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: September 30, 2013

Members of the Thesis Committee:
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

1 Introduction 3
  1.1 Outline ............................................................. 3
  1.2 Hash Functions in Cryptography .................................. 3
    1.2.1 Cryptography .................................................. 3
    1.2.2 Cryptographic Hash Functions ............................... 5
    1.2.3 Digital Signatures and the Hash-And-Sign Paradigm ........ 7
    1.2.4 Collision Resistance and Chosen-Prefix Attacks .......... 8
  1.3 The Hash Function MD5 ........................................... 9
    1.3.1 The Merkle-Damgård Construction ............................ 9
    1.3.2 Outline of MD5 ................................................ 10
    1.3.3 The Compression Function ................................... 11
    1.3.4 Inverting the MD5 Compression Function ..................... 12
  1.4 Cryptanalysis of MD5 ............................................. 13
    1.4.1 Types of Collision Attacks .................................. 13
    1.4.2 Generic Collision Attacks by Birthday Search ............. 13
    1.4.3 A Short History of MD5-Cryptanalysis ....................... 14
    1.4.4 Differential Cryptanalysis ................................... 16
    1.4.5 Binary Signed Digit Representation ......................... 17
    1.4.6 Differential Paths ............................................ 18
    1.4.7 Rotations of Arithmetic Differentials ....................... 19
    1.4.8 Bitconditions ................................................ 23
    1.4.9 Representing Differential Paths with Bitconditions ........ 24
    1.4.10 A Basic Algorithm for Solving Differential Paths ......... 27
    1.4.11 Trivial Differential Steps .................................. 28

2 Attacks on MD5 .................................................... 32
  2.1 The Identical-Prefix Attack by Wang et al. ...................... 32
  2.2 The Chosen-Prefix Attack by Stevens et al. ....................... 32
    2.2.1 Outline of the Attack ....................................... 33
    2.2.2 Elimination Strategy .......................................... 33
    2.2.3 Birthday Search ............................................... 35
    2.2.4 Extending Partial Differential Paths ....................... 40
    2.2.5 Extending a Path Forward .................................... 40
2.2.6 Extending a Path Backward ............................................. 41
2.2.7 Connecting Upper and Lower Paths .................................. 42
2.2.8 Tunnels ........................................................................... 43
2.2.9 Solving Differential Paths .................................................. 46
2.2.10 Cost of the Attack ............................................................ 46
2.3 Counter-Cryptanalysis: Detecting Collision Attacks .................... 48
2.3.1 Counter-Cryptanalysis ......................................................... 48
2.3.2 Collision Detection in MD5 .................................................. 48
3 The Collision Attack for Flame .................................................. 51
3.1 The Flame Collision Attack ..................................................... 51
3.1.1 About Flame .................................................................... 51
3.1.2 The Rogue Certificate ......................................................... 52
3.2 Hypotheses ........................................................................... 54
3.3 Analysis of the Differential Paths ............................................. 55
3.3.1 Some Features of the Near-collision blocks ............................. 55
3.3.2 Connection Steps ............................................................... 57
3.3.3 Estimating the Success Probability ...................................... 58
3.4 Tunnels ................................................................................. 59
3.4.1 Tunnel Strengths in the Near-collision Blocks ......................... 59
3.4.2 Explanation Attempts ......................................................... 61
3.5 Differential Path Family ........................................................ 63
3.5.1 Overview ........................................................................ 63
3.5.2 Reconstructing the End Segments ........................................ 65
3.5.3 Arguments for the Reconstruction ....................................... 71
3.5.4 Birthday Search ................................................................. 73
3.6 Estimating the Cost of Constructing Message Blocks ................. 73
3.6.1 A Formula for the Expected Cost ......................................... 73
3.6.2 Estimating the Expected Number of Attempts at Solving the End-Segment .... 76
3.6.3 Total Cost ........................................................................ 81
3.7 Summary .............................................................................. 83
A Flame Differential Paths ......................................................... 84
B Expected Number of Attempts at Solving the End-Segments ............ 89
Bibliography ............................................................................. 93
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.\textsuperscript{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

\textsuperscript{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 $K$ be some finite set. We call a function $H : K \times \{0, 1\}^* \rightarrow \{0, 1\}^l$ 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 \( k \in K \), we write \( h_k : \{0,1\}^* \to \{0,1\}^l, x \mapsto H(k, x) \). Both \( l \) and \( K \) 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 \( k \in K \) 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 \( k \in K \) 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 \( k \in K \), 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., \(|K| = 1\)) 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. 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

---

4Note 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, \texttt{KeyGen}, \texttt{Sign} and \texttt{Verify}. Using \texttt{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 $C_2$. 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 $P \parallel S$ and $P \parallel S'$ collide. The second, more dangerous, kind is chosen-prefix attacks which take input $(P, P')$ and produce suffixes $(S, S')$ such that $P \parallel S$ and $P' \parallel S'$ 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 $\text{Compress} : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$ 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 to $M$ so that its length is a multiple of $k$; let $M_i$ denote the $i$th $k$-bit block of the padded input (with $i$ starting from 0). Then, we compute a sequence of intermediate hash values (IHV): We set $IHV_0 = IV, IHV_1 = \text{Compress}(IHV_0, M_0), IHV_2 = \text{Compress}(IHV_1, M_1), \ldots$ 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].

---

5In 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.

6By 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 $k \parallel l$ 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 $M' \parallel 0^j$ 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.
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 $\text{MD5Compress}$. 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 $\land$, $\lor$, $\oplus$ and $\overline{}$ (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 $X[31]X[30]...X[0]$, 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 $X[7]X[6]...X[0]X[15]X[14]...X[8]...X[31]X[30]...X[24]$.

We will also sometimes write numbers or bit-strings in the hexadecimal system with digits $0, 1, \ldots, 9, a, b, \ldots, f$. 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 $\text{MD5Compress}$ 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 $\text{MD5Compress}$ 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 $i$th 512-bit block of $M$. We let

$$IV = IHV_0 = (67452301_{16}, \text{efcdab89}_{16}, \text{98badcfe}_{16}, 10325476_{16})$$
which is the bit-string $0123456789\text{abcdeffedcba9876543210}_{16}$ split up into four 32-bit blocks which are then read as 32-bit words using the little endian standard.$^7$

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

4. The output of $\text{MD5}$ 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 $\text{MD5Compress}$. We partition $B$ in 16 consecutive 32-bit strings $m_0, \ldots, m_{15}$ which we call the message words. The compression function is computed in 64 steps numbered 0 to 63; each step updates a 512-bit message block and the words denoted by $W_i$. We let $m = m_0, \ldots, m_{15}$ and the Boolean functions are

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

$$f_t(a, b, c, d) = \begin{cases} (X \land Y) \oplus (\overline{X} \land Z) & \text{for } 0 \leq t \leq 15 \\ (Z \land X) \oplus (\overline{Z} \land Y) & \text{for } 16 \leq t \leq 31 \\ X \oplus Y \oplus Z & \text{for } 32 \leq t \leq 47 \\ Y \oplus (X \lor \overline{Z}) & \text{for } 48 \leq t \leq 63 \end{cases}$$

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 $\text{MD5Compress}$ 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 $\text{MD5Compress}$ as follows:

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

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

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

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 $IHV_{\text{out}}$ an intermediate hash value $IHV_{\text{in}}$ together with a message block $B$ such that $IHV_{\text{out}} = \text{MD5Compress}(IHV_{\text{in}}, B)$. There are still no practical attacks against the pre-image resistance of MD5, but it is possible to invert $\text{MD5Compress}$ in a different sense that is important not for pre-image attacks but for collision attacks. For fixed $IHV_{\text{in}}$, 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 $\text{MD5Compress}(IHV_{\text{in}}, B)$. This correspondence allows us to choose the outcomes of the first 16 steps of a computation of $\text{MD5Compress}(IHV_{\text{in}}, \cdot)$ 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 $\text{MD5Compress}$ 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. \qed
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 $\text{Compress}$ be some compression function. Technically, any two distinct inputs $(IHV, B)$ and $(IHV', B')$ to $\text{Compress}$ with $\text{Compress}(IHV, B) = \text{Compress}(IHV', B')$ 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 $\text{Compress}$ is called a (pseudo-)collision attack. While (pseudo-)collision attacks on $\text{Compress}$ 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 $\text{Compress}$, then $\text{Compress}$ 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 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,

---

*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 $f : \{0, 1\}^k \rightarrow \{0, 1\}^n$ 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 $f = \text{MD5Compress}(IV, \cdot)$. 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 $f : \{0, 1\}^{512} \rightarrow \{0, 1\}^{128}$ by

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

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 $\text{MD5Compress} \sqrt{2} \cdot 2^{64.3} = 2^{64.8}$ times.

### 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

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

we have $\text{MD5Compress}(IHV, B) = \text{MD5Compress}(IHV', B)$. 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 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 $i$th 512-bit block of $M$ and $M'$, respectively, and let $IHV_0 = IHV'_0 = IV$,

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

and

$$IHV'_1 = \text{MD5Compress}(IHV'_0, M'_0), \ldots, IHV'_k = \text{MD5Compress}(IHV'_{k-1}, M'_{k-1})$$

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 $\delta IHV_{k+1}$. In the cases we are interested in, this target value for $\delta IHV_{k+1}$ 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 $\delta IHV_{k+1}$ 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 $\delta IHV_{k+1}$ 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 $\delta IHV_{k+1}$ 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 $\text{MD5}$-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 $Q_1, \ldots, Q_{16}$ and $Q'_1, \ldots, Q'_{16}$ 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 $\text{MD5Compress}(IHV'_k, M'_k)$ and $\text{MD5Compress}(IHV_k, M_k)$ and check if these inputs solve the differential path. If yes, we have found appropriate message blocks. If not, we try different values for $Q_1, \ldots, Q_{16}$ and $Q'_1, \ldots, Q'_{16}$.

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 $Q_1, \ldots, Q_{16}$ and $Q'_1, \ldots, Q'_{16}$ 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 $\text{MD5}$ 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 $Z[i-1] = Z[i+1] = 0$. 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

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

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 $\Delta X$ is a BSDR of $\delta X$, so $\Delta X$ 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 $(21, 17, 15, 13, 2, 1)$.

For a 32-bit word $X$, we call $w(\text{NAF}(X))$ the $\text{NAF}$-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 $\delta X$ 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 $\delta X$ 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 $\delta X$, choose $X$ at random and let $X' = X + \delta X$, lower-weight $\Delta X$ occur with higher probability: The probability that the BSDR-differential is $\Delta X$ 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 $\text{MD5Compress}$. Let

$$(IVH, m_0\parallel \ldots\parallel m_{15}) \text{ and } (IVH', m'_0\parallel \ldots\parallel m'_{15})$$

be two inputs to $\text{MD5Compress}$. 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 $\Delta A$ given by $\Delta A[i] = A'[i] - A[i]$.  

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

- $\Delta Q_t$ for $t = t_0 - 2, \ldots, t_1, \delta Q_{t_0 - 3}$ and $\delta Q_{t_1 + 1}$.
- $\Delta F_t$ for $t = t_0, \ldots, t_1$.
- $\delta m_0, \ldots, \delta m_{15}$ from which $\delta W_0, \ldots, \delta W_{63}$ can be derived.
- $\delta T_t$ and $\delta R_t$ for $t = t_0, \ldots, t_1$.

We say that $\text{MD5Compress}$-inputs $(IVH, B)$ and $(IVH', B')$ (or the computations of $\text{MD5Compress}$ on these inputs) solve steps $t_0', \ldots, t_1'$ of the differential path if $B' - B = (\delta m_0, \ldots, \delta m_{15})$ and for $t = t_0', \ldots, t_1'$, the working states and intermediate variables of the computations on $(IVH, B)$ and $(IVH', B')$ solve the differentials given by the path for $t = t_0', \ldots, t_1'$ 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 $t_0, \ldots, t_1'$ and they are said to simply solve it if they solve steps $t_0, \ldots, t_1$.

A partial differential path is called valid if there is a pair of inputs for $\text{MD5Compress}$ 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 $\delta F_t$. But using bitconditions, it is not possible to fix $\delta F_t$ without fixing $\Delta F_t$. 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 $\delta T_t$, there are (at most) four possible values for the arithmetic difference in the rotated value $R_t$. We can easily calculate these differences and
with what probability they happen and choose a high probability rotation for \( \delta R_t \) (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 \( \delta Q_t \) agrees with the differential path. But the differential path specifies a value for \( \Delta Q_t \) and, unless \( \delta Q_t = 0 \), there are multiple \( \Delta Q_t \) that are compatible with \( \delta Q_t \). Thus, it is possible that the \( \Delta Q_t \) of our computation deviates from the differential path. The likelihood of this event depends on the weight of \( \Delta Q_t \). 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 \( \Delta Q_t = (k + 2, k + 1, \overline{k}) \), and so on. In differential paths for collision attacks, it is therefore generally preferable to have low-weight values for \( \Delta Q_t \).

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 \( \Delta Q_t \).

Suppose now that our computation agrees with the value for \( \Delta Q_t \) from the differential path. This does not guarantee that it agrees with \( \Delta F_t \). 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 \( \Delta F_t \) just by inspecting them individually. Now assume that our inputs agree with \( \Delta F_t \). If our differential path is valid, the value for \( \delta T_t \) in our computation agrees with the path since \( \delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t \). In \( \delta R_t \), 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 \( \delta T_t \) but not with \( \delta R_t \) 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 \( \delta R_t \) from the path, it follows that they must also agree with \( \delta Q_{t+1} \) 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 \( \delta X \) to \( X \). This equation might fail in three different ways, as we will prove shortly:

- The carries introduced by adding \( \delta X \) 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 \( \delta X \) 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 overflowed. Suppose that
\[
X = 11000\|000 \ldots 0
\]
\[
\Rightarrow X' = 11000\|010 \ldots 0
\]
where the ‘∥’ marks the $(32 - n)$-bit boundary. Rotating these words by $n$ swaps the part before the ‘∥’ with the part after the ‘∥’. Thus,
\[
RL(X, n) = 000 \ldots 0\|11000
\]
\[
RL(X', n) = 010 \ldots 0\|11000
\]
and the difference is $RL(X', n) - RL(X, n) = 2^{30} = RL(\delta X, n)$, as expected. Now consider the following value for $X$.
\[
X = 11000\|110 \ldots 0
\]
\[
\Rightarrow X' = 11001\|000 \ldots 0
\]
The rotated words are
\[
RL(X, n) = 110 \ldots 0\|11000
\]
\[
RL(X', n) = 000 \ldots 0\|11001
\]
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
\[
X = 11110\|110 \ldots 0
\]
\[
\Rightarrow X' = 00000\|110 \ldots 0
\]
where the ‘∥’ again indicates the $(32 - n)$-bit boundary. The rotated words are
\[
RL(X, n) = 110 \ldots 0\|11110
\]
\[
RL(X', n) = 110 \ldots 0\|00000
\]
which gives $RL(X', n) - RL(X, n) = -2^4 - 2^3 - 2^2 - 2 = -2^5 + 2$ and we have $RL(\delta X, n) = 2^{28+5} \mod 32 = 2$.

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

20
can be computed as follows. Let \( X_{\text{low}} = \sum_{i=0}^{31-n} \delta X[i] \cdot 2^i \) and \( X_{\text{high}} = \sum_{i=32-n}^{31} \delta X[i] \cdot 2^i \). Then, the \( D_i \) and \( p_i \) are given by the formulas

<table>
<thead>
<tr>
<th>( i )</th>
<th>( D_i )</th>
<th>( p_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( RL(\delta X, n) )</td>
<td>( 2^{-64+n} \cdot (2^{32-n} - X_{\text{low}}) \cdot (2^{32} - X_{\text{high}}) )</td>
</tr>
<tr>
<td>2</td>
<td>( RL(\delta X, n) - 2^n )</td>
<td>( 2^{-64+n} \cdot (2^{32-n} - X_{\text{low}}) \cdot X_{\text{high}} )</td>
</tr>
<tr>
<td>3</td>
<td>( RL(\delta X, n) + 1 )</td>
<td>( 2^{-64+n} \cdot X_{\text{low}} \cdot (2^{32} - X_{\text{high}} - 2^{32-n}) )</td>
</tr>
<tr>
<td>4</td>
<td>( RL(\delta X, n) - 2^n + 1 )</td>
<td>( 2^{-64+n} \cdot X_{\text{low}} \cdot (X_{\text{high}} + 2^{32-n}) )</td>
</tr>
</tbody>
</table>

**Proof.** For a BSDR differential \( \Delta X \), we can easily determine the BSDR differential after the rotation: This is simply \( (\Delta X_{i+n} \mod 32)_{i=0}^{31} \). Two BSDRs \( \Delta X_1 \) and \( \Delta X_2 \) of \( \delta X \) give rise to the same partition if and only if

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

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

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

\((k_i)_{i=0}^{31}\) is a BSDR of \( \delta X \) since \(\alpha + \beta \equiv \delta X \mod 2^{32}\). Conversely, every BSDR of \( \delta X \) gives rise to a partition of \( \delta X \) and two BSDRs result in the same partition \((\alpha, \beta)\) 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 \( \phi : \mathbb{Z}_{32} \rightarrow \{0, \ldots, 2^{32} - 1\} \) as the function that maps elements of \( \mathbb{Z}_{32} \) to their unique representative in the set \( \{0, \ldots, 2^{32} - 1\} \). For fixed \( \alpha \), 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 \(\beta\): The first one is

\[
\beta_0 = (\phi(\delta X) - \alpha) \mod 2^{32}
\]

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 \(\beta\), it follows that \(\delta X - \alpha \mod 2^{32}\) is divided by \(2^{32-n}\). Which values for \(\alpha\) with \(|\alpha| < 2^{32-n}\) achieve this? We must have

\[
\delta X - \alpha \equiv 0 \mod 2^{32-n} \iff \alpha \equiv \delta X \mod 2^{32-n}
\]

and \(|\alpha| < 2^{32-n}\). There are at most two values for \(\alpha\) that satisfy these constraints, namely \(\alpha_0 = X_{\text{low}}\) and, if \(X_{\text{low}} \neq 0\), also \(\alpha_1 = X_{\text{low}} - 2^{32-n}\).
In total, we thus have at most four possibilities:

\[
\begin{align*}
\alpha &= X_{\text{low}}, \beta = X_{\text{high}} \\
\alpha &= X_{\text{low}}, \beta = X_{\text{high}} - 2^{32} \\
\alpha &= X_{\text{low}} - 2^{32-n}, \beta = (X_{\text{high}} + 2^{32-n}) \mod 2^{32} \\
\alpha &= X_{\text{low}} - 2^{32-n}, \beta = (X_{\text{high}} + 2^{32-n}) \mod 2^{32} - 2^{32}
\end{align*}
\]

These correspond to the rotated differentials

\[
D_1 = RL((\delta X, n), X) \quad D_2 = RL((\delta X, n), X - 2^n) \quad D_3 = RL((\delta X, n), X + 2^n + 1)
\]

To compute the probabilities of these rotations, let

\[
p(\alpha, \beta) = \Pr_{X} [RL((\alpha, \beta), n)] = RL(X + \delta X, n) - RL(X, n)
\]

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

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

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

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

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

\[
p(\alpha, \beta) = \frac{2^{32-n} - |\alpha|}{2^{32-n}} \cdot \frac{2^n - |\beta| \cdot 2^{32-n}}{2^n} = \frac{2^{32-n} - |\alpha|}{2^{32}} \cdot \frac{2^{32} - |\beta|}{2^{32}}
\]

and inserting the four possible values for \(\alpha\) and \(\beta\) 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 $\Delta Q_t$ and Boolean function bitconditions that determine the $\Delta F_t$. 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 $q_t$ where the $i$th element $q_t[i]$ is the condition on the $i$th bits of $Q_t$ and $Q_t'$, counted from least to most significant. When listing bitconditions, we write the tuples $q_t$ from right to left, i.e., as $q_t[31] \ldots q_t[0]$, to reflect the significance of the bits.

### Table 1.1: Differential bitconditions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Condition</th>
<th>$\Delta Q_t[i]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>$Q_t[i] = Q_t'[i]$</td>
<td>0</td>
</tr>
<tr>
<td>+</td>
<td>$Q_t[i] = 0 \land Q_t'[i] = 1$</td>
<td>+1</td>
</tr>
<tr>
<td>-</td>
<td>$Q_t[i] = 1 \land Q_t'[i] = 0$</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Condition</th>
<th>Type</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>$Q_t[i] = Q_t'[i]$</td>
<td>direct</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>$Q_t[i] = Q_t'[i] = 0$</td>
<td>direct</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$Q_t[i] = Q_t'[i] = 1$</td>
<td>direct</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>$Q_t[i] = Q_t'[i] = Q_{t-1}[i]$</td>
<td>indirect</td>
<td>backwards</td>
</tr>
<tr>
<td>v</td>
<td>$Q_t[i] = Q_t'[i] = Q_{t+1}[i]$</td>
<td>indirect</td>
<td>forwards</td>
</tr>
<tr>
<td>!</td>
<td>$Q_t[i] = Q_t'[i] = Q_{t-1}[i]$</td>
<td>indirect</td>
<td>backwards</td>
</tr>
<tr>
<td>y</td>
<td>$Q_t[i] = Q_t'[i] = Q_{t+1}[i]$</td>
<td>indirect</td>
<td>forwards</td>
</tr>
<tr>
<td>m</td>
<td>$Q_t[i] = Q_t'[i] = Q_t-2[i]$</td>
<td>indirect</td>
<td>backwards</td>
</tr>
<tr>
<td>w</td>
<td>$Q_t[i] = Q_t'[i] = Q_t+2[i]$</td>
<td>indirect</td>
<td>forwards</td>
</tr>
<tr>
<td>#</td>
<td>$Q_t[i] = Q_t'[i] = Q_t-2[i]$</td>
<td>indirect</td>
<td>backwards</td>
</tr>
<tr>
<td>h</td>
<td>$Q_t[i] = Q_t'[i] = Q_t+2[i]$</td>
<td>indirect</td>
<td>forwards</td>
</tr>
<tr>
<td>?</td>
<td>$Q_t[i] = Q_t'[i] \land (Q_t[i] = 1 \lor Q_t-2[i] = 0)$</td>
<td>indirect</td>
<td>backwards</td>
</tr>
<tr>
<td>q</td>
<td>$Q_t[i] = Q_t'[i] \land (Q_t[i] = 1 \lor Q_t+2[i] = 0)$</td>
<td>indirect</td>
<td>forwards</td>
</tr>
</tbody>
</table>

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 $\Delta Q_t$: If a pair of working states $Q_t, Q'_t$ satisfies $q_t$, then we have

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

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 $\Delta Q_t$, the tuple of differential bitconditions $q_t$ specified by

$$q_t[i] = \begin{cases} + & \text{if } \Delta Q_t[i] = 1 \\ - & \text{if } \Delta Q_t[i] = -1 \\ . & \text{if } \Delta Q_t[i] = 0 \end{cases} \quad (1.1)$$

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

Suppose we have bitconditions ($\cdot, -, -$) 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

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

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 $q_{t_0-3}, \ldots, q_{t_1}$ 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 $\delta Q_{t_0-3}$ from the differential path.
- $Q_{t_1+1}$ and $Q'_{t_1+1}$ satisfy $\delta Q_{t_1+1}$ 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 $\delta Q_{t_0-3}$, $\delta Q_{t_1+1}$ 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 $\delta Q_{t_0-3}$ and $\delta Q_{t_1+1}$ as bitconditions by computing bitconditions from their NAFs using Equation (1.1).

Computing the bitconditions is accomplished by first translating all $\Delta Q_t$ from the differential path into bitconditions using Equation (1.1) and then replacing ‘.’-bitconditions by Boolean function bitconditions so that the $\Delta F_i$ 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 $\gamma, \tau, m, \#$ and $?$.  

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

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

that satisfy the bitconditions $a, b, c$.

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

We define

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

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

A triple $(\delta, e, f) \in L$ is called a solution for step $t$, if it fixes $\Delta F_t$, i.e., if $|V_{t, \delta ef}| = 1$. We let $S_t$ be the set of solutions for step $t$. For $(a, b, c) \in L$ and $g \in V_{t, abc}$, we let $S_{t, abc, g}$ be the set of solutions for step $t$ that strengthen $(a, b, c)$ and that have $V_{t, \delta ef} = \{g\}$. Formally, this can be expressed as

$$S_{t, abc, g} = \{ (\delta, e, f) \in S_t \mid U_{\delta ef} \subseteq U_{abc} \text{ and } V_{t, \delta ef} = \{g\} \}.$$

For any $(a, b, c) \in L$ and any $g \in V_{t, abc}$, it holds that $S_{t, abc, g} \neq \emptyset$ because for any $(x, x', y, y', z, z') \in U_{abc}$ with $f_t(x', y', z') - f_t(x, y, z) = g$, 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 $(\delta, e, f) \in S_{t, abc, g}$ that pick out exactly those elements of $U_{abc}$ which give the correct value for $\Delta F_t[b]$.

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

$$U_{\delta ef} = \{ (x, x', y, y', z, z') \in U_{abc} \mid f_t(x', y', z') - f_t(x, y, z) = g \}$$

(1.2)
The proof of this lemma is by “brute force”. We only give one example here. Let us find \((\partial, \epsilon, \delta)\) for the Boolean function \(F\) of the first round, \(g = 0\) and all \((a, b, c) \in \mathcal{L}\) consisting only of differential bitconditions. We have

\[
\Delta F_t = F(x', y', z') - F(x, y, z) = 0 \iff \begin{cases} 
(x = 0 \wedge x' = 0 \wedge z = z') \\
(x = 0 \wedge x' = 1 \wedge z = y') \\
(x = 1 \wedge x' = 0 \wedge y = z') \\
(x = 1 \wedge x' = 1 \wedge y = y') 
\end{cases}
\]

Suppose first that \(a = \cdot\), so that \(x = x'\). Then, \(b = \cdot\) or \(c = \cdot\), for otherwise \(\Delta F_t = 0\) is not consistent with these bitconditions. If \(b = c = \cdot\), we do not need to add any further bitconditions to fix \(\Delta F_t\). If \(b \neq \cdot\), we have \(y \neq y'\) and thus, we have \(\Delta F_t = 0\) if and only if \(x = x' = 0\). Thus, we set \((\partial, \epsilon, \delta) = (0, b, c)\). If \(c \neq \cdot\), we have \(z \neq z'\), so we have \(\Delta F_t = 0\) if and only if \(x = x' = 1\). Thus, we choose \(\partial = 1, \epsilon = b\) and \(\delta = c\).

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

\[
(\partial, \epsilon, \delta) = \begin{cases} 
(+, \cdot, v) & \text{if we want only forward bitconditions} \\
(+, v, \cdot) & \text{if we want only backward bitconditions}
\end{cases}
\]

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 \(U_{\text{def}} = U_{\text{abc}}\). The case \(a = -\) is treated analogously to \(a = +\).

If there are multiple \((\partial, \epsilon, \delta)\) that fulfill Condition (1.2), we prefer direct bitconditions over indirect ones and among the indirect ones, we prefer the “closer” conditions \(\sim v!y\) that only reach one position backward/forward over \(m\#h?q\) that reach two positions backward/forward. We denote the choice of \(\partial, \epsilon, \delta\) with only forward bitconditions by \(FC(t, ab\epsilon, g)\) and the choice with backward bitconditions by \(BC(t, ab\epsilon, g)\). 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 \(t \in \{t_0, \ldots, t_1\}\), we let \(q_t\) be the differential bitconditions corresponding to \(\Delta Q_t\) according to Equation (1.1). Then, a sequence of working states will satisfy the bitconditions if and only if it solves the \(\Delta Q_t\) from the differential path. For \(t = t_0, \ldots, t_1\) and \(i = 0, \ldots, 31\), we replace bitconditions \((q_t[i], q_{t-1}[i], q_{t-2}[i])\) by \(FC(t, q_t[i], q_{t-1}[i], q_{t-2}[i], \Delta F_t[i])\).

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, \ldots, t_1\}, the bitcondition triple \((q_t[i], q_{t-1}[i], q_{t-2}[i])\) 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 \(\delta Q_{t_0-3}, \Delta Q_{t_0-2}, \ldots, \Delta Q_t\) and \(\Delta F_{t_0}, \ldots, \Delta F_1\) from the differential path.
This statement implies Theorem 1.7 because if the computations on \((IHV, B)\) and \((IHV', B')\) satisfy the \(\Delta Q_t\) and \(\Delta F_t\) in the differential path and \(B, B'\) satisfy the message block differences, then they must satisfy the \(\delta T_i\) of the differential path. The \(\delta R_t\) 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 \((q[t][i], q[t-1][i], q[t-2][i])\) 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 bitconditions 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 \(\Delta F_{t_0}\) from the differential path. Thus, \(FC(t_0, q_{t_0}[i]q_{t_0-1}[i]q_{t_0-2}[i], \Delta F_{t}[i])\) 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 \(\Delta F_t\) 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 \((q_{t-1}[i], q_{t-2}[i], q_{t-3}[i])\) 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 \(q_{t-2}[i] \) is direct or one step forward and that \(q_{t-1}[i] \) is direct. Since \(q_t[i]\) has not been replaced yet, it is direct, too. This shows that \((q_t[i], q_{t-1}[i], q_{t-2}[i])\) 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, \(FC(t_0, q_{t_0}[i]q_{t_0-1}[i]q_{t_0-2}[i], \Delta F_{t}[i])\) 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.

\[\Delta F_{t+1}, \Delta F_{t+2}\]

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

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

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

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

\[\text{1.4.10 A Basic Algorithm for Solving Differential Paths}\]

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

\[IHV = (Q_{-3}, Q_0, Q_{-1}, Q_{-2}), IHV' = (Q'_{-3}, Q'_0, Q'_{-1}, Q_{-2})\]

27
that agree with $q_{-3}, \ldots, q_0$ 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 $Q_1, \ldots, Q_{16}$ that agree with the bitconditions.

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

3. If they do, let $B = m_0 \parallel \ldots \parallel m_{15}, B' = m_0 + \delta m_0 \parallel \ldots \parallel m_{15} + \delta m_{15}$, $IHV = (Q_{-3}, Q_{0}, Q_{-1}, Q_{-2})$ 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 $\Delta Q_t$, if $Q_t$ agrees with $\Delta Q_t$ 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 $\Delta Q_t$ 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 $\delta Q_t$ 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 $\Delta Q_t$ and $\Delta F_t$ 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 \( \delta W_t \) 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 \( \delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t = 0 \). 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 \((31)\). 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 \( \Delta Q_{t-2}, \ldots, \Delta Q_t \in \{(31), (31)\} \), the most significant bits of \( Q'_t, Q'_{t-1} \) and \( Q'_{t-2} \) are \( X, Y \) and \( Z \), respectively.

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

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

which is non-zero if and only if \( Y = X \). Since the only BSDRs for \( 2^{31} \) are \((31)\) and \((31)\), 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 $t \in \{32, \ldots, 47\}$, and $X, Y$ and $Z$ as before, we have

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

and since $A \oplus B = A \oplus B = A \oplus B$, we have

$$X \oplus Y \oplus Z = X \oplus Y \oplus Z = X \oplus Y \oplus Z.$$

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

$$\Delta F_t[31] = Y \oplus (X \lor Z) - Y \oplus (X \lor Z) = Y \oplus (X \lor Z) - Y \oplus (X \lor Z)$$

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

$$(X \lor Z) = (X \lor Z) \iff X = Z.$$

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. \qed

**Remark 1.15.** It is possible to prove a statement similar to 2–4 for $t \in \{0, \ldots, 15\}$, 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 $\delta m_0, \ldots, \delta m_{15}$ and bitconditions $q_{t_0}, \ldots, q_{t_1}$. The following statements hold:

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

   $$q_{t-3}, \ldots, q_{t'+1} = 0, \ldots, 0$$

   *and $\delta W_{t_0}, \ldots, \delta W_{t'} = 0$. 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*

   $$q_{t-3}, \ldots, q_{t+1} = X, 0, \ldots, 0,$$

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

3. *Suppose that for some $t$ with $32 \leq t < 48$, we have bitconditions $q_{t-3}, \ldots, q_t$ 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 $\Delta Q_t$ solve the path up to step $t$.*

30
4. Suppose that, for \( t \) with \( 48 \leq t < 64 \), we have \( q_{t-3}, \ldots, q_t \) as in step 2 and either \( q_t[31] = q_{t-2}[31] \) and \( \delta W_t = 0 \) or \( q_t[31] \neq q_{t-2}[31] \) and \( \delta W_t = 2^{31} \). If an input pair satisfies the path up to step \( t - 1 \) and \( \Delta Q_t \), 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 \( \delta IHV \) 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 \( MD5(M \parallel B_0 \parallel B_1) = MD5(M' \parallel B'_0 \parallel B'_1) \). 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 \( \delta B_0 = (\delta m_0, \ldots, \delta m_{15}) \):

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

The path associated with \( B_1, B'_1 \) begins with \( \delta IHV_{k+1} \) 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.,

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

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 \( 2^{39} \text{MD5Compress}-\text{calls} \).
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 $\text{MD5}(P || S) = \text{MD5}(P' || S')$. 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 $\delta b$ and $\delta c$ are such that $\delta b - \delta c \equiv 0 \mod 2^k$. This condition implies that, on average, the NAFs of $\delta b$ and $\delta c$ 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 \( \delta Q_{64} = 2^r + \sum_{\lambda=0}^{\lambda} s_{\lambda} \cdot 2^{r+21+\lambda \mod 32} \). We explain below how these values are brought about. The following table displays the differentials in the various intermediate variables.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \delta Q_i )</th>
<th>( \delta F_i )</th>
<th>( \delta W_i )</th>
<th>( \delta T_i )</th>
<th>( RC_i )</th>
<th>( \delta R_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( \pm 2^{r-10 \mod 32} )</td>
<td>( \pm 2^{r-10 \mod 32} )</td>
<td>( 10 )</td>
<td>( \pm 2^r )</td>
</tr>
<tr>
<td>62</td>
<td>( \pm 2^r )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 15 )</td>
<td>( 0 )</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>( \pm 2^r )</td>
<td>( \sum_{\lambda=0}^{\lambda} s_{\lambda} \cdot 2^{r+\lambda} )</td>
<td>( 0 )</td>
<td>( \delta F_i )</td>
<td>( 21 )</td>
<td>( \sum_{\lambda=0}^{\lambda} s_{\lambda} \cdot 2^{r+21+\lambda \mod 32} )</td>
</tr>
<tr>
<td>64</td>
<td>( \pm 2^r + \delta R_{63} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These values are brought about as follows: The differential paths have message block differentials \( \delta m_{11} = \pm 2^{r-10 \mod 32} \) 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, \( \delta T_{61} = \delta W_{61} = \delta m_{11} = \pm 2^{r-10 \mod 32} \). The rotation constant is \( RC_{61} = 10 \). Provided that the right rotation happens, we have \( \delta R_{61} = \delta Q_{62} = \pm 2^r \). Then, a likely value for \( \Delta Q_{62} \) is \( \lfloor r \rfloor \) or \( \lceil r \rceil \), respectively. If \( Q_{60}[r] = Q_{60}[r] = 0 \), we have \( \delta F_{62} = 0 \). Also, \( \delta W_i = 0 \). 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 \( \delta Q_{64} \) 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 \( \delta Q_{64} = 2^{11} - 4^3 + 2^2 \).

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \delta Q_i )</th>
<th>( \delta F_i )</th>
<th>( \delta W_i )</th>
<th>( \delta T_i )</th>
<th>( RC_i )</th>
<th>( \delta R_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>61</td>
<td>10101</td>
<td>0</td>
<td>( \pm 2^{r-10 \mod 32} )</td>
<td>( \pm 2^{r-10 \mod 32} )</td>
<td>10</td>
<td>( \pm 2^r )</td>
</tr>
<tr>
<td>62</td>
<td>1.0.+</td>
<td>0</td>
<td>( 0 )</td>
<td>( 15 )</td>
<td>( 0 )</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>+----</td>
<td>( \sum_{\lambda=0}^{\lambda} s_{\lambda} \cdot 2^{r+\lambda} )</td>
<td>( 0 )</td>
<td>( \delta F_i )</td>
<td>( 21 )</td>
<td>( \sum_{\lambda=0}^{\lambda} s_{\lambda} \cdot 2^{r+21+\lambda \mod 32} )</td>
</tr>
<tr>
<td>64</td>
<td>+----</td>
<td>( \delta R_{63} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

We use such near-collision blocks to successively eliminate \( \delta c \) and \( \delta d \) and as much of \( \delta b \) as possible. The remainder of \( \delta b \) is eliminated by introducing a difference in \( \delta c \) and \( \delta d \) and removing it again while making use of the \( s_i \) to eliminate differences in \( \delta b \).

Let us now make the strategy for eliminating the \( IHV \)-difference more precise. Let \( \delta IHV = (0, \delta b, \delta c, \delta d) \) 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 \( \delta c \) to 0. Until \( \delta c = 0 \), repeat the following process:

1. 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 \( \delta T_{61} \) 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 \( s_\lambda = -x_r + 21 \mod 32 \).
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 \( \delta Q_{64} = -y_r \cdot 2^r - \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^r + 21 \mod 32 \).
5. Append the constructed message block pair to the initial messages and update \( \delta IHV \) to the intermediate hash value after the new near-collision blocks.

Each iteration reduces the NAF-weight of \( \delta c \) by one. When \( \delta c \) 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 \( \delta b \). Until \( \delta b = 0 \), repeat the following steps:

1. Pick \( i \in \{0, \ldots, 31\} \) such that \( l_i + 21 \mod 32 \neq 0 \) and \( r = i - 21 \mod 32 \) is minimal. Let \( w' = \min(w, 31 - r) \).
2. For \( \lambda = 0, \ldots, w' \), let \( s_\lambda = -l_i + 21 + \lambda \mod 32 = -l_i + \lambda \mod 32 \).
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 \( \delta Q_{64} = 2^r + \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^r + 21 \mod 32 = 2^r + \sum_{\lambda=0}^{w'} s_\lambda \cdot 2^i + \lambda \mod 32 \).
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 \( \delta IHV \) accordingly.

Each iteration reduces the NAF-weight of \( \delta b \) by at least one. The other words in \( \delta IHV \) remain 0; the contributions of the first and second pair to these differentials cancel each other. Once \( \delta b \) 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 \( \delta IHV \) 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 \( S \) with \(|S| = n\) and we want to find a collision of \( f \), i.e., distinct elements \( x, y \in S \) 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 \( S \) 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 \( x \in S \) 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 \( \theta \).

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 \( \theta \) 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 lose 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” and does not yield a collision of \( f \).

When we have found distinct \( x, y \in S \) 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 \( S \) until the same element is selected twice. Let \( Y \) be the
random variable that counts the number of elements selected uniformly at random from $S$ until the same element is selected twice. By [28, Appendix A],

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

and

$$E[Y] = \sum_{k=0}^{\infty} k \cdot \Pr[Y = k] = \sum_{k=1}^{\infty} k \cdot (\Pr[Y > k-1] - \Pr[Y > k]) = \sum_{k=0}^{\infty} \Pr[Y > k]$$

$$\approx \int_{0}^{\infty} e^{-k^2/(2n)} \, dk = \frac{\sqrt{\pi n}}{2}$$

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

We model our search for a useful collision as follows: Let $S'$ be the set of useful collision values in $S$, so $|S'| = n \cdot p$. At each step, with probability $p$ we select a random element of $S'$ and with probability $1 - p$, we do nothing which simulates selecting a useless value. When we have selected the same element of $S'$ 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 $S'$ until we find a repetition. We expect to make $1/p$ steps in order to select one element of $S'$. 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

$$C_{tr} = \sqrt{\frac{\pi n}{2p}}$$

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.\footnote{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.} Then, we expect to compute $f$ $2/\theta$ times to
recompute the collision and check whether it is useful. Therefore,

\[ C_{coll} = \frac{4}{\theta} \]

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\(^{16}\)

\[ C_{tr} + \frac{C_{coll}}{p} = \sqrt{\frac{\pi n}{2p}} + \frac{4}{\theta p} \]

**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 \(S = \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^k}\). 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

\[ f(x, y, z) = (a, c - d, b - c \mod 2^k) \]

where \((a, b, c, d) = \begin{cases} 
\text{MD5Compress}(IHV, B \parallel x \parallel y \parallel z) & \text{if } x \equiv 0 \mod 2 \\
\text{MD5Compress}(IHV', B' \parallel x \parallel y \parallel 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 \(x \equiv 0 \mod 2\) and the other one has \(x \equiv 1 \mod 2\). Furthermore, we want the differences \(\delta c\) and \(\delta b\) 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

\[ C_{tr}(r, k, w) = \sqrt{\frac{\pi \cdot |S|}{2 \cdot p r,k,w}} \]

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 \(C_{tr} \cdot \theta\). That means, we must choose \(\theta\) such that \(C_{tr} \cdot \theta = M/28\), i.e., \(\theta = M/(28C_{tr})\). Then, the cost for detecting and finding a collision after it has occurred is

\[ C_{coll}(r, k, w, M) = \frac{4 \cdot 28 \cdot C_{tr}(r, k, w) \cdot M}{M} = \frac{112 \cdot C_{tr}(r, k, w) \cdot M}{M} \]

\(^{16}\)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

\[
C(r, k, w, M) = C_{tr}(r, k, w) + \frac{C_{col}(r, k, w, M)}{p_{r,k,w}} = \left(1 + \frac{112}{M \cdot p_{r,k,w}}\right) \cdot C_{tr}(r, k, w)
\]

\[
= \left(1 + \frac{112}{M \cdot p_{r,k,w}}\right) \cdot \sqrt{\frac{\pi \cdot |S|}{2 \cdot p_{r,k,w}}}
\]

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,

\[
C_{tr}(6, 0, 3) = \sqrt{\frac{\pi \cdot 264}{2^{-17.14}}} = \sqrt{\pi \cdot 2^{81.14}} \approx 2^{41.40}.
\]

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

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

**Birthday Search for a Single Block Collision**

There also exists a family of differential paths that can eliminate differences from some set \( D \) with \(|D| = 2^{23.3}\) of intermediate hash value differentials \( \delta \text{IHV} \) that have the property that \( \delta \text{IHV} = (\delta a, \delta b, \delta c, \delta d) \) 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 \( \delta \text{IHV} \) using Birthday Search. To do this, we use search space \( S = \{0, 1\}^{84} \). Let \( B, B' \) be the last pair of message blocks of our prefix and \( \text{IHV} \) and \( \text{IHV}' \) the intermediate hash values after these blocks have been processed. Define a function \( f : S \to S \) by

\[
f(x) = \begin{cases} 
\phi (\text{MD5Compress}(\text{IHV}, B||x) + \delta \hat{\text{IHV}}) & \text{if } \tau(x) = 0 \\
\phi (\text{MD5Compress}(\text{IHV}', B'||x)) & \text{if } \tau(x) = 1
\end{cases}
\]

where \( \tau : S \to \{0, 1\} \) is some balanced selector function, \( \delta \hat{\text{IHV}} = (-2^5, 0, -2^5, -2^5 + 2^{25}) \) and \( \phi(a, b, c, d) = a || d || c \mod 2^{20} \). When we find some collision \( x, y \) with \( \tau(x) = 0 \) and \( \tau(y) = 1 \), then it follows that, for \( (a, b, c, d) = \text{MD5Compress}(\text{IHV}, B||x) \) and \( (a', b', c', d') = \text{MD5Compress}(\text{IHV}', B'||y) \), 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 \(|D| = 2^{23.30}, \) 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 \( \tau \) is balanced, the probability that any given collision is useful is \( p = 2^{-20.70}/2 = 2^{-21.70} \). Plugging \( p \) and \(|S|\) in our formula for \( C_{tr} \), we get

\[
C_{tr} = \sqrt{\frac{\pi \cdot 2^{84}}{2 \cdot 2^{-21.70}}} = \sqrt{\pi \cdot 2^{104.70}} \approx 2^{53.18}
\]

\(^{17}\)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 \(\delta m_0, \ldots, \delta m_{15}\) for the path that we want to build which fixes \(\delta W_0, \ldots, \delta W_{63}\) and with given intermediate hash values. This fixes \(Q_{-3}, \ldots, Q_0\) and \(Q'_{-3}, \ldots, Q'_0\), so we already have \(\Delta Q_{-3}, \ldots, \Delta Q_0\) and corresponding bitconditions \(q_{-3}, \ldots, q_0\) where each \(q_t[i]\) 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 \(q_{-2}, \ldots, q_t\) and \(\delta Q_{-3}, \delta Q_{t+1}\). We also extend the other end of the part backward until we have \(q_{t+3}, \ldots, q_{64}\) 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 \(\Delta F_t\), 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 \(\Delta F_t\) 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 \(\delta Q_{t-3}, \delta Q_t\), bitconditions \(q_{t-2}, q_{t-1}\), and possibly \(q_t\). Extending the path forward one step means doing the following:

- Select bitconditions \(q_t\) on working state \(Q_t\) that enforce \(\delta Q_t\).
- Select a Boolean function differential \(\Delta F_t\) and, if necessary, replace ‘.’-bitconditions by Boolean function bitconditions to enforce it.
- Compute \(\delta T_t\) and the value for \(\delta Q_{t+1}\) that is based on the most likely rotation of it.

Suppose further that for every \(i \in \{0, \ldots, 31\}\), the following invariant holds:
\[
q_t[i] \text{ and } q_{t-1}[i] \text{ are both direct and if } q_{t-2}[i] \text{ is indirect, it only involves } Q_{t-1}[i]. \tag{2.1}
\]
This is the case for \(t = 0\) (since initially, \(q_{-3}, \ldots, q_0\) 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 \(t = 0\), \(q_0\) is already given. In the later steps, we select some BSDR \((Z[i])_{i=0}^{31}\) of \(\delta Q_t\) 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

\[ q_t[i] = \begin{cases} - & \text{if } Z[i] = -1 \\ 0 & \text{if } Z[i] = 0 \\ + & \text{if } Z[i] = 1 \end{cases} \]

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

3. To determine \( \Delta F_t \), we do the following for \( i = 0, \ldots, 31 \).

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

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

(c) Otherwise, we select an arbitrary \(g \in V_{t,abc},t\) and let \((a, c, f) = FC(t,abc,g) \in L\). We then replace \((q_t[i],q_{t-1}[i],q_{t-2}[i])\) by \((a, c, f)\). 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 let \((a, b, c) \in L\) 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 \(q_{t+1}\) 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 \(i = 0, \ldots, 31\), \((q_t[i],q_{t-1}[i],q_{t-2}[i])\) is in \(L\). Thus, we know that if \(q_{t-1}[i]\) is indirect, it may only involve \(Q_{t-2}[i]\) or \(Q_t[i]\). If \(q_{t-1}\) 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 \(q_{t-1}[i]\) is indirect, it only involves \(Q_t[i]\). Likewise, \(q_t[i]\) must be either direct or forward. But since \((q_t[i],q_{t-1}[i],q_{t-2}[i])\) 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 \(q_t, q_{t-1}\) and the differences \(\delta Q_{t+1}\) and \(\delta Q_{t-2}\). We want to extend the differential path backwards by giving bitconditions \(q_{t-2}\), changing ‘+’-bitconditions in \(q_t, q_{t-1}\) to Boolean function bitconditions if necessary and giving a modular differential \(\delta Q_t\). We assume that the conditions in \(q_{t-1}\) are direct and that the indirect conditions in \(q_t\) 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 \(\delta Q_{t-2}\), e.g., \(NAF(\delta Q_{t-2})\), and select \(q_{t-2}\) accordingly. Then we replace for every \(i = 0, \ldots, 31\) the triple \((q_t[i], q_{t-1}[i], q_{t-2}[i])\) by \(BC(t, q_t[i], q_{t-1}[i], q_{t-2}[i], g)\) for some \(g \in V_{t,abc}(q_{t-1}[i], q_{t-2}[i])\).
if \((q[t], q[t-1], q[t-2])\) 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 \(q_{-2}, \ldots, q_t\) and \(\delta Q_{-3}\) and \(\delta Q_{t+1}\). From the backwards extension, we get \(q_{t+3}, \ldots, q_{63}\) and differentials \(\delta Q_{t+2}\) and \(\delta Q_{64}\). Since the bitconditions are computed with the forward and backward extension algorithms, we can assume that all bitconditions in \(q_t\) are direct, that the indirect bitconditions in \(q_{t-1}\) only involve \(Q_t\), that the bit conditions in \(q_{t+3}\) are direct and that all indirect bit conditions in \(q_{t+4}\) only involve \(Q_{t+3}\). Our goal is now to find bitconditions that are compatible with our previous bitconditions and that fix \(\Delta Q_{t+1}, \Delta Q_{t+2}, \Delta F_{t+1}, \Delta F_{t+2}, \Delta F_{t+3}\) and \(\Delta F_{t+4}\) 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 \(\Delta Q_t\) and \(\Delta F_t\) 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 \(\delta R_t = \delta Q_{i+1} - \delta Q_t\).
2. Choose a high-probability rotation difference \(\delta T_t\) from \(dRL(\delta R_t, 32 - RC_t)\).
3. Let \(\delta F_t = \delta T_t - \delta Q_{i-3} - \delta W_t\).

For \(i = 1, \ldots, 32\), let \(U_i\) 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 \(q_j \equiv f_k \equiv 0 \mod 2^t\) such that for \(l = 0, \ldots, i - 1\), there exist bitconditions \(q_{i-1}[l], \ldots, q_{i-4}[l]\) compatible with our previous bitconditions which determine \(\Delta Q_l[l], \Delta Q_{l+1}[l]\) and \(\Delta F_l[l], \ldots, \Delta F_{l+4}[l]\) such that

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

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

Clearly, \(U_0 = \{(\delta Q_{t+1}, \delta Q_{t+2}, \delta F_{t+1}, \delta F_{t+2}, \delta F_{t+3}, \delta F_{t+4})\}\) and \(U_{32}\) 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 \(\Delta Q_i\) and \(\Delta F_i\). Below, we give an algorithm for computing \(U_{i+1}\) from \(U_i\). If we reach a \(U_i = \emptyset\) 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 $\emptyset$;

Let $a = q_{t+4}[i]$, $b = q_{t+3}[i]$, $c = q_t[i]$ and $f = q_{t-1}[i]$;

for $(q_1, q_2, f_1, f_2, f_3, f_4) \in \mathcal{U}_i$ do

    for $e \in \{,\}$ if $q_1[i] = 0$ and $d \in \{,\}$ if $q_2[i] = 0$

        Let $q_1' = 0, 1$ or $-1$ if $e = \cdot$ or $+$ or $-1$, respectively;

        Let $q_2' = 0, 1$ or $-1$ if $d = \cdot$ or $+$ or $-1$, respectively;

        for $f_1' \in \{-f_1[i], f_1[i]\} \cap V_{t+1, ef}$ do

            Let $(d', e', f') = FC(t+1, de, f_1')$;

            for $f_2' \in \{-f_2[i], f_2[i]\} \cap V_{t+2, de'}$ do

                Let $(c', d'', e'') = FC(t+2, cd', f_2')$;

                for $f_3' \in \{-f_3[i], f_3[i]\} \cap V_{t+3, b'd''}$ do

                    Let $(b', c''', d''') = FC(t+3, bc'd'', f_3')$;

                    for $f_4' \in \{-f_4[i], f_4[i]\} \cap V_{t+4, ab'c''}$ do

                        Let $(a', b'', e''') = FC(t+4, ab'c'', f_4')$;

                        insert the element

                        $$(q_1 - q_1' \cdot 2^i, q_2 - q_2' \cdot 2^i, f_1 - f_1' \cdot 2^i, f_2 - f_2' \cdot 2^i, f_3 - f_3' \cdot 2^i, f_4 - f_4' \cdot 2^i)$$

                    into $\mathcal{U}_{i+1}$;

```

Running this algorithm for $i = 0, \ldots, 31$ allows us to determine whether $\mathcal{U}_{32} = \emptyset$. But if $\mathcal{U}_{32} \neq \emptyset$, we also want to find the bitconditions that fix $\Delta Q_t, \Delta Q_{t+1}$ and $\Delta F_t, \Delta F_{t+1}$. 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 $(a', b'', c''', d''', e', f')$ 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 $\Delta Q_{t+i}, \Delta Q_{t+i+2}$ and $\Delta F_{t+i}, \Delta F_{t+i+2}$ 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 $m_0[0] \ldots || m_{15}$. If not, we have to select new working states. However, using tunnels, we often do not have to start over from scratch. A tunnel $T$ 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 $Q_1, \ldots, Q_{64}$ for some $l > 16$. Tunnels in MD5 have been described by Vlastimil Klima in \cite{11}.

We use eight tunnels, labeled $T_1, \ldots, T_8$, 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 $T_8$. Tunnel $T_8$ 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 $Q_{25}, \ldots, Q_{64}$ and message words $m_8, m_9$ and $m_{12}$.

To see why $T_8$ is useful, suppose that we have selected working states $Q_1, \ldots, Q_{16}$ that satisfy the beginning of our differential path. We compute further working states $Q_{17}, \ldots, Q_k$ 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 $Q_1, \ldots, Q_{16}$. 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 $Q_{10}[b] = 0, Q_{11}[b] = 1, q_9[b] = \ldots$ and $Q_9[b]$ is not referenced by any indirect bitcondition, we may flip $Q_9[b]$ and still reuse $Q_{11}, \ldots, Q_{24}$ 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 $T_8$. If we have $Q_1, \ldots, Q_{24}$ that agree with our differential path and $T_8$ 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 $T_8$ works. Let us first see which parts of the message block $m_0, \ldots, m_{15}$ are affected. Clearly, $m_4, m_5, m_7$ 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

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

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

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

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 $Q_{t-3}, \ldots, Q_t$, 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 $W_t = m_{16+5t \mod 16}$ for $t \in \{16, \ldots, 31\}$. The least integer $t > 16$ such that $1+5t \mod 16 \in \{8, 9, 12\}$ 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 $q_k[b] = \ldots$ 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 $Q_i'[b] = 0$ is also satisfied when we have $\Delta Q_i[b] = 1$. We say that a working state bit is active for a tunnel $T_i$ when it is free and the conditions of the tunnel are met for that bit. The expression $Q_{13}[b] = Q_{14}[b]$ in the “Change bit” column for $T_8$ means that the bits $Q_{13}[b]$ 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 $T_6$ means that the bits $Q_8[b]$ and $Q_{12}[b + 22 \mod 32]$ can be changed, but only if they have the same value before and after the change.
Table 2.1: Tunnels of MD5Compress

<table>
<thead>
<tr>
<th>Tunnel</th>
<th>Change bit</th>
<th>Additional bitconditions</th>
<th>Affected states</th>
<th>Affected message words</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$Q_1[b]$</td>
<td>$Q_3[b] = Q_6[b] = 1$</td>
<td>$Q_{21}, \ldots, Q_{64}$</td>
<td>$m_3, m_4, m_5, m_7$</td>
</tr>
<tr>
<td>$T_2$</td>
<td>$Q_5[b]$</td>
<td>$Q_6[b] = 0$</td>
<td>$Q_{21}, \ldots, Q_{64}$</td>
<td>$m_4, m_5, m_7, m_8$</td>
</tr>
<tr>
<td>$T_3$</td>
<td>$Q_3[b] = Q_{14}[b]$</td>
<td>$Q_{15}[b] = Q_{16}[b]$</td>
<td>$Q_{21}, \ldots, Q_{64}$</td>
<td>$m_2, \ldots, m_6, m_{13}, m_{14}, m_{15}$</td>
</tr>
<tr>
<td>$T_4$</td>
<td>$Q_9[b]$</td>
<td>$Q_{10}[b] = Q_{11}[b] = 1$</td>
<td>$Q_{22}, \ldots, Q_{64}$</td>
<td>$m_8, m_9, m_{10}, m_{12}$</td>
</tr>
<tr>
<td>$T_5$</td>
<td>$Q_{10}[b]$</td>
<td>$Q_{11}[b] = 0$</td>
<td>$Q_{22}, \ldots, Q_{64}$</td>
<td>$m_9, m_{10}, m_{12}, m_{13}$</td>
</tr>
<tr>
<td>$T_6$</td>
<td>$Q_9[b] = RR(Q_{12}, 22)[b]$</td>
<td>$Q_{10}[b] = 1$</td>
<td>$Q_{23}, \ldots, Q_{64}$</td>
<td>$m_7, m_8, m_9, m_{12}, \ldots, m_{15}$</td>
</tr>
<tr>
<td>$T_7$</td>
<td>$Q_4[b]$</td>
<td>$Q_5[b] = 0, Q_6[b] = 1$</td>
<td>$Q_{24}, \ldots, Q_{64}$</td>
<td>$m_3, m_4, m_7$</td>
</tr>
<tr>
<td>$T_8$</td>
<td>$Q_9[b]$</td>
<td>$Q_{10}[b] = 0, Q_{11}[b] = 1$</td>
<td>$Q_{25}, \ldots, Q_{64}$</td>
<td>$m_8, m_9, m_{12}$</td>
</tr>
</tbody>
</table>

Let us now, as more difficult examples, verify that $T_3$ and $T_6$ work. Suppose the additional bitconditions for $T_3$ 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 $m_3, \ldots, m_6$ so that no changes occur in the working states $Q_4, \ldots, Q_7$. 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

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

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 $T_6$ 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 $\hat{Q}_8 = Q_8 + (-1)^z \cdot 2^b$. 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

$$F_{10}[b] = (Q_{10}[b] \land Q_9[b]) \oplus (Q_{10}[b] \land \hat{Q}_8[b]) = Q_9[b]$$

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

$$\hat{Q}_{12} = Q_{11} + RL(\hat{T}_{11}, 22)
= Q_{11} + RL(T_{11}, 22) + (-1)^z \cdot 2^{b+22 \bmod 32}
= Q_{12} + (-1)^z \cdot 2^{b+22 \bmod 32}.$$
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 $q_{25}, \ldots, q_{64}$ 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 $q_1, \ldots, q_{16}$ to select $Q_1, \ldots, Q_{16}$. With the help of tunnels $T_1, \ldots, T_7$, we modify our working states so that they also solve the steps given by the bitconditions $q_{17}, \ldots, q_{24}$. We can then vary the bits of tunnel $T_8$ to generate many sequences of working states that all satisfy $q_1, \ldots, q_{24}$. For all these working state sequences, we check whether they cause the right $\delta IHV$. 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 $BC_2$.

 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
 
 Select $Q_1, Q_2, Q_{13}, \ldots, Q_{16}$ that meet bitconditions $q_1, q_2, q_{13}, \ldots, q_{16}$;
 Compute $W_{16} = m_1$ and $Q_{17}$;
 if $Q_{17}$ does not satisfy $q_{17}$ 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 $q_1$ and $q_2$, 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 $q_3$;

 for all $Q_3, \ldots, Q_7$ that satisfy $q_3, \ldots, q_7$ do
  Compute $W_{17} = m_4$ and $Q_{18}$;
  if $Q_{18}$ does not satisfy $q_{18}$ or the wrong rotation occurs at step 17 then
   go to the next loop iteration
  
  for all $Q_8, \ldots, Q_{12}$ meeting $q_8, \ldots, q_{12}$ 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 $(\tilde{Q}_1, \tilde{Q}_2) \in \mathcal{Z}$ do
    Compute $W_{19} = m_9$ and $Q_{20}$;
    if $Q_{20}$ does not meet $q_{20}$ or an incorrect rotation occurs at step 19 then
     go to the next loop iteration
    
    for every available bit of tunnels $T_1, T_2, T_3$ 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 $T_3$ and $T_4$ do
      Compute $W_{21} = m_{10}$ and $Q_{22}$;
      if $Q_{22}$ does not meet $q_{22}$ or an incorrect rotation happened then
       go to the next loop iteration
      
      for every available bit of tunnel $T_5$ 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 $T_6$ do
        Compute $W_{23} = m_4$ and $Q_{24}$;
        if $Q_{24}$ does not meet $q_{24}$ or an incorrect rotation happens then
         go to the next loop iteration
        
        for every available bit of $T_6$ do
         Compute $W_{24} = m_9$ and $Q_{25}$;
         if $Q_{25}$ does not meet $q_{25}$ 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 $Q_{26}, \ldots, Q_{64}$ and $Q'_1, \ldots, Q'_{64}$;
         if for $t = 61, \ldots, 64$, the differences $\delta Q_t = Q'_t - Q_t$ agree with bitconditions $q_t$, then
          return $B = m_0 \parallel m_1 \parallel \ldots \parallel m_{15}$ and 
          $B' = m_0 + \delta m_0 \parallel m_1 + \delta m_1 \parallel \ldots \parallel m_{15} + \delta m_{15}$
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 $k$th 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 \( M \) 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 \( \delta Q_i \) starting at step \( i \). Algorithm 2.3 allows to check whether the \( k \)th 512-bit block \( M_k \) in message \( M \) is the final near-collision block. It works as follows: We first compute \( IHV'_{k+1} = \text{MD5Compress}(IHV_k, M_k) \), storing the working states \( Q_{-3}, \ldots, Q_{64} \). Every element of \( M \) allows us to compute a candidate \( M'_{k} \) for the \( k \)th 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 \( \text{MD5Compress} \) backwards and \( Q'_{i+4}, \ldots, Q'_{64} \) by computing them forwards. From these working states, we obtain \( IHV_k' \) and check if \( IHV_k' = IHV_{k+1}' \). If yes, we conclude that \( M \) was constructed with a collision attack.

**Algorithm 2.3: Detect final near-collision block**

```plaintext
Compute \( IHV'_{k+1} = \text{MD5Compress}(IHV_k, M_k) \) and store the working states \( Q_{-3}, \ldots, Q_{64} \):
for \((\delta m_0, \ldots, \delta m_{15}, i, \delta Q_i) \in M \) 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 \( Q'_{i+1}, \ldots, Q'_{64} \) using the step-function of \( \text{MD5Compress} \);
    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 \( M \), the algorithm will detect that. The cost of computing this algorithm on every block of the message while computing \( \text{MD5}(M) \) is roughly \(|M| + 1 \) times the cost of computing \( \text{MD5}(M) \). Whether this can be improved by early abort is currently investigated. The probability of a false positive is lower bounded by \(|M| \cdot 2^{-128} \). 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 \( \text{MD5} \) 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 \(|M| \cdot 2^{-128} \). 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 $\hat{IHV}'_k$ 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 $\hat{IHV}'_l$, 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.\textsuperscript{19} To circumvent this security measure, the authors of Flame used a previously unknown chosen-prefix collision attack on the hash function $\text{MD5}$ 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 \cite{crysys} by CrySyS Lab at the Budapest University of Technology and Economics and the blog post \cite{kaspersky} 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 \cite{mitigations}, 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 \texttt{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 \cite{iran}) in May 2012. According to \cite{kaspersky}, it was active since at least 2010, but the CrySyS Lab report states that a file named \texttt{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

\textsuperscript{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 $D$. 

51
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. 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.
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 $\delta IHV$. 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 $\text{MD5}$. 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 $\text{MD5Compress}$. We use several different methods to achieve our results: inspection of the differential paths, calculations (under idealizing assumptions based on the pseudo-randomness of $\text{MD5}$), 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 $\delta IHV$, 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 $\delta IHV$ and the lower part designed to provide a certain output $\delta IHV$. 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 $T_8$ 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 $\delta a$ and $\delta d$ in $\delta IHV$ are constant. $\delta b$ and $\delta c$ consist of variable and constant parts. The purpose of the first pair of near-collision blocks is to eliminate $\delta c$ (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 $\delta b$ in the process. The purpose of the second pair is to eliminate $\delta b$, 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 $2^{56.6}$ 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 $i = 25, \ldots, 32$, all four near-collision blocks have the trivial working state differences $\delta Q_i = 0$. For steps $i = 35, \ldots, 59$, 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 $\Delta Q_6$ are maximal in all four near-collision blocks, i.e., for every $i = 0, \ldots, 31$, we have $\Delta Q_6[i] \neq 0$. For $t = 6, \ldots, 32$, the first and third differential path have the same value for $\Delta Q_t$. Likewise, the second and fourth paths have the same value for $\Delta Q_t$ 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 $\delta Q_{22}$. In contrast, the values of $\delta Q_i$ for $i < 6$ are different in all four paths.

Observation 3.8. In the first and third near-collision block, the values of $\Delta F_9, \ldots, \Delta F_{35}$ are identical. In the second and fourth block, the values for $\Delta F_{11}, \Delta F_{12}, \Delta F_{13}, \Delta F_{15}, \ldots, \Delta F_{22}$ and $\Delta F_{24}, \ldots, \Delta F_{35}$ are equal. Also, $\delta F_{14}$ and $\delta F_{23}$ are equal in the second and fourth block, and the values for $\Delta F_{23}$ 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 $Q_9, \ldots, Q_{32}$ are identical. This implies that $\Delta F_{11}, \ldots, \Delta F_{32}$ are identical for the first and third block. To see that the values for $\Delta F_9$ and $\Delta F_{10}$ are identical, first recall that $q[j], q_{-1}[j], q_{-2}[j] \not\in \{+,-\}$ implies $\Delta F_j[j] = 0$. 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 $q_{35}[31]q_{34}[31]q_{33}[31]$ in those two paths cause the same $\Delta F_{35}[31]$.

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

Observation 3.9. The probabilities for the correct rotations of $\delta T_t$ 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 block the steps where the rotation probability falls in the given range, followed by the total number of such steps.
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 $\delta Q_t$ vary among all four blocks. Then, there is a maximal difference in step 6. After that, the values for $\Delta Q_t$ and $\Delta F_t$ 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 $IHV$s 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 $\delta T_t$ as $\delta R_t$. 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: $\Delta Q_6, \Delta Q_7$ and $\Delta Q_8$ appear to belong to the lower differential paths, but $\Delta F_7$ and $\Delta F_8$ vary in all four differential paths, so they do not belong to the lower paths. This makes it seem likely that $\Delta F_7$ and $\Delta F_8$ are varied to achieve the appropriate values for $\Delta Q_7, \Delta Q_8$ and $\Delta Q_9$. To achieve maximal control over $\Delta F_7$, we need to include steps $Q_5$ and $Q_6$ in the connection step. Having maximal differences in $\Delta Q_6$ aids in having many possible alternatives for $\delta F_6$ 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\(^\text{22}\), we

\[^{22}\text{https://code.google.com/p/hashclash}\]
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 \( \delta Q_6 \) should be tried, as in the connection algorithm by Stevens et al. First, we used the values for the \( \delta F_i \) 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.

<table>
<thead>
<tr>
<th></th>
<th>( Q_6 ) fixed</th>
<th>( Q_6 ) not fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 8</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>Block 9</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Block 10</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Block 11</td>
<td>12</td>
<td>96</td>
</tr>
</tbody>
</table>

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 \( \delta F_i \) are taken from the differential paths.

Next, we checked what happens if we compute the \( \delta F_i \) according to Section 2.2.7 instead of taking them from the differential paths. Choosing the \( \delta F_i \) based on the \( \delta T_i \) 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 \( \delta F_i \) 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 \( \delta F_i \) with higher rotation probabilities and still connect the upper and lower paths. In the eighth block, \( \delta F_5, \delta F_6 \) and \( \delta F_7 \) are optimal while \( \delta F_8 \) 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.

<table>
<thead>
<tr>
<th></th>
<th>Choices for ( \delta F_i )</th>
<th>Optimal probabilities</th>
<th>Actual probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block 8</td>
<td>4</td>
<td>( \delta T_5 : .2292, \delta T_6 : .4248, \delta T_7 : .8380, \delta T_8 : .4375 )</td>
<td>( \delta T_8 : .0625 )</td>
</tr>
<tr>
<td>Block 9</td>
<td>32</td>
<td>( \delta T_5 : .3989, \delta T_6 : .4580, \delta T_7 : .9607, \delta T_8 : .4683 )</td>
<td>Optimal</td>
</tr>
<tr>
<td>Block 10</td>
<td>4</td>
<td>( \delta T_5 : .7759, \delta T_6 : .5137, \delta T_7 : .8380, \delta T_8 : .4375 )</td>
<td>Optimal</td>
</tr>
<tr>
<td>Block 11</td>
<td>8</td>
<td>( \delta T_5 : .7489, \delta T_6 : .5181, \delta T_7 : .9607, \delta T_8 : .0317 )</td>
<td>Optimal</td>
</tr>
</tbody>
</table>

Table 3.2: The number of possible choices for the \( \delta F_i \) 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 $\delta Q_6$ is even, throw the path away.
2. For $i = 0, \ldots, 31$, do the following steps.
3. Set the $i$th bitcondition on $Q_6$ to $+$. 
4. If the $i$th bit of $\delta Q_6$ 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 $\delta Q_6$, 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 $T_4$, $T_5$ and $T_8$ 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., $q_9[i] = \cdot$ and $Q_9[i]$ is not referenced by any indirect bitcondition, and if we have $Q_{10}[i], Q_{11}[i] = 1$, tunnel $T_4$ 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 $T_8$ allows us to flip $Q_9[i]$ without changing anything besides $Q_{25}, \ldots, Q_{64}$ and $m_8, m_9, m_{12}$. If $Q_{10}[i]$

---

²³While 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 $T_5$ to flip $Q_{10}[b]$ without changing anything except for $Q_{22}, \ldots, Q_{64}$ and $m_9, m_{10}, m_{12}, m_{13}$.

The Flame authors used tunnel $T_8$, 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 $T_4$ and $T_5$ were used. How exactly $T_8$ 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 $T_8$, 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.*

<table>
<thead>
<tr>
<th>Near-collision Block</th>
<th>Observed strength</th>
<th>Maximal strength</th>
<th>Average strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>17</td>
<td>4.25</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>18</td>
<td>4.5</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>17</td>
<td>4.25</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>18</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Furthermore, we summarize the tunnel strengths of $T_4$ and $T_5$:

**Observation 3.12.** *The strength of tunnel $T_4$ 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 24th bit of $Q_9$ is always active for $T_4$ since the Boolean function bitconditions force $Q_{10}[24], Q_{11}[24] = 1$. Furthermore, in the second and fourth block, we have $q_{11}[18] = \cdot$ and $q_{10}[18] = \cdot$, so $Q_9[18]$ is active for $T_4$ with probability $1/2$. In all other positions $i$ that could be active for $T_4$, we have $q_{10}[i], q_{11}[i] = \cdot$, so these are active with probability $1/4$.***

<table>
<thead>
<tr>
<th>Near-collision Block</th>
<th>Observed strength</th>
<th>Maximal strength</th>
<th>Average strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>19</td>
<td>5.5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>20</td>
<td>5.25</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>19</td>
<td>5.5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>20</td>
<td>5.25</td>
</tr>
</tbody>
</table>

*The strength of $T_5$ 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 $T_5$. In all other positions $i$ that could be active for $T_5$, we have $q_{11}[i] = \cdot$, so they are active with probability $1/2$.***

<table>
<thead>
<tr>
<th>Near-collision Block</th>
<th>Observed strength</th>
<th>Maximal strength</th>
<th>Average strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>19</td>
<td>9.5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>20</td>
<td>10.5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>19</td>
<td>9.5</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>20</td>
<td>10.5</td>
</tr>
</tbody>
</table>

It is clear that the tunnel $T_8$ has been used since the observed tunnel strengths are much larger than the average. The smaller-than-average tunnel strength for $T_4$ and $T_5$ can be explained by the
fact that these three tunnels have conflicting preconditions; we can not conclude that \( T_4 \) and \( T_5 \) 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 \( T_4 \) has strength 3, \( T_5 \) has strength 7.5 and \( T_8 \) 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 \( T_8 \) has strength \( k \) in that partial solution, it gives rise to \( 2^k \) candidates for a complete solution if we use tunnel \( T_8 \). 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 \( T_8 \) 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 \( T_8 \) that is possible with the differential path. Let the random variable \( \text{strength} \) be the tunnel strength of \( T_8 \) in a randomly selected solution for the first 24 steps of the differential path. Let \( \text{solve} \) be the event that a random solution for the first 24 steps gives rise to a solution for the whole path. We assume that \( \Pr[\text{solve} \mid \text{strength} = k] \approx 2^k \cdot p \) for some \( p \) independent of \( k \). Furthermore, we assume that

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

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

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

Using Bayes’ Theorem, we conclude that a solution to the full path has strength \( k \) with probability

\[
\Pr[\text{strength} = k \mid \text{solve}] = \frac{\Pr[\text{strength} = k]}{\Pr[\text{solve}]} \cdot \Pr[\text{solve} \mid \text{strength} = k]
\]

\[
\approx \frac{\Pr[\text{strength} = k]}{\sum_{i=0}^{m} \Pr[\text{strength} = i]} \cdot 2^{k-i}.
\]

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

\[
\sigma = \sqrt{\mathbb{E}[\text{strength}^2] - (\mathbb{E}[\text{strength}])^2}
\]
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 $T_8$. Let $\alpha$ be the probability that a bit which could be active for tunnel $T_8$ is indeed active, so that

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

We searched for values of $\alpha$ 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 $\alpha$ 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 $i$th near-collision block, $\sigma_i$ is the standard deviation of the tunnel strength in the $i$th block and $s_i$ is the observed tunnel strength. By choosing $\alpha = 0.40$, we can minimize $f(\alpha)$. For this value of $\alpha$, all observed tunnel strengths are within 1.3 standard deviations. We summarize the results below:

<table>
<thead>
<tr>
<th>Block</th>
<th>Expectation</th>
<th>Standard deviation ($\sigma$)</th>
<th>Observed strength</th>
<th>Distance from expectation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.7</td>
<td>2.0</td>
<td>7</td>
<td>1.3 $\cdot \sigma$</td>
</tr>
<tr>
<td>2</td>
<td>10.3</td>
<td>2.1</td>
<td>13</td>
<td>1.3 $\cdot \sigma$</td>
</tr>
<tr>
<td>3</td>
<td>9.7</td>
<td>2.0</td>
<td>10</td>
<td>0.2 $\cdot \sigma$</td>
</tr>
<tr>
<td>4</td>
<td>10.3</td>
<td>2.1</td>
<td>9</td>
<td>0.6 $\cdot \sigma$</td>
</tr>
</tbody>
</table>

It is also possible that the first and third block have a different probability than the second and fourth. Let $\alpha_1$ be the probability for the first and third block and $\alpha_2$ the probability for the second and fourth. We searched for $\alpha_1$ and $\alpha_2$ 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 $\alpha_1$, 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 $\alpha_2$, 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:

<table>
<thead>
<tr>
<th>Block</th>
<th>Expectation</th>
<th>Standard deviation ($\sigma$)</th>
<th>Observed strength</th>
<th>Distance from expectation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8.4</td>
<td>2.1</td>
<td>7</td>
<td>0.7 $\cdot \sigma$</td>
</tr>
<tr>
<td>2</td>
<td>11.0</td>
<td>2.1</td>
<td>13</td>
<td>1.0 $\cdot \sigma$</td>
</tr>
<tr>
<td>3</td>
<td>8.4</td>
<td>2.1</td>
<td>10</td>
<td>0.8 $\cdot \sigma$</td>
</tr>
<tr>
<td>4</td>
<td>11.0</td>
<td>2.1</td>
<td>9</td>
<td>1.0 $\cdot \sigma$</td>
</tr>
</tbody>
</table>

62
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 $T_8$ 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

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

But for a chosen-prefix attack, the attackers must have been prepared to eliminate any $\delta IHV$ from some large set. First, a Birthday Search was used to find a value for $\delta IHV$ that could be eliminated; then, the near-collision blocks were constructed. By reconstructing the family of differential paths and the strategy used for eliminating $\delta IHV$, 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 $q_t[31]$ for $t = 62, 63$.

- $\delta a$ and $\delta d$ are fixed constants; $\delta a = -2^5$ and $\delta d = 2^0 - 2^5$. That is, in the Birthday Search, the attackers looked for exactly these differences in $\delta a$ and $\delta d$. 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 $\delta IHVs$ after each near-collision block. After the first block, we have

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

After the second block,

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

After the third block, the differences are

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

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, $\delta a$ and $\delta d$ are 0 and $\delta c$ only has a NAF-weight of 1. The NAF-weight of $\delta b$ 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 $\delta c$ except for a constant term. The second pair then eliminates $\delta b$ 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 $\delta c$ 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 $\delta b$, matters are more complicated. We assume that the changes to $\delta b$ in the first two blocks are mostly random and that the elimination of differences in $\delta b$ really starts at the third near-collision block. This assumption is justified as follows: Most of the bits in $\delta b$ 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 $\delta b$ 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 $\delta b$ 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 $\delta Q_{64}$ 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 $\Delta F_i[j]$. These bitconditions are taken from [24, Table C-4]. The most important observation from the table is the following:

**Observation 3.14.** If $q_i[j]q_{i-1}[j]q_{i-2}[j] = +..$ or $-..$ then we can achieve any $\Delta F_i[j] \in \{-1, 1, 0\}$ by replacing the ‘.’s with appropriate Boolean function bitconditions.

![Table 3.3](image)

Table 3.3: Several tuples of bitconditions and their associated values for $\Delta F$ 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., $q_t[i] \in \{+, -, 0\}$.
- **B**: A variable Boolean function bitcondition. Precisely which bitconditions are relevant at a given position varies, but in general, we have $q_t[i] \in \{0, 1, ?\}$.
- **X**: A non-constant bitcondition, i.e., $q_t[i] \in \{+, -\}$.
- *****: A (for now) irrelevant differential bitcondition.
- **A**: The same differential as above, $q_t[i] = q_{t-1}[i]$.

Note that, since we ultimately only care about the $\delta I HV$ 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 $q_{63}[30]q_{62}[30]q_{61}[30] = +BB$ 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 $\ldots$ and $q_{63}[29]$ 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 $\delta I HV$ 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 \( q_{62}[26] q_{61}[26] q_{60}[26] = -11 \) so that we have \( \Delta F_{62}[26] = 1 \) and hence \( \Delta Q_{63}[9] = 1 \). But we could also achieve the correct \( \delta Q_{63} \) 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 \( \delta c \) and \( \delta b \) 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 \( \delta b \). For \( i=0,...,31 \), 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 \( \delta Q_i \) to be correct.

**Block 1 (Table 3.4)**

<table>
<thead>
<tr>
<th>Steps</th>
<th>Bitconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>59</td>
<td>+............</td>
</tr>
<tr>
<td>60</td>
<td>+BBBB1B.</td>
</tr>
<tr>
<td>61</td>
<td>+BBBB1B.</td>
</tr>
<tr>
<td>62</td>
<td>X+-------.</td>
</tr>
<tr>
<td>63</td>
<td>X.....+.</td>
</tr>
<tr>
<td>64</td>
<td>***.+.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bitconditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

\[
\delta c : +2^{31} + 2^{25} + 2^9 + C_{14} \cdot 2^{14} + \sum_{i=8}^{\delta Q_{62}[25]} C_i \cdot 2^i
\]

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

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 \( \Delta c \) at positions 8,...,8 + \( w_1 \) for a parameter \( w_1 \in \{1,...,5\} \). In the static part, we introduce \( +2^{31} + 2^{25} + 2^9 \) to \( \delta c \). 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 \( \delta Q_{63} = 2^{31} + 2^{25} + \sum_{i=8}^{\delta Q_{62}[25]} C_i \cdot 2^i + 2^9 \) where the \( C_i \) are variable. This difference can be achieved as follows: \( 2^{31} + 2^{25} \) comes from \( \delta Q_{62} \). The term \( \sum_{i=0}^{\delta Q_{62}[25]} C_i \cdot 2^i + 2^9 \) is produced by \( \delta F_{62} \) and the rotation of \( \delta T_{62} \) as follows. Since we have the summand \( 2^{25} \) in \( \delta Q_{62} \), we can have \( \Delta Q_{62}[25],...,\Delta Q_{62}[30] \neq 0 \) and since \( \Delta Q_{61} \) and \( \Delta Q_{60} \) are 0 at these positions, we can get any value for \( \Delta F_{62} \) 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 \( \delta T_{62} = \delta Q_{59} + \delta W_{62} + \delta F_{62} = 2^{31} + \sum_{i=25}^{\delta Q_{62}[25]} \Delta F_{62}[i] \cdot 2^i \). The rotation with constant \( RC_{62} = 15 \) then might give us the desired \( \delta Q_{63} \). 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 \( \delta Q_{64} \).
The static $+2^{25}$ and $+2^9$ will be eliminated as they are for $\delta c$. All differences introduced at the positions marked with $\star$ 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 $\delta Q_{63}$. Consider the case where $C_9$, $\ldots$, $C_{8+w_1} = 1$ and $C_8 \geq 0$. Then, we have $2^9 + \sum_{i=8}^{8+w_1} C_i \cdot 2^i \geq 2^{w_1+1}$ and thus, we need one more carry in step 62 to solve the path. But if $w_1 = 5$, $C_9$, $\ldots$, $C_{14} = 1$ 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 $q_{64}[0], \ldots, q_{64}[3]$ because a higher number forces a higher parameter value in the fourth near-collision block to eliminate the random differences. For $i = 0, \ldots, 4$, we can set $q_{64}[i - 21 \text{mod } 2^{12}] = 0$ to ensure that $\Delta Q_{64}[i] = 0$. Let $v_1$ be the maximal number $\leq 4$ 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 $\Delta Q_{63}$. 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 $\delta Q_{63}$. In the observed path, we have $\delta Q_{63} = 2^{31} + 2^{25} - 2^{14} - 2^{12} + 2^9$. Thus, we need $C_8, \ldots, C_{11} = 0$. 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 $\Delta Q_{64}$ of the observed path does not conform to our template, but only the underlying $\delta Q_{64}$ is relevant and it can be represented by a $\Delta Q_{64}$ 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 $\delta Q_{63}$. 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 $\delta F_{62}$ (rotated to $-2^{24}$) and $-2^{25}$ in $\delta Q_{62}$. We can choose $C_{20}, \ldots, C_{24+w_2}$ from the set $\{-1, 1, 0\}$ using the variable Boolean function bitconditions at positions $5, \ldots, 9 + w_2$. For $\delta Q_{64}$, 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 $q_{63}[5]$ and $q_{63}[6]$. Also, for $i = 20, \ldots, 23$, if $q_{63}[i] \neq \cdots$, we require that $q_{61}[i] = 0$. 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 $\star$ can be corrected in later blocks. We again parametrize the number of irrelevant bitconditions
\[
\delta c : +2^{31} - 2^{26} + 2^{24} - 2^9 + 2^5 + \sum_{i=20}^{24+w_2} C_i \cdot 2^i \\
\delta b : -2^{26} + 2^{24} - 2^9 + 2^5 + \sum_{i=20}^{29} C_i \cdot 2^i + \sum_{i=27}^{29} B_i \cdot 2^i + B_{20} \cdot 2^{20}
\]

where \(w_2\) is the number of carries from position 26. Let \(w_3\) be the number of carries we rely on. In the next step, the variable Boolean function biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). 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 biconditions allow us to control positions 13 to 15 + \(w_3\) in \(\delta Q_{64}\). Due to the static bitconditions, we also get \(2^{30} + 2^{26} - 2^{24} - 2^{14} + 2^9\). Hence,

\[
\delta Q_{63} = \delta Q_{62} + \delta R_{62} = 2^{31} + 2^{25} + 2^{24} - 2^{14} + 2^9 = 2^{31} + 2^{26} - 2^{24} - 2^{14} + 2^9.
\]
\[ \delta c : +2^{31} + 2^{26} - 2^{24} - 2^{14} + 2^9 \]
\[ \delta b : +2^{30} + 2^{26} - 2^{24} + 2^9 - 2^3 + \sum_{i=13}^{15+w_3} B_i \cdot 2^i \]

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)

\[ \delta c : +2^{31} + 2^{24} \]
\[ \delta b : -2^{24} + 2^{14} - 2^9 - 2^5 + 2^3 + \sum_{i=30}^{\min(u_4,1)} B_i \cdot 2^i + \sum_{i=0}^{u_4-2} B_i \cdot 2^i + \sum_{i=3}^{3+w_4} B_i \cdot 2^i \]

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, \ldots, 3 + w_4, 30, 31 in $\Delta Q_{64}$. 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 $-2^5 + 2^3 = -2^4 - 2^2$. 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 $\delta Q_{64}$ 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 $\delta b$, $\delta c$ and $\delta d$, but the differences in $\Delta Q_{62}, \ldots, \Delta Q_{64}$ 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 $\delta IHV$ 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 $\delta IHV$ 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 $q_{62}[25], \ldots, q_{62}[30]$ can then be varied to achieve different values for $\delta F_{62}$ and hence for $\delta Q_{63}$. 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 $q_{62}[31]$ and the two possible rotations $24$, we can get precisely those values for $\delta Q_{63}$ that we give in the template (except for the rare impossible case, when $w_1 = 5$ and $C_9 = \ldots C_{14} = 1$). We assume that the positions marked with a $\ast$ are irrelevant because we can compensate any differences in these positions in later blocks, for appropriate parameter choices.

\[24\text{Since we have, in the terminology of Lemma 1.6, } T_{62, \text{low}} = 0, \text{ only two rotations have non-zero probability. We have } \delta R_{62} \in \{RL(\delta T_{62}, RC_{62}), RL(\delta T_{62}, RC_{62}) - 2RC_{62}\}.\]
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 $\delta Q_{63}$ given in the template. Since we already handled position 14 of $\delta c$ in the first block, we do not need to vary $q_{62}[31]$. 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 $\ast$ can take on differential bitconditions, but are irrelevant because we can handle them in the following blocks. We set $q_{61}[i] = 0$ for $i = 20, \ldots, 24$ so that no changes in $\delta b$ 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., $q_{63}[20]$. Second, the term $2^{12}$ appears both in the observed value for $\delta b$ and $\delta c$. No other terms $2^i$ appear in the NAF of both $\delta b$ and $\delta c$. This might indicate that in the Birthday Search target, $\delta b$ and $\delta c$ are supposed to be identical at this position.

Block 3

In the template, we assume that after Block 2, we have essentially eliminated $\delta c$ 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 $\delta c$ 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 $q_{63}[31]$ allows us to make the changes to $\delta b$ given in the template. We assume that the bitconditions above position 13 are constant. They add the term $-2^3$ to $\delta b$. 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 $\delta Q_{64}$. 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 $-2^4 - 2^3 = -2^5 + 2^5$ 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 $\delta a$ and $\delta d$ 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

$$f(x) = (a, b_{10}, \ldots, b_{13}, b_{16}, c_0, \ldots, c_7, c_{15}, \ldots, c_{31}, d)$$

where $(a, b, c, d) = \begin{cases} 
    \text{MD5Compress} (IHV, B \| x) + (-2^5, 0, -2^5, 2^9 - 2^5) & \text{if } x \text{ is even} \\
    \text{MD5Compress} (IHV', B' \| x) & \text{if } x \text{ is odd}
\end{cases}$

and $\hat{b} = b - c$

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

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

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 $\delta IHV$, but instead, we only require that $\delta a = 0, \delta c = \delta d$ and possibly that $\delta b - \delta c \mod 2^k = 0$ 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 bit conditions are concentrated on the first 16 working states and the tunnel $T_8$ 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 $Q_1, \ldots, Q_{16}$ 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 $T_4$ and $T_5$. 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 $T_8$. 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 $\delta Q_{61}, \ldots, \delta Q_{64}$, 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 $\delta Q_{61}, \ldots, \delta Q_{64}$ 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 $C_{path}$ be the cost of generating the full differential path. We do not know the exact value of $C_{path}$, 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 $Q_1, \ldots, Q_{16}$ 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 $C_t$ 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 $t + 1$. Let $b_t$ 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 $Q_{t+1}, \ldots, Q_{64}$. The expected amortized cost for generating a solution up to step $t$ is

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

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 $C_t + 2^{s_t} - 1$ 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 $q_{-3}, \ldots, q_t$ and with the $\delta Q_{t+1}$ 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 \( C'_t = C_t \). We estimate the expected cost of finding a solution up to step \( t \) as

\[
C_t = \begin{cases} 
0 & \text{for } t = 16 \\
\frac{1}{p_{t-1}} \cdot (C'_{t-1} + 2) & \text{for } 16 < t \leq 32 \\
\frac{1}{p_{t-1}} \cdot (C'_{t-1} + 1) & \text{for } t > 32 
\end{cases}
\]

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 \( C'_{t-1} \). We need to compute one additional step of \texttt{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 \texttt{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.

<table>
<thead>
<tr>
<th>( t )</th>
<th>( p_t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>2^{-6.0}</td>
</tr>
<tr>
<td>17</td>
<td>2^{-3.0}</td>
</tr>
<tr>
<td>18</td>
<td>2^{-2.2}</td>
</tr>
<tr>
<td>19</td>
<td>2^{-2.0}</td>
</tr>
<tr>
<td>20</td>
<td>2^{-3.0}</td>
</tr>
<tr>
<td>21</td>
<td>2^{-2.0}</td>
</tr>
<tr>
<td>22</td>
<td>2^{0.0}</td>
</tr>
<tr>
<td>23</td>
<td>2^{-1.0}</td>
</tr>
</tbody>
</table>

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

\[
\begin{align*}
C_{17} &= C'_{17} \approx 2^{7.0} \\
C_{18} &= C'_{18} \approx 2^{10.0} \\
C_{19} &= C'_{19} \approx 2^{12.2} \\
C_{20} &= C'_{20} \approx 2^{14.2} 
\end{align*}
\]

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

\[
\begin{align*}
C_{22} &= C'_{22} \approx 2^{8.5} \\
C_{23} &= C'_{23} \approx 2^{8.5} 
\end{align*}
\]

and \( C_{24} \approx 2^{9.5} \). The amortized cost is \( C'_{24} \approx 2^{0.9} \).

Since we have only trivial bitconditions with \( \delta Q_i = 0 \) from step 24 to step 33, we have \( C'_{33} = C_{33} = C'_{24} + 17 \approx 2^{14.2} \). In the next step, we have one non-constant bitcondition and a rotation probability of \( 1/2 \), so we get \( C_{34} = C'_{34} \approx 2^{6.4} \). 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 \( \delta Q_i \) are correct. By Theorem 1.14, part 3, we have \( \delta Q_i = 2^{31} \) with certainty up to \( i = 47 \). Thus, \( C_{47} = C'_{47} = C_{34} + 13 \approx 2^{6.6} \). 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 \( \delta W_i = 0 \) or \( \Delta Q_i[31] \neq \Delta Q_{i-2}[31] \) and \( \delta W_i = 2^{31} \). 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 \( C_{i+1} = 2 \cdot (C_i + 1) \) for \( 47 \leq i < 59 \).
That gives us $C_{59} = 2^{19.6}$. 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 $E[\text{attempts}] \cdot (C_{60} + 4) \approx E[\text{attempts}] \cdot 2^{19.6}$. In terms of calls to $\text{MD5Compress}$, the complexity of generating a near-collision block is

$$C_{\text{block}} = C_{\text{path}} + E[\text{attempts}] \cdot 2^{13.6}$$

since $\text{MD5Compress}$ consists of $64 = 2^6$ steps. We will estimate $E[\text{attempts}]$ 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 $\delta Q_{57} = \cdots = Q_{60} = 2^{31}$ 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 $E[\text{attempts}]$ 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 $\tau$ there is a probability $p_\tau$ such that each admissible input pair that we generate hits the target $\tau$ with probability $p_\tau$, independent of the others. Let attempts, be the random variable that counts the number of admissible inputs we have to generate until target $\tau$ is solved. Then, attempts, is distributed geometrically with $\Pr[\text{attempts}, = k] = \Pr[\text{attempts} = k \mid \text{target} = \tau] = (1 - p_\tau)^{k-1}p_\tau$ and $E[\text{attempts},] = E[\text{attempts} \mid \text{target} = \tau] = \tau = p_\tau^{-1}$. By the law of total probability, we can sample attempts by first sampling $\tau \leftarrow \text{target}$ and then sampling attempts,.

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 $\text{MD5}$.

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 $\delta m_i$ 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 $i \not\in \{4, 11, 14\}$.
3. Compute the $\text{MD5}$ 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 $\Delta Q_{58}[31], \ldots, \Delta Q_{60}[31]$. 

76
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 $q_{58}[31]q_{59}[31]q_{60}[31] \in \{+++,+++,-++,...,--\}$. 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.

<table>
<thead>
<tr>
<th>Runs</th>
<th>$\delta m_{11} = 2^{15}$</th>
<th>$\delta m_{11} = -2^{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>162716</td>
<td>161044</td>
</tr>
<tr>
<td>+++</td>
<td>20255</td>
<td>20124</td>
</tr>
<tr>
<td>++-</td>
<td>20506</td>
<td>20055</td>
</tr>
<tr>
<td>+-+</td>
<td>20560</td>
<td>20140</td>
</tr>
<tr>
<td>-++</td>
<td>20458</td>
<td>20052</td>
</tr>
<tr>
<td>-+-</td>
<td>20304</td>
<td>20154</td>
</tr>
<tr>
<td>--+</td>
<td>20000</td>
<td>20321</td>
</tr>
<tr>
<td>---</td>
<td>20307</td>
<td>20198</td>
</tr>
<tr>
<td></td>
<td>20326</td>
<td>20000</td>
</tr>
</tbody>
</table>

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 $E[\text{attempts}]$, we run Algorithm 3.1 until a fixed number $\text{SUCCESSES}$ of targets is solved.

**Algorithm 3.1: Monte Carlo-simulation**

```
s = 0;
trials = 0;
while s < \text{SUCCESSES} do
  Sample $\tau \leftarrow \text{target}$;
  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 $\text{MD5Compress}$;
    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/\text{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 $\mathbb{E}[\text{attempts}]$ 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

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

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 $\delta Q_{61}, \ldots, \delta Q_{64}$ 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 $\delta Q_{63}$. 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.

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

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 \( w_1 = 1, \ldots, 4 \), 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 \( C_9, \ldots, C_{15} = 1 \) and \( C_8 \geq 0 \). To achieve the \( \delta Q_{63} \) of the observed path, we need \( w_1 \geq 4 \). Also, we have \( r_1 \geq 1 \). 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

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

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

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

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 \( 2^{38.3} \).
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Trials (w)</th>
<th>Successes</th>
<th>Success probability</th>
<th>Birthday factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_2 = 0, v_2 = -1$</td>
<td>37689927528</td>
<td>10000</td>
<td>$2^{21.8}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 0$</td>
<td>37689927528</td>
<td>16042</td>
<td>$2^{21.2}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 1$</td>
<td>37689927528</td>
<td>17263</td>
<td>$2^{21.1}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 2$</td>
<td>37689927528</td>
<td>17695</td>
<td>$2^{21.0}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 3$</td>
<td>37689927528</td>
<td>17847</td>
<td>$2^{21.0}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 4$</td>
<td>37689927528</td>
<td>17932</td>
<td>$2^{21.0}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 0$</td>
<td>270004515480</td>
<td>2206</td>
<td>$2^{26.9}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 1$</td>
<td>270004515480</td>
<td>3066</td>
<td>$2^{26.4}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 2$</td>
<td>270004515480</td>
<td>3576</td>
<td>$2^{26.2}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 3$</td>
<td>270004515480</td>
<td>3755</td>
<td>$2^{26.1}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 4$</td>
<td>270004515480</td>
<td>3806</td>
<td>$2^{26.1}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 2$</td>
<td>10415440989433</td>
<td>5008</td>
<td>$2^{31.0}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 3$</td>
<td>10415440989433</td>
<td>6463</td>
<td>$2^{30.6}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 4$</td>
<td>10415440989433</td>
<td>7365</td>
<td>$2^{30.4}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 2$</td>
<td>7392736230494</td>
<td>565</td>
<td>$2^{33.6}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 3$</td>
<td>7392736230494</td>
<td>760</td>
<td>$2^{33.2}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 4$</td>
<td>7392736230494</td>
<td>990</td>
<td>$2^{32.8}$</td>
<td>$2^{1.0}$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Trials (w)</th>
<th>Successes</th>
<th>Trials/Success</th>
<th>Birthday factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_3 = 0$</td>
<td>4231476891</td>
<td>10000</td>
<td>$2^{18.7}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_3 = 1$</td>
<td>16564888766</td>
<td>10000</td>
<td>$2^{20.7}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_3 = 2$</td>
<td>70906580486</td>
<td>10000</td>
<td>$2^{22.8}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_3 = 3$</td>
<td>243313360982</td>
<td>10000</td>
<td>$2^{24.5}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_3 = 4$</td>
<td>275529380806</td>
<td>10000</td>
<td>$2^{24.7}$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

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 $1 \leq w_4 \leq 6$,

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

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 $2^5 - 2^1$. 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 $\delta b$ 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 \( 2^{38.2} \).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Trials</th>
<th>Successes</th>
<th>Trials/Success</th>
<th>Birthday factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_4 = 1, u_4 = 0 )</td>
<td>14986221886</td>
<td>10000</td>
<td>( 2^{20.3} )</td>
<td>( 2^{1.0} )</td>
</tr>
<tr>
<td>( w_4 = 1, u_4 = 1 )</td>
<td>12080362522</td>
<td>10000</td>
<td>( 2^{13.6} )</td>
<td>( 2^{1.0} )</td>
</tr>
<tr>
<td>( w_4 = 1, u_4 = 2 )</td>
<td>40666946131</td>
<td>10000</td>
<td>( 2^{22.0} )</td>
<td>( 2^{1.5} )</td>
</tr>
<tr>
<td>( w_4 = 1, u_4 = 3 )</td>
<td>259469948224</td>
<td>10000</td>
<td>( 2^{24.6} )</td>
<td>( 2^{1.0} )</td>
</tr>
<tr>
<td>( w_4 = 1, u_4 = 4 )</td>
<td>350239928299</td>
<td>10000</td>
<td>( 2^{25.1} )</td>
<td>( 2^{1.5} )</td>
</tr>
<tr>
<td>( w_4 = 2, u_4 = 2 )</td>
<td>422360626733</td>
<td>10000</td>
<td>( 2^{28.7} )</td>
<td>( 2^{1.0} )</td>
</tr>
<tr>
<td>( w_4 = 2, u_4 = 3 )</td>
<td>22126048157</td>
<td>10000</td>
<td>( 2^{31.0} )</td>
<td>( 2^{1.0} )</td>
</tr>
<tr>
<td>( w_4 = 2, u_4 = 4 )</td>
<td>254345380301</td>
<td>10000</td>
<td>( 2^{31.2} )</td>
<td>( 2^{0.5} )</td>
</tr>
<tr>
<td>( w_4 = 3, u_4 = 2 )</td>
<td>105022546473</td>
<td>10000</td>
<td>( 2^{30.0} )</td>
<td>( 2^{1.0} )</td>
</tr>
<tr>
<td>( w_4 = 3, u_4 = 3 )</td>
<td>1228561139591</td>
<td>12000</td>
<td>( 2^{33.3} )</td>
<td>( 2^{0.5} )</td>
</tr>
<tr>
<td>( w_4 = 3, u_4 = 4 )</td>
<td>2419872905855</td>
<td>12000</td>
<td>( 2^{34.2} )</td>
<td>( 1 )</td>
</tr>
</tbody>
</table>

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:

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

Firstly, for \( C_{\text{msg}} \), we choose \( w_1, \ldots, w_4 \) 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

\[
C_{\text{msg}} = 4 \cdot C_{\text{path}} + 2^{11.0} \cdot \frac{2^{44.3}}{\sqrt{p}} + 2^{13.6} \cdot (2^{10.5} + 2^{21.8} + 2^{18.7} + 2^{20.5}) \approx 2^{55.8}
\]

where \( C_{\text{path}} \) 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 \( 4 \cdot C_{\text{path}} \) is insignificant compared to the other costs. Even if that is not the case, our estimate provides a lower bound.

For \( C_{\text{flame}} \), 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

\[
C_{\text{flame}} = 4 \cdot C_{\text{path}} + \frac{2^{48.8}}{\sqrt{p}} + 2^{13.6} \cdot (2^{14.2} + 2^{26.1} + 2^{24.7} + 2^{25.1}) \approx 2^{49.3}
\]

with the same assumptions as before. For \( C_{\text{search}} \), we have a Birthday Factor of 1 and

\[
C_{\text{search}} = 4 \cdot C_{\text{path}} + \frac{2^{44.3}}{\sqrt{p}} + 2^{13.6} \cdot (2^{14.7} + 2^{32.8} + 2^{24.7} + 2^{34.2}) \approx 2^{48.4}
\]

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

\[
C_{\min} = 4 \cdot C_{\text{path}} + \frac{2^{45.8}}{\sqrt{p}} + 2^{13.6} \cdot (2^{14.7} + 2^{30.4} + 2^{24.7} + 2^{31.2}) \approx 2^{46.6}
\]

when we assume that \( p \approx 1/2 \) and that \( C_{\text{path}} \) is insignificant compared to the other terms. We now show that this cost is indeed minimal:

**Theorem 3.15.** Given the values for \( \mathbb{E}[\text{attempts}] \) 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 \( C_{\min} = 2^{46.6} \) 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 \( C_{\min} \).

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 \( C_{\min} \). 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 \( \leq C_{\min} \). 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 \( C_{\min} \).

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 \( \leq C_{\min} \). 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 \( C_{\text{search}} > C_{\min} \). \( \square \)

**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 \( C_{\min} \).

The parameters for \( C_{\min} \) 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 \texttt{MD5Compress}. It is however unlikely that they used the parameters which achieve \( C_{\min} \), 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-$\delta IHV$ and a lower path which causes the desired output-$\delta IHV$. This is similar to the attack by Stevens et al., although $\Delta Q_6$ 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 $\delta T_i$ 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 $T_i$ 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 $\delta T_i$ 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.
<table>
<thead>
<tr>
<th>Steps</th>
<th>Bitconditions</th>
<th>Probability</th>
<th>Conditional estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$+100.00$ . $-0^+11$ . $+0.11$ . $110^+$</td>
<td>opt, 0.273631</td>
<td>1.000000</td>
</tr>
<tr>
<td>1</td>
<td>$0^{+++}$. $-0^{+++}0$ $01^{+}$. $110^{+++}$</td>
<td>0.377524 $\times 0.493162$</td>
<td>0.49707</td>
</tr>
<tr>
<td>2</td>
<td>$+0^{+}$. $0.00$ $-0^{+++}$. $0^{+}$. $1^{+}$. $-0^{++}$</td>
<td>0.246820 $\times 0.628058$</td>
<td>0.166016</td>
</tr>
<tr>
<td>3</td>
<td>$+010$. $000$ $-0^{+++}$. $+1^{+}$ $+0^{+++}$</td>
<td>opt, 0.910804</td>
<td>1.000000</td>
</tr>
<tr>
<td>4</td>
<td>$-00^{+}$ $-0$. $11^{+}$. $+0^{+}$. $110^{+}$</td>
<td>0.381041 $\times 0.561097$</td>
<td>1.000000</td>
</tr>
<tr>
<td>5</td>
<td>$++$. $++$. $-0^{+}$. $-0^{+}$. $000^{+}$</td>
<td>0.229237 $\times 0.45275$</td>
<td>1.000000</td>
</tr>
<tr>
<td>6</td>
<td>$+111^{+}$. $++$. $-0^{+}$. $-0^{+}$. $000^{+}$</td>
<td>0.424792 $\times 0.513688$</td>
<td>1.000000</td>
</tr>
<tr>
<td>7</td>
<td>$00^{+}$. $00000$ $-0^{+}$. $110^{+}$</td>
<td>opt, 0.837956</td>
<td>1.000000</td>
</tr>
<tr>
<td>8</td>
<td>$00^{+}$. $11011101$ $-1101100$. $0111011$</td>
<td>0.062526 $\times 0.443499$</td>
<td>0.170898</td>
</tr>
<tr>
<td>9</td>
<td>$0^{+}$. $000$ $-0^{+}$. $0^{+}$. $+1^{+}$. $0^{+}$. $-0^{+}$</td>
<td>opt, 0.516082</td>
<td>0.56377</td>
</tr>
<tr>
<td>10</td>
<td>$-0^{+}$. $1^{+}$. $-0^{+}$. $0^{+}$. $1^{+}$. $1^{+}$</td>
<td>0.054927 $\times 0.851331$</td>
<td>0.121094</td>
</tr>
<tr>
<td>11</td>
<td>$0^{+}$. $1^{+}$. $+1^{+}$. $01^{+}$. $0^{+}$</td>
<td>opt, 0.822020</td>
<td>0.899414</td>
</tr>
<tr>
<td>12</td>
<td>$1^{+}$. $1^{+}$. $1^{+}$. $1^{+}$. $1^{+}$</td>
<td>opt, 0.710508</td>
<td>0.946289</td>
</tr>
<tr>
<td>13</td>
<td>$+$. $0^{+}$. $0^{+}$. $10^{+}$. $0^{+}$</td>
<td>opt, 0.753908</td>
<td>0.655723</td>
</tr>
<tr>
<td>14</td>
<td>$-0^{+}$. $1^{+}$. $1^{+}$. $1^{+}$. $1^{+}$</td>
<td>opt, 0.514114</td>
<td>0.578125</td>
</tr>
<tr>
<td>15</td>
<td>$0^{+}$. $10^{+}$. $10^{+}$. $1^{+}$. $1^{+}$</td>
<td>opt, 0.99225</td>
<td>0.989258</td>
</tr>
<tr>
<td>16</td>
<td>$1^{+}$. $-0^{+}$. $0^{+}$</td>
<td>opt, 0.87500</td>
<td>0.887959</td>
</tr>
<tr>
<td>17</td>
<td>$1^{+}$. $1^{+}$. $-0^{+}$. $-0^{+}$</td>
<td>opt, 0.999023</td>
<td>1.000000</td>
</tr>
<tr>
<td>18</td>
<td>$0^{+}$. $+1^{+}$. $1^{+}$</td>
<td>opt, 0.998993</td>
<td>0.998047</td>
</tr>
<tr>
<td>19</td>
<td>$0^{+}$</td>
<td>opt, 0.87500</td>
<td>0.864258</td>
</tr>
<tr>
<td>20</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>21</td>
<td>$0^{+}$</td>
<td>opt, 0.50000</td>
<td>0.501953</td>
</tr>
<tr>
<td>22</td>
<td>$0^{+}$</td>
<td>opt, 0.50000</td>
<td>0.517578</td>
</tr>
<tr>
<td>23</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>24</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>25-33</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>34</td>
<td>$-0^{+}$</td>
<td>opt, 0.50000</td>
<td>0.507812</td>
</tr>
<tr>
<td>35-39</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>40</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>41-42</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>43-44</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>45-49</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>50, 52, 54, 56, 58</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>51, 53, 55, 57, 59</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>60</td>
<td>$+1^{+}$. $1^{+}$. $1^{+}$</td>
<td>opt, 0.992125</td>
<td>1.000000</td>
</tr>
<tr>
<td>61</td>
<td>$+1^{+}$. $1^{+}$. $1^{+}$</td>
<td>opt, 0.992125</td>
<td>1.000000</td>
</tr>
<tr>
<td>62</td>
<td>$-0^{+}$. $0^{+}$</td>
<td>0.390625 $\times 0.690375$</td>
<td>0.426758</td>
</tr>
<tr>
<td>63</td>
<td>$+0^{+}$</td>
<td>opt, 0.867188</td>
<td>0.855469</td>
</tr>
<tr>
<td>64</td>
<td>$-0^{+}$</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Table A.1: Differential path of the first near-collision block of the Flame certificate (Block 8)
<table>
<thead>
<tr>
<th>Steps</th>
<th>Bitconditions</th>
<th>Probability</th>
<th>Conditional estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>+.............</td>
<td>opt, 0.676736</td>
<td>0.749023</td>
</tr>
<tr>
<td>-2</td>
<td>-1...+. 1...0</td>
<td>opt, 0.552536</td>
<td>0.425781</td>
</tr>
<tr>
<td>-1</td>
<td>+01.-.+1 0--0</td>
<td>opt, 0.849149</td>
<td>0.492188</td>
</tr>
<tr>
<td>0</td>
<td>1-0...0-0-01</td>
<td>opt, 0.623376</td>
<td>0.833008</td>
</tr>
<tr>
<td>1</td>
<td>10-0...0-0-0-0</td>
<td>0.100466</td>
<td>0.546746</td>
</tr>
<tr>
<td>2</td>
<td>.1-0...0-0-0-0</td>
<td>0.398942</td>
<td>0.431254</td>
</tr>
<tr>
<td>3</td>
<td>.01-0...0-0-0</td>
<td>0.458008</td>
<td>0.518055</td>
</tr>
<tr>
<td>4</td>
<td>00...0-0-0-0-0</td>
<td>0.468258</td>
<td>0.672852</td>
</tr>
<tr>
<td>5</td>
<td>0000...0-0-0-0</td>
<td>opt, 0.960723</td>
<td>1.000000</td>
</tr>
<tr>
<td>6</td>
<td>0000000-0-0-0</td>
<td>opt, 0.468258</td>
<td>0.672852</td>
</tr>
<tr>
<td>7</td>
<td>.00000000-0-0</td>
<td>0.467804</td>
<td>0.495117</td>
</tr>
<tr>
<td>8</td>
<td>.000000000-0</td>
<td>0.910110</td>
<td>0.895508</td>
</tr>
<tr>
<td>9</td>
<td>.0000000000-0</td>
<td>0.877075</td>
<td>0.807617</td>
</tr>
<tr>
<td>10</td>
<td>.00000000000-0</td>
<td>0.875061</td>
<td>1.000000</td>
</tr>
<tr>
<td>11</td>
<td>.000000000000-0</td>
<td>0.996896</td>
<td>1.000000</td>
</tr>
<tr>
<td>12</td>
<td>.0000000000000-0</td>
<td>0.514206</td>
<td>0.586914</td>
</tr>
<tr>
<td>13</td>
<td>.00000000000000-0</td>
<td>0.992128</td>
<td>0.994141</td>
</tr>
<tr>
<td>14</td>
<td>.000000000000000-0</td>
<td>0.875000</td>
<td>0.879883</td>
</tr>
<tr>
<td>15</td>
<td>.0000000000000000-0</td>
<td>0.990023</td>
<td>1.000000</td>
</tr>
<tr>
<td>16</td>
<td>.00000000000000000-0</td>
<td>0.875000</td>
<td>0.895508</td>
</tr>
<tr>
<td>17</td>
<td>.000000000000000000-0</td>
<td>0.990023</td>
<td>0.99923</td>
</tr>
<tr>
<td>18</td>
<td>.0000000000000000000-0</td>
<td>0.875000</td>
<td>0.895508</td>
</tr>
<tr>
<td>19</td>
<td>.00000000000000000000-0</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>20</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>21</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>22</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>23</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>24</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>25-33</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>34</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>35</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>36-40</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>41-43</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>44-45</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>46</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>47</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>48</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>49-58</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>59</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>60</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>61</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>62</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>63</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
<tr>
<td>64</td>
<td>...0...0-0...0</td>
<td>0.500000</td>
<td>0.508789</td>
</tr>
</tbody>
</table>

Table A.2: Differential path of the second near-collision block of the Flame certificate (Block 9)
<table>
<thead>
<tr>
<th>Steps</th>
<th>Bitconditions</th>
<th>Probability</th>
<th>Conditional estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>.1.10100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>.0.0101-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>++1-+++++</td>
<td>opt, 0.833079</td>
<td>1.000000</td>
</tr>
<tr>
<td>1</td>
<td>0-111110</td>
<td>opt, 0.861702</td>
<td>0.96875</td>
</tr>
<tr>
<td>2</td>
<td>10-01110</td>
<td>0.403589, 0.408343</td>
<td>0.374023</td>
</tr>
<tr>
<td>3</td>
<td>-0-01^-1+</td>
<td>opt, 0.941415</td>
<td>1.000000</td>
</tr>
<tr>
<td>4</td>
<td>-0++-00</td>
<td>0.084686, 0.592803</td>
<td>1.000000</td>
</tr>
<tr>
<td>5</td>
<td>-1+++0-1+</td>
<td>opt, 0.775861</td>
<td>1.000000</td>
</tr>
<tr>
<td>6</td>
<td>++---------</td>
<td>opt, 0.516082</td>
<td>0.573242</td>
</tr>
<tr>
<td>7</td>
<td>1000-010</td>
<td>opt, 0.837596</td>
<td>1.000000</td>
</tr>
<tr>
<td>8</td>
<td>11+1.101</td>
<td>opt, 0.437474</td>
<td>0.056406</td>
</tr>
<tr>
<td>9</td>
<td>.0.1..-</td>
<td>opt, 0.516082</td>
<td>0.573242</td>
</tr>
<tr>
<td>10</td>
<td>.0-.1...</td>
<td>0.054927, 0.851331</td>
<td>0.120117</td>
</tr>
<tr>
<td>11</td>
<td>.0-...1+</td>
<td>opt, 0.822020</td>
<td>0.889648</td>
</tr>
<tr>
<td>12</td>
<td>.1-+-1...</td>
<td>opt, 0.710508</td>
<td>0.948242</td>
</tr>
<tr>
<td>13</td>
<td>.0+1-...1</td>
<td>opt, 0.753908</td>
<td>0.631836</td>
</tr>
<tr>
<td>14</td>
<td>.+-+...1</td>
<td>opt, 0.514114</td>
<td>0.585938</td>
</tr>
<tr>
<td>15</td>
<td>.0+...10</td>
<td>opt, 0.992125</td>
<td>0.993164</td>
</tr>
<tr>
<td>16</td>
<td>.1+...0...</td>
<td>opt, 0.875000</td>
<td>0.868164</td>
</tr>
<tr>
<td>17</td>
<td>.1...1.0...</td>
<td>opt, 0.999023</td>
<td>1.000000</td>
</tr>
<tr>
<td>18</td>
<td>.0...+.1...</td>
<td>opt, 0.998933</td>
<td>0.999023</td>
</tr>
<tr>
<td>19</td>
<td>.0........</td>
<td>opt, 0.875000</td>
<td>0.868164</td>
</tr>
<tr>
<td>20</td>
<td>.0........</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>21</td>
<td>.0........</td>
<td>opt, 0.500000</td>
<td>0.495117</td>
</tr>
<tr>
<td>22</td>
<td>............</td>
<td>opt, 0.500000</td>
<td>0.509766</td>
</tr>
<tr>
<td>23</td>
<td>............</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>24</td>
<td>............</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>25-33</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>34</td>
<td>.</td>
<td>opt, 0.500000</td>
<td>0.493164</td>
</tr>
<tr>
<td>35</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>36-41</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>42</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>43</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>44-46</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>47</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>48</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>49-59</td>
<td>.</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>60</td>
<td>.-+-0...</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>61</td>
<td>.0110.0.....</td>
<td>opt, 0.496094</td>
<td>0.514648</td>
</tr>
<tr>
<td>62</td>
<td>.0.01+.......</td>
<td>opt, 0.498047</td>
<td>0.492188</td>
</tr>
<tr>
<td>63</td>
<td>.++.?--......</td>
<td>opt, 0.404300</td>
<td>0.395508</td>
</tr>
<tr>
<td>64</td>
<td>.++.++++.....</td>
<td>opt, 1.000000</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

**Table A.3:** Differential path of the third near-collision block of the Flame certificate (Block 10)
<table>
<thead>
<tr>
<th>Steps</th>
<th>Bitconditions</th>
<th>Probability</th>
<th>Conditional estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>+............</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>+....0+....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>+....++....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>001.1+++. 01^1+11 ++-----0 11++11-</td>
<td>opt, 0.983143</td>
<td>1.000000</td>
</tr>
<tr>
<td>1</td>
<td>011.0.+-. +--+1+1+ ++0000-1 +--+0-11+</td>
<td>opt, 0.905244</td>
<td>0.742388</td>
</tr>
<tr>
<td>2</td>
<td>++--.0--. -+1+0-00 1+1+1++ -1+0++--</td>
<td>opt, 0.691042</td>
<td>0.756836</td>
</tr>
<tr>
<td>3</td>
<td>+++1-1+. +100++ 10---1+0 ---0++1</td>
<td>opt, 0.308664</td>
<td>1.000000</td>
</tr>
<tr>
<td>4</td>
<td>-01+0--. 10-1-01+ 0-000-1- 0+--10-1-</td>
<td>opt, 0.574448</td>
<td>1.000000</td>
</tr>
<tr>
<td>5</td>
<td>+00+00^ 0++11-0 +++0-111 01+100</td>
<td>opt, 0.748917</td>
<td>1.000000</td>
</tr>
<tr>
<td>6</td>
<td>++++++++ ++++++++ ++++++++</td>
<td>opt, 0.518059</td>
<td>0.506836</td>
</tr>
<tr>
<td>7</td>
<td>.111-110 01.010.0 0101111 0-100111</td>
<td>0.031742</td>
<td>0.475575</td>
</tr>
<tr>
<td>8</td>
<td>.111-110 01.010.0 0101111 0-100111</td>
<td>0.031742</td>
<td>0.475575</td>
</tr>
<tr>
<td>9</td>
<td>+...1... -.....1 0.1+... 1...-+</td>
<td>0.031742</td>
<td>0.475575</td>
</tr>
<tr>
<td>10</td>
<td>...0...0 ^0.....0 1..+0... .0....-+</td>
<td>opt, 0.910110</td>
<td>0.895508</td>
</tr>
<tr>
<td>11</td>
<td>...0+..&quot;0-1&quot;&quot;...01...1 &quot;1...&quot;1</td>
<td>opt, 0.877075</td>
<td>0.822266</td>
</tr>
</tbody>
</table>
| 12   | .01+.-0 0...-.1...-... .1...
| 13   | .1-1...0 1...00 100...1 1...1... | opt, 0.996096 | 1.000000 |
| 14   | .-+-10 1...00 1++11 1...1... | 0.031708 | 0.475575 |
| 15   | .0+...0 ++++++ ++++++ ++++++ | opt, 0.992128 | 0.998047 |
| 16   | .-+...0 ++++++ ++++++ ++++++ | opt, 0.875000 | 0.892578 |
| 17   | .1...1...1...0 ++++++ ++++++ | opt, 0.999023 | 1.000000 |
| 18   | .0...-...1 .1...1...1...1...1...1 | opt, 0.875000 | 0.860352 |
| 19   | 0........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 20   | 0........... .1...1...1...1...1...1 | opt, 0.500000 | 0.485352 |
| 21   | 0........... .1...1...1...1...1...1 | opt, 0.500000 | 0.501953 |
| 22   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 23   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 24   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 25-33 | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 34   | +........... .1...1...1...1...1...1 | opt, 0.500000 | 0.498047 |
| 35   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 36-39 | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 40   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 41-43 | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 44-46 | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 47   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 48   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 49-59 | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 60   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 61   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 62   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 63   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |
| 64   | +........... .1...1...1...1...1...1 | opt, 0.100000 | 1.000000 |

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.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>Trials</th>
<th>Successes</th>
<th>Trials/Success</th>
<th>Birthday factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1 = 1, v_1 = -1$</td>
<td>14284746</td>
<td>10000</td>
<td>$2^{10.3}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_1 = 1, v_1 = 0$</td>
<td>14284746</td>
<td>13199</td>
<td>$2^{10.1}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_1 = 1, v_1 = 1$</td>
<td>14284746</td>
<td>14182</td>
<td>$2^{10.0}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_1 = 1, v_1 = 2$</td>
<td>14284746</td>
<td>15637</td>
<td>29.8</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_1 = 1, v_1 = 3$</td>
<td>14284746</td>
<td>22535</td>
<td>$2^{9.3}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_1 = 2, v_1 = -1$</td>
<td>46417453</td>
<td>10000</td>
<td>$2^{12.2}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_1 = 2, v_1 = 0$</td>
<td>46417453</td>
<td>15113</td>
<td>$2^{11.6}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_1 = 2, v_1 = 1$</td>
<td>46417453</td>
<td>17009</td>
<td>$2^{11.4}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_1 = 2, v_1 = 2$</td>
<td>46417453</td>
<td>19427</td>
<td>$2^{11.2}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_1 = 2, v_1 = 3$</td>
<td>46417453</td>
<td>27297</td>
<td>$2^{10.7}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_1 = 3, v_1 = -1$</td>
<td>200785961</td>
<td>10000</td>
<td>$2^{14.3}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_1 = 3, v_1 = 0$</td>
<td>200785961</td>
<td>16419</td>
<td>$2^{13.6}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_1 = 3, v_1 = 1$</td>
<td>200785961</td>
<td>21104</td>
<td>$2^{13.2}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_1 = 3, v_1 = 2$</td>
<td>200785961</td>
<td>25874</td>
<td>$2^{12.9}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_1 = 3, v_1 = 3$</td>
<td>200785961</td>
<td>35030</td>
<td>$2^{12.5}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_1 = 4, v_1 = -1$</td>
<td>777345731</td>
<td>10000</td>
<td>$2^{16.2}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_1 = 4, v_1 = 0$</td>
<td>777345731</td>
<td>16682</td>
<td>$2^{15.5}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_1 = 4, v_1 = 1$</td>
<td>777345731</td>
<td>23470</td>
<td>$2^{15.0}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_1 = 4, v_1 = 2$</td>
<td>777345731</td>
<td>32195</td>
<td>$2^{14.6}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_1 = 4, v_1 = 3$</td>
<td>777345731</td>
<td>42718</td>
<td>$2^{14.2}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_1 = 5, v_1 = -1$</td>
<td>1112515535</td>
<td>10000</td>
<td>$2^{16.8}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_1 = 5, v_1 = 0$</td>
<td>1112515535</td>
<td>16366</td>
<td>$2^{16.1}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_1 = 5, v_1 = 1$</td>
<td>1112515535</td>
<td>22639</td>
<td>$2^{15.6}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_1 = 5, v_1 = 2$</td>
<td>1112515535</td>
<td>30735</td>
<td>$2^{15.1}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_1 = 5, v_1 = 3$</td>
<td>1112515535</td>
<td>40538</td>
<td>$2^{14.7}$</td>
<td>1</td>
</tr>
</tbody>
</table>

Table B.1: Results of the Monte Carlo simulation for the end segment of the first near-collision block.
Table B.2: Results of the Monte Carlo simulation for the second near-collision block.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Trials</th>
<th>Successes</th>
<th>Success probability</th>
<th>Birthday factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_2 = 0, v_2 = -1$</td>
<td>37689927528</td>
<td>10000</td>
<td>$2^{21.8}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 0$</td>
<td>37689927528</td>
<td>16042</td>
<td>$2^{21.2}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 1$</td>
<td>37689927528</td>
<td>17263</td>
<td>$2^{21.1}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 2$</td>
<td>37689927528</td>
<td>17695</td>
<td>$2^{21.0}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 3$</td>
<td>37689927528</td>
<td>17847</td>
<td>$2^{21.0}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 0, v_2 = 4$</td>
<td>37689927528</td>
<td>17932</td>
<td>$2^{21.0}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 1, v_2 = -1$</td>
<td>112707985708</td>
<td>10000</td>
<td>$2^{23.4}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_2 = 1, v_2 = 0$</td>
<td>112707985708</td>
<td>17360</td>
<td>$2^{22.6}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_2 = 1, v_2 = 1$</td>
<td>112707985708</td>
<td>19287</td>
<td>$2^{22.5}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_2 = 1, v_2 = 2$</td>
<td>112707985708</td>
<td>19909</td>
<td>$2^{22.4}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_2 = 1, v_2 = 3$</td>
<td>112707985708</td>
<td>20138</td>
<td>$2^{22.4}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_2 = 1, v_2 = 4$</td>
<td>112707985708</td>
<td>20219</td>
<td>$2^{22.4}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_2 = 2, v_2 = -1$</td>
<td>516950724692</td>
<td>10000</td>
<td>$2^{25.9}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_2 = 2, v_2 = 0$</td>
<td>516950724692</td>
<td>20135</td>
<td>$2^{24.6}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_2 = 2, v_2 = 1$</td>
<td>516950724692</td>
<td>24573</td>
<td>$2^{24.3}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_2 = 2, v_2 = 2$</td>
<td>516950724692</td>
<td>26045</td>
<td>$2^{24.2}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_2 = 2, v_2 = 3$</td>
<td>516950724692</td>
<td>26547</td>
<td>$2^{24.2}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_2 = 2, v_2 = 4$</td>
<td>516950724692</td>
<td>26733</td>
<td>$2^{24.2}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = -1$</td>
<td>270004515480</td>
<td>1000</td>
<td>$2^{28.0}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 0$</td>
<td>270004515480</td>
<td>2206</td>
<td>$2^{26.9}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 1$</td>
<td>270004515480</td>
<td>3066</td>
<td>$2^{26.4}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 2$</td>
<td>270004515480</td>
<td>3576</td>
<td>$2^{26.2}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 3$</td>
<td>270004515480</td>
<td>3755</td>
<td>$2^{26.1}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 3, v_2 = 4$</td>
<td>270004515480</td>
<td>3806</td>
<td>$2^{26.1}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_2 = 4, v_2 = -1$</td>
<td>1415975743623</td>
<td>1000</td>
<td>$2^{30.4}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 4, v_2 = 0$</td>
<td>1415975743623</td>
<td>2170</td>
<td>$2^{29.3}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 4, v_2 = 1$</td>
<td>1415975743623</td>
<td>3038</td>
<td>$2^{28.8}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 4, v_2 = 2$</td>
<td>1415975743623</td>
<td>3926</td>
<td>$2^{28.4}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 4, v_2 = 3$</td>
<td>1415975743623</td>
<td>4456</td>
<td>$2^{28.2}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 4, v_2 = 4$</td>
<td>1415975743623</td>
<td>4637</td>
<td>$2^{28.2}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = -1$</td>
<td>10415440989433</td>
<td>1120</td>
<td>$2^{31.1}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 0$</td>
<td>10415440989433</td>
<td>2524</td>
<td>$2^{31.0}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 1$</td>
<td>10415440989433</td>
<td>3668</td>
<td>$2^{31.0}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 2$</td>
<td>10415440989433</td>
<td>5008</td>
<td>$2^{30.6}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 3$</td>
<td>10415440989433</td>
<td>6463</td>
<td>$2^{30.4}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_2 = 5, v_2 = 4$</td>
<td>10415440989433</td>
<td>7365</td>
<td>$2^{30.4}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = -1$</td>
<td>7392736230494</td>
<td>130</td>
<td>$2^{35.7}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 0$</td>
<td>7392736230494</td>
<td>295</td>
<td>$2^{34.5}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 1$</td>
<td>7392736230494</td>
<td>422</td>
<td>$2^{34.0}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 2$</td>
<td>7392736230494</td>
<td>565</td>
<td>$2^{33.6}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 3$</td>
<td>7392736230494</td>
<td>760</td>
<td>$2^{33.2}$</td>
<td>1</td>
</tr>
<tr>
<td>$w_2 = 6, v_2 = 4$</td>
<td>7392736230494</td>
<td>990</td>
<td>$2^{32.8}$</td>
<td>1</td>
</tr>
</tbody>
</table>


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

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

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Trials</th>
<th>Successes</th>
<th>Trials/Success</th>
<th>Birthday factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_4 = 1, u_4 = 0$</td>
<td>14986221886</td>
<td>10000</td>
<td>$2^{20.5}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_4 = 1, u_4 = 1$</td>
<td>12080362522</td>
<td>10000</td>
<td>$2^{20.2}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_4 = 1, u_4 = 2$</td>
<td>40666946131</td>
<td>10000</td>
<td>$2^{22.0}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_4 = 1, u_4 = 3$</td>
<td>259469948224</td>
<td>10000</td>
<td>$2^{24.6}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_4 = 1, u_4 = 4$</td>
<td>350239928299</td>
<td>10000</td>
<td>$2^{25.1}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_4 = 2, u_4 = 0$</td>
<td>31473408443</td>
<td>10000</td>
<td>$2^{21.6}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_4 = 2, u_4 = 1$</td>
<td>27095762586</td>
<td>10000</td>
<td>$2^{21.4}$</td>
<td>$2^{3.5}$</td>
</tr>
<tr>
<td>$w_4 = 2, u_4 = 2$</td>
<td>89336968111</td>
<td>10000</td>
<td>$2^{23.1}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_4 = 2, u_4 = 3$</td>
<td>573516490688</td>
<td>10000</td>
<td>$2^{25.8}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_4 = 2, u_4 = 4$</td>
<td>785283707839</td>
<td>10000</td>
<td>$2^{26.2}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_4 = 3, u_4 = 0$</td>
<td>107212346493</td>
<td>10000</td>
<td>$2^{23.4}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_4 = 3, u_4 = 1$</td>
<td>80588979585</td>
<td>10000</td>
<td>$2^{22.9}$</td>
<td>$2^{3.0}$</td>
</tr>
<tr>
<td>$w_4 = 3, u_4 = 2$</td>
<td>289767080938</td>
<td>10000</td>
<td>$2^{24.8}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_4 = 3, u_4 = 3$</td>
<td>1729323958821</td>
<td>10000</td>
<td>$2^{27.4}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_4 = 3, u_4 = 4$</td>
<td>2385632329208</td>
<td>10000</td>
<td>$2^{27.8}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_4 = 4, u_4 = 0$</td>
<td>391719772246</td>
<td>10000</td>
<td>$2^{25.2}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_4 = 4, u_4 = 1$</td>
<td>30236921493</td>
<td>10000</td>
<td>$2^{24.8}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_4 = 4, u_4 = 2$</td>
<td>1050401211933</td>
<td>10000</td>
<td>$2^{26.6}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_4 = 4, u_4 = 3$</td>
<td>55771294749</td>
<td>100</td>
<td>$2^{29.1}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_4 = 4, u_4 = 4$</td>
<td>120935577619</td>
<td>100</td>
<td>$2^{30.2}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_5 = 1, u_4 = 0$</td>
<td>1500432742222</td>
<td>10000</td>
<td>$2^{27.2}$</td>
<td>$2^{2.5}$</td>
</tr>
<tr>
<td>$w_5 = 1, u_4 = 1$</td>
<td>1265003442364</td>
<td>10000</td>
<td>$2^{27.0}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_5 = 1, u_4 = 2$</td>
<td>4223606267373</td>
<td>10000</td>
<td>$2^{28.7}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_5 = 1, u_4 = 3$</td>
<td>22126048157</td>
<td>100</td>
<td>$2^{31.0}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_5 = 1, u_4 = 4$</td>
<td>25434380301</td>
<td>100</td>
<td>$2^{31.2}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_5 = 2, u_4 = 0$</td>
<td>72363355156</td>
<td>100</td>
<td>$2^{29.4}$</td>
<td>$2^{2.0}$</td>
</tr>
<tr>
<td>$w_5 = 2, u_4 = 1$</td>
<td>45938374478</td>
<td>100</td>
<td>$2^{28.8}$</td>
<td>$2^{1.5}$</td>
</tr>
<tr>
<td>$w_5 = 2, u_4 = 2$</td>
<td>10502546473</td>
<td>100</td>
<td>$2^{30.0}$</td>
<td>$2^{1.0}$</td>
</tr>
<tr>
<td>$w_5 = 2, u_4 = 3$</td>
<td>1228561139591</td>
<td>120</td>
<td>$2^{33.3}$</td>
<td>$2^{0.5}$</td>
</tr>
<tr>
<td>$w_5 = 2, u_4 = 4$</td>
<td>2419872905855</td>
<td>120</td>
<td>$2^{34.2}$</td>
<td>1</td>
</tr>
</tbody>
</table>
Bibliography


