..... .0...... .^^..... ........`	opt, 0.875000	0.879883
17	`..1..... .1....0. ^......^ ....^...`	opt, 0.999023	1.000000
18	`..0..... .-....1. ........ ........`	opt, 0.999054	0.999023
19	`........ ......-. ........ ........`	opt, 0.875000	0.895508
20	`0....... .^...... ........ ........`	opt, 1.000000	1.000000
21	`0....... ......^. ........ ........`	opt, 0.500000	0.487305
22	`-....... ........ ........ ........`	opt, 0.500000	0.508789
23	`........ ........ ........ ........`	opt, 1.000000	1.000000
24	`^....... ........ ........ ........`	opt, 1.000000	1.000000
25-33	`........ ........ ........ ........`	opt, 1.000000	1.000000
34	`!....... ........ ........ ........`	opt, 0.500000	0.507812
35	`-....... ........ ........ ........`	opt, 1.000000	1.000000
36-40	`+....... ........ ........ ........`	opt, 1.000000	1.000000
41-43	`-....... ........ ........ ........`	opt, 1.000000	1.000000
44-45	`+....... ........ ........ ........`	opt, 1.000000	1.000000
46	`-....... ........ ........ ........`	opt, 1.000000	1.000000
47	`+....... ........ ........ ........`	opt, 1.000000	1.000000
48	`-....... ........ ........ ........`	opt, 1.000000	1.000000
49-58	`+....... ........ ........ ........`	opt, 1.000000	1.000000
59	`+....... ........ ........ ..0.....`	opt, 1.000000	1.000000
60	`+.....0. ........ ...1001. 110.....`	opt, 0.500000	0.506836
61	`-....100 ...0.... ...1..1. 00+.....`	opt, 0.496094	0.749023
62	`+....1-. ........ ...-+++. +--.....`	opt, 0.972412	0.948242
63	`+....++- ...+.... ...???-. ?+-.....`	0.238037\0.269775	0.261719
64	`......-- ..+..... .-....-. .+-....+`		

Table A.2: Differential path of the second near-collision block of the Flame certificate (Block 9)

Steps	Bitconditions	Probability	Conditional estimate
-3	....		
-2	.1.10100 .....11. 10...... ..0.....		
-1	^0.0101- .1.0^10. 11.0.... ..1.100^		
0	++1-++++ 1001---. --.1.... .1+.110-	opt, 0.833079	1.000000
1	0-111110 1-1+1+-^ --1+.... .01^++-0	opt, 0.861702	0.96875
2	10-01110 +++1---+ +10+.... 0-0++++1	0.403589\0.408343	0.374023
3	-0-01^1+ +0+1--10 0-++^^.0 01+0+00.	opt, 0.941415	1.000000
4	--0++-00 0-0+11++ ++-1-+10 -+00+-1.	0.084686\0.592803	1.000000
5	-1++-0-1 +1-00+1- +0++110- -1--1+^^	opt, 0.775861	1.000000
6	++----+- ---+---- -----+++ ++++++++	opt, 0.513692	1.000000
7	1000-010 00.1010. 101-0101 +0001.00	opt, 0.837956	1.000000
8	11+1.101 01011100 -1000101 .1000011	opt, 0.437474	0.0566406
9	..0.1... .....-.. 0.10+... 0-....0.	opt, 0.516082	0.573242
10	..0^...1 ^....0.. 0^0-1... .1....+.	0.054927\0.851331	0.120117
11	..0-...1 +....-.. .+-01... .0..^.1.	opt, 0.822020	0.8896484
12	.1-1..^+ 1....+.. .0+0.... ....+.1.	opt, 0.710508	0.948242
13	.0+1..-+ 1....0.. 100....1 ....0...	opt, 0.753908	0.631836
14	.-+...1. .....1.. 1.+....1 ....1...	opt, 0.514114	0.585938
15	.0+...10 .... -.0....- ....-...	opt, 0.992125	0.993164
16	.1+..... .0...... ..^.....	opt, 0.875000	0.868164
17	..1..... .1....0. ^......^ ....^...	opt, 0.999023	1.000000
18	..0..... .+....1.	opt, 0.998993	0.999023
19	....-.	opt, 0.875000	0.868164
20	0....... .^......	opt, 1.000000	1.000000
21	0....... ......^.	opt, 0.500000	0.495117
22	-.......	opt, 0.500000	0.509766
23	....	opt, 1.000000	1.000000
24	^.......	opt, 1.000000	1.000000
25-33	....	opt, 1.000000	1.000000
34	^.......	opt, 0.500000	0.493164
35	-.......	opt, 1.000000	1.000000
36-41	+.......	opt, 1.000000	1.000000
42	-.......	opt, 1.000000	1.000000
43	+.......	opt, 1.000000	1.000000
44-46	-.......	opt, 1.000000	1.000000
47	+.......	opt, 1.000000	1.000000
48	-.......	opt, 1.000000	1.000000
49-59	+.......	opt, 1.000000	1.000000
60	-.....0. ....1.	opt, 1.000000	1.000000
61	-.0110.0 .... .1....0.	opt, 0.496094	0.514648
62	+..01.+. .... .0....+.	opt, 0.498047	0.492188
63	+.+---?- ....-....+.	opt, 0.404300	0.395508
64	.+...+.- ....++++ -.....+. ....-...		

Table A.3: Differential path of the third near-collision block of the Flame certificate (Block 10)

Steps	Bitconditions	Probability	Conditional estimate
-3	+.......		
-2	+....0+. .... 000+---. ..000..1		
-1	+....+-. 11...-++ ++1101+. 10011..1		
0	001.1+-. 01^.^111 -++----0 11+-+11-	opt, 0.983143	1.000000
1	011.0.+. -+-^++1+ ++0000-1 +--0-11+	opt, 0.905244	0.742188
2	+--.-0-. -+1+0--0 1+1-1-++ -1-00+--	opt, 0.691042	0.756836
3	+--1-^1. .+100--+ 10---1+0 ---0++-1	opt, 0.308664	1.000000
4	-010+-1. 10-1-01+ 0-000-1- 0+-10-1-	opt, 0.574448	1.000000
5	+00-+00^ 0++-11-0 +++0-111 01-+-100	opt, 0.748917	1.000000
6	+-++++++ ++++---- ------+- --+-----	opt, 0.518059	0.506836
7	.111-110 01.010.0 0101-110 1101.011	opt, 0.960723	0.735352
8	11110110 0101000+ -0101111 0-100111	0.031742\0.475575	0.0507812
9	...-1... .-.....1 0..1+... .1....^.	0.467804\0.468719	0.522461
10	...0...0 ^0.....0 1..+0... .0....-.	opt, 0.910110	0.895508
11	..0+..^0 -1...^.. ...01... ......1.	opt, 0.877075	0.822266
12	.001..-+ 0....-.. .111.... ......1.	opt, 0.875061	1.000000
13	.1-1..0- 1....0.. 100....1 ....1...	opt, 0.996096	1.000000
14	.-+...10 .....0.. 1-+....1 ....1...	opt, 0.514206	0.556641
15	.0+....0 .... +01....+ ....-...	opt, 0.992128	0.998047
16	.^+..... .0...... .^^.....	opt, 0.875000	0.892578
17	..1..... .1...0. ^......^ ....^...	opt, 0.999023	1.000000
18	..0..... .-...1.	opt, 0.999054	0.999023
19	....-.	opt, 0.875000	0.860352
20	0....... .^......	opt, 1.000000	1.000000
21	0....... ......^.	opt, 0.500000	0.485352
22	+.......	opt, 0.500000	0.501953
23	....	opt, 1.000000	1.000000
24	^.......	opt, 1.000000	1.000000
25-33	....	opt, 1.000000	1.000000
34	!.......	opt, 0.500000	0.498047
35	+.......	opt, 1.000000	1.000000
36-39	-.......	opt, 1.000000	1.000000
40	+.......	opt, 1.000000	1.000000
41-43	-.......	opt, 1.000000	1.000000
44-46	+.......	opt, 1.000000	1.000000
47	-.......	opt, 1.000000	1.000000
48	+.......	opt, 1.000000	1.000000
49-59	-.......	opt, 1.000000	1.000000
60	+.....0. ....00.	opt, 1.000000	1.000000
61	+.....1. .... 11....1.	opt, 0.496094	0.525391
62	-.....-. .... 10...-+.	opt, 0.500000	0.493164
63	+.....-. .... +-...?-.	opt, 0.499995	0.50293
64	....-++. .... +-....-. ..-.+..+		

Table A.4: Differential path of the fourth near-collision block of the Flame certificate (Block 11)

Appendix B

Expected Number of Attempts at Solving the End-Segments

In the following tables, we give the results of the Monte Carlo-simulations in Section 3.6.2 for *all* parameter choices.

Parameters	Trials	Successes	Trials/Success	Birthday factor
$w_1 = 1, v_1 = -1$	14284746	10000	$2^{10.5}$	$2^{2.0}$
$w_1 = 1, v_1 = 0$	14284746	13199	$2^{10.1}$	$2^{2.0}$
$w_1 = 1, v_1 = 1$	14284746	14182	$2^{10.0}$	$2^{2.0}$
$w_1 = 1, v_1 = 2$	14284746	15637	$2^{9.8}$	$2^{2.0}$
$w_1 = 1, v_1 = 3$	14284746	22535	$2^{9.3}$	$2^{2.0}$
$w_1 = 2, v_1 = -1$	46417453	10000	$2^{12.2}$	$2^{1.5}$
$w_1 = 2, v_1 = 0$	46417453	15113	$2^{11.6}$	$2^{1.5}$
$w_1 = 2, v_1 = 1$	46417453	17009	$2^{11.4}$	$2^{1.5}$
$w_1 = 2, v_1 = 2$	46417453	19427	$2^{11.2}$	$2^{1.5}$
$w_1 = 2, v_1 = 3$	46417453	27297	$2^{10.7}$	$2^{1.5}$
$w_1 = 3, v_1 = -1$	200785961	10000	$2^{14.3}$	$2^{1.0}$
$w_1 = 3, v_1 = 0$	200785961	16419	$2^{13.6}$	$2^{1.0}$
$w_1 = 3, v_1 = 1$	200785961	21104	$2^{13.2}$	$2^{1.0}$
$w_1 = 3, v_1 = 2$	200785961	25874	$2^{12.9}$	$2^{1.0}$
$w_1 = 3, v_1 = 3$	200785961	35030	$2^{12.5}$	$2^{1.0}$
$w_1 = 4, v_1 = -1$	777345731	10000	$2^{16.2}$	$2^{0.5}$
$w_1 = 4, v_1 = 0$	777345731	16682	$2^{15.5}$	$2^{0.5}$
$w_1 = 4, v_1 = 1$	777345731	23470	$2^{15.0}$	$2^{0.5}$
$w_1 = 4, v_1 = 2$	777345731	32195	$2^{14.6}$	$2^{0.5}$
$w_1 = 4, v_1 = 3$	777345731	42718	$2^{14.2}$	$2^{0.5}$
$w_1 = 5, v_1 = -1$	1112515535	10000	$2^{16.8}$	1
$w_1 = 5, v_1 = 0$	1112515535	16366	$2^{16.1}$	1
$w_1 = 5, v_1 = 1$	1112515535	22639	$2^{15.6}$	1
$w_1 = 5, v_1 = 2$	1112515535	30735	$2^{15.1}$	1
$w_1 = 5, v_1 = 3$	1112515535	40538	$2^{14.7}$	1

Table B.1: Results of the Monte Carlo simulation for the end segment of the first near-collision block.

Parameters	Trials	Successes	Success probability	Birthday factor
$w_2 = 0, v_2 = -1$	37689927528	10000	$2^{21.8}$	$2^{3.0}$
$w_2 = 0, v_2 = 0$	37689927528	16042	$2^{21.2}$	$2^{3.0}$
$w_2 = 0, v_2 = 1$	37689927528	17263	$2^{21.1}$	$2^{3.0}$
$w_2 = 0, v_2 = 2$	37689927528	17695	$2^{21.0}$	$2^{3.0}$
$w_2 = 0, v_2 = 3$	37689927528	17847	$2^{21.0}$	$2^{3.0}$
$w_2 = 0, v_2 = 4$	37689927528	17932	$2^{21.0}$	$2^{3.0}$
$w_2 = 1, v_2 = -1$	112707985708	10000	$2^{23.4}$	$2^{2.5}$
$w_2 = 1, v_2 = 0$	112707985708	17360	$2^{22.6}$	$2^{2.5}$
$w_2 = 1, v_2 = 1$	112707985708	19287	$2^{22.5}$	$2^{2.5}$
$w_2 = 1, v_2 = 2$	112707985708	19909	$2^{22.4}$	$2^{2.5}$
$w_2 = 1, v_2 = 3$	112707985708	20138	$2^{22.4}$	$2^{2.5}$
$w_2 = 1, v_2 = 4$	112707985708	20219	$2^{22.4}$	$2^{2.5}$
$w_2 = 2, v_2 = -1$	516950724692	10000	$2^{25.6}$	$2^{2.0}$
$w_2 = 2, v_2 = 0$	516950724692	20135	$2^{24.6}$	$2^{2.0}$
$w_2 = 2, v_2 = 1$	516950724692	24573	$2^{24.3}$	$2^{2.0}$
$w_2 = 2, v_2 = 2$	516950724692	26045	$2^{24.2}$	$2^{2.0}$
$w_2 = 2, v_2 = 3$	516950724692	26547	$2^{24.2}$	$2^{2.0}$
$w_2 = 2, v_2 = 4$	516950724692	26733	$2^{24.2}$	$2^{2.0}$
$w_2 = 3, v_2 = -1$	270004515480	1000	$2^{28.0}$	$2^{1.5}$
$w_2 = 3, v_2 = 0$	270004515480	2206	$2^{26.9}$	$2^{1.5}$
$w_2 = 3, v_2 = 1$	270004515480	3066	$2^{26.4}$	$2^{1.5}$
$w_2 = 3, v_2 = 2$	270004515480	3576	$2^{26.2}$	$2^{1.5}$
$w_2 = 3, v_2 = 3$	270004515480	3755	$2^{26.1}$	$2^{1.5}$
$w_2 = 3, v_2 = 4$	270004515480	3806	$2^{26.1}$	$2^{1.5}$
$w_2 = 4, v_2 = -1$	1415975743623	1000	$2^{30.4}$	$2^{1.0}$
$w_2 = 4, v_2 = 0$	1415975743623	2170	$2^{29.3}$	$2^{1.0}$
$w_2 = 4, v_2 = 1$	1415975743623	3038	$2^{28.8}$	$2^{1.0}$
$w_2 = 4, v_2 = 2$	1415975743623	3926	$2^{28.4}$	$2^{1.0}$
$w_2 = 4, v_2 = 3$	1415975743623	4456	$2^{28.2}$	$2^{1.0}$
$w_2 = 4, v_2 = 4$	1415975743623	4637	$2^{28.2}$	$2^{1.0}$
$w_2 = 5, v_2 = -1$	10415440989433	1120	$2^{33.1}$	$2^{0.5}$
$w_2 = 5, v_2 = 0$	10415440989433	2524	$2^{31.9}$	$2^{0.5}$
$w_2 = 5, v_2 = 1$	10415440989433	3668	$2^{31.4}$	$2^{0.5}$
$w_2 = 5, v_2 = 2$	10415440989433	5008	$2^{31.0}$	$2^{0.5}$
$w_2 = 5, v_2 = 3$	10415440989433	6463	$2^{30.6}$	$2^{0.5}$
$w_2 = 5, v_2 = 4$	10415440989433	7365	$2^{30.4}$	$2^{0.5}$
$w_2 = 6, v_2 = -1$	7392736230494	130	$2^{35.7}$	1
$w_2 = 6, v_2 = 0$	7392736230494	295	$2^{34.5}$	1
$w_2 = 6, v_2 = 1$	7392736230494	422	$2^{34.0}$	1
$w_2 = 6, v_2 = 2$	7392736230494	565	$2^{33.6}$	1
$w_2 = 6, v_2 = 3$	7392736230494	760	$2^{33.2}$	1
$w_2 = 6, v_2 = 4$	7392736230494	990	$2^{32.8}$	1

Table B.2: Results of the Monte Carlo simulation for the second near-collision block.

Parameters	Trials	Successes	Trials/Success	Birthday factor
$w_3 = 0$	4231476891	10000	$2^{18.7}$	$2^{2.0}$
$w_3 = 1$	16564888766	10000	$2^{20.7}$	$2^{1.5}$
$w_3 = 2$	70906580486	10000	$2^{22.8}$	$2^{1.0}$
$w_3 = 3$	243313360982	10000	$2^{24.5}$	$2^{0.5}$
$w_3 = 4$	275529380806	10000	$2^{24.7}$	1

Table B.3: Monte Carlo-simulation results for the third near-collision block.

Parameters	Trials	Successes	Trials/Success	Birthday factor
$w_4 = 1, u_4 = 0$	14986221886	10000	$2^{20.5}$	$2^{4.5}$
$w_4 = 1, u_4 = 1$	12080362522	10000	$2^{20.2}$	$2^{4.0}$
$w_4 = 1, u_4 = 2$	40666946131	10000	$2^{22.0}$	$2^{3.5}$
$w_4 = 1, u_4 = 3$	259469948224	10000	$2^{24.6}$	$2^{3.0}$
$w_4 = 1, u_4 = 4$	350239928299	10000	$2^{25.1}$	$2^{2.5}$
$w_4 = 2, u_4 = 0$	31473408443	10000	$2^{21.6}$	$2^{4.0}$
$w_4 = 2, u_4 = 1$	27095762586	10000	$2^{21.4}$	$2^{3.5}$
$w_4 = 2, u_4 = 2$	89336968111	10000	$2^{23.1}$	$2^{3.0}$
$w_4 = 2, u_4 = 3$	573516490688	10000	$2^{25.8}$	$2^{2.5}$
$w_4 = 2, u_4 = 4$	785283707839	10000	$2^{26.2}$	$2^{2.0}$
$w_4 = 3, u_4 = 0$	107212346493	10000	$2^{23.4}$	$2^{3.5}$
$w_4 = 3, u_4 = 1$	80888979585	10000	$2^{22.9}$	$2^{3.0}$
$w_4 = 3, u_4 = 2$	289767080938	10000	$2^{24.8}$	$2^{2.5}$
$w_4 = 3, u_4 = 3$	1729323958821	10000	$2^{27.4}$	$2^{2.0}$
$w_4 = 3, u_4 = 4$	2385632329208	10000	$2^{27.8}$	$2^{1.5}$
$w_4 = 4, u_4 = 0$	391719772246	10000	$2^{25.2}$	$2^{3.0}$
$w_4 = 4, u_4 = 1$	302369621493	10000	$2^{24.8}$	$2^{2.5}$
$w_4 = 4, u_4 = 2$	1050401211933	10000	$2^{26.6}$	$2^{2.0}$
$w_4 = 4, u_4 = 3$	55771294749	100	$2^{29.1}$	$2^{1.5}$
$w_4 = 4, u_4 = 4$	120935577619	100	$2^{30.2}$	$2^{1.0}$
$w_4 = 5, u_4 = 0$	1500432472222	10000	$2^{27.2}$	$2^{2.5}$
$w_4 = 5, u_4 = 1$	1265003442364	10000	$2^{27.0}$	$2^{2.0}$
$w_4 = 5, u_4 = 2$	4223606267373	10000	$2^{28.7}$	$2^{1.5}$
$w_4 = 5, u_4 = 3$	221260468157	100	$2^{31.0}$	$2^{1.0}$
$w_4 = 5, u_4 = 4$	254345380301	100	$2^{31.2}$	$2^{0.5}$
$w_4 = 6, u_4 = 0$	72363355156	100	$2^{29.4}$	$2^{2.0}$
$w_4 = 6, u_4 = 1$	45938374478	100	$2^{28.8}$	$2^{1.5}$
$w_4 = 6, u_4 = 2$	105022546473	100	$2^{30.0}$	$2^{1.0}$
$w_4 = 6, u_4 = 3$	1228561139591	120	$2^{33.3}$	$2^{0.5}$
$w_4 = 6, u_4 = 4$	2419872905855	120	$2^{34.2}$	1

Table B.4: Results of the Monte Carlo-simulation for the fourth near-collision block

Bibliography

[1] 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.

[2] Magnus Daum and Stefan Lucks. Attacking Hash Functions by Poisoned Messages, "The Story of Alice and her Boss", June 2005. `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/`.

[3] Bert den Boer and Antoon Bosselaers. Collisions for the Compression Function of MD5. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1993.

[4] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[5] Hans Dobbertin. Cryptanalysis of MD5 compress, 1996. presented at the rump session of Eurocrypt'96.

[6] Hans Dobbertin. The Status of MD5 After a Recent Attack. *RSA Laboratories' CryptoBytes*, 2(2), 1996.

[7] Philip Hawkes, Michael Paddon, and Gregory G. Rose. Musings on the Wang et al. MD5 collision. *IACR Cryptology ePrint Archive*, 2004:264, 2004. `http://eprint.iacr.org/2004/264`.

[8] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[9] Kaspersky Lab. The Flame: Questions and Answers, May 2012. Securelist blog, `http://www.securelist.com/en/blog/208193522/The_Flame_Questions_and_Answers`.

[10] Vlastimil Klima. Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications. *IACR Cryptology ePrint Archive*, 2005.

[11] Vlastimil Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. *IACR Cryptology ePrint Archive*, 2006:105, 2006. `http://eprint.iacr.org/2006/105`.

[12] Arjen Lenstra, Xiaoyun Wang, and Benne de Weger. Colliding X.509 Certificates. Cryptology ePrint Archive, Report 2005/067, 2005. `http://eprint.iacr.org/2005/067`.

[13] Arjen K. Lenstra and Benne de Weger. On the Possibility of Constructing Meaningful Hash Collisions for Public Keys. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2005.

[14] Maher. Identification of a New Targeted Cyber-Attack, May 2012. http://certcc.ir/index.php?name=news&file=article&sid=1894.

[15] Ralph C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer, 1989.

[16] Ellen Nakashima, Greg Miller, and Julie Tate. Washington Post, June 19, 2012. http://www.washingtonpost.com/world/national-security/us-israel-developed-computer-virus-to-slow-iranian-nuclear-efforts-officials-say/2012/06/19/gJQA6xBPoV_story.html.

[17] Jonathan Ness. Microsoft certification authority signing certificates added to the Untrusted Certificate Store. TechNet Blogs, Security Research & Defense, 2012. http://blogs.technet.com/b/srd/archive/2012/06/03/microsoft-certification-authority-signing-certificates-added-to-the-untrusted-certificate-store.aspx.

[18] Ron L. Rivest. The MD5 Message-Digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[19] Phillip Rogaway. Formalizing Human Ignorance. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.

[20] Phillip Rogaway and Thomas Shrimpton. Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[21] sKyWIper Analysis Team. sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks. Technical report, Budapest University of Technology and Economics, Laboratory of Cryptography and System Security, May 2012.

[22] Alex Sotirov. Analyzing the MD5 collision in Flame. Presentation at SummerCon, slides available at http://www.trailofbits.com/resources/flame-md5.pdf, 2012.

[23] Marc Stevens. Fast Collision Attack on MD5. Cryptology ePrint Archive, Report 2006/104, 2006.

[24] Marc Stevens. *Attacks on Hash Functions and Applications*. PhD thesis, Universiteit Leiden, 2012.

[25] Marc Stevens. Counter-cryptanalysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 2013.

[26] 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.

[27] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.

[28] Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.

[29] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.

[30] Tao Xie and Dengguo Feng. How To Find Weak Input Differences For MD5 Collision Attacks. Cryptology ePrint Archive, Report 2009/223, 2009.

[31] Jun Yajima and Takeshi Shimoyama. Wang's sufficient conditions of MD5 are not sufficient. Cryptology ePrint Archive, Report 2005/263, 2005.