Attacks on Hash Functions and Applications

Marc Stevens

Attacks on Hash Functions and Applications

PROEFSCHRIFT

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof. mr. P.F. van der Heijden,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 19 juni 2012
klokke 15.00 uur

door

Marc Martinus Jacobus Stevens,

geboren te Hellevoetsluis
in 1981

Samenstelling van de promotiecommissie:

Promotores:

Prof. dr. R. Cramer (CWI & Universiteit Leiden)
Prof. dr. A.K. Lenstra (École Polytechnique Fédérale de Lausanne)

Copromotor:

Dr. B.M.M. de Weger (Technische Universiteit Eindhoven)

Overige leden:

Prof. dr. E. Biham (Technion)
Dr. B. Schoenmakers (Technische Universiteit Eindhoven)
Prof. dr. P. Stevenhagen (Universiteit Leiden)
Prof. dr. X. Wang (Tsinghua University)

The research in this thesis has been carried out at the Centrum Wiskunde & Informatica in the Netherlands and has been funded by the NWO VICI grant of Prof. dr. R. Cramer.

Attacks on Hash Functions and Applications

Attacks on Hash Functions and Applications
Marc Stevens
Printed by Ipskamp Drukkers
AMS 2000 Subj. class. code: 94A60
NUR: 919
ISBN: 978-94-6191-317-3

Universiteit Leiden

Centrum Wiskunde & Informatica

Cover design by Ankita Sonone.

Contents

Part I
Introduction

Pour the initial value in a big cauldron and place it over a nice fire. Now slowly add salt if desired and stir well. Marinade your input bit string by appending some strengthened padding. Now chop the resulting bit string into nice small pieces of the same size and stretch each piece to at least four times its original length. Slowly add each single piece while continually stirring at the speed given by the rotation constants and spicing it up with some addition constants. When the hash stew is ready, extract a nice portion of at least 128 bits and present this hash value on a warm plate with some garnish.

Recipe 1: *Old Hash Recipe*

1 Hash functions

Contents

1.1 Authenticity and confidentiality

Authenticity and confidentiality have been big concerns in communications for millennia. One of the first known examples of confidentially sending a message goes back to around 440 BC to "The Histories" written down by Herodotus. According to Herodotus, Histiaeus sent an important message to his son-in-law Aristagoras. As this message was very sensitive he tattooed it on the shaved head of his most-trusted slave. Later, when enough hair had grown back, he sent his slave to Aristagoras with instructions to shave the head of the slave again.

There is a risk with hidden messages that somebody finds the hidden message and exposes it to unfriendly parties. This is especially the case for regular communications. Eventually the idea arose to encrypt messages so that they cannot be read even if intercepted and this has led to the development of *cryptology*, historically known as *the art of building and breaking ciphers.* Ciphers allow parties to securely communicate by encrypting their messages with some secret knowledge (the secret key) into unreadable ciphertext that can only be read by parties that possess the same secret knowledge. This form of encryption using a single secret key known to both sender and receiver is called *symmetric encryption.* The most famous ancient cipher, the Caesar cipher, is named after Julius Caesar who encrypted his most important messages with a secret number N by cyclically substituting each letter with the N-th next letter in the alphabet (if $N = 3$ then A goes to D, B to E, etc.).

Authentication of messages, the act of confirming that a message really has been sent by a certain party, usually was achieved by a combination of inspecting the message (e.g., verify its signature), the 'envelope' (e.g., is the wax seal intact) and/or

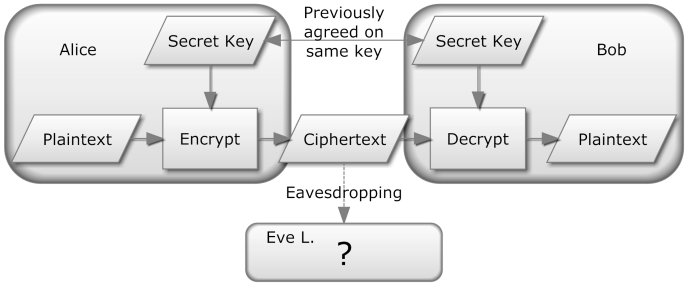

Figure 1: *Symmetric Encryption*

the messenger (e.g., is the messenger known and/or in service of either party). Symmetric encryption also provides some form of authentication. After all, no one else knows the secret key and is able to encrypt messages with it. However, this does not prevent the encrypted message from being purposely changed or simply repeated at an opportune moment by unfriendly parties. In general, authentication of someone (or the sender/creator of something) is achieved through mutual knowledge (such as a secret password), possession of a physical token (such as the king's seal) and/or distinguished marks (such as a known birthmark).

1.2 Rise of a digital world

With the invention of electronics, a new world was born. Digital information can be very cheaply stored and copied without loss of information. The rapid development of computers opened up a huge range of new possibilities in processing information. The explosive growth of telecommunication networks has made it very easy to quickly send information over vast distances to anywhere in the world. So not surprisingly, our society has become more and more dependent on information technologies in just a few decades.

The main advantage of digital information is that there is no longer a link between the information and its carrier. It can be copied endlessly without any loss of quality. The same information can be stored on, e.g., a hard disk, USB disk, mobile phone or a DVD, or sent through an Ethernet, Wi-Fi, GSM or satellite link.

This also introduces a large vulnerability as now authentication of information cannot be guaranteed anymore by simply looking at the physical carrier. Also, ciphers require two parties to first agree on a secret key. Given the vast number of possible parties in the world anyone might want to communicate privately with, establishing and storing secret keys for every possible connection between two parties becomes prohibitive.

Solving these security problems and many more that arose has expanded the field of cryptography. Modern cryptography has grown into a science that encompasses much more than only ciphers and has strong links with mathematics, computer science and (quantum) physics. One of the key innovations of modern cryptography that

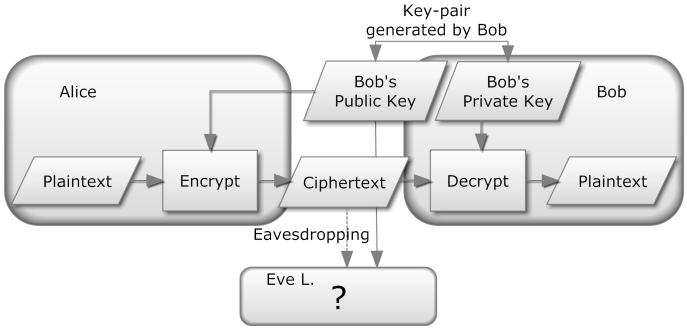

Figure 2: *Asymmetric or Public Key Encryption*

gave rise to new ideas and approaches is *public key encryption* (also called *asymmetric encryption*) [DH76, Mer78, RSA78]. Instead of using one secret key for both encryption and decryption, public key encryption uses a key pair consisting of a public key for encryption and a private key for decryption.

Suppose everyone publishes the public key part of their properly generated key pair in a public key listing (like a phone book). Now, securely sending a message to someone is possible without first exchanging a secret key in advance anymore, as you simply encrypt your message with his public key as found in the listing. Only he can now decrypt your ciphertext using the private key known only to him. This is a remarkable invention, although it depends heavily on the fact that no one can derive the private key from the public key. Furthermore, it requires a guarantee from the public key listing that every listed public key is generated by the associated person and not by an impostor.

Ideas similar to that of public key encryption have also led to *digital signatures* [RSA78]. With digital signatures you have a key pair as well which now consists of a signing key and a verification key. The verification key is made public and should be collected into a public listing such as that for public encryption keys. The signing key remains private and can be used to generate a digital signature for a message, which is then simply appended to the message. Anyone can verify the correctness of the digital signature of a message using the publicly listed verification key and thereby obtains proof of authenticity. However, only someone who knows the signing key can actually generate signatures. Similar to public key encryption, the security of digital signatures depends heavily on the fact that no one can derive the signing key from the verification key and also requires a guarantee from the public key listing that every listed verification key is generated by the associated person and not by an impostor.

1.3 Security frameworks

To analyze the security of cryptographic systems it is important to precisely define the desired notion of security. First, it needs to be clear what a possible adversary can and cannot do, e.g., whether he can eavesdrop communications or whether he is

able to alter messages. Then security notions can be expressed as problems for an adversary, such as finding a private key belonging to a given public key, or decrypting a message without knowledge of the private key, that are hopefully then shown to be hard to solve. However, there is no single good definition of how hard a problem is.

Most of modern cryptography uses one of the following three approaches to analyze the hardness of a problem: the information theoretic approach, the complexity theoretic approach and the system based approach. These approaches differ mainly on their assumptions about the computational limitations of an adversary. The notion of security becomes stronger when fewer limitations are assumed on the adversary. However, most public key cryptography requires at least some reasonable assumption about the limitations on the adversary, i.e., without any assumptions it is even possible that no secure solutions can be found. Each of these three approaches has its advantages and its practical use.

1.3.1 Information theoretic approach

The information theoretic approach makes no assumptions on the computational power of the adversary. This approach has been developed by Claude Shannon [Sha48] to find the fundamental limits on information processing operations such as compression, storage and communication. Shannon introduced the notion of entropy that quantifies information and which intuitively can be seen as the amount of uncertainty an entity has about a piece of information. More practically it can also be seen as the average number of bits you need to store the information.

This approach allows for unconditional security; security which is independent of the computing power of an adversary. It has been shown by Shannon [Sha49] that unconditional (or perfect) privacy protection can be achieved when the length of the uniformly randomly chosen encryption key is at least the length of the plaintext and the encryption key is only used once. More precisely, unconditional privacy requires that the entropy of the key is at least the entropy of the plaintext. Furthermore, unconditional message authentication can also be achieved at the cost of a large secret key.

The main advantage of this approach is that it allows unconditional security. However, the large secret keys which it usually requires makes it rather unpractical.

1.3.2 Complexity theoretic approach

Complexity theory tries to classify computational problems by their inherent difficulty. To quantify this difficulty, this approach utilizes a model of computation such as a Turing machine. In this model, algorithms are measured by the amount of time and/or the amount of space (memory) they use as a function in the size of the input. In general, algorithms are called *efficient* if their runtime and space are bounded by a polynomial function in the size of the input. An adversary is limited in this approach to a certain class of algorithms which is usually the class of efficient algorithms.

Computational problems are seen as an infinite collection of instances with a solution for every instance. To each such computational problem belongs a collection of

algorithms that can solve each instance of the problem (sometimes a certain probability of failure is allowed). As an adversary is limited to the use of efficient algorithms, computational problems that are generally assumed to have no efficient algorithms are of special interest. Examples are the *factoring problem* – finding the prime factors of a composite integer number – and the *discrete logarithm problem* – finding a number x such that $h = g^x \bmod p$ for a given prime p and $g, h \in \{1, \ldots, p-1\}$ [1]. In fact, the well-known RSA public-key encryption system [RSA78] would be broken if the factoring problem has an efficient algorithm. This approach allows the construction of cryptographic primitives (such as public key encryption, digital signatures, zero-knowledge proofs, etc.) for which it can be proven that breaking the cryptographic primitive implies the existence of an efficient algorithm that solves the underlying computational problem. Such primitives are often (a bit misleadingly) called *provably secure* although such proofs only reduce the primitive's security to the *assumed* hardness of some computational problem.

The advantages of this approach are that one can obtain provable security (based on a number of assumptions) and that secret keys can be smaller than in the information theoretic approach. It also has some inherent disadvantages. The distinction of an algorithm being efficient is mainly asymptotic. For concrete input lengths an algorithm with exponential runtime can be a lot faster than an algorithm with polynomial runtime, e.g., due to a large difference in the constant factor. This implies that provable security gives no information on the security of concrete instances. Also, for real world instances an efficient algorithm, such as an encryption algorithm, may have an impractically large runtime; making it impractical for use in cryptographic systems.

1.3.3 System based approach

The system based approach is based on the real world efficiency of software and hardware implementations and tries to produce very practical cryptographic primitives. Analyzing the hardness of a problem instance is based on estimating the necessary real-world resources (in computing power, memory, dedicated hardware, etc) as required by the best known algorithm that solves the problem. Compared to the complexity theoretic approach wherein an algorithm is deemed efficient if its asymptotic complexity is bounded by a polynomial function, the system based approach analyzes concrete problem instances and deems solving a concrete problem instance as feasible if the necessary real-world resources can be obtained within a certain reasonable monetary budget.

Concrete cryptographic primitives are designed to be fast and to make breaking them a hard problem. Similar to a cat-and-mouse game, several attack principles have been invented over the years to analyze and break older primitives, each leading to new designs that try to avoid practical attacks based on those principles.

The advantage of this approach is that it results in very fast cryptographic primitives. The main disadvantages are that the security is mainly based on preventing

1. $g^x \bmod p$ denotes the remainder of g^x after division by p

known attacks and that the designs might seem ad-hoc. The remainder of this thesis mainly uses the system based approach.

1.4 Cryptographic hash functions

1.4.1 One-way functions

In computer science and cryptography there is a type of function that is both theoretically and practically important: *one-way functions*. A one-way function maps a huge (possibly infinite) domain (e.g., of all possible messages) to some (possibly infinite) range (e.g., all bit strings of length 256). The predicate one-way means that such a function is easy to compute whereas it is hard to invert, i.e., for a given output it is hard to find an input that maps to that output. The one-wayness property is not rigorously defined and depends on the security framework used to analyze the hardness of the inverting problem. As the very definition of a one-way function depends on the security framework used, so does the question of their existence.

The information theoretic view leads to the most negative result: one-way functions do not even exist in this view. An adversary with unlimited computing power can simply test every possible input until he finds one that results in the given output.

All known one-way functions in the complexity theoretical approach are actually based on (well-established) assumptions. The existence of one-way functions is thus assumed. Nevertheless, finding an actual proof remains a very interesting problem. In fact, a proof of their existence would also prove that the complexity classes P and NP are distinct which thereby resolves one of the foremost unsolved questions of computer science. The existence of one-way functions also implies the existence of various cryptographic primitives such as pseudo-random generators [ILL89], digital signature schemes [Rom90], zero-knowledge protocols, message authentication codes and commitment schemes.

In cryptography there are several problems that are *assumed* to be hard and can be used to define a one-way function. The earlier mentioned factoring problem leads to the one-way function that maps two very large (say at least 2048-bits) prime numbers p and q to their product $p \cdot q$. Similarly, the discrete logarithm problem leads to the one-way function that maps a number e to $g^e \bmod p$, for a given integer g and a large prime number p.

For cryptography, an important class of one-way functions is the class of one-way *hash* functions, or simply *hash functions*, that operate on bit strings and map bit strings of arbitrary length to a bit string of a fixed length (such as 256 bits) which is called the *hash*.

1.4.2 Cryptographic requirements of hash functions

A major application for hash functions is found in constructing efficient digital signature schemes. In 1978, Rabin [Rab78] introduced the idea of signing the hash of

a document M instead of directly signing it[2]. Signing a large message directly with a public-key crypto system is slow and leads to a signature as large as the message. Reducing a large message to a small hash using a hash function and signing the hash is a lot faster and leads to small fixed-size signatures. Forging a signature then requires either breaking the public-key crypto system or finding a document M' that has the same hash as a different signed document M.

In this case, the security of digital signatures is also based on the hardness of the problem of finding a *collision*, i.e., finding two different documents M and M' that have the same hash. Since if either M or M' is signed, then the corresponding signature is also valid for the other document, resulting in a successful forgery. The actual value of the hash is unimportant as long as both documents have the same hash.

Here it already becomes clear that for cryptographic purposes the one-wayness of a hash function is not enough. For cryptographic use, the following nine cryptographic properties of finite families $\mathcal{F}$ of hash functions with identical output bit length are considered: Pre, aPre, ePre, Sec, aSec, eSec, Coll, MAC and PRF. For a more thorough treatment of these properties, we refer to [RS04], [BR06a] and [BR07].

Pre, ePre and aPre: The first three are based on variations of the problem of finding a pre-image M for a given hash h and are definitions of one-wayness. Pre, which stands for *pre-image resistance*, requires that given a uniformly randomly chosen hash function f from a family $\mathcal{F}$ and a uniformly randomly chosen hash h it is hard to find a pre-image $M \in f^{-1}(h)$. The stronger notion ePre (*everywhere pre-image resistance*) allows the adversary to choose the hash h before f is uniformly randomly chosen from $\mathcal{F}$. The third notion aPre (*always pre-image resistance*) allows the adversary to choose the hash function f before the hash h is uniformly randomly chosen and given to the adversary.[3]

Sec, eSec and aSec: The next three are based on variations of the problem of finding another message M' that has the same hash as a given message M. Sec (*second pre-image resistance*) requires f and M to be uniformly randomly chosen. The stronger eSec (*everywhere Sec*) allows the adversary to choose M before f is uniformly randomly chosen and aSec (*always Sec*) allows the adversary to choose f before M is uniformly randomly chosen and given to the adversary.[3]

Coll: Coll (*collision resistance*) is the property that finding two different messages M and M' that have the same hash $f(M) = f(M')$, where f is a hash function chosen randomly from $\mathcal{F}$ is a hard problem. This property is often the first to be broken and hence requires special attention.

2. Although at the time no such hash functions were available. The first design towards hash functions as used today is MD4 which was introduced in 1990.
3. These variations of Pre (and Sec) may seem to be very similar; nevertheless, their distinctions are theoretically important. For instance in the case of a fixed hash function, where there is no hash function family to speak of, aPre and aSec are the only applicable notions.

MAC: The MAC (*message authentication code unforgeability*) property views the entire hash function family $\mathcal{F}$ as a single keyed hash function f_K where K is a randomly chosen secret key. The adversary does not get direct access to K and f_K, instead the adversary can make queries q_i to an oracle that responds with the output hashes $r_i = f_K(q_i)$. The MAC property requires that the problem of finding a message M and its correct hash under f_K, where M is not one of the queries q_i, is hard.

PRF: For the PRF (*pseudo random function*) property, the problem of distinguishing between an oracle to f_K for a randomly chosen secret K and a random function oracle that responds to queries q_i with randomly chosen hashes r_i must be hard.

In the case of a fixed hash function f instead of a hash function family $\mathcal{F}$, only three properties are considered: pre-image resistance, second pre-image resistance and collision resistance. The pre-image resistance and second pre-image resistance properties are identical to the above aPre and aSec notions, respectively.

However, when formally defining collision resistance for such a fixed hash function f one runs into the *foundations-of-hashing dilemma* [Rog06]. Collision resistance is an intuitive concept and one would like to mathematically define collision resistance as the hardness of some problem or equivalently as some statement "*there is no efficient algorithm that produces collisions*". Unfortunately, for every collision $M \neq M'$ of f, there is a trivial algorithm that simply outputs that collision: M and M'. Nonetheless, if there are no collisions known then one cannot actually write down such a trivial algorithm. Thus one would like to think of collision resistance as the statement "*there is no* known *efficient algorithm that produces collisions*", which may be impossible to define mathematically. Rogaway [Rog06] treats the foundations-of-hashing dilemma in detail and provides a formal way to handle collision resistance, but does not provide a clear definition for collision resistance.

Thus in this thesis we do not define the collision resistance property for a fixed hash function in terms of some mathematical problem that must be hard, rather we view collision resistance as the property that there are no known explicit efficient algorithms. This informal definition of collision resistance is sufficient for the remainder of this thesis as we focus on counter-examples to the collision resistance of fixed hash functions.

1.4.3 General attacks

Even though we would prefer breaking these security properties to be impossible, often there exists a general attack breaking a security property that thereby presents a fundamental upper bound for an adversary's attack complexity. A hash function is called *broken* when there exists a known explicit attack that is faster than the general attack for a security property. It must be noted that even unbroken hash functions may be insecure in the real-world, e.g., a general attack becomes feasible due to a too small hash bit length compared to the possible real-world computational power of adversaries.

The best known general attack to break Pre, aPre, ePre, Sec, aSec and eSec is a brute force search, where hashes $f(M')$ are computed for randomly chosen messages M' until a message M' is found where $f(M')$ is the target hash value (and $M' \neq M$ for Sec, aSec, eSec). For a hash function (family) with an output hash size of N bits, this attack succeeds after approximately 2^N evaluations of the hash function. Already for $N \geq 100$ this attack is clearly infeasible in the real world for the present day and near future.

To find collisions, one can do a lot better with a general attack based on the birthday paradox. The birthday paradox is the counter-intuitive principle that for groups of as few as 23 persons there is already a chance of about one half of finding two persons with the same birthday (assuming all birthdays are equally likely and disregarding leap years). Compared to finding someone in this group with your birthday where you have 23 independent chances and thus a success probability of $\frac{23}{365} \approx 0.06$, this principle is based on the fact that there are $\frac{23*22}{2} = 253$ distinct pairs of persons. This leads to a success probability of about 0.5 (note that this does not equal $\frac{253}{365} \approx 0.7$ since these pairs are not independently distributed).

For a given hash function with output size of N bits, this general algorithm succeeds after approximately $\sqrt{\pi/2} \cdot 2^{N/2}$ evaluations of the hash function [vOW99]. This means that for a hash size of $N = 128$ bits (which was commonly used until recently) finding a collision needs approximately $2^{64.3} \approx 22 \cdot 10^{18}$ evaluations, which is currently just in reach for a large computing project[4], whereas inverting the hash function takes about a factor of $15 \cdot 10^{18}$ longer.

1.4.4 From compression function to hash function

With the need for secure practical hash function designs for use in digital signatures schemes well known [Rab78, Yuv79, DP80, Mer82, Rom90], the first attempts to construct a hash function were made in the 1980s. An immediately recognized design approach to tackle the problem of arbitrary length inputs was to base the security of the hash function on the security of a function with fixed size inputs, such as a block cipher which is an already well studied cryptographic primitive. More generally for the purpose of hash functions, such a fixed input size function is called a *compression function* as it must have an output length smaller than its input length.

The compression function is then repeatedly used in some mode of operation to process the entire message. The well known Merkle-Damgård construction (named after the authors of the independent papers [Mer89, Dam89] published in 1989) describes exactly how to construct a hash function based on a general compression function in an iterative structure as is depicted in Figure 3. Since these papers have proven that the hash function is collision resistant if the underlying compression function is collision resistant, the majority of currently used hash functions are based on

4. The distributed computing project MD5CRK started in March 2004 a brute force search for collisions for the hash function MD5. The project was stopped in August 2004 due to breakthrough cryptanalytic research [WFLY04, WY05]. As a rough estimate, the attack can currently be done in one year on 40.000 quad-core machines or on 1000 high-end graphics cards.

this Merkle-Damgård construction.

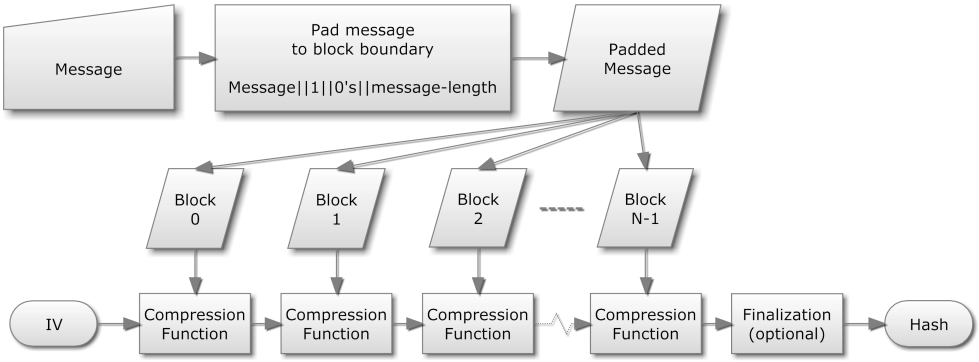

Figure 3: *Merkle-Damgård Construction*

The construction builds a hash function based on a compression function that takes two inputs: a chaining value or IHV_{in} (*Intermediate Hash Value*) of K bits and a message block of N bits, and outputs a new K-bit IHV_{out}. An input message is first padded with a single '1'-bit followed by a number X of '0'-bits and lastly the original message length encoded into 64 bits. The number X of '0'-bits to be added is defined as the lowest possible number so that the entire padded message bit length is an integer multiple of the message block length N. The padded message is now split into blocks of size exactly N bits. The hash function starts with a fixed public value for IHV_0 called the IV (*Initial Value*). For each subsequent message block it calls the compression function with the current IHV_i and the message block M_i and store the output as the new IHV_{i+1}. When we focus only on the compression function, we write IHV_{in} and IHV_{out} for the input IHV_i and the output IHV_{i+1}, respectively. After all blocks are processed it outputs the last IHV_N after an optional finalization transform.

The Merkle-Damgård construction has several weaknesses, none of which pose a real-world threat against the use of currently commonly used hash functions based on the construction. In 2004, Joux [Jou04] showed that finding a *multi-collision* – defined as 2^t messages that all have the same hash value – costs only about t times a single collision attack. Furthermore, cascaded hash functions $f(M) = g(M)||h(M)$, obtained by concatenating the output from two different hash functions, were often expected to yield a more secure hash function than either g and h are. However, if either g or h is based on Merkle-Damgård then using multi-collisions Joux showed that f is as secure as the most secure hash function of g and h.

In 2005, a general second pre-image attack against Merkle-Damgård based hash functions was published [KS05] which can find a second pre-image for a given message consisting of about 2^k blocks in about $2^{n-k+1} + k2^{n/2+1}$ evaluations of the hash function instead of the 2^n evaluations of the brute force attack.

In 2006, it has been shown that the Merkle-Damgård construction preserves also

the ePre property from a compression function besides the Coll property, but fails to preserve Pre, aPre, Sec, aSec and eSec ([ANPS07, BR06a]). Several new constructions have been proposed that preserve many (but not all) of these properties, e.g., ROX [ANPS07], EMD [BR06a] and ESh [BR07].

1.4.5 The MD4 style family of hash functions

Based on the work by Merkle and Damgård and 10 years after the idea of using hash functions in digital signatures [DP80] was introduced, Ron Rivest presented in 1990 the dedicated hash function MD4 [Riv90a, Riv90b] as a first attempt using the Merkle-Damgård construction. MD4 was quickly superseded by MD5 [Riv92] in 1992 due to security concerns [dBB91]. MD5 has found widespread use and remains commonly used worldwide.

MD4 and MD5 have formed an inspiration for other hash function designs that have followed over the years. Several of them can be seen as part of a MD4 style family of hash functions as they are based on Merkle-Damgård and have the same basic design of a compression function:

- SHA-0 [NIS93] (designed by the NSA, the National Security Agency of the USA, in 1993);
- SHA-1 [NIS95] (the replacement of SHA-0 by the NSA in 1995 after undisclosed security concerns were found in SHA-0);
- SHA-2 [NIS02, NIS08, NIS11] consisting of SHA-224, SHA-256, SHA-384 and SHA-512 (designed in 2002 by NSA);
- RIPEMD [BP95];
- RIPEMD-160 [DBP96] consisting of strengthened versions of RIPEMD;
- and many others.

An MD4 style compression function (Figure 4) takes a message block and generates an expanded message block split into R pieces $m_0, \ldots, m_{R-1}$ whose total bit length is at least three times the block length (MD5 and SHA-1 use four and five times, respectively). It initializes a working state with the input IHV called IHV_{in} and iteratively updates this working state with a *step function* and consecutive pieces $m_0, \ldots, m_{R-1}$, very similar to the Merkle-Damgård construction. Lastly it outputs the new IHV called IHV_{out} as the sum of the input IHV_{in} and the last working state. The security of the compression function mainly depends on highly complex bit dependencies in the step function output and a high expansion factor (which is the expanded message block bit length divided by the input message block bit length).

All members of the MD4 style hash function family have different step functions; nevertheless, they also have many similarities. In the case of MD5 the step function is sketched in Figure 5, where the working state consists of four variables A, B, C

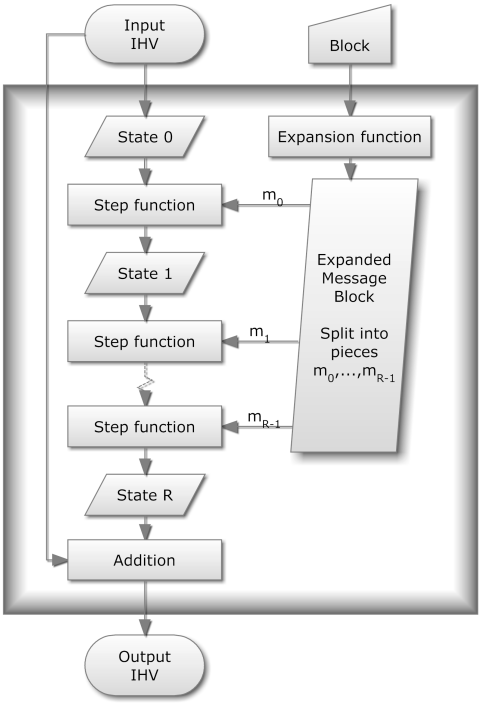

Figure 4: *MD4 Style Compression Function*

and D which are both seen as elements of $\mathbb{Z}_{2^{32}}$ and as 32-bit strings.[5] The variable A is updated by adding a message piece m_i, an addition constant K_i and the result $F(B, C, D)$ of a bit-wise function F (the i-th bit of the result only depends on the i-th bits of the inputs). Next, A is bit-wise left rotated by s (the rotation constant) bits and finally the input value of B is added. Let $\widehat{A}$ denote the updated variable A:

$$\widehat{A} = B + RL(A + m_i + K_i + F(B, C, D), s)$$

The output working state A', B', C' and D' consists of the rotated variables: $A' = D$, $B' = \widehat{A}$, $C' = B$ and $D' = C$. The step function itself varies a little between the R steps in MD5 in that the constant values K_i and s and the bit-wise function F are changed.

The step function is non-linear so that the compression function cannot be simply described as a linear system of equations that can easily be solved and thus inverted. Moreover, it tries to create very complex dependencies of IHV_{out} on all the message block bits. It does so by mixing different mathematical operations (like modular addition, bit-wise functions and bit-wise rotation) for which there is no unified 'calculus' that allows to simplify the mathematical expression defining the compression

5. Throughout this thesis we use $\mathbb{Z}_{2^{32}}$ as shorthand for $\mathbb{Z}/2^{32}\mathbb{Z}$.

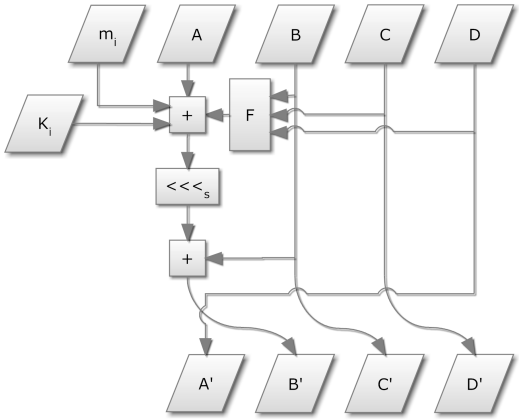

Figure 5: *Step Function of MD5's Compression Function*

function. This effect is amplified by using each message bit multiple times (4 times in MD5) over various spread-out steps. As a result of the lack of such a calculus, solving a set of equations over the compression function that results in a collision or pre-image attack appears to be very difficult.

Note that this step function, like the one of MD4 and other MD4-style hash functions, is efficiently and uniquely invertible given the output working state and the message piece. Therefore, the addition at the end of the compression function is necessary to avoid that inverting the entire compression function is easy. Without this addition, the inverting problem reduces to finding R pre-images of the step function, which can be done just as fast as evaluating the compression function.

1.5 Differential cryptanalysis

The most successful type of cryptographic analysis of the MD4 style hash function family is *differential cryptanalysis*. In differential cryptanalysis one looks at two evaluations of a given hash function at the same time. By analyzing how differences between those two evaluations caused by message differences propagate throughout the computation of hash function, one can try to control those differences and try to build an efficient attack. This type of analysis is mainly used for collision attacks, but sometimes extends to second pre-image attacks.

Differential cryptanalysis has been publicly known since the first published attacks against the DES (Data Encryption Standard) cipher by Biham and Shamir since 1990 [BS90, BS91, BS92]. The initial differential cryptanalysis by Biham and Shamir was based on the bitwise XOR difference and they apply this new technique to a number of ciphers and even a few cipher-based hash functions.

The technique was quickly generalized from the bitwise XOR difference to the modular difference (modulo 2^{32}) and applied to the MD4 style hash function family [Ber92]. Differential cryptanalysis using the modular difference was not successful,

since it was not possible to effectively deal with the bitwise operations (the boolean function and the bitwise rotation). Later, in 1995, Dobbertin [Dob96] was partially successful as he was able to find collisions for the compression function of MD5, but this did not extend to a collision attack against MD5 itself. In the case of MD4 he was more successful. Dobbertin [Dob98] was able to find collisions for MD4 using differential cryptanalysis modulo 2^{32} in 1996.[6]

Major cryptanalytic breakthroughs, based on a combination of the XOR difference and the modular difference, were found in the year 2004 when collisions were presented for SHA-0 by Biham et al. [BCJ+05] and MD4, MD5, HAVAL-128 and RIPEMD [WFLY04] by Xiaoyun Wang et al. These attacks have set off a new innovation impulse in hash function theory, cryptanalysis and practical attacks. The recent advances have resulted in a loss of confidence in the design principles underlying the MD4 style hash functions. In light of this, NIST has started a public competition in 2007 to develop a new cryptographic hash function SHA-3 [NIS07] to be the future de facto standard hash function. At the end of 2008, 51 out of 64 submitted candidate hash functions were selected for the first round of public review. Mid 2009, only 14 were selected to advance to the second round. After another year based on public feedback and internal reviews five SHA-3 finalists were selected at the end of 2010: BLAKE [AHMP10], Grøstl [GKM+11], JH [Wu11], Keccak [BDPA11] and Skein [FLS+10]. NIST will decide the winner in 2012 and name it SHA-3.

6. Hans Dobbertin fell ill suddenly in the late spring of 2005 and passed away on the 2nd of February 2006.

2 MD5 collision attack by Wang et al.

Contents

2.1 Preliminaries

In the upcoming sections we describe the original attack on MD5 by Wang et al. [WY05] in detail. Their original paper is well complemented by [HPR04] which provides many insightful details. First we introduce the necessary preliminaries and the definition of MD5, followed by an overview of the attack and a detailed treatment of the various aspects.

2.1.1 32-bit words

MD5 is designed with a 32-bit computing architecture in mind and operates on *words* $(v_{31} \dots v_0)$ consisting of 32 bits $v_i \in \{0, 1\}$. These 32-bit words are identified with elements $v = \sum_{i=0}^{31} v_i 2^i$ of $\mathbb{Z}_{2^{32}}$ (a shorthand for $\mathbb{Z}/2^{32}\mathbb{Z}$). In this thesis we switch freely between the bitwise and $\mathbb{Z}_{2^{32}}$ representation of 32-bit words. We shall denote a 32-bit word in binary as $\texttt{00000000111111110000111100110101}_2$. For a more compact notation, we also use the well-known hexadecimal form $\texttt{00ff0f35}_{16}$.

For 32-bit words $X = (x_i)_{i=0}^{31}$ and $Y = (y_i)_{i=0}^{31}$ we use the following notation:

- $X \wedge Y = (x_i \wedge y_i)_{i=0}^{31}$ is the bitwise AND of X and Y;
- $X \vee Y = (x_i \vee y_i)_{i=0}^{31}$ is the bitwise OR of X and Y;
- $X \oplus Y = (x_i \oplus y_i)_{i=0}^{31}$ is the bitwise XOR of X and Y;
- $\overline{X} = (\overline{x_i})_{i=0}^{31}$ is the bitwise complement of X;

- $X[i]$ is the i-th bit x_i;
- $X + Y$ and $X - Y$ denote addition and subtraction, respectively, of X and Y in $\mathbb{Z}_{2^{32}}$;
- $RL(X, n)$ and $RR(X, n)$ are the cyclic left and right rotation, respectively, of X by n bit positions:

$$\begin{aligned} & RL(\mathtt{10100100111111111111111100000001}_2,\ 5) \\ = \quad & \mathtt{00111111111111111110000000110100}_2; \end{aligned}$$

- $w(X)$ denotes the Hamming weight $\sum_{i=0}^{31} x_i$ of $X = (x_i)_{i=0}^{31}$.

2.1.2 Endianness

When interpreting a 32-bit word from a string $b_{31} \dots b_0$ of 32 bits and vice versa, there are two main standards that can be used:

Big Endian: This is the most straightforward standard which interprets a bit string in order with the most significant bit first resulting in the 32-bit word $(b_{31} \dots b_0)$.

Little Endian: This standard partitions a bit string into strings of eight consecutive bits and reverses the order of these partitions while maintaining the bit order within each partition. The resulting bit string is then interpreted using Big Endian resulting in the 32-bit word:

$$(b_7 b_6 \dots b_1 b_0\ b_{15} b_{14} \dots b_9 b_8\ b_{23} b_{22} \dots b_{17} b_{16}\ b_{31} b_{30} \dots b_{25} b_{24}).$$

This standard is for example used on the x86 CPU architecture.

2.1.3 Binary signed digit representation

A *binary signed digit representation* (BSDR) for an $X \in \mathbb{Z}_{2^{32}}$ is a sequence $(k_i)_{i=0}^{31}$ such that

$$X = \sum_{i=0}^{31} k_i 2^i, \quad k_i \in \{-1, 0, 1\}.$$

For each non-zero X there exist many different BSDRs. The *weight* $w((k_i)_{i=0}^{31}) = \sum_{i=0}^{31} |k_i|$ of a BSDR $(k_i)_{i=0}^{31}$ is defined as the number of non-zero k_is.

A particularly useful type of BSDR is the Non-Adjacent Form (NAF), where no two non-zero k_i-values are adjacent. For any $X \in \mathbb{Z}_{2^{32}}$ there is no unique NAF, since we work modulo 2^{32} (making $k_{31} = +1$ equivalent to $k_{31} = -1$). However, uniqueness of the NAF can be enforced by the added restriction $k_{31} \in \{0, +1\}$. Among the BSDRs for a given $X \in \mathbb{Z}_{2^{32}}$, the NAF has minimal weight [MS06]. The NAF can be computed easily [Lin98] for a given $X \in \mathbb{Z}_{2^{32}}$ as $\mathrm{NAF}(X) = ((X + Y)[i] - Y[i])_{i=0}^{31}$ where Y is the 32-bit word $(0\ X[31]\ \dots\ X[1])$.

We use the following notation for a 32-digit BSDR Z:

- $Z[i]$ is the i-th signed bit of Z;
- $RL(Z,n)$ and $RR(Z,n)$ are the cyclic left and right rotation, respectively, of Z by n positions;
- $w(Z)$ is the weight of Z.
- $\sigma(Z) = \sum_{i=0}^{31} k_i 2^i \in \mathbb{Z}_{2^{32}}$ is the 32-bit word for which Z is a BSDR.

As a more compact representation for a BSDR, we list all i-values with non-zero k_i and using an overline $\bar{i}$ to indicate a negative k_i. E.g., consider the following BSDR of $2^{23} - 2^0$:

$$\begin{aligned}
&\{0 \ldots 4, \bar{5}, \overline{23} \ldots \overline{27}, 28\} \\
&\quad = \quad \{0, 1, 2, 3, 4, \bar{5}, \overline{23}, \overline{24}, \overline{25}, \overline{26}, \overline{27}, 28\} \\
&\quad = \quad (k_i)_{i=0}^{31} \text{ with } \begin{cases} k_0 = k_1 = k_2 = k_3 = k_4 = k_{28} = +1; \\ k_5 = k_{23} = k_{24} = k_{25} = k_{26} = k_{27} = -1; \\ k_i = 0 \text{ for } 6 \leq i \leq 22 \text{ and } 29 \leq i \leq 31. \end{cases}
\end{aligned}$$

2.1.4 Related variables and differences

In collision attacks we consider two related messages M and M'. In this thesis any variable X related to the message M or its MD5 (or other hash function) calculation may have a corresponding variable X' related to the message M' or its hash calculation. Furthermore, for such a 'matched' variable $X \in \mathbb{Z}_{2^{32}}$ we define $\delta X = X' - X$ and $\Delta X = (X'[i] - X[i])_{i=0}^{31}$, which is a BSDR of δX. For a matched variable Z that is a tuple of 32-bit words, say $Z = (z_1, z_2, \ldots)$, we define δZ and ΔZ as $(\delta z_1, \delta z_2, \ldots)$ and $(\Delta z_1, \Delta z_2, \ldots)$, respectively.

2.2 Description of MD5

2.2.1 MD5 overview

MD5 works as follows on a given bit string M of arbitrary bit length, cf. [Riv92]:

1. *Padding.* Pad the message: first append a '1'-bit, next append the least number of '0'-bits to make the resulting bit length equivalent to 448 modulo 512, and finally append the bit length of the original unpadded message M as a 64-bit little-endian integer[7]. As a result the total bit length of the padded message $\widehat{M}$ is $512N$ for a positive integer N.

7. MD5 inherited the design choice for its predecessor MD4 of using little-endian so that for little-endian machines the conversion between bit strings and words is effortless. Since at the time little-endian machines were considered generally slower than big-endian machines, this conversion advantage was given to little-endian machines instead of big-endian machines[Riv90a], despite the mathematically more aesthetic definition of big-endian.

2. *Partitioning.* Partition the padded message $\widehat{M}$ into N consecutive 512-bit blocks $M_0, M_1, \ldots, M_{N-1}$.

3. *Processing.* To hash a message consisting of N blocks, MD5 goes through $N+1$ states IHV_i, for $0 \leq i \leq N$, called the *intermediate hash values*. Each intermediate hash value IHV_i is a tuple of four 32-bit words (a_i, b_i, c_i, d_i). For $i = 0$ it has a fixed public value called the *initial value* (IV):

$$(a_0, b_0, c_0, d_0) = (\texttt{67452301}_{16}, \texttt{efcdab89}_{16}, \texttt{98badcfe}_{16}, \texttt{10325476}_{16}).$$

   For $i = 1, 2, \ldots, N$ intermediate hash value IHV_i is computed using the MD5 compression function described in detail below:

$$IHV_i = \text{MD5Compress}(IHV_{i-1}, M_{i-1}).$$

4. *Output.* The resulting hash value is the last intermediate hash value IHV_N, expressed as the concatenation of the hexadecimal byte strings of the four words a_N, b_N, c_N, d_N, converted back from their little-endian representation. As an example the IV would be expressed as

$$\texttt{0123456789abcdeffedcba9876543210}_{16}.$$

2.2.2 MD5 compression function

The input for the compression function MD5Compress(IHV, B) consists of an intermediate hash value $IHV_{\text{in}} = (a, b, c, d)$ and a 512-bit message block B. The compression function consists of 64 *steps* (numbered 0 to 63), split into four consecutive *rounds* of 16 steps each. Each step t uses modular additions, a left rotation, and a non-linear function f_t, and involves an *Addition Constant* AC_t and a *Rotation Constant* RC_t. These are defined as follows (see also Table A-1):

$$AC_t = \left\lfloor 2^{32} \left|\sin(t+1)\right|\right\rfloor \ , \qquad 0 \leq t < 64,$$

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

The non-linear function f_t depends on the round:

$$f_t(X, Y, Z) = \begin{cases} F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z) & \text{for } 0 \leq t < 16, \\ G(X, Y, Z) = (Z \wedge X) \oplus (\overline{Z} \wedge Y) & \text{for } 16 \leq t < 32, \\ H(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 32 \leq t < 48, \\ I(X, Y, Z) = Y \oplus (X \vee \overline{Z}) & \text{for } 48 \leq t < 64. \end{cases} \tag{2.1}$$

The 512-bit message block B is partitioned into sixteen consecutive 32-bit strings which are then interpreted as 32-bit words m_0, $m_1, \ldots, m_{15}$ (with little-endian byte ordering), and expanded to 64 words W_t, for $0 \leq t < 64$, (see also Table A-1):

$$W_t = \begin{cases} m_t & \text{for } 0 \leq t < 16, \\ m_{(1+5t) \bmod 16} & \text{for } 16 \leq t < 32, \\ m_{(5+3t) \bmod 16} & \text{for } 32 \leq t < 48, \\ m_{(7t) \bmod 16} & \text{for } 48 \leq t < 64. \end{cases}$$

We follow the description of the MD5 compression function from [HPR04] because its 'unrolling' of the cyclic state facilitates the analysis. For each step t the compression function algorithm uses a working state consisting of four 32-bit words Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3} and calculates a new state word Q_{t+1}. The working state is initialized as $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$. For $t = 0, 1, \ldots, 63$ in succession, Q_{t+1} is calculated as follows:

$$\begin{aligned} F_t &= f_t(Q_t,\ Q_{t-1},\ Q_{t-2}); \\ T_t &= F_t + Q_{t-3} + AC_t + W_t; \\ R_t &= RL(T_t, RC_t); \\ Q_{t+1} &= Q_t + R_t. \end{aligned} \tag{2.2}$$

After all steps are computed, the resulting state words are added to the input intermediate hash value and returned as output:

$$\text{MD5Compress}(IHV_{\text{in}}, B) = (a + Q_{61},\ b + Q_{64},\ c + Q_{63},\ d + Q_{62}). \tag{2.3}$$

2.3 Collision attack overview

The collision attack against MD5 by X. Wang and H. Yu is based on differential cryptanalysis using a combination of the XOR difference and the modular difference, resulting in a differential cryptanalysis that uses the integer difference $(b' - b) \in \{-1, 0, +1\}$ for each bit b. Using this differential cryptanalysis they constructed *differential paths* for the compression function of MD5 which describe precisely how differences between the two input pairs (IHV_{in}, B) and (IHV'_{in}, B') propagate through the compression function's working states Q_t and Q'_t, resulting in a desired difference between the outputs. Based on the differential path they constructed a system of equations, called the *sufficient conditions*, over the bits of the working states Q_t and Q'_t that ensure that the desired differential path happens.[8] The purpose of this set of equations is to facilitate the search for blocks B and B' given IHV_{in} and IHV'_{in} such that the desired

8. As shown later in this thesis, the original analysis did not lead to 'sufficient conditions' that were truly sufficient. In particular, it was indirectly assumed that the desired propagation of a modular difference through the bitwise rotation in MD5 occurs with probability 1, even though in most cases this happens with probability slightly less than 1.

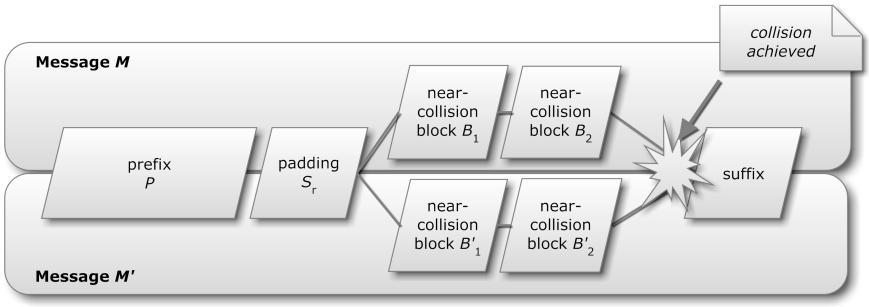

Figure 6: *Identical-prefix collision sketch*

differential path occurs, thus resulting in the desired output difference. Such an attack against the compression function, where for given IHV_{in} and IHV'_{in} a differential path is used to find message blocks B and B' such that a desired δIHV_{out} is obtained, is called a *near-collision attack*.

The attack takes as input two prefixes described as two equal-length sequences of message blocks $M_0, \ldots, M_{k-1}$ and $M'_0, \ldots, M'_{k-1}$ that resulting in identical IHV_k and IHV'_k. This condition is most easily fulfilled when both prefixes are identical, therefore collision attacks requiring $IHV_k = IHV'_k$ are called *identical-prefix collision attacks*.[9] For an arbitrary prefix P, we can obtain said message block sequences by appending a random bit string S_{r} of bit length less than 512 to P, such that the bit length of $P\|S_{\text{r}}$ is an integer multiple of 512. The padded prefix $P\|S_{\text{r}}$ is then split into message blocks $M'_0 = M_0, \ldots, M'_{k-1} = M_{k-1}$ of 512 bits.

A collision is generated using two consecutive pairs of message blocks (B_0, B'_0) and (B_1, B'_1), where $B'_0 \neq B_0$ and $B'_1 \neq B_1$, that are appended to the two prefixes. Together they result in a collision: $IHV'_{k+2} = IHV_{k+2}$. The collision attack outputs two messages $M = P||B_0||B_1$ and $M' = P'||B'_0||B'_1$ that result in the same MD5 hash. Due to the incremental nature of MD5, these messages can be further extended using an identical suffix S, resulting in the colliding messages $P||B_0||B_1||S$ and $P'||B'_0||B'_1||S$.

Thus in the most straightforward way, the attack can be used to create two messages M and M' with the same MD5 hash that only differ slightly in two subsequent blocks, as shown in Figure 6.

Note that all blocks $M_0, \ldots, M_{k-1}$ and $M_{k+2}, \ldots, M_{N-1}$ can be chosen arbitrarily and that only B_0, B'_0, B_1 and B'_1 are generated by the collision finding algorithm using the value of $IHV'_k = IHV_k$.

In the following few subsections we explain the various parts of the collision attack in more detail.

9. Since this collision attack requires $IHV_k = IHV'_k$ and results in $IHV_{k+2} = IHV'_{k+2}$, the attack can be chained endlessly.

2.4 Two message block collision

Wang et al.'s attack consists of two differential paths for two subsequent message blocks, which we refer to in order as the first and second differential path. We refer to B_0 and B_1 as the first and second near-collision block, respectively. The first differential path starts with any given $IHV_k = IHV'_k$ and introduces a difference between IHV_{k+1} and IHV'_{k+1}:

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

These differences are canceled again by the second differential path. The first differential path is based on the following differences in the message block:

$$\delta m_4 = 2^{31}, \quad \delta m_{11} = 2^{15}, \quad \delta m_{14} = 2^{31}, \quad \delta m_i = 0 \text{ for } i \notin \{4, 11, 14\}.$$

The second differential path is based on the *negated* message block differences:

$$\delta m_4 = 2^{31}, \quad \delta m_{11} = -2^{15}, \quad \delta m_{14} = 2^{31}, \quad \delta m_i = 0 \text{ for } i \notin \{4, 11, 14\}.$$

Note that $-2^{31} = 2^{31}$ in $\mathbb{Z}_{2^{32}}$, so in fact δm_4 and δm_{14} are not changed by the negation.

These are very specific message block differences that were selected to ensure a low complexity for the collision finding algorithm as is shown later. In Section 6.4, we introduce even better message block differences.

2.5 Differential paths

As said before, this attack uses a combination of XOR differential cryptanalysis and modular differential cryptanalysis. The combination of both kinds of differences gives more information than each by themselves and both are aimed at a different representation of a 32-bit word: as a string of 32 bits or as an element of $\mathbb{Z}_{2^{32}}$. So instead of only the integer modular difference between two related 32-bit words X and X', this combination uses the integer differences ($-1, 0$ or $+1$) between each pair of bits $X[i]$ and $X'[i]$ for $0 \leq i \leq 31$. This difference is represented in a natural manner using BSDRs and thus can be denoted as ΔX, which is a BSDR of $\delta X = X' - X$:

$$\Delta X = (k_i), \quad k_i = X'[i] - X[i] \text{ for } 0 \leq i \leq 31.$$

Of all variables used in MD5's compression function, this combined differential cryptanalysis is only used on the Q_t variables since these are used in both the bitwise boolean function and modular addition. The R_t and T_t variables are used in bitwise rotation and modular addition. Nevertheless, only the modular differential is used for R_t and T_t as a simplification, since $\delta R_t = RL(T_t + \delta T_t, RC_t) - RL(T_t, RC_t)$ already holds with high probability for properly chosen δR_t and δT_t. The modular differential is used for the remaining variables as these are only used in modular addition.

The differential paths for both blocks (Table 2-3 and Table 2-5) were constructed specifically to create a collision in this manner. The differential paths describe precisely for each of the 64 steps of MD5 what the differences are in the working state and

how these differences pass through the boolean function and the rotation. More precisely, for MD5 a differential path can be described through the following sequences of differences: $(\delta m_t)_{t=0}^{15}$, $(\Delta Q_t)_{t=-3}^{64}$, $(\Delta F_t)_{t=0}^{63}$. Other differences can easily be derived: $\delta T_t = \delta F_t + \delta Q_{t-3} + \delta W_t$ and $\delta R_t = \delta Q_{t+1} - \delta Q_t$.

The first differential path starts without differences as $\delta IHV_k = 0$, but differences are introduced in step $t = 4$ by $\delta m_4 = 2^{31}$. The second differential path starts with the given δIHV_{k+1}. In both paths, all differences in the working state are canceled at step $t = 25$ by δm_{14}. Furthermore, from step $t = 34$, both paths use the same differential path, although with opposite signs. This structure can easily be seen in Table 2-3 and Table 2-5.

Below we show a fraction of the first differential path:

Table 2-1: *Fraction from Table 2-3*

$\mathbf{t}$	$\Delta\mathbf{Q_t}$ **(BSDR of** $\delta\mathbf{Q_t}$**)**	$\delta\mathbf{F_t}$	$\delta\mathbf{w_t}$	$\delta\mathbf{T_t}$	$\mathbf{RC_t}$
13	$\{\overline{24}, 25, 31\}$	$-2^{13}+2^{31}$	0	-2^{12}	12
14	$\{31\}$	$2^{18}+2^{31}$	2^{31}	$2^{18}-2^{30}$	17
15	$\{3, \overline{13}, 31\}$	$2^{25}+2^{31}$	0	$-2^{7}-2^{13}+2^{25}$	22
16	$\{\overline{29}, 31\}$	2^{31}	0	2^{24}	5
17	$\{31\}$	2^{31}	0	0	9
18	$\{31\}$	2^{31}	2^{15}	2^{3}	14
19	$\{17, 31\}$	2^{31}	0	-2^{29}	20

These two differential paths were made by hand with great skill and intuition by Wang et al. In order to construct faster collisions attacks or for collision attacks that do not require $IHV'_k = IHV_k$ we need to be able to construct differential paths preferably algorithmically instead of by hand.

2.6 Sufficient conditions

Wang et al. use *sufficient conditions* to efficiently search for message blocks for which these differential paths hold. These sufficient conditions guarantee that the necessary carries and correct boolean function differences happen. Each condition gives the value of a bit $Q_t[i]$ of the working state either directly or indirectly as shown in Table 2-2. Later on, we generalize and extend these conditions to also include the value of the related bit $Q'_t[i]$.

These conditions are only used to find a block B on which the message differences can be applied to find B' and should guarantee that the differential path happens. They can be derived for any differential path and there can be many different possible sets of sufficient conditions.

However, it should be noted that the sufficient conditions given by Wang et al. are not sufficient at all, as those conditions do not guarantee that in each step the differences are rotated correctly. In fact, as we show later on, one does not want

Table 2-2: *Sufficient bitconditions.*

Symbol	condition on $\mathbf{Q_t[i]}$	direct/indirect
.	no condition	direct
0	$Q_t[i] = 0$	direct
1	$Q_t[i] = 1$	direct
^	$Q_t[i] = Q_{t-1}[i]$	indirect

See Table B-1 for a complete listing of all possible bitconditions used in this thesis.

sufficient conditions for the full differential path as this increases the collision finding complexity significantly.

2.7 Collision finding

Using these sufficient conditions one can efficiently search for a block B. The most basic *collision finding algorithm* chooses random working state words $Q_1, \ldots, Q_{16}$ that fulfill their sufficient conditions for the first round. The corresponding message words can directly be computed:

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

As all the bits in the message block are now determined, the other working state words $Q_{17}, \ldots, Q_{64}$ are now also completely determined. Thus the remaining sufficient conditions have to be fulfilled probabilistically and directly result in the complexity of this basic collision finding algorithm. Wang et al. used the following improvements over this basic algorithm:

Message modification: When a certain condition in the second round fails, one can use message modification. This is a substitution formula specially made for this condition on the message block B. In the case that this condition does not hold applying this substitution has the effect that this condition now does hold without interfering with other previous conditions. To use message modification, auxiliary conditions have to be imposed on the first round. Specifically, at least one auxiliary condition should be imposed to avoid testing the same message block B twice (once directly, once through another message block $\widehat{B}$ to which this substitution is applied).

Early stop: Stop at the step where the first sufficient condition fails and there is no message modification. This improvement reduces the average number of MD5 compression function steps that have to be evaluated.

An example of message modification is the following. When testing a block for the sufficient conditions in Table 2-4, suppose $Q_{17}[31] = 1$ instead of 0. This can be corrected by replacing m_1, m_2, m_3, m_4, m_5 with:

1. $\widehat{m}_1 \leftarrow (m_1 + 2^{26})$ which results in a different $\widehat{Q}_2$;

2. $\widehat{m}_2 \leftarrow RR(Q_3 - \widehat{Q}_2, 17) - Q_{-1} - F(\widehat{Q}_2, Q_1, Q_0) - AC_2$;
3. $\widehat{m}_3 \leftarrow RR(Q_4 - Q_3, 22) - Q_0 - F(Q_3, \widehat{Q}_2, Q_1) - AC_3$;
4. $\widehat{m}_4 \leftarrow RR(Q_5 - Q_4, 7) - Q_1 - F(Q_4, Q_3, \widehat{Q}_2) - AC_4$;
5. $\widehat{m}_5 \leftarrow RR(Q_6 - Q_5, 12) - \widehat{Q}_2 - F(Q_5, Q_4, Q_3) - AC_5$.

The first line is the most important, here m_1 is changed such that $\widehat{Q}_{17}[31] = 0$, assuming Q_{13} up to Q_{16} remain unaltered. The added $+2^{26}$ in m_1 results in an added $+2^{31}$ in $Q_{17}[31]$, hence $\widehat{Q}_{17}[31] = 0$. The four other lines simply change m_2, m_3, m_4 and m_5 such that Q_3 up to Q_{16} remain unaltered by the change from Q_2 to $\widehat{Q}_2$. Since there are no conditions in the first block that involve Q_2, all sufficient conditions up to Q_{17} are left unaltered. However, there is a side-effect: with probability 1/2 the rotation in step 16 actually results in an additional added $+2^0$ in $Q_{17}[31]$, which then with probability 1/8 interferes with the condition imposed on $Q_{17}[3]$. Fortunately, this side-effect can be avoided entirely by negating the substitution whenever $T_{16}[26] = 1$.

Comparing Table 2-4 and Table 2-6, there are fewer sufficient conditions for the first block. In general, this means that more message modification techniques can be applied on the first block, resulting in a larger speedup compared to the second block. However, there are eight sufficient bitconditions for the second block on the input IHV_{k+1}. Since the value of IHV_{k+1} depends only on the values of IHV_k and the first block B_0, these eight sufficient bitconditions have to be fulfilled probabilistically by the first block rather than the second block.

The original attack can find collisions for MD5 in about 15 minutes up to an hour on an IBM P690 with a computational cost equivalent to about 2^{39} compression function calls. Since then many improvements have been made [YS05, SNKO05, LL05, Kli05, Ste06, Kli06]. Currently, collisions for MD5 based on these differential paths can be found in several seconds on a single powerful PC with a computational cost equivalent to about $2^{24.1}$ compression function calls. These faster attacks use techniques based on *Tunnels* [Kli06], controlling rotations in the first round [Ste06] and additional differential paths. Our fastest collision attack [SSA$^+$09b] is based on slightly different message block differences and has a theoretical computational cost of about 2^{16} compression function calls (see Section 6.4). Xie and Feng [XF09] claim to have found message differences that may lead to an even faster attack. Based on a rather optimistic cost function they claim an estimated complexity of 2^{10} compressions. Besides this claim, they have constructed a MD5 collision attack with a complexity of about $2^{20.96}$ MD5 compressions using less promising message differences. Recently even single-block identical-prefix collisions have been found by Xie and Feng [XF10], although they do not present their new techniques 'for security reasons'.

2.8 Tables: differential paths and sufficient conditions

Table 2-3: *Wang et al.'s first differential path*

$\mathbf{t}$	$\Delta\mathbf{Q_t}$ (BSDR of $\delta\mathbf{Q_t}$)	$\delta\mathbf{F_t}$	$\delta\mathbf{w_t}$	$\delta\mathbf{T_t}$	$\mathbf{RC_t}$
$0-3$	0	0	0	0	RC_t
4	0	0	2^{31}	2^{31}	7
5	$\{6\dots21,\overline{22}\}$	$2^{11}+2^{19}$	0	$2^{11}+2^{19}$	12
6	$\{\overline{6},23,31\}$	$-2^{10}-2^{14}$	0	$-2^{10}-2^{14}$	17
7	$\{0\dots4,\overline{5},6\dots10,\overline{11},$ $\overline{23}\dots\overline{25},26\dots31\}$	$-2^{2}+2^{5}+2^{10}$ $+2^{16}-2^{25}-2^{27}$	0	$-2^{2}+2^{5}+2^{10}$ $+2^{16}-2^{25}-2^{27}$	22
8	$\{0,15,\overline{16},17,18,$ $19,\overline{20},\overline{23}\}$	$2^{6}+2^{8}+2^{10}$ $+2^{16}-2^{24}+2^{31}$	0	$2^{8}+2^{10}+2^{16}$ $-2^{24}+2^{31}$	7
9	$\{\overline{0},1,6,7,\overline{8},\overline{31}\}$	$2^{0}+2^{6}-2^{20}$ $-2^{23}+2^{26}+2^{31}$	0	$2^{0}-2^{20}+2^{26}$	12
10	$\{\overline{12},13,31\}$	$2^{0}+2^{6}+2^{13}-2^{23}$	0	$2^{13}-2^{27}$	17
11	$\{30,31\}$	$-2^{0}-2^{8}$	2^{15}	$-2^{8}-2^{17}-2^{23}$	22
12	$\{7,\overline{8},13\dots18,\overline{19},31\}$	$2^{7}+2^{17}+2^{31}$	0	$2^{0}+2^{6}+2^{17}$	7
13	$\{\overline{24},25,31\}$	$-2^{13}+2^{31}$	0	-2^{12}	12
14	$\{31\}$	$2^{18}+2^{31}$	2^{31}	$2^{18}-2^{30}$	17
15	$\{3,\overline{15},31\}$	$2^{25}+2^{31}$	0	$-2^{7}-2^{13}+2^{25}$	22
16	$\{\overline{29},31\}$	2^{31}	0	2^{24}	5
17	$\{31\}$	2^{31}	0	0	9
18	$\{31\}$	2^{31}	2^{15}	2^{3}	14
19	$\{17,31\}$	2^{31}	0	-2^{29}	20
20	$\{31\}$	2^{31}	0	0	5
21	$\{31\}$	2^{31}	0	0	9
22	$\{31\}$	2^{31}	0	2^{17}	14
23	0	0	2^{31}	0	20
24	0	2^{31}	0	0	5
25	0	0	2^{31}	0	9
$26-33$	0	0	0	0	RC_t
34	0	0	2^{15}	2^{15}	16
35	$\delta Q_{35}=2^{31}$	2^{31}	2^{31}	0	23
36	$\delta Q_{36}=2^{31}$	0	0	0	4
37	$\delta Q_{37}=2^{31}$	2^{31}	2^{31}	0	11
$38-49$	$\delta Q_t=2^{31}$	2^{31}	0	0	RC_t
50	$\delta Q_{50}=2^{31}$	0	2^{31}	0	15
$51-59$	$\delta Q_t=2^{31}$	2^{31}	0	0	RC_t
60	$\delta Q_{60}=2^{31}$	0	2^{31}	0	6
61	$\delta Q_{61}=2^{31}$	2^{31}	2^{15}	2^{15}	10
62	$\delta Q_{62}=2^{31}+2^{25}$	2^{31}	0	0	15
63	$\delta Q_{63}=2^{31}+2^{25}$	2^{31}	0	0	21
64	$\delta Q_{64}=2^{31}+2^{25}$	$\times$	$\times$	$\times$	$\times$

Table 2-4: *Wang et al.'s first block sufficient bitconditions*

t	Conditions on Q_t: $b_{31} \dots b_0$	#
3	....0... ....0... .0......	3
4	1....... 0^^^1^^^ ^^^^1^^^ ^0......	19
5	1...1.0. 01..0000 00000000 001..1.1	22
6	0000001^ 01111111 10111100 0100^0^1	32
7	00000011 11111110 11111000 00100000	32
8	00000001 1..10001 0.0.0101 01000000	28
9	11111011 ...10000 0.1^1111 00111101	28
10	01...... 0..11111 1101...0 01....00	17
11	00...... ....0001 1100...0 11....10	15
12	00....^^ ....1000 0001...1 0.......	14
13	01....01 ....1111 111....0 0...1...	14
14	0.0...00 ....1011 111....1 1...1...	14
15	0.1...01 .... 1....... ....0...	6
16	0.1.....	2
17	0....... ......0. ^....... ....^...	4
18	0.^..... ......1.	3
19	0....... ......0.	2
20	0.......	1
21	0....... ......^.	2
22	0.......	1
23	0.......	1
24	1.......	1
25 − 45	....	0
46	....	0
47	....	0
48	m.......	1
49	m.......	1
50	#.......	1
51	m.......	1
52	m.......	1
53	m.......	1
54	m.......	1
55	m.......	1
56	m.......	1
57	m.......	1
58	m.......	1
59	m.......	1
60	#.....0.	2
61	m.....1.	2
62	m.....0.	2
63	m.....0.	2
64	....	0
	Sub-total # IV conditions in Table 2-6	**8**
	Total # conditions	**289**

Uses bitconditions as defined in Table B-1 limited to only the variables $Q_i[j]$, not $Q'_i[j]$.

Table 2-5: *Wang et al.'s second differential path*

$\mathbf{t}$	$\Delta\mathbf{Q_t}$ (BSDR of $\delta\mathbf{Q_t}$)	$\delta\mathbf{F_t}$	$\delta\mathbf{w_t}$	$\delta\mathbf{T_t}$	$\mathbf{RC_t}$
-3	$\{31\}$	$\times$	$\times$	$\times$	$\times$
-2	$\{25, 31\}$	$\times$	$\times$	$\times$	$\times$
-1	$\{\overline{25}, 26, 31\}$	$\times$	$\times$	$\times$	$\times$
0	$\{25, 31\}$	2^{31}	0	0	7
1	$\{25, 31\}$	2^{31}	0	2^{25}	12
2	$\{\overline{5}, 25, 31\}$	2^{25}	0	$2^{31}+2^{26}$	17
3	$\{\overline{5}, \overline{6}, 7, \overline{11}, 12, \overline{16} \ldots \overline{20}, 21, \overline{25} \ldots \overline{29}, 30, 31\}$	$-2^{11}-2^{21}+2^{25}$ $-2^{27}+2^{31}$	0	$-2^{11}-2^{21}-2^{26}$	22
4	$\{1, 2, 3, \overline{4}, 5, \overline{25}, 26, 31\}$	$2^{1}-2^{3}-2^{18}$ $+2^{26}+2^{30}$	2^{31}	$2^{1}+2^{2}-2^{18}$ $+2^{25}+2^{26}+2^{30}$	7
5	$\{0, \overline{6}, 7, 8, \overline{9}, \overline{10}, \overline{11}, 12, 31\}$	$-2^{4}-2^{5}-2^{8}-2^{20}$ $-2^{25}-2^{26}+2^{28}+2^{30}$	0	$-2^{4}-2^{8}-2^{20}$ $-2^{26}+2^{28}-2^{30}$	12
6	$\{16, \overline{17}, 20, \overline{21}, 31\}$	$2^{3}-2^{5}-2^{10}-2^{11}$ $-2^{16}-2^{21}-2^{25}$	0	$2^{3}-2^{10}-2^{21}-2^{31}$	17
7	$\{6, 7, 8, \overline{9}, 27, \overline{28}, 31\}$	$2^{16}-2^{27}+2^{31}$	0	$-2^{1}+2^{5}+2^{16}$ $+2^{25}-2^{27}$	22
8	$\{\overline{15}, 16, \overline{17}, 23, 24, 25, \overline{26}, 31\}$	$-2^{6}+2^{16}+2^{25}$	0	$2^{0}+2^{8}+2^{9}$ $+2^{16}+2^{25}-2^{31}$	7
9	$\{\overline{0}, 1, \overline{6}, \overline{7}, \overline{8}, 9, 31\}$	$2^{0}+2^{16}-2^{26}+2^{31}$	0	$2^{0}-2^{20}-2^{26}$	12
10	$\{12, 31\}$	$2^{6}+2^{31}$	0	-2^{27}	17
11	$\{31\}$	2^{31}	-2^{15}	$-2^{17}-2^{23}$	22
12	$\{\overline{7}, 13 \ldots 18, \overline{19}, 31\}$	$2^{17}+2^{31}$	0	$2^{0}+2^{6}+2^{17}$	7
13	$\{\overline{24} \ldots \overline{29}, 30, 31\}$	$-2^{13}+2^{31}$	0	-2^{12}	12
14	$\{31\}$	$2^{18}+2^{30}$	2^{31}	$2^{18}+2^{30}$	17
15	$\{3, 15, 31\}$	$-2^{25}+2^{31}$	0	$-2^{7}-2^{13}-2^{25}$	22
16	$\{\overline{29}, 31\}$	2^{31}	0	2^{24}	5
17	$\{31\}$	2^{31}	0	0	9
18	$\{31\}$	2^{31}	-2^{15}	2^{3}	14
19	$\{17, 31\}$	2^{31}	0	-2^{29}	20
20	$\{31\}$	2^{31}	0	0	5
21	$\{31\}$	2^{31}	0	0	9
22	$\{31\}$	2^{31}	0	2^{17}	14
23	0	0	2^{31}	0	20
24	0	2^{31}	0	0	5
25	0	0	2^{31}	0	9
$26-33$	0	0	0	0	RC_t
34	0	0	-2^{15}	-2^{15}	16
35	$\delta Q_{35} = 2^{31}$	2^{31}	2^{31}	0	23
36	$\delta Q_{36} = 2^{31}$	0	0	0	4
37	$\delta Q_{37} = 2^{31}$	2^{31}	2^{31}	0	11
$38-49$	$\delta Q_t = 2^{31}$	2^{31}	0	0	RC_t
50	$\delta Q_{50} = 2^{31}$	0	2^{31}	0	15
$51-59$	$\delta Q_t = 2^{31}$	2^{31}	0	0	RC_t
60	$\delta Q_{60} = 2^{31}$	0	2^{31}	0	6
61	$\delta Q_{61} = 2^{31}$	2^{31}	-2^{15}	-2^{15}	10
62	$\delta Q_{62} = 2^{31} - 2^{25}$	2^{31}	0	0	15
63	$\delta Q_{63} = 2^{31} - 2^{25}$	2^{31}	0	0	21
64	$\delta Q_{64} = 2^{31} - 2^{25}$	$\times$	$\times$	$\times$	$\times$

Table 2-6: *Wang et al.'s second block sufficient bitconditions*

t	Conditions on Q_t: $b_{31} \dots b_0$	#
-2	`......0. ........ ........ ........`	(1)
-1	`^....01. ........ ........ ........`	(3)
0	`^....00. ........ ........ ..0.....`	(4)
	Total # IV conditions for first block	(8)
1	`!...010. ..1..... ....0... .10.....`	8
2	`^^^^110. ..0^^^^0 ...^1... ^10..00.`	20
3	`^011111. ..011111 ...01..1 011^^11.`	23
4	`^011101. ..000100 ...00^^0 0001000^`	26
5	`!10010.. ..101111 ...01110 01010000`	25
6	`^..0010. 1.10..10 1..01100 01010110`	24
7	`!..1011^ 1.00..01 1..11110 00.....1`	20
8	`^..00100 0.11..10 1.....11 11....^0`	18
9	`^..11100 0.....01 0..^..01 11....01`	17
10	`^....111 1....011 1100..11 11....00`	18
11	`^....... ....^101 1100..11 11....11`	15
12	`^^^^^^^^ ....1000 0001.... 1.......`	17
13	`!0111111 ....1111 111..... 0...1...`	17
14	`^1000000 ....1011 111..... 1...1...`	17
15	`01111101 ........ 0....... ....0...`	10
16	`0.1..... ........ ........ ........`	2
17	`0....... ......0. ^....... ....^...`	4
18	`0.^..... ......1. ........ ........`	3
19	`0....... ......0. ........ ........`	2
20	`0....... ........ ........ ........`	1
21	`0....... ......^. ........ ........`	2
22	`0....... ........ ........ ........`	1
23	`0....... ........ ........ ........`	1
24	`1....... ........ ........ ........`	1
$25-45$	`........ ........ ........ ........`	0
46	`........ ........ ........ ........`	0
47	`........ ........ ........ ........`	0
48	`m....... ........ ........ ........`	1
49	`m....... ........ ........ ........`	1
50	`#....... ........ ........ ........`	1
51	`m....... ........ ........ ........`	1
52	`m....... ........ ........ ........`	1
53	`m....... ........ ........ ........`	1
54	`m....... ........ ........ ........`	1
55	`m....... ........ ........ ........`	1
56	`m....... ........ ........ ........`	1
57	`m....... ........ ........ ........`	1
58	`m....... ........ ........ ........`	1
59	`m....... ........ ........ ........`	1
60	`#.....0. ........ ........ ........`	2
61	`m.....1. ........ ........ ........`	2
62	`m.....1. ........ ........ ........`	2
63	`m.....1. ........ ........ ........`	2
64	`........ ........ ........ ........`	0
	Total # conditions	**312**

Uses bitconditions as defined in Table B-1 limited to only the variables $Q_i[j]$, not $Q'_i[j]$.

Part II
Contributions

3 Our contributions

Contents

3.1 Overview

In this section we list our main contributions. First we present applications for collision attacks that we have successfully implemented. This is followed by our main results for MD5 and SHA-1. Next, we present more general results that apply to certain classes of hash functions. Finally, we outline the remainder of this thesis.

3.2 Chosen-prefix collision applications

Several abuse scenarios were presented based on Wang et al.'s collision attack, such as to mislead integrity checking software [Kam04, Mik04], constructing different PostScript documents [DL05] that collide under MD5, constructing two different *X.509 certificates* [CSF+08] with identical Distinguished Names and identical MD5-based digital signatures but different public keys [LdW05], etc. Although none of these abuse scenarios spoke in favor of continued usage of MD5, the abuse potential of Wang et al.'s identical-prefix collision attack remains limited.

The most important contribution of this thesis is the construction and application of *chosen-prefix collision* attacks. This removes the $IHV_{\text{in}} = IHV'_{\text{in}}$ restriction of identical-prefix collisions: given any two chosen message prefixes P and P', a chosen-prefix collision attack constructs suffixes S and S' such that the concatenated values $P||S$ and $P'||S'$ form a collision.

Using our chosen-prefix collision attack against MD5 we discuss and implement several new abuse scenarios in Chapter 4 that are impossible using identical-prefix collisions. The main abuse scenarios that we present and successfully have implemented are:

- Construction of colliding X.509 certificates with different distinguished names;
- Construction of colliding documents or executables where the malicious payload is not present in the harmless counterpart;

- Finally, our most convincing abuse scenario is the construction of a rogue CA (*Certification Authority*) X.509 certificate that undermines the core of the X.509 public key infrastructure most commonly known for its use to secure websites (`https://`);

3.3 Results on MD5

For MD5 we present a practical chosen-prefix collision attack and an identical-prefix collision attack which is faster than Wang et al.'s collision attack:

Theorem 3.1 (MD5 collision attacks)**.** *There exist an identical-prefix collision attack and a chosen-prefix collision attack against MD5 with average complexities equivalent to* 2^{16} *and* 2^{39} *calls to the compression function of MD5, respectively.*

Furthermore, there exists a near-collision attack against the compression function of MD5, with an average complexity equivalent to $2^{14.8}$ *calls to the compression function of MD5* [10]*, that for given* $IHV_{in} = IHV'_{in}$ *searches for a pair of message blocks that results in*

$$\delta IHV_{out} = (-2^5, -2^5 + 2^{25} + 2^{23} + 2^{26} - 2^{14}, -2^5 + 2^{25} + 2^{23}, -2^5 + 2^{25}).$$

This near-collision attack can be altered to work for any given IHV_{in} *and* IHV'_{in}*. The resulting near-collision attack has the same runtime complexity and searches for a pair of message blocks that result in*

$$\delta IHV_{out} = \delta IHV_{in} + (-2^5, -2^5 + 2^{25} + 2^{23} + 2^{26} - 2^{14}, -2^5 + 2^{25} + 2^{23}, -2^5 + 2^{25}).$$

The proof by construction of Theorem 3.1 is given in Chapter 6.

Lastly, we present a technique that can detect all known feasible identical-prefix and chosen-prefix collision attacks against MD5 given only one message from a colliding message pair:

Theorem 3.2 (Detecting MD5 collision attacks)**.** *Given a message* M*, it is possible to detect, with an average complexity of about* 224 *times the complexity of computing the MD5 hash of* M *and error probability approximately* 2^{-128}*, whether* M *has been constructed by a collision attack based on message differences given in one of the following papers: [dBB93], [WY05], [SLdW07c], [XFL08], [XLF08], [VJBT08], [XF09], [SSA+09b] and [XF10]. Furthermore, future collision attacks can easily be added in the detection if they are not detected already.*

The proof by construction of Theorem 3.2 is given in Section 8.3.

3.4 Results on SHA-1

For SHA-1 we present a near-collision attack which is used to construct an identical-prefix and a chosen-prefix collision attack:

10. MD5's compression function requires 500 arithmetic and bitwise word operations.

Theorem 3.3 (SHA-1 collision attacks)**.** *There exists an identical-prefix collision attack against SHA-1 with an average complexity between* $2^{60.3}$ *and* $2^{65.3}$ *calls to the compression function of SHA-1. Also, there exists a chosen-prefix collision attack against SHA-1 with an average complexity of* $2^{77.1}$ *calls to the compression function of SHA-1.*

There exists a near-collision attack against the compression function of SHA-1 that, for given $IHV_{in} = IHV'_{in}$ *satisfying the bitconditions given in Table 7-6 (p. 169), searches for a pair of message blocks so that the resulting* δIHV_{out} *equals one of the 192 target* δIHV_{diff} *values given in Table 7-5 (p. 167). It has an average complexity equivalent to* $2^{57.5}$ *calls to the compression function* [11]*.*

The proof by construction of Theorem 3.3 is given in Chapter 7.

Our attacks against SHA-1 are based on *local collisions* that consist of a single disturbance in some step followed by corrections in the next five steps which cancel out this disturbance. *Disturbance vectors* mark these disturbances and thus the start of local collisions in patterns that are compatible with SHA-1's message expansion. In Section 7.5 we introduce a new precise SHA-1 disturbance vector analysis for the use of near-collision attacks with the following features that improve upon the literature:

- takes into account the dependence of local collision (so far independence has been assumed);
- allows to determine $\delta IHV_{\text{diff}} = \delta IHV_{\text{out}} - \delta IHV_{\text{in}}$ with higher success probabilities than those as prescribed by the disturbance vector;
- allows to determine the optimal message bitrelations that lead to the highest success probability (so far only treated for individual local collisions).

Using our new analysis we show in Section 7.5 that higher success probabilities can be obtained compared to the product of the success probabilities of the individual local collisions.

Furthermore, we present a technique that can detect possibly feasible identical-prefix and chosen-prefix collision attacks against SHA-1 given only one message from a colliding message pair:

Theorem 3.4 (Detecting SHA-1 collision attacks)**.** *Given a message* M*, it is possible to detect, with an average complexity of about* 15 *times the complexity of computing the SHA-1 hash of* M *and error probability approximately* 2^{-160}*, whether* M *is constructed by a possibly feasible collision attack against SHA-1. Furthermore, future collision attacks can easily be added in the detection if they are not detected already.*

The proof by construction of Theorem 3.4 is given in Section 8.4.

11. SHA-1's compression function requires 941 arithmetic and bitwise word operations.

3.5 General results

Our following theorem enables one to construct complete differential paths for a certain class of compression functions.

Theorem 3.5 (Differential path construction)**.** *For the family $\mathcal{F}_{md4cf}$ of compression functions defined in Section 5.3 that includes those of MD4, MD5 and SHA-1 there exists an algorithm which, from given IHV_{in}, IHV'_{in} and a partial differential path beginning at some early step and ending at the last step, constructs complete differential paths.*

Proof is given in Chapter 5 with specific more efficient algorithms for MD5 and SHA-1 presented in Section 6.2 and Section 7.4, respectively. These algorithms were key to the construction of the identical-prefix and chosen-prefix collision attacks against MD5 and SHA-1 mentioned in Theorems 3.1 and 3.3.

As a more general result we can construct chosen-prefix collisions for other non-collision-resistant hash functions:

Theorem 3.6 (Generic chosen-prefix collision attack)**.** *Let H be a hash function with L-bit hash values based on the Merkle-Damgård construction with compression function*

$$Compress : \{0,1\}^L \times \{0,1\}^N \to \{0,1\}^L, \quad 1 \leq L \leq N$$

that uses an IHV and message block of bit size L and N, respectively. Addition and subtraction of IHV-values are defined through some additive abelian group $(\{0,1\}^L, +)$. There exists a chosen-prefix collision attack against H that is faster than a brute force collision attack against H if H satisfies the following criteria:

1. *there is a set $\mathcal{I}$ of at least four IHV differences D for which there exists near-collision attacks against Compress that for fixed IHV_{in} and IHV'_{in} searches for message blocks B and B' such that*

   $$IHV'_{out} - IHV_{out} = IHV'_{in} - IHV_{in} - D$$

   where

   $$IHV_{out} = Compress(IHV_{in}, B), \quad IHV'_{out} = Compress(IHV'_{in}, B').$$

2. *each of the above near-collision attacks can be modified to start with any given $\widehat{IHV}_{in}$ and $\widehat{IHV}'_{in}$ such that the total average complexity in calls to Compress, consisting of the average cost of modifying the near-collision attack together with the average runtime complexity of the modified near-collision attack, is at most $\sqrt{\pi \cdot 2^{L-1}}/4$.*

3. *Given any $IHV_{in}, IHV'_{in} \in \{0,1\}^L$, we can assume that the probability that $Compress(IHV_{in}, B) = Compress(IHV'_{in}, B')$ for uniformly-randomly chosen $B \neq B' \in \{0,1\}^N$ is 2^{-L}.*

4. *there exists a function* $\phi : \{0,1\}^L \times \{0,1\} \to \{0,1\}^K$ *with* $K < L$ *such that the probability* $p_{\texttt{useful}}$ *that* $y - x \in \mathcal{I}$ *for IHV-values* x *and* y *with* $\phi(x,0) = \phi(y,1)$ *is at least* $\max(8 \cdot 2^{K-L}, \pi \cdot 2^{-K})$.

The second criterion can be partly fulfilled by Theorem 3.5 as it provides a way to modify a differential path underlying a near-collision attack to start with any given $\widehat{IHV}_{\text{in}}$ and $\widehat{IHV}'_{\text{in}}$. The proof of Theorem 3.6 depends on the following lemma:

Lemma 3.7 (Birthday search [vOW99])**.** *Let* V *be a finite set of size at least 3. Let* $f : V \to V$ *be a function such that* $\Pr[f(a) = f(b) \mid a, b \in V, a \neq b] = 1/|V|$. *Let* $\texttt{Pred} : V \times V \to \{0,1\}$ *be a boolean predicate defined on* $V \times V$ *such that the probability* $q = \Pr[\texttt{Pred}(a,b) = 1 \mid a, b \in V, a \neq b, f(a) = f(b)]$ *is non-zero.*[12]

Consider the following experiment. For $i = 1, 2, \ldots$, *sample* q_i *uniformly random from* $V \setminus \{q_1, \ldots, q_{i-1}\}$, *evaluate* f *on* q_i *and store* $(q_i, f(q_i))$. *The experiment continues until there exists an* $j \in \{1, \ldots, i-1\}$ *such that* $f(q_i) = f(q_j)$ *and* $\texttt{Pred}(q_i, q_j) = 1$.

Then the expected number of evaluations of f *required to find a pair* $(a,b) \in V \times V$ *such that* $a \neq b$, $f(a) = f(b)$ *and* $\texttt{Pred}(a,b) = 1$ *is* $\sqrt{\pi \cdot |V|/(2 \cdot q)}$.

We refer to [vOW99] for the proof of Lemma 3.7 and a practical probabilistic algorithm to perform a birthday search.

One can directly construct a brute force collision search against H (if it satisfies the third criteria of Theorem 3.6) with an average complexity of $\sqrt{\pi \cdot 2^{L-1}}$ calls to Compress, by using the above lemma with $V = \{0,1\}^L$ and $q = 1$ (a trivial predicate).

Proof of Theorem 3.6. Let $K < L$, $\phi : \{0,1\}^L \times \{0,1\} \to \{0,1\}^K$ and p_{useful} be as in Theorem 3.6. The chosen-prefix collision attack consists of a birthday search followed by a single near-collision attack. For given prefixes P and P', we first append bit strings S_{r} and S'_{r} such that bit lengths of $P||S_{\text{r}}$ and $P'||S'_{\text{r}}$ are both equal to $c \cdot N - K$ for some $c \in \mathbb{N}^+$ (note that $K < L \leq N$). Let IHV_{in} and IHV'_{in} be the intermediate hash values after processing the first $c-1$ blocks, i.e., the first $(c-1) \cdot N$ bits, of $P||S_{\text{r}}$ and $P'||S'_{\text{r}}$, respectively. Furthermore, let B and B' be the last $N - K$ bits of $P||S_{\text{r}}$ and $P'||S'_{\text{r}}$, respectively.

The birthday search is defined using ϕ and a birthday search space $V = \{0,1\}^K$. Let $\tau : V \to \{0,1\}$ be some balanced selector function, i.e., $|\tau^{-1}(0)| = |\tau^{-1}(1)|$. Then the birthday step function $f : V \to V$ is defined as

$$f(v) = \begin{cases} \phi(\text{Compress}(IHV_{\text{in}}, B||v), 0) & \text{if } \tau(v) = 0; \\ \phi(\text{Compress}(IHV'_{\text{in}}, B'||v), 1) & \text{if } \tau(v) = 1. \end{cases}$$

A birthday search collision $f(v) = f(w)$ with $v \neq w$ satisfies predicate `useful` if $\tau(v) \neq \tau(w)$ and $IHV'_{\text{out}} - IHV_{\text{out}} \in \mathcal{I}$ where

$$IHV_{\text{out}} = \text{Compress}(IHV_{\text{in}}, B||v), \quad IHV'_{\text{out}} = \text{Compress}(IHV'_{\text{in}}, B'||w),$$

12. This implies that collisions $a \neq b$ with $f(a) = f(b)$ and $\texttt{Pred}(a,b) = 1$ exist.

assuming without loss of generality that $\tau(v) = 0$ and $\tau(w) = 1$. As $f(v) = f(w)$ and $\tau(v) = 0 \neq \tau(w)$ implies that $\phi(IHV_{\text{out}}, 0) = \phi(IHV'_{\text{out}}, 1)$, it follows that the probability of a birthday search collision satisfying predicate `useful` is $p_{\texttt{useful}}/2$. Using Lemma 3.7, we can conclude that the average birthday search complexity is

$$\sqrt{\frac{\pi \cdot 2^K}{2 \cdot p_{\texttt{useful}}/2}} = \sqrt{\frac{\pi}{p_{\texttt{useful}}}} \cdot 2^{K/2}.$$

As $p_{\texttt{useful}} \geq \pi \cdot 2^{-K}$, one can expect a 'useful' birthday collision before the entire search space has been exhausted. Furthermore, since $p_{\texttt{useful}} \geq 8 \cdot 2^{K-L}$, the average birthday search complexity is at most half that of a brute force collision attack against H.

A successful birthday search gives us bit strings $S_{\text{b}} = v$ and $S'_{\text{b}} = w$ such that $D = IHV'_c - IHV_c \in \mathcal{I}$, where IHV_c and IHV'_c are the intermediate hash values after processing the first c blocks, i.e., the first $c \cdot N$ bits, of $P||S_{\text{r}}||S_{\text{b}}$ and $P'||S'_{\text{r}}||S'_{\text{b}}$, respectively. To finish we need to construct a near-collision attack that starts with IHV-values IHV_c and IHV'_c and searches for message blocks S_{c} and S'_{c} such that

$$IHV'_{c+1} - IHV_{c+1} = IHV'_c - IHV_c - D = 0 \quad \Rightarrow \quad IHV'_{c+1} = IHV_{c+1},$$

where

$$IHV_{c+1} = \text{Compress}(IHV_c, S_{\text{c}}), \quad IHV'_{c+1} = \text{Compress}(IHV'_c, S'_{\text{c}}).$$

After this successful near-collision attack we have obtained our chosen-prefix collision:

$$H(P||S_{\text{r}}||S_{\text{b}}||S_{\text{c}}) = H(P'||S'_{\text{r}}||S'_{\text{b}}||S'_{\text{c}}).$$

Due to the second criteria of Theorem 3.6, the complexity of constructing and executing this near-collision attack is at most one quarter of the complexity of a brute force collision attack against H.

Our chosen-prefix collision attack against H has an average complexity of three quarters of the complexity of a brute force collision attack against H. □

3.6 Recommendations

Based on the results presented in this thesis, we offer the following recommendations on the use of hash functions.

1. Given the current state-of-art in the cryptanalysis of MD5, we urgently recommend to immediately replace MD5 by a more secure hash function (e.g., SHA-2) in all applications, unless it is known that the application's security does not depend on the collision resistance of MD5.

   Especially in the case of MD5, it has been argued at the time of the first collision attack that such collision attacks do not pose a realistic threat due to attack

complexity, the problem of creating meaningful collisions and other technical constraints. However, we provide several examples in Chapter 4 that show that these arguments did not withstand the progress in cryptanalytic techniques and the creativity of researchers in constructing realistic threats.

2. Given the current state-of-art in the cryptanalysis of SHA-1, we strongly recommend to replace SHA-1 by a more secure hash function (e.g., SHA-2) in all applications, unless it is known that the application's security does not depend on the collision resistance of SHA-1.

   It can be argued, as it has been before with MD5 and was later refuted, that the current collision attacks on SHA-1 do not pose a threat due to too high attack complexities. However, our new differential cryptanalysis presented in Section 7.5 provides the exact analysis of combinations of local collisions that has been lacking so far. Furthermore, we provide several possibilities to improve our first attempt at a SHA-1 collision attack. Thus the SHA-1 collision attack complexity will very likely decrease in the future.

3. Furthermore, we recommend to use hash functions in a randomized hashing scheme (c.f. [HK06]), where possible, in such a manner that the randomness cannot be influenced by a possible adversary.

   In particular this holds for digital signatures as one should never sign any document whose entire contents can be controlled or determined by a possibly malicious party (e.g., see Section 4.2).

4. It is often observed that replacing a cryptographic primitive in applications can be a lengthy process, therefore we recommend to start such a replacement process as soon as indications of security weaknesses arise and not wait until sufficient proof has been given that these security weaknesses can be abused.

5. In the event the above recommendations apply but cannot be implemented in the near-future (e.g., due to compatibility issues, hardware designs, etc.), we offer the recommendation of implementing a barrier against collision-attacks using the algorithms presented in Chapter 8.

3.7 Directions for further research

We also offer some possible directions for further research:

1. Our disturbance vector analysis presented in Section 7.5 might be improved by a more compact definition of differential paths over steps $t_b, \ldots, t_e$ by replacing ΔQ_{t_e} with the pair $(\delta Q_{t_e}, \delta RL(Q_{t_e}, 5))$ which represents several possible allowed values for ΔQ_{t_e}. The definition for the probability of these differential paths should be adjusted accordingly. This improvement will reduce the runtime and memory cost of the analysis.

2. The reduction algorithm of Section 7.5.4 can be extended to remove more differences of a differential path $\mathcal{P}$ as long as:

$$\Pr[\mathcal{P}_{\text{ext}}] = \Pr[\mathcal{P}_{\text{redext}}] \cdot \frac{\Pr[\mathcal{P}]}{\Pr[\mathit{Reduce}(\mathcal{P})]},$$

for all (forward or backward) extensions $\mathcal{P}_{\text{ext}}$ of $\mathcal{P}$ and the analogical extensions $\mathcal{P}_{\text{redext}}$ of $\mathit{Reduce}(\mathcal{P})$. This improvement will reduce the runtime and memory cost of the analysis.

3. As described in Section 7.6.8 one can also use a tunnel if it breaks only message bitrelations over m_{14} and m_{15}. Such tunnels can be used to speed up steps 14 and 15. However, the true effect of these tunnels on the optimization and runtime complexity of the near-collision attack is subject for further research.

4. So far we have analyzed one disturbance vector and made an early attempt at a near-collision attack in Section 7.6. More research is necessary to analyze other disturbance vectors and other combinations of tunnels and speedup techniques. This would be aided greatly by an algorithmic construction of a (near-)optimal near-collision attack given disturbance vector, differential path, message bitrelations and a set of tunnels.

5. The implementation of our MD5 chosen-prefix collision attack [HC] can make use of massively parallel computing architectures, such as CELL SPU (PlayStation 3) and CUDA (graphic cards), that offer a significantly higher performance per cost ratio compared to common computing architectures such as x86, AMD64, PowerPC, etc. However, so far this speedup is strictly limited to the birthday search. Further research would be necessary to determine to what extent the near-collision attacks against MD5 and SHA-1 could benefit from massively parallel computing architectures.

6. The SHA-1 chosen-prefix collision attack complexity presented in Section 7.7 is dominated by the complexity of the birthday search, which is directly related to the size of the reasonably structured set $\mathcal{I}$ of δIHV_{diff}-values that can be efficiently eliminated with near-collision attacks. It is unclear which disturbance vector leads to the largest such set $\mathcal{I}$ while keeping the near-collision attack complexity relatively low. Furthermore, one could even combine several such sets $\mathcal{I}$ from different disturbance vectors similar to how the chosen-prefix collision attack on MD5 uses 32 possible message differences. This could significantly decrease the complexity for SHA-1 chosen-prefix collisions.

7. How the cryptanalytical techniques used for MD5 and SHA-1 can aid the cryptanalysis of other hash functions or other cryptographic primitives.

3.8 Thesis outline

The remainder of this thesis consists of the following parts: Chapter 4 discusses the application of chosen-prefix collision attacks in a number of abuse scenarios. In

particular it describes in detail our most important successful application of a chosen-prefix collision attack in Section 4.2, namely the construction of a rogue Certification Authority certificate.

In Chapter 5 we introduce the definition for the family $\mathcal{F}_{\text{md4cf}}$ of compression functions and differential paths thereof. Furthermore, we prove Theorem 3.5 by presenting an algorithm for constructing complete differential paths which is a generalization of the algorithms for MD5 and SHA-1 presented in Chapter 6 and Chapter 7, respectively.

Chapter 6 focuses on MD5 and provides an algorithm for constructing complete differential paths for MD5 together with a collision finding algorithm algorithm that searches for actual near-collision blocks given a differential path. Finally this section proves Theorem 3.1 by presenting the practical identical-prefix and chosen-prefix collision attacks with the respective claimed complexities.

Similarly, Chapter 7 focuses on SHA-1 and provides an algorithm for constructing complete differential paths for SHA-1. This is followed by the presentation of our SHA-1 disturbance vector analysis. At the end of Chapter 7 we present the construction of a practical near-collision attack which proves Theorem 3.3.

Chapter 8 presents a technique to detect an identical-prefix or chosen-prefix collision given only one message of the collision pair. Such a technique may prevent abuse of collision attacks. We apply this technique to MD5 and SHA-1 in Sections 8.3 and 8.4, respectively.

4 Chosen-prefix collision abuse scenarios

Contents

4.1 Survey

When exploiting collisions in real world applications two major obstacles must be overcome.

- The problem of constructing *meaningful collisions*. Given current methods, collisions require appendages consisting of unpredictable and mostly uncontrollable bit strings. These must be hidden in the usually heavily formatted application data structure without raising suspicion.
- The problem of constructing *realistic attack scenarios*. As we do not have effective attacks against MD5's (second) pre-image resistance but only collision attacks, we cannot target existing MD5 hash values. In particular, the colliding data structures must be generated simultaneously, along with their shared hash, by the adversary.

In this section several chosen-prefix collision applications are surveyed where these problems are addressed with varying degrees of success. Sections 4.2, 4.3, and 4.4 describe the three most prominent applications in more detail. These applications are the result of work done jointly with Arjen Lenstra and Benne de Weger, Section 4.2 is the result of joint work with Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik and Benne de Weger [SLdW07c, SSA+09b, SLdW12]. The theory, algorithms, implementation and practical execution of the underlying collision attacks of these applications as described in Chapter 6 and Chapter 7 are the work of the author of this thesis.

Digital certificates. Given how heavily they rely on cryptographic hash functions, digital certificates are the first place to look for applications of chosen-prefix collisions. Two X.509 certificates are said to collide if their to-be-signed parts have the same hash and consequently their digital signatures, as provided by the CA (Certification Authority), are identical. In [LdW05] it was shown how identical-prefix collisions can be used to construct colliding X.509 certificates with different RSA moduli but identical Distinguished Names. Here the RSA moduli absorbed the random-looking near-collision blocks, thus inconspicuously and elegantly solving the meaningfulness problem. Allowing different Distinguished

Names required chosen-prefix collisions, as we have shown in [SLdW07c] in collaboration with Arjen Lenstra and Benne de Weger. The certificates resulting from both constructions do not contain spurious bits, so superficial inspection at bit level of either of the certificates does not reveal the existence of a sibling certificate that collides with it signature-wise. Nevertheless, for these constructions to work the entire to-be-signed parts, and thus the signing CA, must be fully under the attacker's control, thereby limiting the practical attack potential.

A related but in detail rather different construction was carried out in collaboration with Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik and Benne de Weger, as reported in [SSA+09a, SSA+09b] and in Section 4.2. Although in practice a certificate's to-be-signed part cannot be for 100% under control of the party that submits the certification request, for some commercial CAs (that still used MD5 for their digital signature generation) the entire to-be-signed part could be predicted reliably enough to make the following guess-and-check approach practically feasible: prepare the prefix of the to-be-signed part of a legitimate certification request including a guess for the part that will be included by the CA upon certification, prepare a rogue to-be-signed prefix, determine different collision-causing and identical collision-maintaining appendages to complete two colliding to-be-signed parts, and submit the legitimate one for certification. If upon receipt of the legitimate certificate the guess turns out to have been correct, then the rogue certificate can be completed by pasting the CA's signature of the legitimate data onto the rogue data: because the data collide, the signature is equally valid for both. Otherwise, if the guess is incorrect, another attempt is made. Using this approach we managed (upon the fourth attempt) to trick a commercial CA into providing a signature valid for a rogue CA certificate. For the intricate details of the construction we refer to Section 4.2.

A few additional remarks about this construction are in order here. We created not just a rogue certificate, but a rogue CA certificate, containing identifying information and public key material for a rogue CA. The private key of this rogue CA is under our control. Because the commercial CA's signature is valid for the rogue CA certificate, all certificates issued by the rogue CA are trusted by anybody trusting the commercial CA. As the commercial CA's root certificate is present in all major browsers, this gives us in principle the possibility to impersonate any certificate owner. This is certainly a realistic attack scenario. The price that we have to pay is that the meaningfulness problem is only adequately – and most certainly not elegantly – solved: as further explained in the next paragraph, one of the certificates contains a considerable number of suspicious-looking bits.

To indicate that a certificate is a CA certificate, a certain bit has to be set in the certificate's to-be-signed-part. According to the X.509v3 standard [CSF+08], this bit comes after the public key field. Because it is unlikely that a commercial CA accepts a certification request where the CA bit is set, the bit must not be set

in the legitimate request. For our rogue CA certificate construction, the fact that the two to-be-signed parts must contain a different bit *after* the public key field causes an incompatibility with our 'usual' colliding certificate construction as in [SLdW07c]. In that construction the collision-causing appendages correspond to the high order bits of RSA moduli, and they are followed by identical collision-maintaining appendages that transform the two appendages into valid RSA moduli. Anything following after the moduli must remain identical lest the collision property goes lost. As a consequence, the appendages on the rogue side can no longer be hidden in the public key field and some other field must be found for them. Such a field may be specially defined for this purpose, or an existing (proprietary) extension may be used. The Netscape Comment extension is a good example of the latter, as we found that it is ignored by the major certificate processing software. The upshot is, however, that as the appendages have non-negligible length, it will be hard to define a field that will not look suspicious to someone who looks at the rogue certificate at bit level.

Colliding documents. In [DL05] (see also [GIS05]) it was shown how to construct a pair of PostScript files that collide under MD5, but that display different messages when viewed or printed. These constructions use identical-prefix collisions and thus the only difference between the colliding files is in the generated collision bit strings. It follows that they have to rely on the presence of both messages in each of the colliding files and on macro-functionalities of the document format used to show either one of the two messages. Obviously, this raises suspicion upon inspection at bit level. With chosen-prefix collisions, one message per colliding document suffices and macro-functionalities are no longer required. For example, using a document format that allows insertion of color images (such as Microsoft Word or Adobe PDF), inserting one message per document, two documents can be made to collide by appending carefully crafted color images after the messages. A short one pixel wide line will do – for instance hidden inside a layout element, a company logo, or a nicely colored barcode – and preferably scaled down to hardly visible size (or completely hidden from view, as possible in PDF). An extension of this construction is presented in the paragraphs below and set forth in detail in Section 4.3.

Hash based commitments. Kelsey and Kohno [KK06] presented a method to first commit to a hash value, and next to construct faster than by a trivial pre-image attack a document with the committed hash value, and with any message of one's choice as a prefix. The method applies to any Merkle-Damgård hash function, such as MD5, that given an IHV and a suffix produces some IHV. Omitting details involving message lengths and padding, the idea is to commit to a hash value based on an IHV at the root of a tree, either that IHV itself or calculated as the hash of that IHV and some suffix at the root. The tree is a complete binary tree and is calculated from its leaves up to the root, so the IHV at the root will be one of the last values calculated. This is done in such a way that each node of the tree is associated with an IHV along with a

suffix that together hash to the *IHV* associated with the node's parent. Thus, two siblings have *IHV* values and suffixes that collide under the hash function. The *IHV* values at the leaves may be arbitrarily chosen but are, preferably, all different. Given a prefix of one's choice one performs a brute-force search for a suffix that, when appended to the prefix and along with the standard *IHV*, results in the *IHV* at one of the leaves (or nodes) of the tree. Appending the suffixes one encounters on one's way from that leave or node to the root, results in a final message with the desired prefix and committed hash value.

Originally based on a birthday search, the construction of the tree can be done more efficiently by using chosen-prefix collisions to construct sibling node suffixes based on their *IHV* values. For MD5, however, it remains far from feasible to carry out the entire construction in practice. In a variant that *is* feasible, one commits to a prediction by publishing its hash value. In due time one reveals the correct prediction, chosen from among a large enough preconstructed collection of documents that, due to tree-structured chosen-prefix collision appendages, all share the same published hash value. In section 4.3 we present an example involving 12 documents.

Software integrity checking. In [Kam04] and [Mik04] it was shown how any existing MD5 collision, such as the ones originally presented by Xiaoyun Wang at the Crypto 2004 rump session, can be abused to mislead integrity checking software that uses MD5. A similar application, using freshly made collisions, was given on [Sel06]. As shown on [Ste09] this can even be done within the framework of Microsoft's Authenticode code signing program. All these results use identical-prefix collisions and, similar to the colliding PostScript application mentioned earlier, differences in the colliding inputs are used to construct deviating execution flows.

Chosen-prefix collisions allow a more elegant approach, since common operating systems ignore bit strings that are appended to executables: the programs will run unaltered. Thus, using tree-structured chosen-prefix collision appendages as above, any number of executables can be made to have the same MD5 hash value or MD5-based digital signature. See Section 4.4 for an example.

One can imagine two executables: a 'good' one (say Word.exe) and a 'bad' one (the attacker's Worse.exe). A chosen-prefix collision for those executables is computed, and the collision-causing bit strings are appended to both executables. The resulting altered file Word.exe, functionally equivalent to the original Word.exe, can be offered to a code signing program such as Microsoft's Authenticode and receive an 'official' MD5-based digital signature. This signature will then be equally valid for the attacker's Worse.exe, and the attacker might be able to replace Word.exe by his Worse.exe (renamed to Word.exe) on the appropriate download site. This construction affects a common functionality of MD5 hashing and may pose a practical threat. It also allows people to get many executables signed at once at the cost of getting a single such executable signed,

bypassing verification of any kind (e.g., authenticity, quality, compatibility, non-spyware, non-malware) by the signing party of the remaining executables.

Computer forensics. In computer forensics so-called *hash sets* are used to quickly identify known files. For example, when a hard disk is seized by law enforcement officers, they may compute the hashes of all files on the disk, and compare those hashes to hashes in existing hash sets: a whitelist (for known harmless files such as operating system and other common software files) and a blacklist (for previously identified harmful files). Only files whose hashes do not occur in either hash set have to be inspected further. A useful feature of this method of recognizing files is that the file name itself is irrelevant, since only the content of the file is hashed.

MD5 is a popular hash function for this application. Examples are NIST's National Software Reference Library Reference Data Set[13] and the US Department of Justice's Hashkeeper application[14].

A conceivable, and rather obvious, attack on this application of hashes is to produce a harmless file (e.g., an innocent picture) and a harmful one (e.g., an illegal picture), and insert collision blocks that will not be noticed by common application software or human viewers. In a learning phase the harmless file might be submitted to the hash set and thus the common hash may end up on the whitelist. The harmful file will be overlooked from then on.

Peer to peer software. Hash sets are also used in peer to peer software. A site offering content may maintain a list of pairs (file name, hash). The file name is local only, and the peer to peer software uniquely identifies the file's content by means of its hash. Depending on how the hash is computed such systems may be vulnerable to a chosen-prefix attack. Software such as eDonkey and eMule use MD4 to hash the content in a two stage manner: the identifier of the content $c_1\|c_2\|\ldots\|c_n$ is $\mathrm{MD4}(\mathrm{MD4}(c_1)\|\ldots\|\mathrm{MD4}(c_n))$, where the chunks c_i are about 9 MB each. One-chunk files, i.e., files not larger than 9 MB, are most likely vulnerable; whether multi-chunk files are vulnerable is open for research. We have not worked out the details of a chosen-prefix collision attack against MD4, but this seems very well doable by adapting our methods and should result in an attack that is considerably faster than our present one against MD5.

Content addressed storage. In recent years *content addressed storage* is gaining popularity as a means of storing fixed content at a physical location of which the address is directly derived from the content itself. For example, a hash of the content may be used as the file name. See [PD05] for an example. Clearly, chosen-prefix collisions can be used by an attacker to fool such storage systems, e.g., by first preparing colliding pairs of files, by then storing the harmless-looking first one, and later overwriting it with the harmful second one.

13. `http://www.nsrl.nist.gov/`
14. `http://www.usdoj.gov/ndic/domex/hashkeeper.htm`

Further investigations are required to assess the impact of chosen-prefix collisions. We leave it to others to study to what extent commonly used protocols and message formats such as TLS, S/MIME (CMS), IPSec and XML Signatures (see [BR06b] and [HS05]) allow insertion of random looking data that may be overlooked by some or all implementations. The threat posed by identical-prefix collisions is not well understood either: their application may be more limited, but for MD5 they can be generated almost instantaneously and thus allow real-time attacks on the execution of cryptographic protocols, and, more importantly, for SHA-1 they may soon be feasible. We present a possible countermeasure against identical-prefix and chosen-prefix collision attacks for MD5 and SHA-1 in Chapter 8.

4.2 Creating a rogue Certification Authority certificate

In our conference paper [SLdW07c, Section 4.1] we daydreamed:

> "Ideally, a realistic attack targets the core of PKI: provide a relying party with trust, beyond reasonable cryptographic doubt, that the person indicated by the Distinguished Name field has exclusive control over the private key corresponding to the public key in the certificate. The attack should also enable the attacker to cover his trails."

Our dream scenario has been, mainly, realized with the construction of a rogue CA certificate. With the private key of a CA under our control, and the public key appearing in a certificate with a valid signature of a commercial CA that is trusted by all major browsers, we can create 'trusted' certificates at will. When scrutinized at bit level, however, our rogue CA certificate may look suspicious which may, ultimately, expose us. Bit level inspection is not something many users will engage in – if they know the difference between `https` and `http` to begin with – and, obviously, the software that is supposed to inspect a certificate's bits is expertly guided around the suspicious ones. So, it may be argued that our construction has a non-negligible attack potential. Below we discuss some possibilities in this direction. Upfront, however, we like to point out that our rogue CA is nothing more than a proof of concept that is incapable of doing much harm, because it expired, on purpose, in September of 2004, i.e., more than four years before it was created.

Any website secured using TLS can be impersonated using a rogue certificate issued by a rogue CA. This is irrespective of which CA issued the website's true certificate and of any property of that certificate (such as the hash function it is based upon – SHA-256 is not any better in this context than MD4). Combined with redirection attacks where `http` requests are redirected to rogue web servers, this leads to virtually undetectable phishing attacks.

But any application involving a Certification Authority that provides MD5-based certificates with sufficiently predictable serial number and validity period may be vulnerable. In contexts different from TLS this may include signing or encryption of e-mail or software, non-repudiation services, etc.

As pointed out earlier, bit-level inspection of our rogue CA certificate will reveal a relatively large number of bits that may look suspicious – and that *are* suspicious.

This could have been avoided if we had chosen to create a rogue certificate for a regular website, as opposed to a rogue CA certificate, because in that case we could have hidden all collision causing bits inside the public keys. Nevertheless, even if each resulting certificate by itself looks unsuspicious, as soon as a dispute arises, the rogue certificate's legitimate sibling can be located with the help of the CA, and the fraud becomes apparent by putting the certificates alongside, thus exposing the party responsible for the fraud.

Our attack relies on our ability to predict the content of the certificate fields inserted by the CA upon certification: if our prediction is correct with non-negligible probability, a rogue certificate can be generated with the same non-negligible probability. Irrespective of the weaknesses, known or unknown, of the cryptographic hash function used for digital signature generation, our type of attack becomes effectively impossible if the CA adds a sufficient amount of fresh randomness to the certificate fields before the public key fields. Relying parties, however, cannot verify this randomness. Also, the trustworthiness of certificates should not crucially depend on such secondary and circumstantial aspects. We would be in favor of a more fundamental solution – along with a strong cryptographic hash function – possibly along the lines as proposed in [HK06]. Generally speaking, it is advisable not to sign data that is completely determined by some other party. Put differently, a signer should always make a few trivial and unpredictable modifications before digitally signing a document provided by someone else.

The issue in the previous paragraph was recognized and the possibility of the attack presented in this paper anticipated in the catalogue [Bun08] of algorithms suitable for the German Signature Law ('Signaturgesetz'). This catalogue includes conditions and time frames for cryptographic hash algorithms to be used in legally binding digital signatures in Germany. One of the changes introduced in the 2008 version of the catalog is an explicit condition on the usage of SHA-1: only until 2010, and only for so-called "qualified certificates" that contain at least 20 bits of entropy in their serial numbers. We are grateful to Prof. Werner Schindler of the BSI for bringing this to our attention and for confirming that this change was introduced to thwart exactly the type of rogue certificates that we present here for MD5.

We stress that our attack on MD5 is not a pre-image or second pre-image attack. We cannot create a rogue certificate having a signature in common with a certificate that was not especially crafted using our chosen-prefix collision. In particular, we cannot target any existing, independently created certificate and forge a rogue certificate that shares its digital signature with the digital signature of the targeted certificate. Given any certificate with an MD5-based digital signature, so far a relying party cannot easily recognize if it is trustworthy or, on the contrary, crafted by our method. However, in Chapter 8 we present a method to distinguish near-collision attacks given only either certificate. This method could both be used to prevent legitimate-looking but malicious certificates to be signed by CAs and to block malicious certificates in the end-users applications. Nevertheless, we repeat our urgent recommendation not to use MD5 for new X.509 certificates. How existing MD5 certificates should be handled is a subject of further research. We also urgently recommend reconsidering usage of

MD5 in other applications. Proper alternatives are available; but compatibility with existing applications is obviously another matter.

The first colliding X.509 certificate construction was based on an identical-prefix collision, and resulted in two certificates with different public keys, but identical Distinguished Name fields [LdW05]. As a first application of chosen-prefix collisions we showed how the Distinguished Name fields could be chosen differently as well [SLdW07c]. In this section we describe the details of a colliding certificate construction that goes one step further by also allowing different "basic constraints" fields. This allows us to construct one of the certificates as an ordinary website certificate, but the other one as a CA certificate, the contents of both certificates can be found in Appendix E. As already pointed out in Section 4.1, this additional difference required a radical departure from the traditional construction methods from [LdW05] and [SLdW07c]. Also, unlike our previous colliding certificate constructions where the CA was under our control, a commercial CA provided the digital signature for the (legitimate) website certificate. This required us to sufficiently accurately predict its serial number and validity period well before the certification request was submitted to the signing CA.

We exploited the following weaknesses of the commercial CA that carried out the legitimate certification request:

- Its usage of the cryptographic hash function MD5 to generate digital signatures for new certificates.
- Its fully automated way to process online certification requests that fails to recognize anomalous behavior of requesting parties.
- Its usage of sequential serial numbers and its usage of validity periods that are determined entirely by the date and time in seconds at which the certification request is processed.
- Its failure to enforce, by means of the "basic constraints" field in its own certificate, a limit on the length of the chain of certificates that it can sign.

The first three points are further discussed below. The last point, if properly handled, could have crippled our rogue CA certificate but does not affect its construction. A certificate contains a "basic constraints" field where a bit is set to indicate if the certificate is a CA certificate. With the bit set, a "path length constraint" subfield may be present, specifying an integer that indicates how many CAs may occur in the chain between the CA certificate in question and end-user certificates. The commercial CA that we interacted with failed to use this option in its own certificate, implying that any number of intermediate CAs is permitted. If the "path length constraint" would have been present and set at 0 (zero), then our rogue CA certificate could still have been constructed. But whether or not the rogue CA certificate or certificates signed by it can then also be used depends on (browser-)software actually checking the "path length constraint" subfields in chains of certificates. Thus a secondary "defense

in depth" mechanism was present that could have foiled our attack, but failed to do so simply because it was not used.

Before describing the construction of the colliding certificates, we briefly discuss the parameter choices used for the chosen-prefix collision search. First, the number of near-collision blocks is denoted by r and can be used to trade-off between birthday search time complexity and the cost of finding the r near-collision blocks. Second, k defines the birthday search space (its size is $64 + k$) and the birthday iteration function and can be used to trade-off between birthday search time complexity, birthday search memory complexity and average number of required near-collisions per birthday collision. Third, w defines the family of differential paths that can be used to construct the near-collision blocks and is the number of bit positions where arbitrary bit differences are allowed. It can be used to trade-off between the average number of required near-collision blocks per birthday collision and the cost of finding the r near-collision blocks. For more details on r, k and w we refer to Sections 6.5.2 and 6.5.3.

The 2048-bit upper bound on the length of RSA moduli, as enforced by some CAs, combined with other limitations of our certificate construction, implied we could allow for at most three near-collision blocks. Opting for the least difficult possibility (namely, three near-collision blocks), we had to decide on values for k and the aimed for value for w that determine the costs of the birthday search and the near-collision block constructions, respectively. Obviously, our choices were influenced by our computational resources, namely a cluster of 215 PlayStation 3 (PS3) game consoles. When running Linux on a PS3, applications have access to 6 Synergistic Processing Units (SPUs), a general purpose CPU, and about 150MB of RAM per PS3. For the birthday search, the 6×215 SPUs are computationally equivalent to approximately 8600 regular 32-bit cores, due to each SPU's 4×32-bit wide SIMD architecture. The other parts of the chosen-prefix collision construction are not suitable for the SPUs, but we were able to use the 215 PS3 CPUs for the construction of the actual near-collision blocks. With these resources, the choice $w = 5$ still turned out to be acceptable despite the 1000-fold increase in the cost of the actual near-collision block construction. This is the case even for the hard cases with many differences between IHV and IHV': as a consequence the differential paths contain many bitconditions, which leaves little space for the tunnels, thereby complicating the near-collision block construction.

For the targeted three near-collision blocks, the entries for $w = 5$ in the first table in Appendix D show the time-memory trade-off when the birthday search space is varied with k. With 150MB at our disposal per PS3, for a total of about 30GB, we decided to use $k = 8$ as this optimizes the overall birthday search complexity for the plausible case that the birthday search takes $\sqrt{2}$ times longer than expected. The resulting overall chosen-prefix collision construction takes on average less than a day on the PS3-cluster. In theory we could have used 1TB (or more) of hard drive space, in which case it would have been optimal to use $k = 0$ for a birthday search of about 20 PS3 days which is about 2.3 hours on the PS3-cluster.

We summarize the construction of the colliding certificates in the sequence of steps below, and then describe each step in more detail.

1. Construction of templates for the two to-be-signed parts, as outlined in Figure 7. Note that we distinguish between a 'legitimate' to-be-signed part on the left hand side, and a 'rogue' to-be-signed part on the other side.
2. Prediction of serial number and validity period for the legitimate part, thereby completing the chosen prefixes of both to-be-signed parts.
3. Computation of the two different collision-causing appendages.
4. Computation of a single collision-maintaining appendage that will be appended to both sides, thereby completing both to-be-signed parts.
5. Preparation of the certification request for the legitimate to-be-signed part.
6. Submission of the certification request and receipt of the new certificate.
7. If serial number and validity period of the newly received certificate are as predicted, then the rogue certificate can be completed. Otherwise return to Step 2.

The resulting rogue CA certificate and the end-user certificate, together with the differential paths used for the three near-collision blocks, can be found in Appendix E.

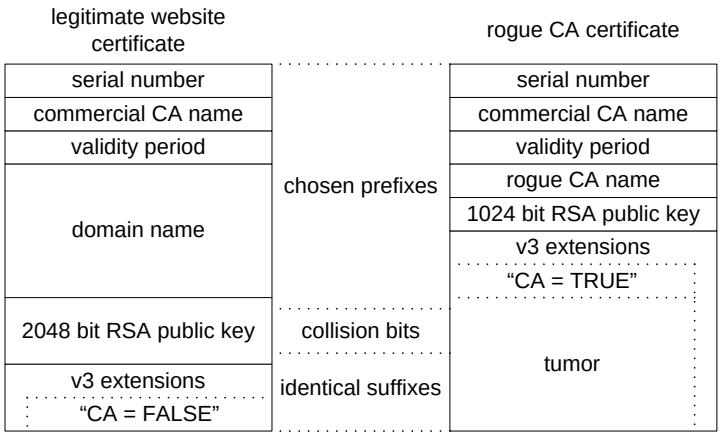

Figure 7: *The to-be-signed parts of the colliding certificates.*

Step 1. Templates for the to-be-signed parts. In this step all bits are set in the two to-be-signed parts, except for bits that are determined in later steps. For the latter bits space is reserved here. On the legitimate side the parts to be filled in later are the predictions for the serial number and validity period, and most bits of the

public key. On the rogue side the largest part of the content of an extension field of the type "Netscape Comment" is for the moment left undetermined. The following roughly describes the sequence of steps.

- On the legitimate side, the chosen prefix contains space for serial number and validity period, along with the exact Distinguished Name of the commercial CA where the certification request will be submitted. This is followed by a subject Distinguished Name that contains a legitimate website domain name (owned by one of us) consisting of as many characters as allowed by the commercial CA (in our case 64), and concluded by the first 208 bits of an RSA modulus, the latter all chosen at random after the leading '1'-bit. These sizes were chosen in order to have as many corresponding bits as possible on the rogue side, while fixing as few bits as possible of the RSA modulus on the legitimate side (see Step 4 for the reason why).

- The corresponding bits on the rogue side contain an arbitrarily chosen serial number, the same commercial CA's Distinguished Name, an arbitrarily chosen validity period (actually chosen as indicating "August 2004", to avoid abuse of the rogue certificate), a short rogue CA name, a 1024-bit RSA public key generated using standard software, and the beginning of the X.509v3 extension fields. One of these fields is the "basic constraints" field, a bit that we set to indicate that the rogue certificate will be a CA certificate (in Figure 7 this bit is denoted by "CA=TRUE").

- At this point the entire chosen prefix is known on the rogue side, but on the legitimate side predictions for the serial number and validity period still need to be inserted. That is done in Step 2.

- The various field sizes were selected so that on both sides the chosen prefixes are now 96 bits short of the same MD5 block boundary. On both sides these 96 bit positions are reserved for the birthday bits. Because only $64 + k = 72$ birthday bits per side are needed (and appended in Step 3) the first 24 bits at this point are set to 0. On the legitimate side these 96 bits are part of the RSA modulus, on the rogue side they are part of an extension field of the type "Netscape Comment", denoted as 'tumor' in Figure 7.

- From here on forward, everything that goes to the rogue side is part of the "Netscape Comment" field, as it is not meaningful for the rogue CA certificate but only appended to cause and maintain a collision with bits added to the legitimate side. On the legitimate side we first make space for 3 near-collision blocks of 512 bits each (calculated in Step 3) and for 208 bits used to complete a 2048-bit RSA modulus (determined in Step 4), and then set the RSA public exponent (for which we took the common choice 65537) and the X.509v3 extensions including the bit indicating that the legitimate certificate will be an end-user certificate (in Figure 7 denoted by "CA=FALSE").

Step 2. Prediction of serial number and validity period. Based on repeated certification requests submitted to the targeted commercial CA, it turned out that the validity period can very reliably be predicted as the period of precisely one year plus one day, starting exactly six seconds after a request is submitted. So, to control that field, all we need to do is select a validity period of the right length, and submit the legitimate certification request precisely six seconds before it starts. Though occasional accidents may happen in the form of one-second shifts, this was the easy part.

Predicting the serial number is harder but not impossible. In the first place, it was found that the targeted commercial CA uses sequential serial numbers. Being able to predict the next serial number, however, is not enough: the construction of the collision can be expected to take at least a day, before which the serial number and validity period have to be fixed, and only after which the to-be-signed part of the certificate will be entirely known. As a consequence, there will have been a substantial and uncertain increment in the serial number by the time the collision construction is finished. So, another essential ingredient of our construction was the fact that the CA's weekend workload is quite stable: it was observed during several weekends that the increment in serial number over a weekend does not vary a lot. This allowed us to pretty reliably predict Monday morning's serial numbers on the Friday afternoon before. Thus, on Friday afternoon we selected a number at the high end of the predicted range for the next Monday morning, and inserted it in the legitimate to-be-signed part along with a validity period starting that same Monday morning at the time corresponding to our serial number prediction. See Step 6 how we then managed, after the weekend, to target precisely the selected serial number and validity period.

Step 3. Computation of the collision. At this point both chosen prefixes have been fully determined so the chosen-prefix collision can be computed: first the 72 birthday bits per side, calculated in parallel on the 1290 SPUs of a cluster of 215 PS3s, followed by the calculation of 3 pairs of 512-bit near-collision blocks on a quad-core PC and the 215 PS3 CPUs. The entire calculation takes on average about a day.

Given that we had a weekend available, and that the calculation can be expected to take just a day, we sequentially processed a number of chosen-prefixes, each corresponding to different serial numbers and validity periods (targeting both Monday and Tuesday mornings). So, a near-collision block calculation on the CPUs would always run simultaneously with a birthday search on the SPUs for the 'next' attempt.

Step 4. Finishing the to-be-signed parts. At this point the legitimate and rogue sides collide under MD5, so that from here on only identical bits may be appended to both sides.

With $208 + 24 + 72 + 3 * 512 = 1840$ bits set, the remaining $2048 - 1840 = 208$ bits need to be set for the 2048-bit RSA modulus on the legitimate side. Because in the next step the RSA private exponent corresponding to the RSA public exponent is needed, the full factorization of the RSA modulus needs to be known, and the factors must be compatible with the choice of the RSA public exponent. Because common

CAs (including our targeted commercial CA) do not check for compositeness of RSA moduli in certification requests, we could simply have added 208 bits to make the RSA modulus a prime. We found that approach unsatisfactory, and opted for the rather crude but trivial to program method sketched below that results in a 224-bit prime factor with a prime 1824-bit cofactor. Given that at the time this work was done the largest factor found using the elliptic curve integer factorization method was 222 bits long, a 224-bit smallest prime factor keeps the resulting modulus out of reach of common factoring efforts. We could have used a relatively advanced lattice-based method to try and squeeze in a 312-bit prime factor along with a prime 1736-bit cofactor. Given only 208 bits of freedom to select a 2048-bit RSA modulus, it is unlikely that a more balanced solution can efficiently be found. Thus the reason why as few bits as possible should be fixed in Step 1, is that it allows us to construct a slightly less unbalanced RSA modulus.

Let N be the 2048-bit integer consisting of the 1840 already determined bits of the RSA modulus-to-be, followed by 208 one bits. We select a 224-bit integer p at random until $N = a \cdot p + b$ with $a \in \mathbb{N}$ and $b < 2^{208}$, and keep doing this until both p and $q = \lfloor N/p \rfloor$ are prime and the RSA public exponent is coprime to $(p-1)(q-1)$. Once such primes p and q have been found, the number pq is the legitimate side's RSA modulus, the leading 1840 bits of which are already present in the legitimate side's to-be-signed part, and the 208 least significant bits of which are inserted in both to-be-signed parts.

To analyze the required effort somewhat more in general, 2^{k-208} integers of k bits (with $k > 208$) need to be selected on average for pq to have the desired 1840 leading bits. Since an ℓ-bit integer is prime with probability approximately $1/\log(2^\ell)$, a total of $k(2048-k)2^{k-208}(\log 2)^2$ attempts may be expected before a suitable RSA modulus is found. The coprimality requirement is a lower order effect that we disregard. Note that for $k(k-2048)(\log 2)^2$ of the attempts the k-bit number p has to be tested for primality, and that for $(2048-k)\log 2$ of those q needs to be tested as well (on average, obviously). For $k = 224$ this turned out to be doable in a few minutes on a standard PC.

This completes the to-be-signed parts on both sides. Now it remains to be hoped that the legitimate part that actually will be signed corresponds, bit for bit, with the legitimate to-be-signed part that we concocted.

Step 5. Preparing the certification request. Using the relevant information from the legitimate side's template, i.e., the subject Distinguished Name and the public key, a PKCS#10 Certificate Signing Request is prepared. The CA requires proof of possession of the private key corresponding to the public key in the request. This is done by signing the request using the private key – this is the sole reason that we need the RSA private exponent.

Step 6. Submission of the certification request. The targeted legitimate to-be-signed part contains a very specific validity period that leaves no choice for the moment at which the certification request needs to be submitted to the CA. Just hoping that at that time the serial number would have precisely the predicted value is unlikely to work, so a somewhat more elaborate approach is used. About half an

hour before the targeted submission moment, the same request is submitted, and the serial number in the resulting certificate is inspected. If it is already too high, the entire attempt is abandoned. Otherwise, the request is repeatedly submitted, with a frequency depending on the gap that may still exist between the serial number received and the targeted one, and taking into account possible certification requests by others. In this way the serial number is slowly nudged toward the right value at the right time. Although there is nothing illegal about repeated certification requests, it should be possible for a CA to recognize the somewhat anomalous behavior sketched above and to take appropriate countermeasures (such as random delays or jumps in serial numbers) if it occurs.

Various types of accidents may happen, of course, and we experienced some of them, such as another CA customer 'stealing' our targeted serial number just a few moments before our attempt to get it, thereby wasting that weekend's calculations. But, after the fourth weekend it worked as planned, and we managed to get an actually signed part that exactly matched our predicted legitimate to-be-signed part.

Step 7. Creation of the rogue certificate. Given the perfect match between the actually signed part and the hoped for one, and the MD5 collision between the latter and the rogue side's to-be-signed part, the MD5-based digital signature present in the legitimate certificate as provided by the commercial CA is equally valid for the rogue side. To finish the rogue CA certificate it suffices to copy the digital signature to the right spot in the rogue CA certificate.

4.3 Nostradamus attack

In the original Nostradamus attack from [KK06] one first commits to a certain hash value, and afterwards for any message constructs a document that not only contains that message but that also has the committed hash value. In its full generality, this attack is at this point in time not feasible for MD5. It is easily doable, though, if a limited size message space has been defined upfront.

Suppose there are messages $m_1, m_2, \ldots, m_r$, then using $r-1$ chosen-prefix collisions we can construct r suffixes $s_1, s_2, \ldots, s_r$ such that the r documents $d_i = m_i \| s_i$ all have the same hash. After committing to the common hash, afterwards any of the r documents $d_1, d_2, \ldots, d_r$ can be shown, possibly to achieve some malicious goal. The other documents will remain hidden and their contents, i.e., the m_i-parts, cannot be derived – with overwhelming probability – from the single published document or from the common hash value.

To show the practicality of this variant, we have made an example consisting of 12 different PDF documents with a common MD5 hash value, where each document predicts a different outcome of the 2008 US presidential elections. The PDF format is convenient for this purpose because it allows insertion of extra image objects that are unreferenced in the resulting document and thus invisible to the viewer in any common PDF reader. The common MD5 hash value of our 12 colliding PDF documents containing our predictions is

$$\texttt{3d515dead7aa16560aba3e9df05cbc80}_{16}.$$

See [SLdW07a] for the actual PDF documents, one of which correctly predicted the outcome one year before the elections took place.

For each of the 11 collisions required for this example we used a 64-bit birthday search (on a single PS3) aiming for about 11 near-collision blocks (constructed on a quad-core PC). It took less than two days per chosen-prefix collision. Since we performed those computations our methods have improved as described in this thesis, so this attack would now run much faster.

Given the structure of PDF documents it is not entirely straightforward to insert different chosen-prefix collision blocks, while keeping the parts following those blocks identical in order to maintain the collision. The relevant details of both the PDF structure and our construction are covered here.

A PDF document is built up from the following four consecutive parts: a fixed header, a part consisting of an arbitrary number of numbered objects, an object lookup table and, finally, a trailer. The trailer specifies the number of objects, which of the objects is the unique root object (containing the document content) and which is the info object (containing the document's meta information such as authors and title etc.), and contains a filepointer to the start of the object lookup table.

Given a file containing a PDF document, additional objects can be inserted, as long as they are added to the object lookup table and the corresponding changes are made to the number of objects and the filepointer in the trailer. A template for an image object is given in Table 4-1. With the exception of binary images, the format is mostly text based. Any binary image is put between single line-feed characters (ASCII code 10) and the result is encapsulated by the keywords `stream` and `endstream`. The keyword `/Length` must specify the byte length of the image. Because in our case, the image is uncompressed and each pixel requires three bytes (‘`RGB`’), the image byte length must be three times the product of the specified width and height. The object number (`42` in the example object header) must be set to the next available object number.

Table 4-1: *An example numbered image object in the PDF format.*

Part	Contents
object header	`42 0 obj`
image header	`<< /ColorSpace /DeviceRGB /Subtype /Image`
image size	`/Length 9216 /Width 64 /Height 48 /BitsPerComponent 8`
image contents	`>> stream...endstream`
object footer	`endobj`

When constructing colliding PDF files they must be equal after the collision-causing data. The object lookup tables and trailers for all files must therefore be the same. This was achieved as follows:

- Because all documents must have the same number of objects, dummy objects are inserted where necessary.

- Because all root objects must have the same object number, they can be copied if necessary to objects with the next available object number.
- The info objects are treated in the same way as the root objects.
- To make sure that all object lookup tables and filepointers are identical, the objects can be sorted by object number and if necessary padded with spaces after their `obj` keyword to make sure that all objects with the same object number have the same file position and byte length in all files.
- Finally, the object lookup tables and trailers need to be adapted to reflect the new situation – as a result they should be identical for all files.

Although this procedure works for basic PDF files (such as PDF version 1.4 as we produced using pdflatex), it should be noted that the PDF document format allows additional features that may cause obstructions.

Given r LaTeX files with the desired subtle differences (such as names of r different candidates), r different PDF files are produced using a version of LaTeX that is suitable for our purposes (cf. above). In all these files a binary image object with a fixed object number is then inserted, and the approach sketched above is followed to make the lookup tables and trailers for all files identical. Since this binary image object is present but not used in the PDF document, it remains hidden from view in a PDF reader. To ensure that the files are identical after the hidden image contents, their corresponding objects were made the last objects in the files. This then leads to r chosen prefixes consisting of the leading parts of the PDF files up to and including the keyword `stream` and the first line-feed character. After determining $r - 1$ chosen-prefix collisions resulting in r collision-causing appendages, the appendages are put in the proper binary image parts, after which all files are completed with a line-feed character, the keywords `endstream` and `endobj`, and the identical lookup tables and trailers.

Note that the `Length` etc. fields have to be set before collision finding, and that the value of `Length` will grow logarithmically with r and linearly in the number of near-collision blocks one is aiming for.

4.4 Colliding executables

Using the same set-up as used for the Nostradamus attack reported in Section 4.3, i.e., 64-bit birthday search on a PS3 followed by the construction of about 12 near-collision blocks on a quad-core PC, it took us less than 2 days to create two different Windows executables with the same MD5 hash. Initially both 40960 bytes large, 13×64 bytes had to be appended to each executable, for a resulting size of just 41792 bytes each, to let the files collide under MD5 without changing their functionality.

See [SLdW07b] for details. As noted above, it has been shown on [Ste09] that this attack can be elevated to one on a code signing scheme.

As usual, the following remarks apply:

- An existing executable with a known and published hash value not resulting from this construction cannot be targeted by this attack: our attack is not a pre-image or second pre-image attack. In order to attack a software integrity protection or code signing scheme using this approach, the attacker must be able to manipulate the files before they are hashed (and, possibly, signed). Given the level of access required to realize the attack an attacker can probably do more harm in other simpler and more traditional ways.
- Any number r of executables can be made to collide, at the cost of $r-1$ chosen-prefix collisions and an $O(\log r)$-byte appendage to each of the r original executables.

In Chapter 8 we present a method that allows to distinguish near-collision attacks and thus distinguish potentially malicious MD5-based certificates or executables. It is better, however, not to rely on cryptographic primitives such as MD5 and SHA-1 that fail to meet their design criteria.

5 Differential cryptanalysis and paths

Contents

5.1 Introduction

In this thesis we limit ourselves to hash functions of the MD4 family and focus mainly on MD5 and SHA-1. As can be seen in Chapter 2, the construction of a near-collision attack for MD4 style compression functions has many aspects that all have to fit together. Nevertheless there is one aspect that is the key to the entire attack: the differential path. In this section we introduce a definition of a compression function family $\mathcal{F}_{\mathrm{md4cf}}$ that includes an important subset of the family of MD4 style compression functions, namely those of MD4, MD5 and SHA-1. We complement this family with a definition of differential paths for these compression functions. Such a differential path is called *valid* if there exists a solution to the differential path under a

relaxation of the message expansion. The most important contribution of this section is an algorithm that searches for complete valid differential paths for which the beginning and ending part are predetermined. In its generic form, this algorithm provides insights in differential cryptanalysis applied to compression functions. We apply this differential cryptanalysis and improve the differential path construction algorithm for MD5 and SHA-1 in Sections 6 and 7, respectively.

For MD4 style compression functions, a differential path designed for a near-collision attack consists of three segments. The most important segment, namely the end segment, consists of all steps after a certain step K whose success probability directly contributes reciprocally to the final attack complexity. E.g., the collision finding algorithm for MD5 in Chapter 2 easily fulfills the first 16 steps of the differential path and with effort can fulfill a small number of steps thereafter using message modification techniques. Since message modification techniques do not affect the fulfillment of the differential path over all steps starting at K, where K is approximately 27 for MD5, the differential path over those steps has to be fulfilled probabilistically. The expected number of attempts required, which is the reciprocal of the success probability over those steps, is a direct factor of the attack complexity. Hence, to build an efficient collision attack the success probability of a differential path over those steps should be as high as possible. Specifically, this probability must be at least $\pi^{-0.5}2^{-N/2}$ for the resulting attack not to be slower than a brute-force attack.

The first segment of the differential path consists of the initial working state defined by the input pair IHV_{in} and IHV'_{in}. Since for a near-collision attack the input pair IHV_{in} and IHV'_{in} are given, this segment is also a given.

The remaining segment thus 'connects' the begin segment and end segment of the differential path. Whereas the end segment is constructed and specifically optimized for a low attack complexity and the begin segment is pre-determined, this segment must be constructed using both extremal segments in order to build a valid complete differential path. There is no trivial solution to the problem of constructing such a connecting segment, due to the fact that each working state variable affects multiple steps of the differential path.

The first ones to solve the problem of constructing a connecting segment were Xiaoyun Wang and her team. Their paper [WY05] describes their methodology which depends mainly on expertise, intuition and patience. With their methodology they enabled the construction of identical-prefix collisions for MD5. In this section we present an algorithmic solution to this problem for a certain class of compression functions which we call $\mathcal{F}_{\text{md4cf}}$. To this end, we first define this class $\mathcal{F}_{\text{md4cf}}$ of compression functions which includes those of MD4, MD5 and SHA-1, and more formally define the concept of a differential path for this class.

Besides providing a much easier and faster way of constructing full differential paths, our algorithmic solution also allows optimization of differential paths with respect to given message modification techniques. Furthermore, the most important result of our algorithmic solution is the construction of a chosen-prefix collision attack against MD5, where the differential paths are necessarily created during the attack instead of precomputed (see Section 6.5). Compared to identical-prefix colli-

sion attacks such as the collision attack in Chapter 2, a chosen-prefix collision attack removes the identical input IHV_k and IHV'_k requirement at the start of the attack. The two equal-length input message prefixes P and P' (resulting in IHV_k and IHV'_k) can therefore be chosen independently.

First in Section 5.2 we introduce some basic definitions and notations necessary for this section. In Section 5.3 we present the formal definition of the class $\mathcal{F}_{\text{md4cf}}$ of MD4 based compression functions. This is followed by a discussion of three important properties of the step functions of the compression functions in $\mathcal{F}_{\text{md4cf}}$ in Section 5.4. Next, we formally define differential paths for the class $\mathcal{F}_{\text{md4cf}}$ of compression functions in Section 5.5. Finally, we present our differential path construction algorithm in Section 5.6.

5.2 Definitions and notation

Throughout this thesis we denote $a \bmod b$ for the least non-negative residue of a modulo b where $a \in \mathbb{Z}$ and $b \in \mathbb{N}^+$.

5.2.1 N-bit words and operators

Similar to MD5, the class $\mathcal{F}_{\text{md4cf}}$ is based on the integer working registers of modern CPU architectures. Whereas MD5 used 32-bit words, here we generalize this to N-bit words. We use the shorter notation $\mathbb{Z}_{2^N}$ for $\mathbb{Z}/2^N\mathbb{Z}$. An N-bit word $(v_i)_{i=0}^{N-1}$ consists of N bits $v_i \in \{0,1\}$. These N-bit words are identified with elements $v = \sum_{i=0}^{N-1} v_i 2^i$ of $\mathbb{Z}_{2^N}$ and we switch freely between these two representations.

On N-bit words $X = (x_i)_{i=0}^{N-1}$ and $Y = (y_i)_{i=0}^{N-1}$, we define the following operations:

- $X \wedge Y = (x_i \wedge y_i)_{i=0}^{N-1}$ is the bitwise AND of X and Y;
- $X \vee Y = (x_i \vee y_i)_{i=0}^{N-1}$ is the bitwise OR of X and Y;
- $X \oplus Y = (x_i \oplus y_i)_{i=0}^{N-1}$ is the bitwise XOR of X and Y;
- $\overline{X} = (1 - x_i)_{i=0}^{N-1}$ is the bitwise complement of X;
- $X[i]$ is the i-th bit x_i;
- $X + Y$ and $X - Y$ denote addition and subtraction, respectively, of X and Y in $\mathbb{Z}_{2^N}$;
- $RL(X,n) = (x_{(i+n \bmod N)})_{i=0}^{N-1}$ is the cyclic left rotation of X by $0 \leq n < N$ bit positions;
- $RR(X,n) = (x_{(i-n \bmod N)})_{i=0}^{N-1}$ is the cyclic right rotation of X by $0 \leq n < N$ bit positions. Equivalent to cyclic left rotation over $(N-n \bmod N)$ bit positions;
- $w(X)$ denotes the Hamming weight $\sum_{i=0}^{N} x_i$ of $X = (x_i)_{i=0}^{N}$.

5.2.2 Binary signed digit representations

We extend the notion of the BSDR to N-bit words $X \in \mathbb{Z}_{2^N}$ as a sequence $(k_i)_{i=0}^{N-1}$ such that

$$X = \sum_{i=0}^{N-1} k_i 2^i, \quad k_i \in \{-1, 0, 1\}.$$

For each non-zero $X \in \mathbb{Z}_{2^N}$ there exist many different BSDRs. The *weight* $w((k_i)_{i=0}^{N-1})$ of a BSDR $(k_i)_{i=0}^{N-1}$ is defined as the number of non-zero k_is.

A particularly useful type of BSDR is the Non-Adjacent Form (NAF), where no two non-zero k_i-values are adjacent. For any $X \in \mathbb{Z}_{2^N}$ there is no unique NAF, since we work modulo 2^N (making $k_{N-1} = +1$ equivalent to $k_{N-1} = -1$). However, uniqueness of the NAF can be enforced by the added restriction $k_{N-1} \in \{0, +1\}$. Among the BSDRs for a given $X \in \mathbb{Z}_{2^N}$, the NAF has minimal weight [MS06]. The NAF can be computed easily [Lin98] for a given $X \in \mathbb{Z}_{2^N}$ as $\mathrm{NAF}(X) = ((X+Y)[i] - Y[i])_{i=0}^{N-1}$ where Y is the N-bit word $(0\ X[N-1]\ \ldots\ X[1])$.

We use the following notation for an N-digit BSDR Z:

- $Z[i]$ is the i-th signed bit k_i of $Z = (k_i)_{i=0}^{N-1}$;
- $RL(Z, n) = (x_{(i+n) \bmod N})_{i=0}^{N-1}$ is the cyclic left rotation of Z by $0 \leq n < N$ digit positions;
- $RR(Z, n) = (x_{(i-n) \bmod N})_{i=0}^{N-1}$ is the cyclic right rotation of Z by $0 \leq n < N$ digit positions and is equivalent to $RL(Z, (N - n \bmod N))$;
- $w(Z) = \sum_{i=0}^{N-1} |k_i|$ is the weight of Z.
- $\sigma(Z) = \sum_{i=0}^{N-1} k_i 2^i \in \mathbb{Z}_{2^N}$ is the N-bit word for which Z is a BSDR.

5.2.3 Function families over N-bit words

As an aid we define three families $\mathcal{F}_{\mathrm{sumrot}}$, $\mathcal{F}_{\mathrm{bool}}$ and $\mathcal{F}_{\mathrm{boolrot}}$ of functions

$$f : (\mathbb{Z}_{2^N})^J \to \mathbb{Z}_{2^N}, \quad J \in \mathbb{N}$$

that map an input tuple of J N-bit words to a single N-bit word.

The family $\mathcal{F}_{\mathrm{sumrot}}$ consists of functions f that perform a selective sum over bitwise cyclic rotated input words and a chosen constant value $C \in \mathbb{Z}_{2^N}$:

$$f(X_1, \ldots, X_J) \mapsto C + \sum_{j=1}^{J} c_j \cdot RL(X_j, r_j), \quad c_j \in \{-1, 0, 1\},\ r_j \in \{0, \ldots, N-1\}.$$

We restrict r_j to zero whenever $c_j = 0$ for $j = 1, \ldots, J$, so that all non-trivial rotations ($r_j \neq 0$) contribute in the sum.

The family $\mathcal{F}_{\text{bool}}$ consists of functions f that extend a boolean functions $g : \{0,1\}^J \to \{0,1\}$ to words:

$$f(X_1, \ldots, X_J) \mapsto (g(X_1[i], \ldots, X_J[i]))_{i=0}^{N-1}.$$

The family $\mathcal{F}_{\text{boolrot}}$ consists of functions that first cyclically rotate their input words and then pass them to a function $g \in \mathcal{F}_{\text{bool}}$:

$$f(X_1, \ldots, X_J) \mapsto g(RL(X_1, r_1), \ldots, RL(X_J, r_J)), \quad r_j \in \{0, \ldots, N-1\}.$$

Observation 5.1. *For any $f \in \mathcal{F}_{sumrot}$ if we fix all input values except X_i, $i \in \{1, \ldots, J\}$, then the resulting f_i is either bijective and easily invertible or a constant function with respect to the remaining single input value. More formally, let $f \in \mathcal{F}_{sumrot}$:*

$$f(X_1, \ldots, X_J) = C + \sum_{j=1}^{J} c_j \cdot RL(X_j, r_j), \quad c_j \in \{-1, 0, 1\},\ r_j \in \{0, \ldots, N-1\}.$$

For any $i \in \{1, \ldots, J\}$ and given values of all inputs except X_i we define the function $f_i : \mathbb{Z}_{2^N} \to \mathbb{Z}_{2^N}$ as:

$$f_i : X_i \mapsto f(X_1, \ldots, X_J) = C_i + c_i \cdot RL(X_i, r_i),$$

where

$$C_i = C + \sum_{\substack{j=0 \\ j \neq i}}^{J} c_j \cdot RL(X_j, r_j).$$

If $c_i = 0$ then $f_i(X_i) = C_i$ is a constant function. Otherwise, if $c_i \in \{-1, 1\}$ then $f_i(X_i) = C_i + c_i RL(X_i, r_i)$ is bijective and easily invertible:

$$f_i^{-1} : A \mapsto RR(c_i \cdot (A - C_i), r_i).$$

5.3 $\mathcal{F}_{\text{md4cf}}$: MD4 based compression functions

The class $\mathcal{F}_{\text{md4cf}}$ of compression functions can be seen as a subfamily of the family of MD4-style compression functions and consists of all compression functions Compress as defined in this section. The class $\mathcal{F}_{\text{md4cf}}$ includes the compression functions of MD4, MD5 and SHA-1. It does not include the compression functions of the SHA-2 family, since these compression functions update two state variables per step instead of one.

A compression function Compress uses only fixed sized N-bit words, $N \in \mathbb{N}^+$, and the above listed N-bit word operations. From now on, we also use the shorthand *word* for N-bit word.

For fixed $L, K \in \mathbb{N}^+$, Compress(IHV_{in}, M) maps an L tuple of words IHV_{in} and a K tuple of words M to an L tuple of words referred to as IHV_{out}:

$$\text{Compress} : (\mathbb{Z}_{2^N})^L \times (\mathbb{Z}_{2^N})^K \to (\mathbb{Z}_{2^N})^L.$$

To compute IHV_{out} given IHV_{in} and M, Compress computes a sequence of $S+L$ working state words $(Q_i)_{i=-L+1}^{S} \in (\mathbb{Z}_{2^N})^{S+L}$, where S is a fixed multiple of K. The first L words $Q_{-L+1}, \ldots, Q_0$ are initialized using IHV_{in} in Section 5.3.1. The remaining S words $Q_1, \ldots, Q_S$ are sequentially computed using *step functions* in Section 5.3.4. The output L tuple IHV_{out} is computed based on the last L words $Q_{S-L+1}, \ldots, Q_S$ and IHV_{in} in Section 5.3.2.

In the computation of each word Q_i for $i = 1, \ldots, S$ a single word W_i is used that is derived from the K tuple M. Section 5.3.3 describes the mapping of M to $(W_i)_{i=0}^{S}$ called the *message expansion*. Note that the problem of endianness (Section 2.1.2) is avoided by defining M as a tuple of words instead of a bit string.

As an example, the value of the tuple (N, L, K, S) for MD4, MD5 and SHA-1 is $(32, 4, 16, 48)$, $(32, 4, 16, 64)$ and $(32, 5, 16, 80)$, respectively.

5.3.1 Working state initialization

Compress initializes the first L working state words $Q_{-L+1}, \ldots, Q_0$ using the L-tuple $IHV_{\text{in}} = (ihvin_0, \ldots, ihvin_{L-1})$:

$$Q_i = f_{\text{in},i}(ihvin_0, \ldots, ihvin_{L-1}), \quad f_{\text{in},i} \in \mathcal{F}_{\text{sumrot}},$$

for $i \in \{-L+1, \ldots, 0\}$ such that

$$(f_{\text{in},i})_{i=-L+1}^{0} : (\mathbb{Z}_{2^N})^L \to (\mathbb{Z}_{2^N})^L$$

is bijective.

In the case of MD4, MD5 and SHA-1, this initialization function $(f_{\text{in},i})_{i=-L+1}^{0}$ forms a non-trivial permutation of the L input words.

5.3.2 Finalization

After all S steps are performed, the output $IHV_{\text{out}} = (ihvout_0, \ldots, ihvout_{L-1})$ is determined as a function of the last L working state words $Q_{S-L+1}, \ldots, Q_S$ and IHV_{in}:

$$ihvout_i = f_{\text{out},i}(ihvin_0, \ldots, ihvin_{L-1}, Q_{S-L+1}, \ldots, Q_S), \quad f_{\text{out},i} \in \mathcal{F}_{\text{sumrot}},$$

for $i \in \{0, \ldots, L-1\}$. For all values of $Q_{S-L+1}, \ldots, Q_S$ we require that the following mapping is bijective:

$$(ihvin_0, \ldots, ihvin_{L-1}) \mapsto (f_{\text{out},i}(ihvin_0, \ldots, ihvin_{L-1}, Q_{S-L+1}, \ldots, Q_S))_{i=0}^{L-1}.$$

Also, for all values of $ihvin_0, \ldots, ihvin_{L-1}$ we require that the following mapping is bijective:

$$(Q_{S-L+1}, \ldots, Q_S) \mapsto (f_{\text{out},i}(ihvin_0, \ldots, ihvin_{L-1}, Q_{S-L+1}, \ldots, Q_S))_{i=0}^{L-1}.$$

The finalization of the compression functions of MD4, MD5 and SHA-1 is as follows. First the inverse initialization permutation, namely $((f_{\text{in},i})_{i=-L+1}^{0})^{-1}$, is applied

to $(Q_{S-L+1}, \ldots, Q_S)$ resulting in $(\widehat{Q}_{S-L+1}, \ldots, \widehat{Q}_S)$. The value of IHV_{out} is computed as the word-wise sum of the tuples $(\widehat{Q}_{S-L+1}, \ldots, \widehat{Q}_S)$ and $(ihvin_0, \ldots, ihvin_{L-1})$, thus $ihvout_i = \widehat{Q}_{S-L+1+i} + ihvin_i$ for $i = 0, \ldots, L-1$.

5.3.3 Message expansion

In each of the S steps $t = 0, \ldots, S-1$ a single word W_t is used that is derived from the message word tuple M:

$$W_t = f_{\text{msgexp},t}(m_0, \ldots, m_{K-1}) \in \mathbb{Z}_{2^N}.$$

The functions $f_{\text{msgexp},t}$ are arbitrary functions

$$f_{\text{msgexp},t} : (\mathbb{Z}_{2^N})^K \to \mathbb{Z}_{2^N}, \quad t \in \{0, \ldots, S-1\}$$

under the restriction that for $k = 0, K, \ldots, S-K$ the following function is bijective:

$$f_{\text{msgexpblock},k}(M) = (f_{\text{msgexp},k}(M), f_{\text{msgexp},k+1}(M), \ldots, f_{\text{msgexp},k+K-1}(M)).$$

MD4 and MD5 divide their S steps into S/K rounds, thus $48/16 = 3$ and $64/16 = 4$ rounds respectively. The first round uses the message words in order: $W_0 = m_0, \ldots, W_{15} = m_{15}$. The remaining rounds $r = 1, \ldots, \frac{S}{16} - 1$ apply a fixed permutation on the message words $(m_0, \ldots, m_{15})$ to obtain $(W_{r \cdot K}, \ldots, W_{r \cdot K 15})$. SHA-1 also uses $W_0 = m_0, \ldots, W_{15} = m_{15}$, however it computes the remaining words with the following linear relation:

$$W_i = RL(W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}, 1), \quad \text{for } i = 16, \ldots, 79.$$

This linear relation can be used backwards:

$$W_{i-16} = RR(W_i, 1) \oplus W_{i-3} \oplus W_{i-8} \oplus W_{i-14}, \quad \text{for } i = 16, \ldots, 79.$$

It follows that any 16 consecutive $W_i, \ldots, W_{i+15}$ fully determine $W_0, \ldots, W_{79}$ and in particular $W_0, \ldots, W_{15}$. This implies that for $k = 0, 16, 32, 48, 64$ the mapping from $m_0, \ldots, m_{15}$ to $W_{k+0}, \ldots, W_{k+15}$ is bijective.

5.3.4 Step function

In each of the S steps $t = 0, \ldots, S-1$, Compress computes a single boolean function over the last $L-1$ state words $Q_{t-L+2}, \ldots, Q_t$:

$$F_t = f_{\text{bool},t}(Q_{t-L+2}, \ldots, Q_t) \in \mathbb{Z}_{2^N}, \quad f_{\text{bool},t} \in \mathcal{F}_{\text{boolrot}}.$$

Each step t computes $V \in \mathbb{N}^+$ (V is fixed and independent of the step t) intermediate variables $T_{t,i}$ where $i \in \{1, \ldots, V\}$ starting with $T_{t,0} = 0$:

$$T_{t,i} \quad = \quad f_{\text{temp},t,i}(Q_{t-L+1}, \ldots, Q_t, F_t, W_t, T_{t,i-1}), \quad f_{\text{temp},t,i} \in \mathcal{F}_{\text{sumrot}}.$$

The new working state word Q_{t+1} is set to the final intermediate variable $T_{t,V}$. Furthermore, the functions $f_{\text{temp},t,i}$ have the following restrictions:

- In each function $f_{\text{temp},t,i}$, the selective sum coefficient of $T_{t,i-1}$ is non-zero.
- Over all selective sums in $(f_{\text{temp},t,i})_{i=1}^{V}$, the variable Q_{t-L+1} is selected exactly once. More formally, for $i = 1, \ldots, V$, let $c_i \in \{-1, 0, 1\}$ be the selective sum coefficient of Q_{t-L+1} in $f_{\text{temp},t,i}$ (see Section 5.2.3) then $\sum_{i=1}^{V} |c_i|$ must be 1.
- For $i = 1, \ldots, V$, let c_i' be the selective sum coefficient of W_t in $f_{\text{temp},t,i}$ then $\sum_{i=1}^{V} |c_i'|$ must be 1.
- For $i = 1, \ldots, V$, let c_i'' be the selective sum coefficient of F_t in $f_{\text{temp},t,i}$ then $\sum_{i=1}^{V} |c_i''|$ must be 1.

For MD4, $V = 2$ and functions $f_{\text{bool},t}$, $f_{\text{temp},t,1}$ and $f_{\text{temp},t,2}$ are defined as:

$$\begin{aligned} f_{\text{temp},t,1}(\ldots) &= Q_{t-3} + F_t + W_t + T_{t,0} + AC_t; \\ f_{\text{temp},t,2}(\ldots) &= RL(T_{t,1}, RC_t); \\ f_{\text{bool},t}(Q_{t-2}, Q_{t-1}, Q_t) &= \\ &\begin{cases} (Q_t \wedge Q_{t-1}) \oplus (\overline{Q_t} \wedge Q_{t-2}) & \text{for } 0 \le t < 16, \\ (Q_t \wedge Q_{t-1}) \vee (Q_t \wedge Q_{t-2}) \vee (Q_{t-1} \wedge Q_{t-2}) & \text{for } 16 \le t < 32, \\ Q_t \oplus Q_{t-1} \oplus Q_{t-2} & \text{for } 32 \le t < 48, \end{cases} \end{aligned}$$

where $AC_t \in \mathbb{Z}_{2^{32}}$ and $RC_t \in \{0, \ldots, 31\}$ are constants. For MD5, also $V = 2$ and the functions $f_{\text{bool},t}$, $f_{\text{temp},t,1}$ and $f_{\text{temp},t,2}$ are defined as:

$$\begin{aligned} f_{\text{temp},t,1}(\ldots) &= Q_{t-3} + F_t + W_t + T_{t,0} + AC_t; \\ f_{\text{temp},t,2}(\ldots) &= Q_t + RL(T_{t,1}, RC_t); \\ f_{\text{bool},t}(Q_{t-2}, Q_{t-1}, Q_t) &= \\ &\begin{cases} (Q_t \wedge Q_{t-1}) \oplus (\overline{Q_t} \wedge Q_{t-2}) & \text{for } 0 \le t < 16, \\ (Q_{t-2} \wedge Q_t) \oplus (\overline{Q_{t-2}} \wedge Q_{t-1}) & \text{for } 16 \le t < 32, \\ Q_t \oplus Q_{t-1} \oplus Q_{t-2} & \text{for } 32 \le t < 48, \\ Q_{t-1} \oplus (Q_t \vee \overline{Q_{t-2}}) & \text{for } 48 \le t < 64, \end{cases} \end{aligned}$$

where $AC_t \in \mathbb{Z}_{2^{32}}$ and $RC_t \in \{0, \ldots, 31\}$ are constants. For SHA-1, $V = 1$ and the functions $f_{\text{bool},t}$ and $f_{\text{temp},t,1}$ are defined as:

$$\begin{aligned} f_{\text{temp},t,1}(\ldots) &= RL(Q_{t-4}, 30) + RL(Q_t, 5) + F_t + W_t + T_{t,0} + AC_t; \\ f_{\text{bool},t}(\ldots) &= f_{\text{sha1},t}(Q_{t-1}, RL(Q_{t-2}, 30), RL(Q_{t-3}, 30)); \\ f_{\text{sha1},t}(X, Y, Z) &= \\ &\begin{cases} F(X,Y,Z) = Z \oplus (X \wedge (Y \oplus Z)) & \text{for } 0 \le t < 20, \\ G(X,Y,Z) = X \oplus Y \oplus Z & \text{for } 20 \le t < 40, \\ H(X,Y,Z) = (X \wedge Y) \vee (Z \wedge (X \vee Y)) & \text{for } 40 \le t < 60, \\ I(X,Y,Z) = X \oplus Y \oplus Z & \text{for } 60 \le t < 80, \end{cases} \end{aligned}$$

where $AC_t \in \mathbb{Z}_{2^{32}}$ is a constant.

5.4 Properties of the step functions

For each $t = 0, \ldots, S-1$, the mapping

$$f_t : (Q_{t-L+1}, \ldots, Q_t, F_t, W_t) \mapsto Q_{t+1}$$

as defined by the sequence $(f_{\text{temp},t,i})_{i=1}^{V}$ has three properties that are crucial to our differential path construction algorithm and are design criteria to thwart certain attacks. The first property of f_t is that the output Q_{t+1} can take on all possible values in $\mathbb{Z}_{2^N}$ by varying only $Q_{t-L+1} \in \mathbb{Z}_{2^N}$ and fixing all other input values. An important implication of this property is that Q_{t-L+1} can be uniquely determined given the output value Q_{t+1} and all other input values. The other two properties are similar to the first with respect to F_t and W_t instead of Q_{t-L+1}. These three properties are treated in detail in Section 5.4.1, 5.4.2 and 5.4.3.

5.4.1 Full dependency on Q_{t-L+1}

Theorem 5.1 (Full dependency on Q_{t-L+1})**.** *Given values of the variables Q_{t-L+2}, ..., Q_t, F_t, W_t the following mapping is bijective:*

$$f_{t,Q_{t-L+1}} : Q_{t-L+1} \mapsto Q_{t+1} = f_t(Q_{t-L+1}, \ldots, Q_t, F_t, W_t).$$

Furthermore, there exists a sequence of functions $(f_{invQ,t,i})_{i=1}^{2V+1} \in \mathcal{F}_{sumrot}^{2V+1}$ that computes the inverse of $f_{t,Q_{t-L+1}}$ given values of $Q_{t-L+2}, \ldots, Q_t$, F_t, W_t and the value of Q_{t+1}:

$$\begin{aligned} T_{invQ,t,0} &= 0; \\ T_{invQ,t,i} &= f_{invQ,t,i}(Q_{t-L+2}, \ldots, Q_t, F_t, W_t, Q_{t+1}, T_{invQ,t,0}, \ldots, T_{invQ,t,i-1}); \\ Q_{t-L+1} &= T_{invQ,t,2V+1}. \end{aligned}$$

Proof. We prove the theorem by providing a construction of a sequence of functions $(f_{\text{invQ},t,i})_{i=1}^{2V+1}$ that computes the inverse of $f_{t,Q_{t-L+1}}$. In our construction not all $f_{\text{invQ},t,i}$ may be needed in which case we define those as the zero-function. Using the second restriction in Section 5.3.4, let $j \in \{1, \ldots, V\}$ be the unique index such that the selection coefficient[15] of Q_{t-L+1} in $f_{\text{temp},t,j}$ is non-zero. Given values of $Q_{t-L+2}, \ldots, Q_t, F_t$ and W_t, we can compute the values of $T_{t,0}, \ldots, T_{t,j-1}$, since these do not depend on the value of Q_{t-L+1}. We define $f_{\text{invQ},t,1}, \ldots, f_{\text{invQ},t,j-1}$ to compute the values of $T_{t,0}, \ldots, T_{t,j-1}$:

$$f_{\text{invQ},t,i}(\ldots) = f_{\text{temp},t,i}(0, Q_{t-L+2}, \ldots, Q_t, F_t, W_t, T_{\text{invQ},t,i-1}),\ i \in \{1, \ldots, j-1\}.$$

Thus $T_{\text{invQ},t,j-1} = T_{t,j-1}$ after these steps. The functions $(f_{\text{invQ},t,i})_{i=j}^{2j-2}$ are not needed and defined as the zero function.

15. See Section 5.2.3.

For $i = j+1, \ldots, V$ the values of all inputs except T_{i-1} and Q_{t-L+1} of $f_{\text{temp},t,i}$ are known, but Q_{t-L+1} can be ignored as these functions do not depend on the value of Q_{t-L+1}. Using Observation 5.1, the following mappings are bijective:

$$g_i : T_{i-1} \mapsto f_{\text{temp},t,i}(0, Q_{t-L+2}, \ldots, Q_t, F_t, W_t, T_{i-1}), \quad i \in \{j+1, \ldots, V\}.$$

By inverting $g_V, \ldots, g_{j+1}$ we can compute the values of $T_{t,V-1}, \ldots, T_{t,j}$ in that order. First let $f_{\text{invQ},t,2j-1}(\ldots) = T_{t,V} = Q_{t+1}$. For $i = V, \ldots, j+1$, let $k = j+V-i$ and let c_i and r_i be the select coefficient and rotation constant of the variable $T_{t,i-1}$ in $f_{\text{temp},t,i}$ then we define $f_{\text{invQ},t,2(j+V-i)}$ and $f_{\text{invQ},t,2(j+V-i)+1}$ as follows:

$$\begin{aligned} f_{\text{invQ},t,2k}(\ldots) &= c_i \cdot T_{\text{invQ},t,2k-1} \\ &\quad -c_i \cdot f_{\text{temp},t,i}(0, Q_{t-L+2}, \ldots, Q_t, F_t, W_t, 0); \\ f_{\text{invQ},t,2k+1}(\ldots) &= RL(T_{\text{invQ},t,2k}, N - r_i). \end{aligned}$$

Similar to Observation 5.1, the first and second function compute the $c_j \cdot (A - C_j)$ part and the RR part, respectively, of the inverse of g_i. Given the values of $Q_{t-L+2}, \ldots, Q_t$, F_t, W_t and $T_{\text{invQ},t,2k-1} = T_{t,i}$, this results in $T_{\text{invQ},t,2k+1} = T_{t,i-1}$. Thus for $i = j+1$ this results in $T_{\text{invQ},t,2V-2} = T_j$.

Again using Observation 5.1, the following mapping is bijective:

$$g_j : Q_{t-L+1} \mapsto f_{\text{temp},t,j}(Q_{t-L+1}, \ldots, Q_t, F_t, W_t, T_{j-1}).$$

Thus it follows that $f_{t,Q_{t-L+1}}$ is bijective as

$$f_{t,Q_{t-L+1}}(Q_{t-L+1}) = g_V(\cdots(g_{j+1}(g_j(Q_{t-L+1})))\cdots).$$

Let c_Q and r_Q be the select and rotation constant of the variable Q_{t-L+1} in $f_{\text{temp},t,j}$ then we define $f_{\text{invQ},t,2V}$ and $f_{\text{invQ},t,2V+1}$ as follows:

$$\begin{aligned} f_{\text{invQ},t,2V}(\ldots) &= c_Q \cdot T_{\text{invQ},t,2V-1} \\ &\quad -c_Q \cdot f_{\text{temp},t,j}(0, Q_{t-L+2}, \ldots, Q_t, F_t, W_t, T_{\text{invQ},t,j-1}); \\ f_{\text{invQ},t,2V+1}(\ldots) &= RL(T_{\text{invQ},t,2V}, N - r_Q). \end{aligned}$$

Given the values of $Q_{t-L+2}, \ldots, Q_t$, F_t, W_t, $T_{\text{invQ},t,2V-1} = T_{t,j}$ and $T_{\text{invQ},t,j-1} = T_{t,j-1}$, this results in $T_{\text{invQ},t,2V+1} = Q_{t-L+1}$. □

Using the above theorem it follows that the following mapping is bijective and its inverse is easily computable using only additions in $\mathbb{Z}_{2^N}$ and bitwise rotations for all values of $Q_{t-L+2}, \ldots, Q_t, F_t, W_t \in \mathbb{Z}_{2^N}$:

$$f_{t,Q_{t-L+1}} : Q_{t-L+1} \mapsto f_t(Q_{t-L+1}, \ldots, Q_t, F_t, W_t).$$

5.4.2 Full dependency on F_t

Theorem 5.2 (Full dependency on F_t)**.** *Given values of the variables $Q_{t-L+1}, \ldots, Q_t$ and W_t the following mapping is bijective:*

$$f_{t,F_t} : F_t \mapsto Q_{t+1} = f_t(Q_{t-L+1}, \ldots, Q_t, F_t, W_t).$$

Furthermore, there exists a sequence of functions $(f_{invF,t,i})_{i=1}^{2V+1} \in \mathcal{F}_{sumrot}^{2V+1}$ that computes the inverse of f_{t,F_t} given values of $Q_{t-L+1}, \ldots, Q_t$, W_t and the value of Q_{t+1}:

$$\begin{aligned} T_{invF,t,0} &= 0; \\ T_{invF,t,i} &= f_{invF,t,i}(Q_{t-L+1}, \ldots, Q_t, W_t, Q_{t+1}, T_{invF,t,0}, \ldots, T_{invF,t,i-1}); \\ F_t &= T_{invF,t,2V+1}. \end{aligned}$$

The proof is analogous to the proof of Theorem 5.1.

5.4.3 Full dependency on W_t

Theorem 5.3 (Full dependency on W_t)**.** *Given values of the variables $Q_{t-L+1}, \ldots, Q_t$ and F_t the following mapping is bijective:*

$$f_{t,W_t} : W_t \mapsto Q_{t+1} = f_t(Q_{t-L+1}, \ldots, Q_t, F_t, W_t).$$

Furthermore, there exists a sequence of functions $(f_{invW,t,i})_{i=1}^{2V+1} \in \mathcal{F}_{sumrot}^{2V+1}$ that computes the inverse of f_{t,W_t} given values of $Q_{t-L+1}, \ldots, Q_t$, F_t and the value of Q_{t+1}:

$$\begin{aligned} T_{invW,t,0} &= 0; \\ T_{invW,t,i} &= f_{invW,t,i}(Q_{t-L+1}, \ldots, Q_t, W_t, Q_{t+1}, T_{invW,t,0}, \ldots, T_{invW,t,i-1}); \\ W_t &= T_{invW,t,2V+1}. \end{aligned}$$

The proof is analogous to the proof of Theorem 5.1.

5.4.4 Properties as step function design criteria

The above three properties can be seen as step function design criteria, since without these properties the compression function would be more vulnerable to attacks. Below we outline how the security of the compression function may be compromised if either one of these three properties does not hold. These design criteria may not be often pointed out, since the topic of step function design (criteria) is not treated very well in the literature. Nevertheless it should come as no surprise that these criteria hold for all members of the MD4 hash function family.

Full dependency on Q_{t-L+1}. From a hash function design perspective it is important that the mapping $f_{t,Q_{t-L+1}}$ is bijective. Suppose this mapping is not bijective, then there are values of $Q_{t-L+2}, \ldots, Q_t, F_t, W_t$ and Q_{t+1} such that inverting $f_{t,Q_{t-L+1}}$ results in multiple pre-images:

$$|\{Q_{t-L+1} \in \mathbb{Z}_{2^N} \mid f_t(Q_{t-L+1}, \ldots, Q_t, F_t, W_t) = Q_{t+1}\}| > 1.$$

Barring further complications, this leads to a pre-image attack of Compress which is about 2^{K-L} faster than the brute-force pre-image attack. The pre-image attack finds a message M given values $\widehat{IHV}_{\text{in}}$, $\widehat{IHV}_{\text{out}}$ such that $\widehat{IHV}_{\text{out}} = \text{Compress}(\widehat{IHV}_{\text{in}}, M)$ with high probability using about $2^{(N \cdot L)-(K-L)}$ calls to Compress. Below we sketch this attack.

Since finalization is bijective and based on $\mathcal{F}_{\text{sumrot}}$, it is easy to find values of $Q_{S-L+1}, \ldots, Q_S$ such that

$$\widehat{ihvout}_i = f_{\text{out},i}(\widehat{ihvin}_0, \ldots, \widehat{ihvin}_{L-1}, Q_{S-L+1}, \ldots, Q_S), \quad i \in \{0, \ldots, L-1\}.$$

Furthermore, we can expect that an attacker has a sufficiently fast way of choosing the last $K-L$ words $\widehat{W}_{S-K+L}, \ldots, \widehat{W}_{S-1}$ such that inverting the last $K-L$ steps leads to at least 2^{K-L} different pre-images $(Q_{S-K+1}, \ldots, Q_{S-K+l})$.[16]

The third property in Section 5.4.3 implies that in each step t all values of Q_{t+1} can be obtained by varying W_t. Therefore we can safely assume that on average a message M has probability $2^{(K-L)-(N \cdot L)}$ of resulting in one of the 2^{K-L} found pre-images $(Q_{S-K+1}, \ldots, Q_{S-K+l})$ during the computation of $\text{Compress}(\widehat{IHV}_{\text{in}}, M)$. By choosing only messages M such that

$$(W_{S-K+L}, \ldots, W_{S-1}) = (\widehat{W}_{S-K+L}, \ldots, \widehat{W}_{S-1}),$$

then with probability $2^{(K-L)-(N \cdot L)}$ the computation of $\text{Compress}(\widehat{IHV}_{\text{in}}, M)$ results in one of the 2^{K-L} found pre-images and thereby also results in $IHV_{\text{out}} = \widehat{IHV}_{\text{out}}$. There are in total $2^{N \cdot L}$ such messages M among which we can expect to find 2^{K-L} solutions. It follows that finding a single pre-image of Compress takes about $2^{(N \cdot L)-(K-L)}$ attempts.

Full dependency on F_t. From a hash function designers perspective, the boolean function is used as a security measure to prevent the hash function to be described as a linear system over $\mathbb{Z}_{2^N}$. If not all possible values of Q_{t+1} can be obtained by varying F_t then F_t does not fully contribute to Q_{t+1} and the security measure is not used to its full potential. An obvious attack technique is to replace the boolean function with a linear function that approximates the boolean function. In general it is easier to linearly approximate the boolean function as the influence of F_t on the output Q_{t+1} gets smaller.

16. Actually, instead of 2^{K-L} pre-images, any set of $2^Z > 2$ pre-images qualifies and here 'sufficiently fast' simply requires that finding the $\widehat{W}_i$ and the 2^Z pre-images is faster than performing $2^{(N \cdot L)-Z}$ calls to Compress.

Full dependency on W_t. Designing a hash function without having the message words W_t fully contribute in each step t implies that IHV_{out} depends in a simpler manner on the message than possible. This increases the likelihood of significant weaknesses.

In extreme cases, say only the first bit $W_t[0]$ actually contributes, this reduces the complexity of a brute force (second) pre-image attack to $\min(2^S, 2^{L \cdot N})$ calls to Compress. $S = 1/2(L \cdot N)$ for common designs like MD5 and SHA-1, thus this attack is significantly faster than the desired $2^{L \cdot N}$. In each step there are only two possible outcomes for Q_{t+1}, as in each step $t \in \{0, \ldots, S-1\}$ only a single bit of the message is used. Over S steps this means that given IHV_{in} there are at most 2^S possible $IHV_{\text{out}} = \text{Compress}(IHV_{\text{in}}, M)$ by varying M instead of the expected $2^{L \cdot N}$ possible IHV_{out}. Hence, a brute force (second) pre-image attack has a success probability of at most $\max(2^{-S}, 2^{-L \cdot N})$ for each try of M.

Rather than brute-forcing it, for given IHV_{in} and M in each step $t \in \{0, \ldots, S-1\}$ there are $2^{N-1} - 1$ other $W'_t \neq W_t$ that result in the same Q_{t+1}. Let $\mathcal{W}_{\text{sec},t}$ be the set of all W'_t that result in the same Q_{t+1} in step t. Finding a second pre-image $M' \neq M$ is thus reduced to the problem of finding $M' \neq M$ such that its message expansion $(W'_t)_{t=0}^{S-1}$ is in the set $\prod_{t=0}^{S-1} \mathcal{W}_{\text{sec},t}$. In the case of the message expansion of either MD5 or SHA-1 this would be a simple if not trivial exercise.

5.4.5 Properties in differential cryptanalysis

The reason we treat these three properties in detail is that we need them in our differential cryptanalysis. As outlined above, these three properties can be seen as step function design criteria and as such are inherent to the MD4 hash function family. Below we outline the use of these properties in our differential cryptanalysis.

Full dependency on Q_{t-L+1}. In our algorithm to construct full differential paths, we need to be able to extend partial differential paths over steps $t = A, \ldots, B$ with the preceding step $t = A - 1$. This can easily be done due to the existence of the functions $(f_{\text{invQ},t,i})_{i=1}^{2V+1} \in \mathcal{F}_{\text{sumrot}}^{2V+1}$ that compute the inverse of $f_{t,Q_{t-L+1}}$.

Full dependency on F_t. In our algorithm to construct full differential paths, we construct two partial differential paths independently, the first over steps $t = 0, \ldots, A$ and the second over steps $t = A + L + 1, \ldots, S - 1$. This implies that the difference in any Q_i is either given by the first or the second partial differential path. Due to the existence of the functions $(f_{\text{invF},t,i})_{i=1}^{2V+1} \in \mathcal{F}_{\text{sumrot}}^{2V+1}$, one can determine target differences for the boolean function outcomes $F_{A+1}, \ldots, F_{A+L}$. A full valid differential path is obtained whenever these boolean function outcome differences and the Q_i differences can be simultaneously fulfilled. Construction of a full differential path can thus be reduced to finding two partial differential paths for which these target differences in $F_{A+1}, \ldots, F_{A+L}$ can actually be obtained given the Q_i differences.

Full dependency on W_t. Although this property is not directly used in our algorithm to construct full differential paths, it is important in collision finding algorithms that search for messages M that fulfill a given differential path. This property allows one to choose working state words $Q_{t-L+1}, \ldots, Q_{t+1}$ which fulfill sufficient conditions for a given differential path and then compute the corresponding W_t.

5.5 Differential paths

In this section we formalize the concept of a differential path as a precise description of how differences propagate through the S steps of Compress $\in \mathcal{F}_{\text{md4cf}}$. These differences are taken between instances Compress(IHV_{in}, M) and Compress(IHV'_{in}, M'). For each variable $X \in \mathbb{Z}_{2^N}$ in the computation of Compress(IHV_{in}, M) we denote the respective variable in the computation of Compress(IHV'_{in}, M') as X'. Their difference is denoted as $\delta X = X' - X$ and the bitwise integer difference as $\Delta X = (X'[i] - X[i])_{i=0}^{N-1}$ which is a BSDR of δX. In the following differential cryptanalysis we refer to such δX and ΔX as variables themselves without directly implying values for X and X'.

First we analyze the only two non-trivial operations of Compress with respect to differences in $\mathbb{Z}_{2^N}$: the bitwise rotations $RL(X, n)$ in all $((f_{\text{temp},t,i})_{i=1}^{V})_{t=0}^{S-1}$ and the boolean functions $(f_{\text{bool},t})_{t=0}^{S-1}$.

5.5.1 Rotation of word differences

To determine the difference $RL(X', n) - RL(X, n)$ given δX, we distinguish two situations. First, suppose besides δX also ΔX is known. In this case

$$\begin{aligned}
\Delta RL(X, n) &= (RL(X', n)[i] - RL(X, n)[i])_{i=0}^{N-1} \\
&= (X'[(i+n) \bmod N] - X[(i+n) \bmod N])_{i=0}^{N-1} \\
&= (\Delta X[(i+n) \bmod N])_{i=0}^{N-1} \\
&= RL(\Delta X, n).
\end{aligned}$$

The remaining case is where ΔX is undetermined. In this case, up to four different $\delta Y = RL(X', n) - RL(X, n)$ are possible. Here, we determine the possible δY and their probabilities $\Pr_X[\delta Y = RL(X + \delta X, n) - RL(X, n)]$. We define $dRL(Z, n)$ for $Z \in \mathbb{Z}_{2^N}, n \in \{0, \ldots, N-1\}$ as the set of possible differences $RL(X+Z, n) - RL(X, n)$ after rotation with non-zero probability as in the following lemma.

Lemma 5.4 (Rotation of differences)**.** *Given $\delta X \in \mathbb{Z}_{2^N}$ and rotation constant $n \in \{0, \ldots, N-1\}$ then for uniformly chosen random $X \in \mathbb{Z}_{2^N}$, there are at most four possible differences $\delta Y = RL(X + \delta X, n) - RL(X, n)$ after rotation. Let $Z_{low} = \sum_{i=0}^{N-n-1} 2^i \cdot \delta X[i] \in \mathbb{Z}$ and $Z_{high} = \sum_{i=N-n}^{N-1} 2^i \cdot \delta X[i] \in \mathbb{Z}$. Then δY as above may*

attain one of four values as given below along with the corresponding probabilities:

δY		$\Pr_X[RL(X+Z,n) - RL(X,n) = \delta Y]$
$D_1 =$	$RL(\delta X, n)$	$2^{-2N+n} \cdot (2^{N-n} - Z_{low}) \cdot (2^N - Z_{high})$
$D_2 =$	$RL(\delta X, n) - 2^n$	$2^{-2N+n} \cdot (2^{N-n} - Z_{low}) \cdot Z_{high}$
$D_3 =$	$RL(\delta X, n) + 1$	$2^{-2N+n} \cdot Z_{low} \cdot (2^N - Z_{high} - 2^{N-n})$
$D_4 =$	$RL(\delta X, n) - 2^n + 1$	$2^{-2N+n} \cdot Z_{low} \cdot (Z_{high} + 2^{N-n})$

Depending on the value of δX, some of these probabilities may be zero thus reducing the number of possible outcomes, thus:

$$dRL(\delta X, n) = \left\{ D_i \;\middle|\; \Pr_X[RL(X+Z,n) - RL(X,n) = D_i] \neq 0, i \in \{1,2,3,4\} \right\}.$$

This lemma follows directly from Corollary 4.12 of Magnus Daum's Ph.D. thesis [Dau05]. We provide a different proof using BSDRs:

Proof. As above, any BSDR $(k_i)_{i=0}^{N-1}$ of δX gives rise to a candidate δY given by the BSDR

$$RL((k_i), n) = (k_{N-n-1}, \ldots, k_0, k_{N-1}, \ldots, k_{N-n}).$$

Two BSDRs $(k_i)_{i=0}^{N-1}$ and $(l_i)_{i=0}^{N-1}$ of δX result in the same δY if and only if

$$\sum_{i=0}^{N-n-1} 2^i k_i = \sum_{i=0}^{N-n-1} 2^i l_i \quad \text{and} \quad \sum_{i=N-n}^{N-1} 2^i k_i = \sum_{i=N-n}^{N-1} 2^i l_i. \tag{5.1}$$

To analyze this property, we define a *partition* as a pair $(\alpha, \beta) \in \mathbb{Z}^2$ such that $\alpha + \beta = \delta X \bmod 2^N$, $|\alpha| < 2^{N-n}$, $|\beta| < 2^N$ and $2^{N-n} | \beta$. For any partition (α, β), values $k_i \in \{-1, 0, 1\}$ for $0 \leq i < N$ can be found such that

$$\alpha = \sum_{i=0}^{N-n-1} 2^i k_i \quad \text{and} \quad \beta = \sum_{i=N-n}^{N-1} 2^i k_i. \tag{5.2}$$

In particular, there is at least one such $(k_i)_{i=0}^{N-1}$ which can be constructed by looking at the binary representation of $|\alpha|$ and $|\beta|$ and applying the sign of α and β on the respective bits. With $\alpha + \beta = \delta X \bmod 2^N$ it follows that any such $(k_i)_{i=0}^{N-1}$ is a BSDR of δX. Conversely, with Equation 5.2 any BSDR (k_i) of δX defines a partition, which we denote $(k_i) \equiv (\alpha, \beta)$. Moreover, any BSDR (k_i) of δX that leads to the same partition gives rise to the same candidate δY due to Equation 5.1. The rotation of a partition (α, β) is defined as

$$RL((\alpha, \beta), n) = 2^n \alpha + 2^{n-N} \beta \in \mathbb{Z}_{2^N}.$$

If $(k_i) \equiv (\alpha, \beta)$, this matches $RL((k_i), n)$.

The partition constraints imply there are at most two possible values for α such that $\alpha + \beta = \delta X$, namely Z_{low} and when $Z_{\text{low}} \neq 0$ also $Z_{\text{low}} - 2^{N-n}$. For each value of

α there are also at most two possible values for β namely $(\delta X - \alpha \bmod 2^N)$ and when this is non-zero also $(\delta X - \alpha \bmod 2^N) - 2^N$. Note that $Z_{\text{high}} = (\delta X - Z_{\text{low}} \bmod 2^N)$ and let $Z'_{\text{high}} = (\delta X - Z_{\text{low}} + 2^{N-n} \bmod 2^N)$. Together this gives rise to at most 4 partitions:

$$\begin{array}{lll}
\text{p1.} & (\alpha,\beta) = (Z_{\text{low}}, Z_{\text{high}}); & \\
\text{p2.} & (\alpha,\beta) = (Z_{\text{low}}, Z_{\text{high}} - 2^N), & \text{if } Z_{\text{high}} \neq 0; \\
\text{p3.} & (\alpha,\beta) = (Z_{\text{low}} - 2^{N-n}, Z'_{\text{high}}), & \text{if } Z_{\text{low}} \neq 0; \\
\text{p4.} & (\alpha,\beta) = (Z_{\text{low}} - 2^{N-n}, Z'_{\text{high}} - 2^N), & \text{if } Z_{\text{low}} \neq 0 \wedge Z'_{\text{high}} \neq 0.
\end{array}$$

Note that $RL((Z_{\text{low}}, Z_{\text{high}}), n) = RL(\delta X, n)$.

To find the probability of each δY, we define

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

and show how $p_{(\alpha,\beta)}$ can be calculated. For each of the four possibilities this is done by counting the number of N-bit words X such that the BSDR defined by $k_i = (X+\delta X)[i] - X[i]$ satisfies $(k_i) \equiv (\alpha,\beta)$. This can be expressed in two equations:

$$\alpha = \sum_{i=0}^{N-n-1} ((X + \alpha + \beta)[i] - X[i])2^i, \quad \beta = \sum_{i=N-n}^{N-1} ((X + \alpha + \beta)[i] - X[i])2^i.$$

Since $2^{N-n} | \beta$, the β term can be ignored in the first equation. The first equation then implies that adding α to X only affects the low-order $N-n$ bits and thus α can be ignored in the second equation:

$$\alpha = \sum_{i=0}^{N-n-1} ((X + \alpha)[i] - X[i])2^i, \quad \beta = \sum_{i=N-n}^{N-1} ((X + \beta)[i] - X[i])2^i.$$

These two equations hold if and only if:

$$0 \leq \alpha + \sum_{i=0}^{N-n-1} X[i]2^i < 2^{N-n}, \quad 0 \leq \beta + \sum_{i=N-n}^{N-1} X[i]2^i < 2^N.$$

Considering the $(N-n)$ low-order bits, we determine the number r of X_{low}-values with $0 \leq X_{\text{low}} < 2^{N-n}$ such that $0 \leq \alpha + X_{\text{low}} < 2^{N-n}$: if $\alpha < 0$ then $r = 2^{N-n} + \alpha$ and if $\alpha \geq 0$ then $r = 2^{N-n} - \alpha$. Hence only $r = 2^{N-n} - |\alpha|$ out of 2^{N-n} possible X_{low}-values satisfy $0 \leq \alpha + X_{\text{low}} < 2^{N-n}$. The same argument can be used for the n high-order bits to determine that there are exactly $2^n - |\beta|2^{n-N}$ number of values for $X_{\text{high}} = 2^{n-N} \cdot \sum_{i=N-n}^{N-1} X[i]2^i$ such that $0 \leq 2^{n-N} \cdot \beta + X_{\text{high}} < 2^n$. Hence, we conclude

$$p_{(\alpha,\beta)} = \frac{2^{N-n} - |\alpha|}{2^{N-n}} \cdot \frac{2^n - |\beta|2^{n-N}}{2^n} = \frac{2^{N-n} - |\alpha|}{2^{N-n}} \cdot \frac{2^N - |\beta|}{2^N}.$$

Assume that $Z'_{\text{high}} \neq 0$, then $Z'_{\text{high}} = Z_{\text{high}} + 2^{N-n}$. One can now immediately verify that D_1, D_2, D_3 and D_4 match the rotations of the 4 partitions p1, p2, p3 and p4, respectively. Furthermore, the partition probabilities match those given in the lemma, and whenever a partition is excluded, its probability is zero.

When $Z'_{\text{high}} = 0$, then $Z_{\text{high}} = 2^N - 2^{N-n}$. Now as above, D_1 and D_2 match the rotations of partitions p1 and p2, respectively. We can show that D_4 matches the rotation of partition p3:

$$\begin{aligned} RL((Z_{\text{low}} - 2^{N-n}, Z'_{\text{high}}), n) &= 2^n \cdot (Z_{\text{low}} - 2^{N-n}) + 2^{n-N} \cdot (0) \\ &= 2^n \cdot Z_{\text{low}} + 2^{n-N} \cdot (Z_{\text{high}} - 2^N + 2^{N-n}) \\ &= 2^n \cdot Z_{\text{low}} + 2^{n-N} \cdot Z_{\text{high}} - 2^n + 1 \\ &= RL(\delta X, n) - 2^n + 1. \end{aligned}$$

And also the probability of partition p3 matches the probability of D_4 given in the lemma:

$$\begin{aligned} p_{\text{p3}} &= 2^{n-N} \cdot (2^{N-n} - (Z_{\text{low}} - 2^{N-n})) \cdot 2^{-N} \cdot (2^N - Z'_{\text{high}}) \\ &= 2^{-2N+n} \cdot Z_{\text{low}} \cdot (2^N) \\ &= 2^{-2N+n} \cdot Z_{\text{low}} \cdot (Z_{\text{high}} + 2^{N-N}). \end{aligned}$$

The remaining partition p4 is excluded and D_3 is also excluded as its probability is zero. □

5.5.2 Boolean function differences

For any step t, let $f_{\text{bool},t} \in \mathcal{F}_{\text{boolrot}}$ be given as:

$$f_{\text{bool},t}(Q_{t-L+2}, \ldots, Q_t) = \sum_{i=0}^{N-1} 2^i \cdot g(RL(Q_{t-L+2}, r_{t-L+2})[i], \ldots, RL(Q_t, r_t)[i]),$$

where $g : \{0,1\}^{L-1} \to \{0,1\}$ is an arbitrary boolean function. Since each bit of the output depends only on a single bit of each of the variables $Q_{t-L+2}, \ldots, Q_t$, for a precise differential description we need the bitwise integer difference ΔQ_i for these inputs and the output difference is determined bitwise as ΔF_t.

For $j \in \{0, \ldots, L-2\}$, define $X_j = RL(Q_{t-L+2+j}, r_{t-L+2+j})$ and thus $\Delta X_j = RL(\Delta Q_{t-L+2+j}, r_{t-L+2+j})$ to simplify notation a bit, then the difference ΔF_t can be determined using

$$\Delta F_t[i] = g(X'_0[i], \ldots, X'_{L-2}[i]) - g(X_0[i], \ldots, X_{L-2}[i]).$$

Let $i \in \{0, \ldots, N-1\}$, we consider the set U_i of possible input values:

$$U_i = \{((X'_0[i], \ldots, X'_{L-2}[i]), (X_0[i], \ldots, X_{L-2}[i])) \mid X'_j[i] = X_j[i] + \Delta X_j[i]\}.$$

This set U_i may be further restricted to $\widetilde{U}_i$ by auxiliary conditions imposed over the bits $X_j[i]$ for which $\Delta X_j[i] = 0$. (When $\Delta X_j[i] \neq 0$, $X'_j[i]$ and $X_j[i]$ are already

determined and otherwise $X'_j[i] = X_j[i]$ holds.) The cardinality of the set $\widetilde{U}_i$ indicates the amount of freedom left. If $\widetilde{U}_i = \emptyset$ then the auxiliary conditions together with ΔQ_j are contradictory and are thus of no interest.

The set $\widetilde{U}_i$ of possible input values induces a set $V_{\widetilde{U}_i}$ of possible output differences:

$$V_{\widetilde{U}_i} = \{g(X'_0[i], \ldots) - g(X_0[i], \ldots) \mid ((X'_0[i], \ldots), (X_0[i], \ldots)) \in \widetilde{U}_i\}.$$

If $|V_{\widetilde{U}_i}| = 1$ then $\Delta F_t[i] \in V_{\widetilde{U}_i}$ is uniquely determined and no auxiliary conditions are necessary. This is always the case if $\Delta X_0[i] = \ldots = \Delta X_{L-2}[i] = 0$ as then $V_{\widetilde{U}_i} = \{0\}$. Otherwise one can choose any value $\Delta\widehat{F}_t[i] \in V_{\widetilde{U}_i}$ and pose auxiliary conditions on $X_0[i], \ldots, X_{L-2}[i]$ such that the resulting new set of possible input values $\widehat{U}_i$:

- uniquely determines $\Delta F_t[i]$: $V_{\widehat{U}_i} = \{\Delta\widehat{F}_t[i]\}$.
- is as large as possible: $V_{\widetilde{U}_i \setminus \widehat{U}_i} \cap V_{\widehat{U}_i} = \emptyset$.

Once all $\Delta F_t[i]$ are uniquely determined by adding auxiliary conditions on Q_{t-L+2}, …, Q_t as necessary, also ΔF_t is completely determined. Depending on the boolean function g, some of the input variables of $Q_{t-L+2}, \ldots, Q_t$ may not affect the value of $f_{\mathrm{bool},t}(Q_{t-L+2}, \ldots, Q_t)$ and thus their respective variables $Q_j, Q'_j, \Delta Q_j$, can be ignored entirely in the above analysis.

5.5.3 Differential steps

Differential paths essentially are a sequence of *differential steps*. The message differences are chosen in some clever manner and must be given in the form of a sequence $(\mathcal{W}_t)_{t=0}^{S-1}$ of allowed message word differences $\delta W_t \in \mathcal{W}_t$ for each step t. A differential step is a precise description of how differences in the working state propagate through a single step $t \in \{0, \ldots, S-1\}$ and is defined as the tuple

$$((\delta Q_j)_{j=t-L+1}^{t+1},\ (\Delta Q_j)_{j \in I_t},\ \delta W_t,\ \Delta F_t,\ ((\delta Y_{j,i})_{i=1}^{L+3})_{j=1}^{V}),$$

where

- $I_t \subseteq \{t-L+2, \ldots, t\}$ is the set of indices j such that the input variable Q_j affects the outcome $f_{\mathrm{bool},t}(Q_{t-L+2}, \ldots, Q_t)$;
- For $j \in I_t$, ΔQ_j is a BSDR of δQ_j;
- $\delta W_t \in \mathcal{W}_t$;
- $\Delta F_t[i] \in \{g(X'_0[i], \ldots) - g(X_0[i], \ldots) \mid X'_j[i] = X_j[i] + \Delta X_j[i]\}$ where g is the underlying boolean function of $f_{\mathrm{bool},t}$ and $RL(Q_{t-L+2+j}, r_{t-L+2+j})$ is denoted by X_j as in Section 5.5.2;
- For $j \in \{1, \ldots, V\}$ and $i \in \{1, \ldots, L+3\}$, let $r_{j,i}$ and $c_{j,i}$ denote the rotation constant and selection constant associated with the i-th input variable in $f_{\mathrm{temp},t,j}$;

- The variable $Y_{j,i}$ denotes the i-th input variable of $f_{\text{temp},t,j}$ *after* the rotation as in $f_{\text{temp},t,j}$, thus for $i = 1, \ldots, L$, $Y_{j,i}$ denotes $RL(Q_{t-L+i}, r_{j,i})$. The remaining $Y_{j,L+1}$, $Y_{j,L+2}$, $Y_{j,L+3}$ denote $RL(F_t, r_{j,L+1})$, $RL(W_t, r_{j,L+2})$ and $RL(T_{t,j-1}, r_{j,L+3})$, respectively.

The intermediate variable differences $\delta T_{t,j}$ as in Section 5.3.4 are hereby determined: $\delta T_{t,0} = 0$ by definition, $\delta T_{t,j} = \sum_{i=1}^{L+3} c_{j,i} \delta Y_{j,i}$ for $j = 1, \ldots, V$. Thus together with δQ_j, δW_t and ΔF_t all differences of the inputs of $f_{\text{temp},t,j}$ and the differences $\delta Y_{j,j}$ of the rotations of these inputs are known.

A differential step is called *valid* if there exist values $(\widehat{Q}_j)_{j=t-L+1}^{t+1}$, $(\widehat{Q}'_j)_{j=t-L+1}^{t+1}$, $\widehat{W}_t$, $\widehat{W}'_t$ such that:

- $\widehat{Q}_{t+1}$ and $\widehat{Q}'_{t+1}$ are the correct outputs of the stepfunction in Section 5.3.4 for inputs $((\widehat{Q}_j)_{j=t-L+1}^{t}, \widehat{W}_t)$ and $((\widehat{Q}'_j)_{j=t-L+1}^{t}, \widehat{W}'_t)$, respectively;
- $\delta Q_j = \widehat{Q}'_j - \widehat{Q}_j$ for $j = t - L + 1, \ldots, t + 1$;
- $\Delta Q_j[i] = \widehat{Q}'_j[i] - \widehat{Q}_j[i]$ for $j \in I_t$ and $i = 0, \ldots, N - 1$;
- $\delta W_t = \widehat{W}'_t - \widehat{W}_t$;
- $\Delta F_t[i] = \widehat{F}'_t[i] - \widehat{F}_t[i]$ for $i = 0, \ldots, N - 1$, where $\widehat{F}_t = f_{\text{bool},t}(\widehat{Q}_{t-L+2}, \ldots, \widehat{Q}_t)$ and $\widehat{F}'_t = f_{\text{bool},t}(\widehat{Q}'_{t-L+2}, \ldots, \widehat{Q}'_t)$;
- $\delta Y_{j,i} = RL(\widehat{Q}'_{t-L+i}, r_{j,i}) - RL(\widehat{Q}_{t-L+i}, r_{j,i})$ for $j = 1, \ldots, V$ and $i = 1, \ldots, L$;
- $\delta Y_{j,L+1} = RL(\widehat{F}'_t, r_{j,L+1}) - RL(\widehat{F}_t, r_{j,L+1})$ for $j = 1, \ldots, V$;
- $\delta Y_{j,L+2} = RL(\widehat{W}'_t, r_{j,L+2}) - RL(\widehat{W}_t, r_{j,L+2})$ for $j = 1, \ldots, V$;
- $\delta Y_{j,L+3} = RL(\widehat{T}'_{t,j-1}, r_{j,L+3}) - RL(\widehat{T}_{t,j-1}, r_{j,L+3})$ for $j = 1, \ldots, V$, where $\widehat{T}_{t,i}$ and $\widehat{T}'_{t,i}$ for $i = 0, \ldots, V$ are computed as in Section 5.3.4;

Such values $(\widehat{Q}_j)_{j=t-L+1}^{t+1}$, $(\widehat{Q}'_j)_{j=t-L+1}^{t+1}$, $\widehat{W}_t$, $\widehat{W}'_t$ are called a solution for the differential step. This implies that for $j = 1, \ldots, V$ and each rotation in $f_{\text{temp},t,j}$, the difference $\delta Y_{j,i}$ is a possible outcome after rotation using the respective input difference: δQ_{t-L+1}, …, δQ_t, δF_t, δW_t or $\delta T_{t,j}$. Moreover this also implies that if a particular BSDR Z is given for such an input difference associated with $\delta Y_{j,i}$, then $RL(Z, r_{j,i})$ is a BSDR of $\delta Y_{j,i}$.

A partial differential path is defined as a sequence of differential steps for a subrange $t = t_{\text{begin}}, \ldots, t_{\text{end}}$, where $t_{\text{begin}}, t_{\text{end}} \in \{0, \ldots, S - 1\}$, $t_{\text{end}} \geq t_{\text{begin}}$. A partial differential path over the range $t = t_{\text{begin}}, \ldots, t_{\text{end}}$ is called *valid* if there exists a solution consisting of values

$$(\widehat{Q}_j)_{j=t_{\text{begin}}-L+1}^{t_{\text{end}}+1}, \quad (\widehat{Q}'_j)_{j=t_{\text{begin}}-L+1}^{t_{\text{end}}+1}, \quad (\widehat{W}_j)_{j=t_{\text{begin}}}^{t_{\text{end}}}, \quad (\widehat{W}'_j)_{j=t_{\text{begin}}}^{t_{\text{end}}}$$

which simultaneously provides solutions for all differential steps of the partial differential path. Such a solution does not imply a solution for the near-collision attack as $(\widehat{W}_j)_{j=t_{\text{begin}}}^{t_{\text{end}}}$ and $(\widehat{W}'_j)_{j=t_{\text{begin}}}^{t_{\text{end}}}$ with almost certainty are not part of valid message expansions for two message blocks M and M'. A (full) differential path is defined as a partial differential path over the range $t = 0, \ldots, S-1$.

5.6 Differential path construction

In this section we present the algorithm for constructing valid full differential paths for a compression function Compress $\in \mathcal{F}_{\text{md4cf}}$ based on two valid partial differential paths: $\mathcal{P}_{\text{l}}$ over steps $t = 0, \ldots, t_b$ and $\mathcal{P}_{\text{u}}$ over steps $t = t_e, \ldots, S-1$ where $t_b < t_e - L$. It will do so by independently extending $\mathcal{P}_{\text{l}}$ and $\mathcal{P}_{\text{u}}$ to $t = 0, \ldots, t_c - 1$ and $t_c + L, \ldots, S-1$, respectively, for some chosen value of t_c so they do not overlap. Actually, we construct large sets $\mathcal{E}_{\text{l}}$ and $\mathcal{E}_{\text{u}}$ of such extended differential paths. For the remaining steps $t = t_c, \ldots, t_c + L - 1$ we check for combinations $\mathcal{P}_{\text{l}} \in \mathcal{E}_{\text{l}}$, $\mathcal{P}_{\text{u}} \in \mathcal{E}_{\text{u}}$ whether a valid full differential path can be constructed.

5.6.1 Forward

For $\widetilde{t} = t_b + 1, \ldots, t_c - 1$, we extend a valid partial differential path $\mathcal{P}$ over steps $t = 0, \ldots, \widetilde{t} - 1$ with step $\widetilde{t}$. The partial differential path gives us values $(\delta Q_t)_{t=-L+1}^{\widetilde{t}}$ and $(\Delta Q_t)_{t \in I}$ where $I = \bigcup_{t=0}^{\widetilde{t}-1} I_t$. For $i \in \{\widetilde{t} - L + 1, \ldots, \widetilde{t} - 1\}$, $\mathcal{P}$ also gives us (multiple) differences after rotation of δQ_i. For each non-trivial rotation of δQ_i, which we index by $j \in J_{\mathcal{P},i}$, we denote the rotation constant as $r_{\mathcal{P},i,j}$ and the outcome difference as $\delta Y_{\mathcal{P},i,j} \in dRL(\delta Q_i, r_{\mathcal{P},i,j})$.[17] Since such values can also be determined from given IHV_{in} and IHV'_{in} we can also allow t_b to be -1.

For $i \in I_{\widetilde{t}} \setminus I$, we need to choose a BSDR ΔQ_i of δQ_i. We may choose any BSDR ΔQ_i of δQ_i such that $RL(\Delta Q_i, r_{\mathcal{P},i,j})$ is a BSDR of $\delta Y_{\mathcal{P},i,j}$ for all $j \in J_{\mathcal{P},i}$. Low weight BSDRs are the preferable choice as they in general lead to fewer sufficient conditions for the differential path.

Let X_j denote $RL(Q_{\widetilde{t}-L+2+j}, r_{\text{bool},\widetilde{t}-L+2+j})$ and g be the underlying boolean function of $f_{\text{bool},\widetilde{t}}$ as in Section 5.5.2. For $i = 0, \ldots, N-1$, the set

$$U_i = \{((X'_0[i], \ldots), (X_0[i], \ldots)) \mid X'_j[i] = X_j[i] + \Delta X_j[i]\}$$

defines all possible input bit values associated with the output bit $F_t[i]$. Previous steps restrict this set U_i further to $\widetilde{U}_i$ by allowing only those values for which there is a matching solution of $\mathcal{P}$. Choose any

$$k_i \in \left\{ g(X'_0[i], \ldots) - g(X_0[i], \ldots) \mid ((X'_0[i], \ldots), (X_0[i], \ldots)) \in \widetilde{U}_i \right\}.$$

After all k_i have been chosen, $\Delta F_t = (k_i)_{i=0}^{N-1}$ is fully determined. For $j \in \{1, \ldots, V\}$, let $\delta Y_{j,L+1} = \sigma(RL(\Delta F_t, r_{\text{temp},j,L+1}))$ where $r_{\text{temp},j,L+1}$ is the rotation constant as-

17. For dRL see Section 5.5.1.

sociated with F_t in $f_{\mathrm{temp},\widetilde{t},j}$.[18] As F_t is selected only once in total, there is at most one non-trivial case for which $r_{\mathrm{temp},j,L+1} \neq 0$.

Choose any $\delta W_{\widetilde{t}} \in \mathcal{W}_{\widetilde{t}}$. For $j \in \{1,\ldots,V\}$, let $r_{\mathrm{temp},j,L+2}$ and $c_{\mathrm{temp},j,L+2}$ be the rotation and selection constant associated with $W_{\widetilde{t}}$ in $f_{\mathrm{temp},\widetilde{t},j}$. Choose any difference $\delta Y_{j,L+2} \in dRL(\delta W_{\widetilde{t}}, r_{\mathrm{temp},j,L+2})$, preferably one with the highest probability. As W_t is selected only once in total, there is at most one non-trivial case for which $r_{\mathrm{temp},j,L+2} \neq 0$.

For $k \in \{1,\ldots,V\}$ and $i \in \{\widetilde{t}-L+1,\ldots,\widetilde{t}\}$, let $r_{\mathrm{temp},k,i-\widetilde{t}+L}$ be the rotation constant associated with Q_i in $f_{\mathrm{temp},\widetilde{t},k}$. If $i \notin (I_{\widetilde{t}} \cup I)$ then ΔQ_i is not yet chosen, we need to choose V differences

$$\delta Y_{k,(i-\widetilde{t}+L)} \in dRL(\delta Q_i, r_{\mathrm{temp},k,i-\widetilde{t}+L}).$$

Consider the set of BSDRs Z of δQ_i such that $\sigma(RL(Z, r_{P,i,j})) = \delta Y_{P,i,j}$ for all $j \in J_{P,i}$. Each such BSDR Z leads directly to values

$$(\delta Y_{k,i-\widetilde{t}+L})_{k=1}^{V} = (\sigma(RL(Z, r_{\mathrm{temp},k,i-\widetilde{t}+L})))_{k=1}^{V}.$$

One can choose any such values $(\delta Y_{k,i-\widetilde{t}+L})_{k=1}^{V}$, the preferable choice is one which results from the largest number of possible Z-values. Otherwise, if $i \in (I_{\widetilde{t}} \cup I)$ let $\delta Y_{k,i-\widetilde{t}+L} = \sigma(RL(\Delta Q_i, r_{\mathrm{temp},k,i-\widetilde{t}+L}))$.

The differential step $t = \widetilde{t}$ is now easily determined. By definition $\delta T_{\widetilde{t},0} = 0$. For $i = 1,\ldots,V$, choose a $\delta Y_{i,L+3} \in dRL(\delta T_{\widetilde{t},i-1}, r_{\mathrm{temp},i,L+3})$ with the highest probability where $r_{\mathrm{temp},i,L+3}$ is the rotation constant associated with $T_{\widetilde{t},i-1}$ in $f_{\mathrm{temp},\widetilde{t},i}$. Note that:

$$\begin{aligned}
\delta T_{\widetilde{t},i} &= f_{\mathrm{temp},\widetilde{t},i}(Q'_{\widetilde{t}-L+1},\ldots,Q'_{\widetilde{t}},F'_{\widetilde{t}},W'_{\widetilde{t}},T'_{\widetilde{t},i-1}) \\
&\quad -f_{\mathrm{temp},\widetilde{t},i}(Q_{\widetilde{t}-L+1},\ldots,Q_{\widetilde{t}},F_{\widetilde{t}},W_{\widetilde{t}},T_{\widetilde{t},i-1}) \\
&= \sum_{j=1}^{L} c_{i,j}(RL(Q'_{\widetilde{t}-L+j}, r_{\mathrm{temp},i,j}) - RL(Q_{\widetilde{t}-L+j}, r_{\mathrm{temp},i,j})) \\
&\quad +c_{i,L+1}(RL(F'_{\widetilde{t}}, r_{\mathrm{temp},i,L+1}) - RL(F_{\widetilde{t}}, r_{\mathrm{temp},i,L+1})) \\
&\quad +c_{i,L+2}(RL(W'_{\widetilde{t}}, r_{\mathrm{temp},i,L+2}) - RL(W_{\widetilde{t}}, r_{\mathrm{temp},i,L+2})) \\
&\quad +c_{i,L+3}(RL(T'_{\widetilde{t},i-1}, r_{\mathrm{temp},i,L+3}) - RL(T_{\widetilde{t},i-1}, r_{\mathrm{temp},i,L+3})) \\
&= \sum_{j=1}^{L+3} c_{i,j}\delta Y_{i,j}
\end{aligned}$$

All $\delta Y_{i,j}$ have been determined already. Thus $\delta T_{\widetilde{t},i}$ is hereby determined and for $i = V$ also $\delta Q_{\widetilde{t}+1}$.

The extended differential path $\widetilde{\mathcal{P}}$ consists of $\mathcal{P}$ and the differential step $t = \widetilde{t}$. If there exists no solution of $\widetilde{\mathcal{P}}$ then the extended differential path is not valid and of no further interest. We collect many valid differential paths $\widetilde{\mathcal{P}}$ by varying the

18. For σ see Section 5.2.2.

choices made and for different input differential paths $\mathcal{P}$. As this set can theoretically grow exponentially over subsequent steps $\widetilde{t}$, keep only the R "best" differential paths for some feasibly large R. Here "best" should be tailored toward the near-collision attack construction, which implies a low number of sufficient conditions for the partial differential path and a large degree of freedom for message modification techniques.

5.6.2 Backward

For $\widetilde{t} = t_e - 1, \ldots, t_c + L$, we extend a valid partial differential path $\mathcal{P}$ over steps $t = \widetilde{t}+1, \ldots, S-1$ with step $\widetilde{t}$. The partial differential path gives us values $(\delta Q_t)_{t=\widetilde{t}-L+2}^{S}$, $(\Delta Q_t)_{t\in I}$ where $I = \bigcup_{t=\widetilde{t}+1}^{S-1} I_t$. For $i \in \{\widetilde{t} - L + 2, \ldots, \widetilde{t} + 1\}$, the differential path $\mathcal{P}$ also gives us (multiple) differences after rotation of δQ_i. For each non-trivial rotation of δQ_i, which we index by $j \in J_{\mathcal{P},i}$, we denote the rotation constant as $r_{\mathcal{P},i,j}$ and the outcome difference as $\delta Y_{\mathcal{P},i,j} \in dRL(\delta Q_i, r_{\mathcal{P},i,j})$. The following steps are basically the same as in Section 5.6.1 except initially $(f_{\mathrm{invQ},\widetilde{t},k})_{k=1}^{2V+1}$ (as constructed in Section 5.4.1) are used instead of $(f_{\mathrm{temp},\widetilde{t},k})_{k=1}^{V}$. Only at the end the results are translated to a differential step over $(f_{\mathrm{temp},\widetilde{t},k})_{k=1}^{V}$.

For $i \in I_{\widetilde{t}} \setminus I$, we need to choose a BSDR ΔQ_i of δQ_i. We may choose any BSDR ΔQ_i of δQ_i such that $RL(\Delta Q_i, r_{\mathcal{P},i,j})$ is a BSDR of $\delta Y_{\mathcal{P},i,j}$ for all $j \in J_{\mathcal{P},i}$. As before, low weight BSDRs are the preferable choice as they in general lead to fewer sufficient conditions for the differential path.

Let X_j denote $RL(Q_{\widetilde{t}-L+2+j}, r_{\mathrm{bool},\widetilde{t}-L+2+j})$ and g be the underlying boolean function of $f_{\mathrm{bool},\widetilde{t}}$ as in Section 5.5.2. For $i = 0, \ldots, N-1$, the set

$$U_i = \{((X'_0[i], \ldots), (X_0[i], \ldots)) \mid X'_j[i] = X_j[i] + \Delta X_j[i]\}$$

defines all possible input bit values associated with the output bit $F_t[i]$. The known differential steps $t = \widetilde{t}+1, \ldots, S-1$ restrict this set U_i further to $\widetilde{U}_i$ by allowing only those values for which there is a matching solution of $\mathcal{P}$. Choose any

$$k_i \in \left\{ g(X'_0[i], \ldots) - g(X_0[i], \ldots) \mid ((X'_0[i], \ldots), (X_0[i], \ldots)) \in \widetilde{U}_i \right\}.$$

After $k_0, \ldots, k_{N-1}$ have been chosen, $\Delta F_t = (k_i)_{i=0}^{N-1}$ is fully determined. For $j \in \{1, \ldots, 2V+1\}$, let $\delta Y_{j,L+1} = \sigma(RL(\Delta F_t, r_{\mathrm{invQ},j,L+1}))$ where $r_{\mathrm{invQ},j,L+1}$ is the rotation constant associated with F_t in $f_{\mathrm{invQ},\widetilde{t},j}$.

Choose any $\delta W_{\widetilde{t}} \in \mathcal{W}_{\widetilde{t}}$. For $k \in \{1, \ldots, 2V+1\}$, let $r_{\mathrm{invQ},k,L+2}$ and $c_{\mathrm{invQ},k,L+2}$ be the rotation and selection constant associated with $W_{\widetilde{t}}$ in $f_{\mathrm{invQ},\widetilde{t},k}$. Choose any difference $\delta Y_{k,L+2} \in dRL(\delta W_{\widetilde{t}}, r_{\mathrm{invQ},k,L+2})$, preferably one with the highest probability.

For $k \in \{1, \ldots, 2V+1\}$ and $i \in \{\widetilde{t} - L + 2, \ldots, \widetilde{t} + 1\}$, let $r_{\mathrm{invQ},k,(i-\widetilde{t}+L-1)}$ be the rotation constant associated with Q_i in $f_{\mathrm{invQ},\widetilde{t},k}$. If $i \notin (I_{\widetilde{t}} \cup I)$ then ΔQ_i is not yet chosen, we need to choose $2V+1$ differences

$$\delta Y_{k,(i-\widetilde{t}+L-1)} \in dRL(\delta Q_i, r_{\mathrm{invQ},k,(i-\widetilde{t}+L-1)}).$$

Consider the set of BSDRs Z of δQ_i such that $\sigma(RL(Z, r_{\mathcal{P},i,j})) = \delta Y_{\mathcal{P},i,j}$ for all $j \in J_{\mathcal{P},i}$. Each such BSDR Z leads directly to values

$$(\delta Y_{k,(i-\widetilde{t}+L-1)})_{k=1}^{2V+1} = (\sigma(RL(Z, r_{\mathrm{invQ},k,(i-\widetilde{t}+L-1)})))_{k=1}^{2V+1}.$$

One can choose any such values $(\delta Y_{k,(i-\widetilde{t}+L-1)})_{k=1}^{2V+1}$, the preferable choice is one which results from the largest number of possible Z-values. Otherwise, if $i \in (I_{\widetilde{t}} \cup I)$ let $\delta Y_{k,(i-\widetilde{t}+L-1)} = \sigma(RL(\Delta Q_i, r_{\mathrm{invQ},k,(i-\widetilde{t}+L-1)}))$.

By definition $\delta T_{\mathrm{invQ},\widetilde{t},0} = 0$. For $i = 1, \ldots, 2V+1$ and $j = 0, \ldots, i-1$, let $r_{\mathrm{T},i,j}$ be the rotation constant associated with $T_{\mathrm{invQ},\widetilde{t},j}$ in $f_{\mathrm{invQ},\widetilde{t},i}$. Choose a most probable $\delta Y_{i,L+3+j} \in dRL(\delta T_{\mathrm{invQ},\widetilde{t},j}, r_{\mathrm{T},i,j})$. Note that:

$$\begin{aligned}
\delta T_{\mathrm{invQ},\widetilde{t},i} &= \sum_{j=1}^{L} c_{i,j}(RL(Q'_{\widetilde{t}-L+j+1}, r_{\mathrm{invQ},i,j}) - RL(Q_{\widetilde{t}-L+j+1}, r_{\mathrm{invQ},i,j})) \\
&\quad + c_{i,L+1}(RL(F'_{\widetilde{t}}, r_{\mathrm{invQ},i,L+1}) - RL(F_{\widetilde{t}}, r_{\mathrm{invQ},i,L+1})) \\
&\quad + c_{i,L+2}(RL(W'_{\widetilde{t}}, r_{\mathrm{invQ},i,L+2}) - RL(W_{\widetilde{t}}, r_{\mathrm{invQ},i,L+2})) \\
&\quad + \sum_{j=0}^{i-1} c_{i,L+3+j}(RL(T'_{\mathrm{invQ},\widetilde{t},j}, r_{\mathrm{T},i,j}) - RL(T_{\mathrm{invQ},\widetilde{t},j}, r_{\mathrm{T},i,j})) \\
&= \sum_{j=1}^{L+2+i} c_{i,j}\delta Y_{i,j}.
\end{aligned}$$

All $\delta Y_{i,j}$ have been determined already. Thus $\delta T_{\mathrm{invQ}\widetilde{t},i}$ is hereby determined and for $i = 2V+1$ also $\delta Q_{\widetilde{t}-L+1}$.

Most of the information for the differential step $t = \widetilde{t}$ is a direct result of the above steps: the values $(\delta Q_j)_{j=\widetilde{t}-L+1}^{\widetilde{t}+1}$, $(\Delta Q_j)_{j \in I_t}$, δW_t and ΔF_t. It remains to determine the $\delta \widetilde{Y}_{i,j}$ corresponding to differences after rotation of the inputs of $f_{\mathrm{temp},t,i}$. In the proof of Theorem 5.1 the original inputs of $f_{\mathrm{temp},t,j}$ are used in the $f_{\mathrm{invQ},t,i}$ in one of two ways. In the first way they are used as a direct input to a call of $f_{\mathrm{temp},t,j}$ within $f_{\mathrm{invQ},t,i}$. In the second way, they form the outcome of some $f_{\mathrm{invQ},t,i}$ that is used together with $f_{\mathrm{invQ},t,i-1}$ to invert $f_{\mathrm{temp},t,j}$. Thus all differences $\delta \widetilde{Y}_{i,j}$ after rotation of the inputs of $f_{\mathrm{temp},t,i}$ are already determined above.

The extended differential path $\widetilde{\mathcal{P}}$ consists of $\mathcal{P}$ and the differential step $t = \widetilde{t}$. If there exists no solution of $\widetilde{\mathcal{P}}$ then the extended differential path is not valid and of no further interest. Similar to extending forward, we collect many valid differential paths $\widetilde{\mathcal{P}}$ by varying the choices made and for different input differential paths $\mathcal{P}$. As this set can theoretically grow exponentially over subsequent steps $\widetilde{t}$, keep only the R "best" differential paths for some feasibly large R. Here "best" differential paths should be read as the differential paths with the highest success probability.

5.6.3 Connect

In the final stage, we try many combinations of lower valid differential paths $\mathcal{P}_\mathrm{l}$ over steps $t = 0, \ldots, t_c - 1$ and upper valid differential paths $\mathcal{P}_\mathrm{u}$ over steps $t_c + L, \ldots, S - 1$. For each such combination we have differential steps $t = 0, \ldots, t_c - 1, t_c + L, \ldots, S - 1$. This means that all $(\delta Q_t)_{t=-L+1}^{S}$ are known either from $\mathcal{P}_\mathrm{l}$ or $\mathcal{P}_\mathrm{u}$, there is no overlap. Together $\mathcal{P}_\mathrm{l}$ and $\mathcal{P}_\mathrm{u}$ also provide values $(\Delta Q_t)_{t \in I}$ where $I = \bigcup_{t=t_c+L}^{S-1} I_t \cup \bigcup_{t=0}^{t_c-1} I_t$. We use $((f_{\mathrm{invF},j,i})_{i=1}^{2V+1})_{j=t_c}^{t_c+L-1}$ to determine target differences $\delta F_{t_c}, \ldots, \delta F_{t_c+L-1}$. By using the remaining freedom in yet undetermined BSDRs $\Delta F_{t_c}, \ldots, \Delta F_{t_c+L-1}$, we try to achieve these target differences.

For $\widetilde{t} = t_c, \ldots, t_c + L - 1$, we do the following and exhaustively try all possible choices. For $i \in I_{\widetilde{t}} \setminus (I \cup \bigcup_{t=t_c}^{\widetilde{t}-1} I_t)$, we need to choose a BSDR ΔQ_i of δQ_i. For $i \in \{\widetilde{t} - L + 1, \ldots, \widetilde{t} + 1\}$, the preceding differential steps $t = t_c, \ldots, \widetilde{t} - 1$ together with $\mathcal{P}_\mathrm{l}$ and $\mathcal{P}_\mathrm{u}$ give us (multiple) differences after rotation of δQ_i. For each non-trivial rotation of δQ_i, which we index by $j \in J_{\mathcal{P},i}$, we denote the rotation constant as $r_{\mathcal{P},i,j}$ and the outcome difference as $\delta Y_{\mathcal{P},i,j} \in dRL(\delta Q_i, r_{\mathcal{P},i,j})$. Choose any BSDR ΔQ_i of δQ_i such that $RL(\Delta Q_i, r_{\mathcal{P},i,j})$ is a BSDR of $\delta Y_{\mathcal{P},i,j}$ for all $j \in J_{\mathcal{P},i}$.

Choose any $\delta W_{\widetilde{t}} \in \mathcal{W}_{\widetilde{t}}$. For $k \in \{1, \ldots, 2V + 1\}$, let $r_{\mathrm{invF},k,L+2}$ and $c_{\mathrm{invF},k,L+2}$ be the rotation and selection constant associated with $W_{\widetilde{t}}$ in $f_{\mathrm{invF},\widetilde{t},k}$. Choose any difference $\delta Y_{k,L+2} \in dRL(\delta W_{\widetilde{t}}, r_{\mathrm{invF},k,L+2})$.

For $k \in \{1, \ldots, 2V + 1\}$ and $i \in \{\widetilde{t} - L + 1, \ldots, \widetilde{t} + 1\}$, let $r_{\mathrm{invF},k,(i-\widetilde{t}+L)}$ be the rotation constant associated with Q_i in $f_{\mathrm{invF},\widetilde{t},k}$. If $i \notin (I \cup \bigcup_{t=t_c}^{\widetilde{t}} I_t)$ then ΔQ_i is not yet chosen and we need to choose $2V + 1$ differences

$$\delta Y_{k,(i-\widetilde{t}+L)} \in dRL(\delta Q_i, r_{\mathrm{invF},k,(i-\widetilde{t}+L)}).$$

Consider the set of BSDRs Z of δQ_i such that $\sigma(RL(Z, r_{P,i,j})) = \delta Y_{P,i,j}$ for all $j \in J_{P,i}$. Each such BSDR Z leads directly to values

$$(\delta Y_{k,(i-\widetilde{t}+L)})_{k=1}^{2V+1} = (\sigma(RL(Z, r_{\mathrm{invF},k,(i-\widetilde{t}+L)})))_{k=1}^{2V+1}.$$

Choose any such values $(\delta Y_{k,(i-\widetilde{t}+L)})_{k=1}^{2V+1}$. Otherwise, if $i \in (I \cup \bigcup_{t=t_c}^{\widetilde{t}} I_t)$ then let $\delta Y_{k,(i-\widetilde{t}+L)} = \sigma(RL(\Delta Q_i, r_{\mathrm{invF},k,(i-\widetilde{t}+L)}))$.

By definition $\delta T_{\mathrm{invF},\widetilde{t},0} = 0$. For $i = 1, \ldots, 2V + 1$ and $j = 0, \ldots, i - 1$, let $r_{\mathrm{T},i,j}$ be the rotation constant associated with $T_{\mathrm{invF},\widetilde{t},j}$ in $f_{\mathrm{invF},\widetilde{t},i}$. Choose any $\delta Y_{i,L+3+j} \in dRL(\delta T_{\mathrm{invF},\widetilde{t},j}, r_{\mathrm{T},i,j})$, here we do not limit the choices to high probability values to increase the success probability of our connection algorithm. This can be compensated by searching for many valid full differential paths and choosing the one most suitable

for a near-collision attack. Note that:

$$\begin{aligned}\delta T_{\mathrm{invF},\widetilde{t},i} &= \sum_{j=1}^{L+1} c_{i,j}(RL(Q'_{\widetilde{t}-L+j}, r_{\mathrm{invF},i,j}) - RL(Q_{\widetilde{t}-L+j}, r_{\mathrm{invF},i,j})) \\ &\quad + c_{i,L+2}(RL(W'_{\widetilde{t}}, r_{\mathrm{invF},i,L+2}) - RL(W_{\widetilde{t}}, r_{\mathrm{invF},i,L+2})) \\ &\quad + \sum_{j=0}^{i-1} c_{i,L+3+j}(RL(T'_{\mathrm{invF},\widetilde{t},j}, r_{\mathrm{T},i,j}) - RL(T_{\mathrm{invF},\widetilde{t},j}, r_{\mathrm{T},i,j})) \\ &= \sum_{j=1}^{L+2+i} c_{i,j}\delta Y_{i,j}.\end{aligned}$$

All $\delta Y_{i,j}$ have been determined already. Thus $\delta T_{\mathrm{invF}\widetilde{t},i}$ is hereby determined and for $i = 2V + 1$ also $\delta F_{\widetilde{t}}$.

Let X_j denote $RL(Q_{\widetilde{t}-L+2+j}, r_{\mathrm{bool},\widetilde{t}-L+2+j})$ and g be the underlying boolean function of $f_{\mathrm{bool},\widetilde{t}}$ as in Section 5.5.2. For $i = 0, \ldots, N-1$, the set

$$U_i = \{((X'_0[i], \ldots), (X_0[i], \ldots)) \mid X'_j[i] = X_j[i] + \Delta X_j[i]\}$$

defines all possible input bit values associated with the output bit $F_t[i]$. The known differential steps restrict this set U_i further to $\widetilde{U}_i$ by allowing only those values for which there is a matching solution of those differential steps. Let

$$V_i = \left\{ g(X'_0[i], \ldots) - g(X_0[i], \ldots) \mid ((X'_0[i], \ldots), (X_0[i], \ldots)) \in \widetilde{U}_i \right\}.$$

Now it remains to verify whether $\delta F_{\widetilde{t}}$ can be achieved by one of the possible BSDRs Z such that $Z[i] \in V_i$ for $i = 0, \ldots, N-1$.

To this end we desire to construct sets

$$\mathcal{Z}_i = \left\{ (k_j)_{j=0}^{i} \;\middle|\; k_j \in V_j \wedge \left(\sum_{j=0}^{i} k_j 2^j \equiv \delta F_{\widetilde{t}} \bmod 2^{i+1} \right) \right\}, \quad i \in \{0, \ldots, N-1\}.$$

It follows that $\mathcal{Z}_0 = \{(k_0) \mid k_0 \in V_0 \wedge (k_0 \equiv \delta F_{\widetilde{t}} \bmod 2)\}$. For $i = 1, \ldots, N-1$, we construct $\mathcal{Z}_i$ as

$$\mathcal{Z}_i = \left\{ (k_j)_{j=0}^{i} \;\middle|\; (k_j)_{j=0}^{i-1} \in \mathcal{Z}_{i-1} \wedge k_i \in V_i \wedge \left(\sum_{j=0}^{i} k_j 2^j \equiv \delta F_{\widetilde{t}} \bmod 2^{i+1} \right) \right\}.$$

If $\mathcal{Z}_i = \emptyset$ then $\delta F_{\widetilde{t}}$ cannot be achieved and other choices above have to be tried. If all choices above have been exhausted then we try a untried combination of $\mathcal{P}_{\mathrm{l}}$ and $\mathcal{P}_{\mathrm{u}}$. Otherwise, $\delta F_{\widetilde{t}}$ is achieved by any of the BSDRs $Z \in \mathcal{Z}_{N-1}$. Choose any $\Delta F_{\widetilde{t}} \in \mathcal{Z}_{N-1}$.

Most of the information for the differential step $t = \widetilde{t}$ is a direct result of the above steps: the values $(\delta Q_j)_{j=\widetilde{t}-L+1}^{\widetilde{t}+1}$, $(\Delta Q_j)_{j \in I_t}$, δW_t and ΔF_t. It remains to determine

the $\delta\widetilde{Y}_{i,j}$ corresponding to differences after rotation of the inputs of $f_{\text{temp},t,i}$. As the proof of Theorem 5.2 is analogous to that of Theorem 5.1, the original inputs of $f_{\text{temp},t,j}$ are used in the $f_{\text{invF},t,i}$ in one of two ways. In the first way they are used as a direct input to a call of $f_{\text{temp},t,j}$ within $f_{\text{invF},t,i}$. In the second way, they form the outcome of some $f_{\text{invQ},t,i}$ that is used together with $f_{\text{invQ},t,i-1}$ to invert $f_{\text{temp},t,j}$.

Thus all differences $\delta\widetilde{Y}_{i,j}$ after rotation of the inputs of $f_{\text{temp},t,i}$ are already determined above. It remains to verify whether there exists a simultaneous solution for $\mathcal{P}_\text{l}$, $\mathcal{P}_\text{u}$ and the differential steps $t = t_c, \ldots, \widetilde{t}$. If there is no such solution then this differential step is of no further interest and we try different choices above.

We can now try the next remaining differential step until $\widetilde{t} = t_c + L - 1$ in which case we have a full valid differential path.

5.6.4 Complexity

The complexity of the above procedure is of great interest, however it depends on many factors which are still undecided. It is common in attacks against hash functions to describe the complexity as the equivalent runtime cost in compression function calls. However, the parameters N, L, K and S, and the initialization, finalization, message expansion and step functions of the compression function remain to be chosen, thereby making a thorough complexity analysis infeasible. Nevertheless, below we comment on several significant factors that contribute to the complexity of the above differential path construction.

Connect search. The connect search tries combinations of lower and upper partial differential paths. The average runtime cost for each combination of lower and upper partial differential paths depends among others on the degrees of freedom. Less freedom implies a lower average runtime cost for each combination, however the average success probability per combination also decreases. The expected number of combinations to try is determined by the average success probability multiplied by the desired number of valid full differential paths. This desired number of valid full differential paths can be one, but often a lot more than one are desired as this leaves additional freedom when implementing a collision attack.

The degrees of freedom can mostly be found in the number of allowed BSDRs ΔQ_i for $i \in \{t_c - L + 2, \ldots, t_c + L\} \setminus \bigcup_{t=t_c}^{t_c+L-1} I_t$ and the average size of the boolean function output bit difference sets $\widetilde{U}_j$. At least two ΔQ_i are not uniquely determined yet: $i = t_c$ and $i = t_c + 1$. On average, a higher weight $\text{NAF}(\delta Q_i)$ results in a larger number of allowed BSDRs ΔQ_i. Therefore a high weight $\text{NAF}(\delta Q_i)$ for these values of i is beneficiary to the connect search, which is in contrast with the desire to obtain differential paths with as few as possible number of bitconditions.

Forward and backward search. The forward and backward search have very similar complexity characteristics. The complexity of the forward and backward search for each step consists of the following factors:

- number of input partial differential paths: by keeping only the R "best" partial differential paths this factor is upper bounded by R. Let R_{f} and R_{b} be this upper bound for the forward and backward search, respectively. It follows that $R_{\mathrm{f}} \cdot R_{\mathrm{b}}$ must be chosen large enough such that the desired number of valid full differential paths in the connect search may be obtained.

- average number of BSDRs $\Delta Q_{\tilde{t}}$ of $\delta Q_{\tilde{t}}$: this is lower bounded by $2^{w(\mathrm{NAF}(\delta Q_{\tilde{t}}))}$. Since low weight BSDRs are preferred, the set of allowed BSDRs $\Delta Q_{\tilde{t}}$ can be bounded for instance by taking only the B lowest weight BSDRs, limiting the maximum weight $w(\Delta Q_{\tilde{t}}) \leq B$, limiting the maximum offset weight $w(\Delta Q_{\tilde{t}}) \leq w(\mathrm{NAF}(\delta Q_{\tilde{t}})) + B$ or any combination thereof.

- average size κ of the possible boolean function output bit difference sets $\widetilde{U}_j$ resulting in a total factor of κ^N.

- size of the set of possible message word differences $\mathcal{W}_{\tilde{t}}$.

5.6.5 Specializations

In Chapters 6 and 7 we provide specializations of the differential cryptanalysis and differential path construction for MD5 and SHA-1. In particular, we use the concept of bitconditions to describe the differential path. These bitconditions also provide a means to efficiently determine the set of possible boolean function output bit differences and the additional bitconditions necessary to enforce the chosen boolean function output bit difference. Moreover, we improve the connect search such that it deals with the to be determined BSDRs ΔQ_i and ΔF_j in a per-digit manner instead of a per-BSDR manner.

6 MD5

Contents

6.1 Overview

In Section 6.2 we apply and improve the differential path analysis and algorithms of Chapter 5 for MD5 specifically. To complete the construction of near-collision attacks, we augment the differential path construction algorithms with a collision finding algorithm in Section 6.3. In Section 6.4 we present a near-collision attack based on new message differences that leads to an identical-prefix collision attack with estimated complexity of 2^{16} MD5 compressions. We present the chosen-prefix collision attack in Section 6.5 with estimated complexity $2^{39.1}$ MD5 compressions. Finally, we introduce a variant chosen-prefix collision attack which uses only a single near-collision block in Section 6.5.4 with estimated complexity $2^{53.2}$ MD5 compressions.

6.2 Differential path construction

First, we repeat the definition of MD5Compress. Then we introduce the concept of bitconditions as a means to describe (partial) differential paths. In Sections 6.2.3, 6.2.4 and 6.2.5 we refine the algorithms of Sections 5.6.1, 5.6.2 and 5.6.3, respectively, for MD5.

6.2.1 Definition of MD5Compress

MD5Compress uses solely 32-bit words. The input for the compression function MD5Compress(IHV_{in}, B) consists of an intermediate hash value $IHV_{\text{in}} = (a, b, c, d)$ consisting of four words and a 512-bit message block B. The compression function consists of 64 *steps* (numbered 0 to 63), split into four consecutive *rounds* of 16 steps each. Each step t uses modular additions, a left rotation, and a non-linear function f_t, and involves an *Addition Constant* AC_t and a *Rotation Constant* RC_t. These are defined as follows (see also Table A-1):

$$AC_t = \left\lfloor 2^{32} \left|\sin(t+1)\right| \right\rfloor \ , \qquad 0 \le t < 64,$$

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

The non-linear function f_t depends on the round:

$$f_t(X, Y, Z) = \begin{cases} F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z) & \text{for } 0 \le t < 16, \\ G(X, Y, Z) = (Z \wedge X) \oplus (\overline{Z} \wedge Y) & \text{for } 16 \le t < 32, \\ H(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 32 \le t < 48, \\ I(X, Y, Z) = Y \oplus (X \vee \overline{Z}) & \text{for } 48 \le t < 64. \end{cases} \tag{6.1}$$

The message block B is partitioned into sixteen consecutive words m_0, $m_1, \ldots, m_{15}$ (with little-endian byte ordering, see Section 2.1.2), and expanded to 64 words W_t, for $0 \le t < 64$, of 32 bits each (see also Table A-1):

$$W_t = \begin{cases} m_t & \text{for } 0 \le t < 16, \\ m_{(1+5t) \bmod 16} & \text{for } 16 \le t < 32, \\ m_{(5+3t) \bmod 16} & \text{for } 32 \le t < 48, \\ m_{(7t) \bmod 16} & \text{for } 48 \le t < 64. \end{cases}$$

We follow the description of the MD5 compression function from [HPR04] because its 'unrolling' of the cyclic state facilitates the analysis and because it matches the definitions in Section 5.3, thus MD5Compress $\in \mathcal{F}_{\text{md4cf}}$. For each step t the compression function algorithm maintains a working register with four state words Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3} and calculates a new state word Q_{t+1}. With $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$, for $t = 0, 1, \ldots, 63$ in succession Q_{t+1} is calculated as follows:

$$\begin{aligned} F_t &= f_t(Q_t,\ Q_{t-1},\ Q_{t-2}); \\ T_t &= F_t + Q_{t-3} + AC_t + W_t; \\ R_t &= RL(T_t, RC_t); \\ Q_{t+1} &= Q_t + R_t. \end{aligned} \tag{6.2}$$

After all steps are computed, the resulting state words are added to the intermediate hash value and returned as output:

$$\text{MD5Compress}(IHV_{\text{in}}, B) = (a + Q_{61},\ b + Q_{64},\ c + Q_{63},\ d + Q_{62}). \tag{6.3}$$

6.2.2 Bitconditions

Since MD5Compress $\in \mathcal{F}_{\text{md4cf}}$, we use the differential path definition from Section 5.5 and the differential path construction algorithm from Section 5.6. The differential path construction in Section 5.6 does not provide an algorithmic solution to determine whether a differential path is valid or to determine the sets $\widetilde{\mathcal{U}}_i$ in Section 5.6. In this section we define the concept of *bitconditions* as a way to describe differential paths. We use this description of a differential path in bitconditions to deal with the two above-mentioned issues.

Differential paths for MD5 are described using bitconditions $\mathfrak{q}_t = (\mathfrak{q}_t[i])_{i=0}^{31}$ on (Q_t, Q'_t), where each bitcondition $\mathfrak{q}_t[i]$ specifies a restriction on the bits $Q_t[i]$ and $Q'_t[i]$ possibly including values of other bits $Q_l[i]$. As we show in this section, we can specify the values of ΔQ_t, ΔF_t for all t using bitconditions on (Q_t, Q'_t), which with given δW_t determine $\delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t$ and $\delta R_t = \sigma(\Delta Q_{t+1}) - \sigma(\Delta Q_t)$. Thus, a partial differential path over steps $t = t_b, \ldots, t_e$ can be seen as a $(t_e - t_b + 5) \times 32$ matrix $(\mathfrak{q}_t)_{t=t_b-3}^{t_e+1}$ of bitconditions. A full differential paths is thus a 68×32 matrix $(\mathfrak{q}_t)_{t=-3}^{64}$. As for each step t only $\Delta Q_t, \Delta Q_{t-1}$ and ΔQ_{t-2} are required in a differential step, in such a differential path $\mathfrak{q}_{t_b-3}$ and $\mathfrak{q}_{t_e+1}$ are used only to represent δQ_{t_b-3} and δQ_{t_e+1} instead of BSDRs. In general, the four rows $(\mathfrak{q}_t)_{t=-3}^{0}$ are fully determined by the values of IHV_{in} and IHV'_{in}.

Table 6-1: *Differential bitconditions*

$\mathfrak{q}_t[i]$	condition on $(Q_t[i], Q'_t[i])$	k_i
`.`	$Q_t[i] = Q'_t[i]$	0
`+`	$Q_t[i] = 0, \quad Q'_t[i] = 1$	+1
`-`	$Q_t[i] = 1, \quad Q'_t[i] = 0$	−1

Note that $\delta Q_t = \sum_{i=0}^{31} 2^i k_i$ and $\Delta Q_t = (k_i)$.

Bitconditions are denoted using symbols such as '`0`', '`1`', '`+`', '`-`', '`^`', ..., as defined in Tables 6-1 and 6-2, to facilitate the representation of a differential path. A *direct* bitcondition $\mathfrak{q}_t[i]$ does not involve any other indices than t and i, whereas an *indirect* bitcondition involves one of the row indices $t \pm 1$ or $t \pm 2$ as well. Table 6-1 lists *differential* bitconditions $\mathfrak{q}_t[i]$, which are direct bitconditions that specify the value $k_i = Q'_t[i] - Q_t[i]$. A full row of differential bitconditions $\mathfrak{q}_t$ fixes a BSDR $(k_i)_{i=0}^{31}$ of $\delta Q_t = \sum_{i=0}^{31} 2^i k_i$. Table 6-2 lists *boolean function* bitconditions, which are direct or indirect. They are used to resolve a possible ambiguity in

$$\Delta F_t[i] = f_t(Q'_t[i], Q'_{t-1}[i], Q'_{t-2}[i]) - f_t(Q_t[i], Q_{t-1}[i], Q_{t-2}[i]) \in \{-1, 0, +1\}$$

Table 6-2: *Boolean function bitconditions*

$\mathtt{q}_t[i]$	condition on $(Q_t[i], Q'_t[i])$	direct/indirect	direction
`0`	$Q_t[i] = Q'_t[i] = 0$	direct	
`1`	$Q_t[i] = Q'_t[i] = 1$	direct	
`^`	$Q_t[i] = Q'_t[i] = Q_{t-1}[i]$	indirect	backward
`v`	$Q_t[i] = Q'_t[i] = Q_{t+1}[i]$	indirect	forward
`!`	$Q_t[i] = Q'_t[i] = \overline{Q_{t-1}[i]}$	indirect	backward
`y`	$Q_t[i] = Q'_t[i] = \overline{Q_{t+1}[i]}$	indirect	forward
`m`	$Q_t[i] = Q'_t[i] = Q_{t-2}[i]$	indirect	backward
`w`	$Q_t[i] = Q'_t[i] = Q_{t+2}[i]$	indirect	forward
`#`	$Q_t[i] = Q'_t[i] = \overline{Q_{t-2}[i]}$	indirect	backward
`h`	$Q_t[i] = Q'_t[i] = \overline{Q_{t+2}[i]}$	indirect	forward
`?`	$Q_t[i] = Q'_t[i] \wedge (Q_t[i] = 1 \vee Q_{t-2}[i] = 0)$	indirect	backward
`q`	$Q_t[i] = Q'_t[i] \wedge (Q_{t+2}[i] = 1 \vee Q_t[i] = 0)$	indirect	forward

that may be caused by different possible values for $Q_j[i], Q'_j[i]$ given differential bitconditions $\mathtt{q}_j[i]$. As an example, for $t = 0$ and $(\mathtt{q}_t[i], \mathtt{q}_{t-1}[i], \mathtt{q}_{t-2}[i]) = (\mathtt{.}, \mathtt{+}, \mathtt{-})$ (cf. Table 6-1) there is an ambiguity:

$$\begin{aligned} \text{if } Q_t[i] = Q'_t[i] = 0 \text{ then } \Delta F_t[i] &= f_t(0,1,0) - f_t(0,0,1) = -1, \\ \text{but if } Q_t[i] = Q'_t[i] = 1 \text{ then } \Delta F_t[i] &= f_t(1,1,0) - f_t(1,0,1) = +1. \end{aligned}$$

To resolve this ambiguity the triple of bitconditions (`.`,`+`,`-`) can be replaced by (`0`,`+`,`-`) or (`1`,`+`,`-`) for the two cases given above, respectively.

All boolean function bitconditions include the constant bitcondition $Q_t[i] = Q'_t[i]$, so boolean function bitconditions do not affect δQ_t. Furthermore, the indirect boolean function bitconditions never involve bitconditions '`+`' or '`-`', since those bitconditions can always be replaced by one of the direct ones '`.`', '`0`' or '`1`'. For the indirect bitconditions we distinguish between 'forward' and 'backward' ones, because that makes it easier to resolve an ambiguity in our step-wise approach. In a valid (partial) differential path one can easily convert forward bitconditions into backward bitconditions and vice versa.

To resolve some ambiguity or simply to determine ΔF_t for a valid partial differential path we proceed as follows. For some t and i, let $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathtt{q}_t[i], \mathtt{q}_{t-1}[i], \mathtt{q}_{t-2}[i])$ be any triple of bitconditions such that all indirect bitconditions involve only $Q_t[i]$, $Q_{t-1}[i]$ or $Q_{t-2}[i]$. For any such triple $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ let $U_{\mathfrak{abc}}$ denote the set of tuples of values

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

satisfying the bitconditions:

$$U_{\mathfrak{abc}} = \left\{(x, x', y, y', z, z') \in \{0,1\}^6 \text{ satisfies bitconditions } (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})\right\}.$$

The cardinality of $U_{\mathfrak{abc}}$ indicates the amount of freedom left by $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$. Triples $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ for which $U_{\mathfrak{abc}} = \emptyset$ cannot be part of a valid differential path and are thus of no interest. The set of all triples $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ as above and with $U_{\mathfrak{abc}} \neq \emptyset$ is denoted by $\mathcal{L}$ which thus consists of valid triples of bitconditions that are "local" in the sense that either they are direct or involve only one of the other two bits.

Each $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ induces a set $V_{t,\mathfrak{abc}}$ of possible boolean function differences $\Delta F_t[i] = f_t(x', y', z') - f_t(x, y, z)$:

$$V_{t,\mathfrak{abc}} = \{f_t(x', y', z') - f_t(x, y, z) \mid (x, x', y, y', z, z') \in U_{\mathfrak{abc}}\} \subset \{-1, 0, +1\}.$$

A triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{L}$ with $|V_{t,\mathfrak{def}}| = 1$ leaves no ambiguity in $\Delta F_t[i]$ and is therefore called a *solution*. Let $\mathcal{S}_t \subset \mathcal{L}$ be the set of solutions for step t.

For arbitrary $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ and for each $g \in V_{t,\mathfrak{abc}}$, we define $\mathcal{S}_{t,\mathfrak{abc},g}$ as the subset of $\mathcal{S}_t$ consisting of all solutions that are compatible with $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ and that have g as boolean function difference:

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

For each $g \in V_{t,\mathfrak{abc}}$ there is always a triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{S}_{t,\mathfrak{abc},g}$ consisting of direct bitconditions `01+-` that suffices, i.e., fixes a certain tuple in $U_{\mathfrak{abc}}$. This implies that $\mathcal{S}_{t,\mathfrak{abc},g} \neq \emptyset$. Despite this fact, we are specifically interested in bitconditions $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{S}_{t,\mathfrak{abc},g}$ that maximize $|U_{\mathfrak{def}}|$ as such bitconditions maximize the amount of freedom in the bits of Q_t, Q_{t-1}, Q_{t-2} while fully determining $\Delta F_t[i]$.

The direct and forward (respectively backward) boolean function bitconditions were chosen such that for all t, i and $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$ and for all $g \in V_{t,\mathfrak{abc}}$ there exists a triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}) \in \mathcal{S}_{t,\mathfrak{abc},g}$ consisting only of direct and forward (respectively backward) bitconditions such that

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

These boolean function bitconditions allow one to resolve an ambiguity in $\Delta F_t[i]$ in an optimal way in the sense that they are sufficient *and* necessary.

If the triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f})$ is not unique, then for simplicity we prefer direct over indirect bitconditions and short indirect bitconditions (`vy^!`) over long indirect ones (`whqm#?`). For given t, bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$, and $g \in V_{t,\mathfrak{abc}}$ we define $FC(t, \mathfrak{abc}, g) = (\mathfrak{d}, \mathfrak{e}, \mathfrak{f})$ as the preferred triple $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f})$ consisting of direct and forward bitconditions. Similarly, we define $BC(t, \mathfrak{abc}, g)$ as the preferred triple consisting of direct and backward bitconditions. These functions are easily determined and should be precomputed. They have been tabulated in Appendix C in Tables C-1, C-2, C-3 and C-4 grouped according to the four different round functions F, G, H, I, and per table for all 27 possible triples $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ of differential bitconditions.

Using this procedure we can easily determine for a partial differential path over steps $t = t_b, \ldots, t_e$ whether all $\Delta F_t[i]$ are unambiguously determined. When all ΔQ_t and ΔF_t have been determined by bitconditions then also $\delta T_t = \delta Q_{t-3} + \delta F_t + \delta W_t$ and $\delta R_t = \delta Q_{t+1} - \delta Q_t$ can be determined, which together describe the bitwise rotation of δT_t in each step. This does, however, not imply that $\delta R_t \in dRL(\delta T_t, RC_t)$ or with

what probability a correct rotation of differences happens. We refer to Lemma 5.4 (p. 74) for $dRL(\delta T_t, RC_t)$ and the probabilities of each possible output difference.

To determine whether a partial differential path is valid one needs to check whether there exist values $(Q_t)_{t=t_b-3}^{t_e+1}$ and $(Q'_t)_{t=t_b-3}^{t_e+1}$ satisfying the bitconditions $(\mathfrak{q}_t)_{t=t_b-3}^{t_e+1}$ that follow the prescribed path. If for all $t = t_b, \ldots, t_e$ and $i = 0, \ldots, 31$ we have that $|V_{t,\mathfrak{q}_t[i],\mathfrak{q}_{t-1}[i],\mathfrak{q}_{t-2}[i]}| = 1$ then those bitconditions are not contradicting and $\Delta F_t[i]$ is unambiguously determined. Moreover, then also any values $(Q_t)_{t=t_b-3}^{t_e+1}$ and $(Q'_t)_{t=t_b-3}^{t_e+1}$ satisfying the bitconditions $(\mathfrak{q}_t)_{t=t_b-3}^{t_e+1}$ trivially satisfy the ΔQ_t and ΔF_t as given by the differential path. In which case, the only remaining obstacle for a differential path to be valid are the rotations.

6.2.3 Extending differential paths forward

When constructing a differential path one must first fix the message block differences $\delta m_0, \ldots, \delta m_{15}$. They result directly in the differences δW_t for $t = 0, \ldots, 63$. We use an adaptation of the algorithm from Section 5.6.1 to extend a partial differential path forward using bitconditions. Suppose we have a partial differential path consisting of at least bitconditions $\mathfrak{q}_{t-1}$ and $\mathfrak{q}_{t-2}$ and that the differences δQ_t and δQ_{t-3} are known. We want to extend this partial differential path forward with step t resulting in the difference δQ_{t+1}, bitconditions $\mathfrak{q}_t$, and additional bitconditions $\mathfrak{q}_{t-1}, \mathfrak{q}_{t-2}$.

We assume that all indirect bitconditions in $\mathfrak{q}_{t-1}$ and $\mathfrak{q}_{t-2}$ are forward and involve only bits of Q_{t-1}. If we already have $\mathfrak{q}_t$ as opposed to just the value δQ_t (e.g., $\mathfrak{q}_0$ resulting from given values $IHV_{\text{in}}, IHV'_{\text{in}}$) then we can skip the remaining part of this paragraph. Otherwise, we first select bitconditions $\mathfrak{q}_t$ based on the value δQ_t. Since we want to construct differential paths with as few bitconditions as possible, but also want to be able to randomize the process, any low weight BSDR (such as the NAF) of δQ_t may be chosen, which then translates into a possible choice for $\mathfrak{q}_t$ as in Table 6-1. For instance, with $\delta Q_t = 2^8$, we may choose $\mathfrak{q}_t[8] =$ '+', or $\mathfrak{q}_t[8] =$ '-' and $\mathfrak{q}_t[9] =$ '+' (with in either case all other $\mathfrak{q}_t[i] =$ '.').

To determine the differences $\Delta F_t = (g_i)_{i=0}^{31}$ we proceed as follows. For $i = 0, 1, 2, \ldots, 31$ we assume that we have valid bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ where only $\mathfrak{c}$ may be indirect. If $\mathfrak{c}$ is indirect then we assume it involves $Q_{t-1}[i]$. Otherwise, $\mathfrak{c}$ is direct and no further assumption is required. Therefore $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$. If $|V_{t,\mathfrak{abc}}| = 1$, then there is no ambiguity and $\{g_i\} = V_{t,\mathfrak{abc}}$ and $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$. Otherwise, if $|V_{t,\mathfrak{abc}}| > 1$, then we choose g_i arbitrarily from $V_{t,\mathfrak{abc}}$ and we resolve the ambiguity by replacing bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ by $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = FC(t, \mathfrak{abc}, g_i)$. Note that in the next step $t+1$ our assumptions hold again, since $\widehat{\mathfrak{a}}$, $\widehat{\mathfrak{b}}$ and $\widehat{\mathfrak{c}}$ will be either direct or forward indirect bitconditions. More explicitly, $\widehat{\mathfrak{a}}$ is a direct bitcondition and if $\widehat{\mathfrak{b}}$ is indirect then it involves $\widehat{\mathfrak{a}}$.

Once all g_i and thus ΔF_t have been determined, δT_t is determined as $\delta F_t + \delta Q_{t-3} + \delta W_t$. We choose a high probability $\delta R_t \in dRL(\delta T_t, RC_t)$ and we determine $\delta Q_{t+1} = \delta Q_t + \delta R_t$.

6.2.4 Extending differential paths backward

Having dealt with the forward extension, we now consider the backward extension of a differential path. The backward construction follows the same approach as the forward one and is an adaptation of the algorithm in Section 5.6.2 using bitconditions.

Suppose we have a partial differential path consisting of at least bitconditions $\mathfrak{q}_t$ and $\mathfrak{q}_{t-1}$ and that the differences δQ_{t+1} and δQ_{t-2} are known. We want to extend this partial differential path backward with step t resulting in the difference δQ_{t-3}, bitconditions $\mathfrak{q}_{t-2}$, and additional bitconditions $\mathfrak{q}_t, \mathfrak{q}_{t-1}$. We assume that all indirect bitconditions in $\mathfrak{q}_t$ and $\mathfrak{q}_{t-1}$ are backward and only involve bits of Q_{t-1}.

We choose a low weight BSDR (such as the NAF) of δQ_{t-2}, which then translates into a possible choice for $\mathfrak{q}_{t-2}$ as in Table 6-1.

The differences $\Delta F_t = (g_i)_{i=0}^{31}$ are determined by assuming for $i = 0, 1, \ldots, 31$ that we have valid bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ where only $\mathfrak{a}$ may be indirect. If $\mathfrak{a}$ is indirect then we assume it involves $Q_{t-1}[i]$. Otherwise, $\mathfrak{a}$ is direct and no further assumption is required. Therefore $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$. If $|V_{t,\mathfrak{abc}}| = 1$, then there is no ambiguity and $\{g_i\} = V_{t,\mathfrak{abc}}$ and $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$. Otherwise, if $|V_{t,\mathfrak{abc}}| > 1$, then we choose g_i arbitrarily from $V_{t,\mathfrak{abc}}$ and we resolve the ambiguity by replacing bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ by $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = BC(t, \mathfrak{abc}, g_i)$.

To rotate $\delta R_t = \delta Q_{t+1} - \delta Q_t$ over $32 - RC_t$ bits, we choose a high probability $\delta T_t \in dRL(\delta R_t, 32 - RC_t)$. Finally, we determine $\delta Q_{t-3} = \delta T_t - \delta F_t - \delta W_t$ to extend our partial differential path backward with step t. Note that here also in the next step $t-1$ our assumptions hold again, since $\widehat{\mathfrak{a}}$, $\widehat{\mathfrak{b}}$ and $\widehat{\mathfrak{c}}$ will be either direct or backward indirect bitconditions. More explicitly, $\widehat{\mathfrak{c}}$ is a direct bitcondition and if $\widehat{\mathfrak{b}}$ is indirect then it involves $\widehat{\mathfrak{c}}$.

6.2.5 Constructing full differential paths

In this section we present an adaptation of the algorithm in Section 5.6.3 to connect a lower and upper partial differential path. The adapted algorithm in this section not only uses bitconditions, it also operates in a bit-wise manner instead of a word-wise manner. More precisely, instead of directly searching for compatible values ΔQ_i and ΔF_j that result in a valid full differential path as in Section 5.6.3, the algorithm presented here considers only bit position 0 in the beginning and searches for compatible bit values $\Delta Q_i[0]$ and $\Delta F_j[0]$ that could lead to a valid full differential path. It then iteratively extends such values with higher bit positions as long as there exist compatible values that could lead to a valid full differential path. This significantly reduces the cost of determining whether a lower and upper partial differential path can be connected.

Constructing a full valid differential path for MD5 can be done as follows. Assume that for some δQ_{-3} and bitconditions $\mathfrak{q}_{-2}$, $\mathfrak{q}_{-1}$, $\mathfrak{q}_0$ the forward construction has been carried out up to some step t. By default, we choose $t = 11$ such that this construction leaves as much freedom for message modification techniques as possible. Furthermore, assume that for some δQ_{64} and bitconditions $\mathfrak{q}_{63}, \mathfrak{q}_{62}, \mathfrak{q}_{61}$ the backward construction has been carried out down to step $t+5$. For each combination of forward and backward

Algorithm 6-1 Construction of $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$ for MD5.

Suppose $\mathcal{U}_i$ is given as $\{(\delta Q_{t+1},\ \delta Q_{t+2},\ \delta F_{t+1},\ \delta F_{t+2},\ \delta F_{t+3},\ \delta F_{t+4})\}$ if $i = 0$ or if $i > 0$ constructed inductively based on $\mathcal{U}_{i-1}$ by means of this algorithm. For each tuple $(q_1, q_2, f_1, f_2, f_3, f_4) \in \mathcal{U}_i$ do the following:

1. Let $\mathcal{U}_{i+1} = \emptyset$ and $(\mathfrak{a}, \mathfrak{b}, \mathfrak{e}, \mathfrak{f}) = (\mathfrak{q}_{t+4}[i], \mathfrak{q}_{t+3}[i], \mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i])$
2. For each bitcondition $\mathfrak{d} = \mathfrak{q}_{t+1}[i] \in \begin{cases} \{.\} & \text{if } q_1[i] = 0 \\ \{-,+\} & \text{if } q_1[i] = 1 \end{cases}$ do
3. Let $q_1' = 0, -1$ or $+1$ depending on whether $\mathfrak{d} =$ '.', '-' or '+', respectively
4. For each different $f_1' \in \{-f_1[i], +f_1[i]\} \cap V_{t+1,\mathfrak{def}}$ do
5. Let $(\mathfrak{d}', \mathfrak{e}', \mathfrak{f}') = FC(t+1, \mathfrak{def}, f_1')$
6. For each bitcondition $\mathfrak{c} = \mathfrak{q}_{t+2}[i] \in \begin{cases} \{.\} & \text{if } q_2[i] = 0 \\ \{-,+\} & \text{if } q_2[i] = 1 \end{cases}$ do
7. Let $q_2' = 0, -1$ or $+1$ depending on whether $\mathfrak{c} =$ '.', '-' or '+'
8. For each different $f_2' \in \{-f_2[i], +f_2[i]\} \cap V_{t+2,\mathfrak{cd}'\mathfrak{e}'}$ do
9. Let $(\mathfrak{c}', \mathfrak{d}'', \mathfrak{e}'') = FC(t+2, \mathfrak{cd}'\mathfrak{e}', f_2')$
10. For each different $f_3' \in \{-f_3[i], +f_3[i]\} \cap V_{t+3,\mathfrak{bc}'\mathfrak{d}''}$ do
11. Let $(\mathfrak{b}', \mathfrak{c}'', \mathfrak{d}''') = FC(t+3, \mathfrak{bc}'\mathfrak{d}'', f_3')$
12. For each different $f_4' \in \{-f_4[i], +f_4[i]\} \cap V_{t+4,\mathfrak{ab}'\mathfrak{c}''}$ do
13. Let $(\mathfrak{a}', \mathfrak{b}'', \mathfrak{c}''') = FC(t+4, \mathfrak{ab}'\mathfrak{c}'', f_4')$
14. Insert $(q_1 - 2^i q_1', q_2 - 2^i q_2', f_1 - 2^i f_1', f_2 - 2^i f_2', f_3 - 2^i f_3', f_4 - 2^i f_4')$ in $\mathcal{U}_{i+1}$

partial differential paths thus found, this leads to bitconditions $\mathfrak{q}_{-2}, \mathfrak{q}_{-1}, \ldots, \mathfrak{q}_t$ and $\mathfrak{q}_{t+3}, \mathfrak{q}_{t+4}, \ldots, \mathfrak{q}_{63}$ and differences δQ_{-3}, δQ_{t+1}, δQ_{t+2}, δQ_{64}.

It remains to try and glue together each of these combinations by finishing steps $t+1, t+2, t+3, t+4$ until a full differential path is found. First, as in the backward extension, for $i = t+1, t+2, t+3, t+4$ we set $\delta R_i = \delta Q_{i+1} - \delta Q_i$, choose a high probability $\delta T_i \in dRL(\delta R_i, 32 - RC_i)$, and determine $\delta F_i = \delta T_i - \delta W_i - \delta Q_{i-3}$.

We aim to complete the differential path by finding new bitconditions $\mathfrak{q}_{t-1}$, $\mathfrak{q}_t, \ldots,$ $\mathfrak{q}_{t+4}$ that are compatible with the original bitconditions and that result in the required δQ_{t+1}, δQ_{t+2}, δF_{t+1}, δF_{t+2}, δF_{t+3}, δF_{t+4}.

An efficient way to find the missing bitconditions is to first test if they exist, and if so to backtrack to actually construct them. For $i = 0, 1, \ldots, 32$ we attempt to construct a set $\mathcal{U}_i$ consisting of all tuples $(q_1, q_2, f_1, f_2, f_3, f_4)$ of 32-bit words with $q_j \equiv f_k \equiv 0 \bmod 2^i$ for $j = 1, 2$ and $k = 1, 2, 3, 4$ such that for all $\ell = 0, 1, \ldots, i-1$ there exist compatible bitconditions $\mathfrak{q}_{t-1}[\ell], \mathfrak{q}_t[\ell], \ldots, \mathfrak{q}_{t+4}[\ell]$ that determine $\Delta Q_{t+j}[\ell]$ and $\Delta F_{t+k}[\ell]$ below, and such that

$$\begin{cases} \delta Q_{t+j} &= q_j + \sum_{\ell=0}^{i-1} 2^\ell \Delta Q_{t+j}[\ell], \quad j = 1, 2, \\ \delta F_{t+k} &= f_k + \sum_{\ell=0}^{i-1} 2^\ell \Delta F_{t+k}[\ell], \quad k = 1, 2, 3, 4. \end{cases} \tag{6.4}$$

From these conditions it follows that $\mathcal{U}_0$ must be chosen as $\{(\delta Q_{t+1}, \delta Q_{t+2}, \delta F_{t+1}, \delta F_{t+2}, \delta F_{t+3}, \delta F_{t+4})\}$. For $i = 1, 2, \ldots, 32$, we attempt to construct $\mathcal{U}_i$ based on $\mathcal{U}_{i-1}$ using Algorithm 6-1. Because per j there are at most two q_j-values and per k there are at most two f_k-values that can satisfy the above relations, we have that $|\mathcal{U}_i| \leq 2^6$ for each i, $0 \leq i \leq 32$. On the other hand, for each tuple in $\mathcal{U}_i$ there may in principle be many different compatible sets of bitconditions, thus providing a significant speedup compared to Section 5.6.3.

As soon as we encounter an i for which $\mathcal{U}_i = \emptyset$, we know that the desired bitconditions do not exist, and that we should try another combination of forward and backward partial differential paths. If, however, we find $\mathcal{U}_{32} \neq \emptyset$ then it must be the case that $\mathcal{U}_{32} = \{(0, 0, 0, 0, 0, 0)\}$. Furthermore, in that case, every set of bitconditions that leads to this non-empty $\mathcal{U}_{32}$ gives rise to a full differential path, since equations (6.4) hold with $i = 32$. Thus, if $\mathcal{U}_{32} \neq \emptyset$, there exists at least one valid trail $u_0, u_1, \ldots, u_{32}$ with $u_i \in \mathcal{U}_i$. For each valid trail, the desired new bitconditions $(\mathfrak{q}_{t+4}[i], \mathfrak{q}_{t+3}[i], \ldots, \mathfrak{q}_{t-1}[i])$ are $(\mathfrak{a}', \mathfrak{b}'', \mathfrak{c}''', \mathfrak{d}''', \mathfrak{e}'', \mathfrak{f}')$, which can be found at step 13 of Algorithm 6-1.

Note that in Algorithm 6-1 we have that $(\mathfrak{d}, \mathfrak{e}, \mathfrak{f}), (\mathfrak{c}, \mathfrak{d}', \mathfrak{e}') \in \mathcal{L}$ for the same reasons as in Section 6.2.3. Using the same assumption as in Section 6.2.4 that $\mathfrak{b}$ is a direct bitcondition and $\mathfrak{a}$ is either a direct bitcondition or a backward indirect bitcondition involving $\mathfrak{b}$. It follows that also $(\mathfrak{b}, \mathfrak{c}', \mathfrak{d}''), (\mathfrak{a}, \mathfrak{b}', \mathfrak{c}'') \in \mathcal{L}$.

6.2.6 Complexity

The complexity to construct valid full differential paths for MD5 depends on many factors as is explained in Section 5.6.4. For our chosen-prefix collision construction (see Section 6.5), we have heuristically found parameter choices that are sufficient for most cases and lead to an average complexity equivalent to about 2^{35} MD5 compression function calls:

- *number of differential paths*: 10^6 lower differential paths and $0.5 \cdot 10^6$ upper differential paths. We chose for a relatively lower number of upper differential paths as these include more differential steps and thus require more memory to be stored.

- *differential path weight function*: the weight of a partial differential path over steps $t = b, \ldots, e$ is defined as the number of non-constant bitconditions in $\mathfrak{q}_{b-2}, \ldots, \mathfrak{q}_e$ which excludes the weight of the NAF of δQ_{b-3} and δQ_{e+1}. The output per differential step of the forward and backward construction is thus the 10^6 or $0.5 \cdot 10^6$, respectively, partial differential paths with the lowest such weight. This particular choice allows us to efficiently determine an upper limit for this weight such that the desired number of differential paths can be found for each differential step in the forward and backward construction. Having this upper weight limit, the forward and backward construction can skip all input differential paths and/or $\Delta F_t[i]$ choices that lead to an extended differential path with weight above the limit. This leads to a significant speedup.

- *BSDRs of δQ_i*: the set of allowed BSDRs of δQ_i where $i = t$ or $i = t - 2$ in the forward or backward construction, respectively, is determined as the set of all BSDRs of δQ_i having weight less than or equal to ω. The limit ω is initially chosen as $w(\mathrm{NAF}(\delta Q_i)) + 1$ to allow at least some variability. Then ω is repeatedly increased by 1 while both $\omega < 14$ and

$$\left|\left\{\text{BSDR } X \;\middle|\; \sigma(X) = \delta Q_i \wedge w(X) \leq \omega + 1\right\}\right| \leq 16.$$

- *message modification freedom*: the message modification techniques shown in Section 6.3.1 depend on bitconditions. The maximum possible combined tunnel strength can be determined after each step of the forward and connect construction and every (partial) differential path that falls below a certain threshold can be skipped.

6.3 Collision finding

Collision finding is the process of finding an actual message block pair B, B' that satisfies a given δB and a differential path based on a given $IHV_{\text{in}}, IHV'_{\text{in}}$.

6.3.1 Tunnels

To speed up to search of collision blocks we make extensive use of so-called *tunnels* [Kli06]. A tunnel allows one to make small changes in a certain first round Q_t, in specific bits of Q_t that are determined by the full differential path $\mathfrak{q}_{-3}, \mathfrak{q}_{-2}, \ldots, \mathfrak{q}_{64}$ under consideration, while causing changes in the second round only after some step l that depends on the tunnel. However, each tunnel implies that additional first-round bitconditions have to be taken into account in the differential path, while leaving freedom of choice for some of the bits in Q_t that may be changed. A tunnel's *strength* is the number of independent bits that can be changed in this first round Q_t. Thus, a tunnel of strength k allows us to generate 2^k different message blocks that all satisfy the differential path up to and including step l in the second round.

The tunnels used in our collision finding algorithm are shown in Table 6-3. For example, the first tunnel ($\mathcal{T}_1$) allows changes in bits of Q_4, in such a way that if $Q_4[b]$ is changed for some bit position b with $0 \leq b < 32$, this causes extra bitconditions $Q_5[b] = 1$ and $Q_6[b] = 1$, which have to be incorporated in the differential path. Furthermore, because tunnel $\mathcal{T}_1$ affects after the first round only Q_{21} through Q_{64} we have that $l = 20$, and $\mathcal{T}_1$ can be used to change message blocks m_3, m_4, m_5, and m_7. To determine the strength of a tunnel one first needs to incorporate the tunnel's extra bitconditions in the full differential path, and then count the remaining amount of freedom in the first round Q_t that is changed by the tunnel.

The most effective tunnel is $\mathcal{T}_8$. As indicated in the table, it affects after the first round only $Q_{25}, \ldots, Q_{64}$. Over these rounds, Wang et al.'s original differential paths have 20 bitconditions whereas the chosen-prefix collision differential paths that we manage to construct have approximately 27 bitconditions. It follows that, given

Table 6-3: *Collision finding tunnels for MD5.*

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

$\star$ The extra bitconditions refer only to $Q_t[b]$ and not to $Q'_t[b]$, thus $Q_6[b] = 0$ is met by both $\mathfrak{q}_6[b] =$ '0' and $\mathfrak{q}_6[b] =$ '+'.
† Bitcondition $\mathfrak{q}_3[b] =$ '.' and no other indirect bitconditions may involve $Q_3[b]$. Set $Q_3[b] = Q_{14}[b]$ to avoid carries in Q_3.
‡ Bitcondition $\mathfrak{q}_{12}[b - 22 \bmod 32] =$ '.' and no other indirect bitconditions may involve $Q_{12}[b - 22 \bmod 32]$. Set $Q_{12}[b - 22 \bmod 32] = Q_8[b]$ to avoid carries in Q_{12}.

enough tunnel strength, especially for $\mathcal{T}_7$ and $\mathcal{T}_8$, collision finding can be done efficiently.

6.3.2 Algorithm

The conditions on the differential path imposed by Algorithm 6-2 can easily be met because forward and backward bitconditions in the differential path are interchangeable. Steps 10 through 15 of Algorithm 6-2 are its most computationally intensive part, in particular for the toughest differential paths in a chosen-prefix collision, so they should be optimized. Greater tunnel strength significantly reduces the expected time spent there, because all tunnels are used in steps 16 through 26. This implies that there are significantly more chances at success in step 32 per successful step 15.

Note that in Algorithm 6-2, computation of m_i and Q_i is performed at $t = i$ and $t = i - 1$, respectively. Furthermore, we assume that the rotations in the first round have probability very close to 1 to be correct, and therefore do not verify them. This is further explained in Section 6.3.3.

6.3.3 Rotation bitconditions

As mentioned below Algorithm 6-2, it is assumed there that all rotations in the first round are correct with probability very close to 1. In Algorithm 6-2, $Q_1, \ldots, Q_{16}$ are chosen in a non-sequential order and also changed at various steps in the algorithm. Ensuring correct rotations in the first round would be cumbersome and it would hardly avoid wasting time in a state where one or more rotations in the first round would fail due to the various tunnels. However, if we use additional bitconditions $\mathfrak{q}_t[i]$ we can (almost) ensure correct rotations in the first round, thereby (almost) eliminating both the effort to verify rotations and the wasted computing time. This is explained below.

Algorithm 6-2 Collision finding algorithm.

Given a full differential path $\mathtt{q}_{-3}, \ldots, \mathtt{q}_{64}$ consisting of only direct and backward bitconditions and the set $\mathcal{T}_1, \ldots, \mathcal{T}_8$ of tunnels from Table 6-3, perform the following steps:

1. Determine for all tunnels for which bits b the extra bitconditions as shown in Table 6-3 can be met. For each possible case, apply compatible bitconditions to enforce the extra bitconditions and change the bitconditions $\mathtt{q}_t[b]$ of the changed or affected $Q_t[b]$ in the first round from '`.`' to '`0`'.
2. Perform the steps below until a collision block has been found.
3. Select $Q_1, Q_2, Q_{13}, \ldots, Q_{16}$ such that $\mathtt{q}_1, \mathtt{q}_2, \mathtt{q}_{13}, \ldots, \mathtt{q}_{16}$ hold.
4. Compute m_1, Q_{17}.
5. If $\mathtt{q}_{17}$ holds and the rotation for $t = 16$ is successful, then proceed.
6. Store the set $\mathcal{Z}$ of all pairs (Q_1, Q_2) meeting $\mathtt{q}_1, \mathtt{q}_2$ that do not change m_1 and bits of Q_2 involved in $\mathtt{q}_3$.
7. For all $Q_3, \ldots, Q_7$ meeting $\mathtt{q}_3, \ldots, \mathtt{q}_7$ do:
8. Compute m_6, Q_{18}.
9. If $\mathtt{q}_{18}$ holds and the rotation for $t = 17$ is successful, then proceed.
10. For all $Q_8, \ldots, Q_{12}$ meeting $\mathtt{q}_8, \ldots, \mathtt{q}_{12}$ do:
11. Compute m_{11}, Q_{19}.
12. If $\mathtt{q}_{19}$ holds and the rotation for $t = 18$ is successful, then proceed.
13. For all (Q_1, Q_2) in $\mathcal{Z}$ do:
14. Compute m_0, Q_{20}.
15. If $\mathtt{q}_{20}$ holds and the rotation for $t = 19$ is successful, then proceed.
16. For all values of the bits of tunnels $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3$ do:
17. Set the bits to those values and compute m_5, Q_{21}.
18. If $\mathtt{q}_{21}$ holds and the rotation for $t = 20$ is successful, then proceed.
19. For all values of the bits of tunnels $\mathcal{T}_4, \mathcal{T}_5$ do:
20. Set the bits to those values and compute m_{10}, Q_{22}.
21. If $\mathtt{q}_{22}$ holds and the rotation for $t = 21$ is successful, then proceed.
22. For all values of the bits of tunnel $\mathcal{T}_6$ do:
23. Set the bits to those values and compute m_{15}, Q_{23}.
24. If $\mathtt{q}_{23}$ holds and the rotation for $t = 22$ is successful, then proceed.
25. For all values of the bits of tunnel $\mathcal{T}_7$ do:
26. Set the bits to those values and compute m_4, Q_{24}.
27. If $\mathtt{q}_{24}$ holds and the rotation for $t = 23$ is successful, then proceed.
28. For all values of the bits of tunnel $\mathcal{T}_8$ do:
29. Set the bits to those values and compute m_9, Q_{25}.
30. If $\mathtt{q}_{25}$ holds and the rotation for $t = 24$ is successful, then proceed.
31. Compute $m_0, \ldots, m_{15}$, $Q_{26}, \ldots, Q_{64}$ and $Q'_1, \ldots, Q'_{64}$.
32. If $\delta\widehat{Q}_t = Q'_t - Q_t$ agrees with $\mathtt{q}_t$ for $t = 61, 62, 63, 64$, return B, B'.

Using notation as in the proof of Lemma 5.4, given δT_t and δR_t it is easy to determine which partition (α, β) satisfies $RL((\alpha, \beta), RC_t) = \delta R_t$. The probability that this correct rotation holds is not necessarily $p_{(\alpha,\beta)}$ because it may be assumed that bitconditions $\mathfrak{q}_t$ and $\mathfrak{q}_{t+1}$ hold and these directly affect $R_t = Q_{t+1} - Q_t$ and thus $T_t = RR(R_t, RC_t)$. Hence, using bitconditions $\mathfrak{q}_t$ and $\mathfrak{q}_{t+1}$ we can try and increase the probability of a correct rotation in step t to (almost) 1 in the following way.

The other three partitions (of the four listed in Lemma 5.4) correspond to the incorrect rotations. Those partitions are of the form

$$(\widehat{\alpha}, \widehat{\beta}) = (\alpha - \lambda_0 2^{32-RC_t}, \beta + \lambda_0 2^{32-RC_t} + \lambda_{RC_t} 2^{32}), \quad \lambda_0, \lambda_{RC_t} \in \{-1, 0, +1\}$$

where either $\lambda_0 \neq 0$ or $\lambda_{RC_t} \neq 0$. They result in incorrect $\delta\widehat{R}_t$ of the form

$$\delta\widehat{R}_t = RL((\widehat{\alpha}, \widehat{\beta}), RC_t) = \delta R_t + \lambda_0 2^0 + \lambda_{RC_t} 2^{RC_t}.$$

They are caused by a carry when adding δT_t to T_t that does or does not propagate: from bit position $32 - RC_t - 1$ to $32 - RC_t$ for $\lambda_0 \neq 0$ and from bit position 31 to 32 for $\lambda_{RC_t} \neq 0$. Since we chose the partition (α, β) with highest probability, this usually means that we have to prevent instead of ensure those propagations in order to decrease the probability that $\lambda_0 \neq 0$ or $\lambda_{RC_t} \neq 0$.

To almost guarantee proper rotations in each step of Algorithm 6-2, additional bitconditions can be determined by hand. Adding bitconditions on Q_t, Q_{t+1} around bit positions $31 - RC_t + i$ and lower helps preventing $\lambda_i \neq 0$. This can be automated using a limited brute-force search, separately handling the cases $\lambda_0 \neq 0$ and $\lambda_{RC_t} \neq 0$.

Let $i \in \{0, RC_t\}$. Given bitconditions $\mathfrak{q}_t, \mathfrak{q}_{t+1}$, we estimate $\Pr[\lambda_i \neq 0 | \mathfrak{q}_t, \mathfrak{q}_{t+1}]$ by sampling a small set of $\widehat{Q}_t, \widehat{Q}_{t+1}$ satisfying $\mathfrak{q}_t, \mathfrak{q}_{t+1}$, and determining the fraction where $\lambda_i = \mathrm{NAF}(\delta\widehat{R}_t - \delta R_t)[i] \neq 0$ using

$$\begin{aligned} \widehat{T}_t &= RR(\widehat{Q}_{t+1} - \widehat{Q}_t, RC_t); \\ \delta\widehat{R}_t &= RL(\widehat{T}_t + \delta T_t, RC_t) - RL(\widehat{T}_t, RC_t). \end{aligned}$$

Using this approach, we estimate the probability that $\lambda_i = 0$ by selecting a small search bound B and exhaustively trying all combinations of additional bitconditions on $Q_t[b], Q_{t+1}[b]$ for $b = 31 - RC_t + i - B, \ldots, 31 - RC_t + i$. Finally, if there are any bitconditions $(\mathfrak{q}'_t, \mathfrak{q}'_{t+1})$ for which $\Pr[\lambda_i \neq 0 | \mathfrak{q}'_t, \mathfrak{q}'_{t+1}]$ is negligible, we select the pair $(\mathfrak{q}'_t, \mathfrak{q}'_{t+1})$ that leads to the smallest number of additional bitconditions and for which $\Pr[\lambda_0 = \lambda_{RC_t} = 0 | \mathfrak{q}_{t-1}, \mathfrak{q}'_t]$ and $\Pr[\lambda_0 = \lambda_{RC_t} = 0 | \mathfrak{q}'_{t+1}, \mathfrak{q}_{t+2}]$ do not decrease significantly for step $t - 1$ and $t + 1$, respectively.

6.4 Identical-prefix collision attack

We first present a new collision attack for MD5 with complexity of approximately 2^{16} MD5 compressions improving upon the $2^{20.96}$ MD5 compressions required in [XLF08]. Our starting point is the partial differential path for MD5 given in Table 6-4 with

success probability approximately $2^{-14.5}$.[19] It is based on message differences $\delta m_2 = 2^8$, $\delta m_4 = \delta m_{14} = 2^{31}$ and $\delta m_{11} = 2^{15}$ which is very similar to those used by Wang et al. in [WY05] for the first collision attack against MD5. This partial differential path can be used for a near-collision attack with complexity of approximately $2^{14.8}$ MD5 compressions. This complexity estimate is based on the partial differential path success probability and the influence of additional bitconditions prior step 28, tunnels and the early-stop technique as in Algorithm 6-2. An example full differential path based on this partial differential path, but with alternate steps 60–63, is given in Table 6-8 (p. 113). This example full differential path has very few bitconditions on $Q_{18}, \ldots, Q_{24}$ so that most of the time is spent around tunnel $\mathcal{T}_8$ (in this case with strength 13) which alters Q_{25}. From step 24 onwards, the average number of steps computed per value of $\mathcal{T}_8$ is approximately 3. The complexity over steps 24–63 is thus the average complexity over steps 24–64 expressed as compression function calls (3/64) divided by the success probability over steps 24–63 ($2^{-14.5} \cdot 2^{-4} = 2^{-18.5}$) leading to $(3/64) \cdot 2^{-18.5} = 2^{14.1}$. Together with previous steps this leads to a near-collision complexity equivalent to approximately $2^{14.8}$ compression function calls.

This leads in the usual fashion to an identical-prefix collision attack for MD5 that requires approximately 2^{16} MD5 compressions, since one has to do it twice: first to add differences to δIHV and then to eliminate them again. It should be noted that usually bitconditions are required on the IHV and IHV' between the two collision blocks which imply an extra factor in complexity. In the present case, however, we can construct a large set of differential paths for the second near-collision block that cover all IHV and IHV' values that are likely to occur, thereby avoiding the extra complexity at the cost of precomputations.

6.5 Chosen-prefix collision attack

This section is dedicated to the removal of the identical prefix condition to attain MD5 chosen-prefix collisions. We show how *any* pair of IHV values can be made to collide under MD5 by appending properly chosen collision blocks. More precisely, we show how, for any two chosen message prefixes P and P', suffixes S and S' can be constructed such that the concatenated values $P\|S$ and $P'\|S'$ form a chosen-prefix collision for MD5.

Given two arbitrarily chosen messages P and P', to construct a chosen-prefix collision, we first apply padding to the shorter of the two, if any, to make their lengths equal. This ensures that the Merkle-Damgård strengthening – which is applied after the last bits of the message and involves the message's bitlength – is identical for the two messages resulting from this construction. We apply additional padding such that both resulting messages are a specific number of bits (such as 64 or 96) short of a whole number of blocks. In principle this can be avoided, but it leads to an efficient method that allows relatively easy presentation. All these requirements can easily be met, also in applications with stringent formatting restrictions.

19. As steps 29–33 hold with probability 1, steps 34–63 hold with probability $2^{-14.5}$.

Table 6-4: *Partial differential path for fast near-collision attack.*

t	δQ_t	δF_t	δW_t	δT_t	δR_t	RC_t
26	-2^8					
27	0					
28	0					
29	0	0	2^8	0	0	9
$30-33$	0	0	0	0	0	·
34	0	0	2^{15}	2^{15}	2^{31}	16
35	2^{31}	2^{31}	2^{31}	0	0	23
36	2^{31}	0	0	0	0	4
37	2^{31}	2^{31}	2^{31}	0	0	11
$38-46$	2^{31}	2^{31}	0	0	0	·
47	2^{31}	2^{31}	2^8	2^8	2^{31}	23
48	0	0	0	0	0	6
49	0	0	0	0	0	10
50	0	0	2^{31}	0	0	15
$51-59$	0	0	0	0	0	·
60	0	0	2^{31}	2^{31}	-2^5	6
61	-2^5	0	2^{15}	2^{15}	2^{25}	10
62	-2^5+2^{25}	0	2^8	2^8	2^{23}	15
63	$-2^5+2^{25}+2^{23}$	2^5-2^{23}	0	2^5-2^{23}	$2^{26}-2^{14}$	21
64	$-2^5+2^{25}+2^{23}+2^{26}-2^{14}$					

Partial differential path for $t = 29, \ldots, 63$ using message differences $\delta m_2 = 2^8$, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = 2^{15}$. The probability that it is satisfied is approximately $2^{-14.5}$. It leads to a identical-prefix collision attack of approximated complexity 2^{16} MD5 compressions.

Given this message pair, we modify a suggestion by Xiaoyun Wang (private communication) by finding a pair of k-bit values that, when appended to the last incomplete message blocks, results in a specific form of difference vector between the IHV values after application of the MD5 compression function to the extended message pair. Finding the k-bit appendages can be done using a birthday search procedure.

The specific form of difference vector between the IHV values that is aimed for during the birthday search is such that the difference pattern can be removed using a to-be defined family of differential paths. Removing the difference pattern is done by further appending to the messages a sequence of *near-collision blocks*. Each pair of near-collision blocks targets a specific subpattern of the remaining differences. For each such subpattern we construct a new differential path, as described in detail in Section 6.2, and subsequently use the differential path to construct a pair of near-collision blocks. Appending those blocks to the two messages results in a new difference vector between the new IHV values from which the targeted subpattern has been eliminated compared to the previous difference vector between the IHV values. The construction continues as long as differences exist.

How the various steps involved in this construction are carried out and how their parameters are tuned depends on what needs to be optimized. Extensive birthday searching can be used to create difference patterns that require a small number of pairs of near-collision blocks. When combined with a properly chosen large family of differential paths, a single pair of near-collision blocks suffices to complete the collision right away. One can trade-off between the size s of the family of differential paths and the birthday search cost: $2^{64.8}/\sqrt{|s|}$ MD5 compressions. However, it may make the actual near-collision block construction quite challenging, which leads to the intuitively expected result that finding very short chosen-prefix collision-causing appendages is relatively costly. On the other side of the spectrum, fast birthday searching combined with a smaller family of differential paths leads to the need for many successive pairs of near-collision blocks, each of which can quickly be found: if one is willing to accept long chosen-prefix collision-causing appendages, the overall construction can be done quite fast. Between the two extremes almost everything can be varied: number of near-collision blocks, their construction time given the differential path, time to find the full differential path, birthday search time, birthday search space requirements, etc., leading to a very wide variety of 'optimal' choices depending on what needs to be optimized.

Eliminating the birthday search entirely may be possible. However, this is not interesting, since it requires additional more-complex differential paths and needs on average even more near-collision blocks. In comparison, the lowest birthday search cost of $2^{32.9}$ MD5 compressions will not be a significant portion of the total chosen-prefix complexity.

6.5.1 Construction details

As shown in Figure 8, a chosen-prefix collision for MD5 is a pair of messages M and M' that consist of arbitrarily chosen prefixes P and P' (not necessarily of the same length), together with constructed suffixes S and S', such that $M = P\|S$, $M' = P'\|S'$, and $\mathrm{MD5}(M) = \mathrm{MD5}(M')$. The suffixes consist of three parts: padding bit strings $S_\mathrm{r}, S'_\mathrm{r}$, followed by 'birthday' bit strings $S_\mathrm{b}, S'_\mathrm{b}$ both of bit length $64 + k$, where $0 \leq k \leq 32$ is a parameter, followed by bit strings $S_\mathrm{c}, S'_\mathrm{c}$ each consisting of a sequence of near-collision blocks. The padding bit strings are chosen such that the bit lengths of $P\|S_\mathrm{r}$ and $P'\|S'_\mathrm{r}$ are both equal to $512n - 64 - k$ for a positive integer n. The birthday bit strings $S_\mathrm{b}, S'_\mathrm{b}$ are determined in such a way that application of the MD5 compression function to $P\|S_\mathrm{r}\|S_\mathrm{b}$ and $P'\|S'_\mathrm{r}\|S'_\mathrm{b}$ results in IHV_n and IHV'_n, respectively, for which δIHV_n has a certain desirable property that is explained below.

The idea is to eliminate the difference δIHV_n in r consecutive steps, for some r, by writing $S_\mathrm{c} = S_{\mathrm{c},1}\|S_{\mathrm{c},2}\|\ldots\|S_{\mathrm{c},r}$ and $S'_\mathrm{c} = S'_{\mathrm{c},1}\|S'_{\mathrm{c},2}\|\ldots\|S'_{\mathrm{c},r}$ for r pairs of near-collision blocks $(S_{\mathrm{c},j}, S'_{\mathrm{c},j})$ for $1 \leq j \leq r$. For each pair of near-collision blocks $(S_{\mathrm{c},j}, S'_{\mathrm{c},j})$ we need to construct a differential path such that the difference vector δIHV_{n+j} has lower weight than δIHV_{n+j-1}, until after r pairs we have reached $\delta IHV_{n+r} = (0, 0, 0, 0)$.

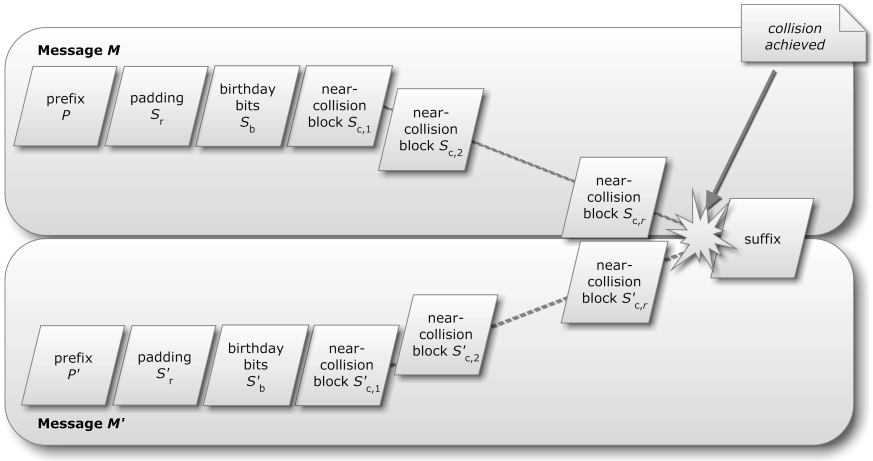

Figure 8: *Chosen-prefix collision sketch*

Fix some j and let $S_{\mathrm{c},j}$ consist of 32-bit words m_i, for $0 \leq i < 16$. We fix fifteen of the δm_i as 0 and allow only δm_{11} to be $\pm 2^{p-10 \bmod 32}$ with as yet unspecified p with $0 \leq p < 32$. This was suggested by Xiaoyun Wang because with this type of message difference the number of bitconditions over the final two and a half rounds can be kept low, which turns out to be helpful while constructing collisions. For steps $t = 34$ up to $t = 61$ the differential path is fully determined by δm_{11} as illustrated in Table 6-5. The greater variability for the steps not specified in Table 6-5 does not need to be fixed at this point. In the last two steps there is a greater degree of freedom specified by the integer $w \geq 0$ that determines which and how many IHV differences can be eliminated per pair of near-collision blocks. A larger w allows more eliminations by means of additional differential paths. The latter have, however, a smaller chance to be satisfied because they depend on more (and thus less likely) carry propagations in ΔQ_{62} and ΔQ_{63}. This effect contributes to the complexity of finding the near-collision blocks satisfying the differential paths. Varying w therefore leads to a trade-off between fewer near-collision blocks and increased complexity to find them.

This entire construction of the pair of near-collision blocks $(S_{\mathrm{c},j}, S'_{\mathrm{c},j})$ is done in a fully automated way based on the choice of w and the values of IHV_{n+j-1} and IHV'_{n+j-1} as specified. It follows from equation 6.3 (p. 91) and the rows for $t \geq 61$ in Table 6-5 that a differential path with $\delta m_{11} = \pm 2^{p-10 \bmod 32}$ would add a tuple

$$\pm \left(0, 2^p + \sum_{\lambda=0}^{w'} s_\lambda 2^{p+21+\lambda \bmod 32}, 2^p, 2^p \right)$$

to δIHV_{n+j-1}, with notation as in Table 6-5. This is set forth in more detail below. A sequence of such tuples is too restrictive to eliminate arbitrary δIHV_n: although

Table 6-5: *Family of partial differential paths using* $\delta m_{11} = \pm 2^{p-10 \bmod 32}$.

t	δQ_t	δF_t	δW_t	δT_t	δR_t	RC_t
31	$\mp 2^{p-10 \bmod 32}$					
32	0					
33	0					
34	0	0	$\pm 2^{p-10 \bmod 32}$	0	0	16
$35-60$	0	0	0	0	0	$\cdot$
61	0	0	$\pm 2^{p-10 \bmod 32}$	$\pm 2^{p-10 \bmod 32}$	$\pm 2^p$	10
62	$\pm 2^p$	0	0	0	0	15
63	$\pm 2^p$	0	0	0	0	21
64	$\pm 2^p$ $+\sum_{\lambda=0}^{w'} s_\lambda 2^{p+21+\lambda \bmod 32}$					

Here $s_0, \ldots, s_{w'} \in \{-1, 0, +1\}$ and $w' = \min(w, 31-p)$ for a fixed $w \geq 0$. Interesting values for the parameter w are between 2 and 5.

differences in the b component can be handled using a number of near-collision block pairs, only identical differences can be removed from the c and d components and the a-component differences are not affected at all. We therefore make sure that δIHV_n has the desirable property, as referred to above, that it *can* be eliminated using these tuples. This is done in the birthday search step where birthday bit strings S_b and S'_b are determined such that $\delta IHV_n = (0, \delta b, \delta c, \delta c)$ for some δb and δc. A δIHV_n of this form corresponds to a collision $(a, c-d) = (a', c'-d')$ between $IHV_n = (a, b, c, d)$ and $IHV'_n = (a', b', c', d')$. With a search space of only 64 bits, such a collision can easily be found. Since the number of near-collision block pairs and the effort required to find them depends in part on the number of bit differences between δb and δc, it may pay off to lower that number at the cost of extending the birthday search space. For instance, for any k with $0 \leq k \leq 32$, a collision

$$(a, c-d, c-b \bmod 2^k) = (a', c'-d', c'-b' \bmod 2^k)$$

with a $(64+k)$-bit search space results in $\delta c - \delta b \equiv 0 \bmod 2^k$ and thus, on average, just $(32-k)/3$ bit differences between δb and δc. Determining such S_b and S'_b can be expected to require on the order of $\sqrt{2\frac{\pi}{2}2^{64+k}} = \sqrt{\pi}2^{32+(k/2)}$ calls to the MD5 compression function. More on the birthday search in Section 6.5.2.

Let δIHV_n be of the form $(0, \delta b, \delta c, \delta c)$, $\delta c = \sum_i k_i 2^i$ and $\delta b - \delta c = \sum_i l_i 2^i$, where $(k_i)_{i=0}^{31}$ and $(l_i)_{i=0}^{31}$ are NAFs. If $\delta c \neq 0$, let i be such that $k_i \neq 0$. Using a differential path from Table 6-5 with $\delta m_{11} = -k_i 2^{i-10 \bmod 32}$ we can eliminate the difference $k_i 2^i$ in δc and δd and simultaneously change δb by

$$k_i 2^i + \sum_{\lambda=i+21 \bmod 32}^{i+21+w' \bmod 32} l_\lambda 2^\lambda,$$

where $w' = \min(w, 31-i)$. Here one needs to be careful that each non-zero l_λ is

Algorithm 6-3 Construction of pairs of near-collision blocks.

Given n-block $P\|S_\mathrm{r}\|S_\mathrm{b}$ and $P'\|S'_\mathrm{r}\|S'_\mathrm{b}$, the corresponding resulting IHV_n and IHV'_n, and a value for w, a pair of bit strings $S_\mathrm{c}, S'_\mathrm{c}$ is constructed consisting of sequences of near-collision blocks such that $M = P\|S_\mathrm{r}\|S_\mathrm{b}\|S_\mathrm{c}$ and $M' = P'\|S'_\mathrm{r}\|S'_\mathrm{b}\|S'_\mathrm{c}$ satisfy $\mathrm{MD5}(M) = \mathrm{MD5}(M')$. This is done by performing in succession steps 1, 2 and 3 below.

1. Let $j = 0$ and let S_c and S'_c be two bit strings of length zero.
2. Let $\delta IHV_{n+j} = (0, \delta b, \delta c, \delta c)$. If $\delta c = 0$ then proceed to step 3. Let $(k_i)_{i=0}^{31} = \mathrm{NAF}(\delta c)$ and $(l_i)_{i=0}^{31} = \mathrm{NAF}(\delta b - \delta c)$. Choose any i for which $k_i \neq 0$ and let $w' = \min(w, 31-i)$. Perform steps (a) through (f):
   (a) Increase j by 1.
   (b) Let $\delta S_{\mathrm{c},j} = (\delta m_0, \delta m_1, \ldots, \delta m_{15})$ with $\delta m_{11} = -k_i 2^{i-10 \bmod 32}$ and $\delta m_t = 0$ for $0 \leq t < 16$ and $t \neq 11$.
   (c) Given $\delta IHV_{n+j-1} = IHV'_{n+j-1} - IHV_{n+j-1}$ and $\delta S_{\mathrm{c},j}$, construct a few differential paths based on Table 6-5 with
   $$\delta Q_{61} = 0, \quad \delta Q_{64} = -k_i 2^i - \sum_{\lambda = i+21 \bmod 32}^{i+21+w' \bmod 32} l_\lambda 2^\lambda, \quad \delta Q_{63} = \delta Q_{62} = -k_i 2^i.$$
   (d) Find message blocks $S_{\mathrm{c},j}$ and $S'_{\mathrm{c},j} = S_{\mathrm{c},j} + \delta S_{\mathrm{c},j}$ that satisfy one of the constructed differential paths. If proper message blocks cannot be found, back up to step (c) to find more differential paths.
   (e) Compute $IHV_{n+j} = \mathrm{MD5Compress}(IHV_{n+j-1}, S_{\mathrm{c},j})$, $IHV'_{n+j} = \mathrm{MD5Compress}(IHV'_{n+j-1}, S'_{\mathrm{c},j})$, and append $S_{\mathrm{c},j}$ and $S'_{\mathrm{c},j}$ to S_c and S'_c, respectively.
   (f) Repeat step 2
3. Let $\delta IHV_{n+j} = (0, \delta\hat{b}, 0, 0)$. If $\delta\hat{b} = 0$ then terminate. Let $(l_i)_{i=0}^{31} = \mathrm{NAF}(\delta\hat{b})$. Choose i such that $l_i \neq 0$ and $i - 21 \bmod 32$ is minimal and let $w' = \min(w, 31 - (i - 21 \bmod 32))$. Perform steps (a) through (e) as above with $\delta m_{11} = 2^{i-31 \bmod 32}$ as opposed to $\delta m_{11} = -k_i 2^{i-10 \bmod 32}$ in step (b) and in steps (c) and (d) with
   $$\begin{aligned} \delta Q_{61} &= 0; \\ \delta Q_{64} &= 2^{i-21 \bmod 32} - \sum_{\lambda = i}^{i+w' \bmod 32} l_\lambda 2^\lambda; \\ \delta Q_{63} &= 2^{i-21 \bmod 32}; \\ \delta Q_{62} &= 2^{i-21 \bmod 32}. \end{aligned}$$
   Perform steps (a) through (e) again with $\delta m_{11} = -2^{i-31 \bmod 32}$ in step (b) and
   $$\delta Q_{61} = 0, \quad \delta Q_{64} = \delta Q_{63} = \delta Q_{62} = -2^{i-21 \bmod 32}$$
   in steps (c) and (d). Repeat step 3.

eliminated only once in the case when multiple i-values allow the elimination of l_λ. Doing this for all k_i that are non-zero in the NAF of δc results in a difference vector $(0, \widehat{\delta b}, 0, 0)$ where $\widehat{\delta b}$ may be different from δb, and where the weight $w(\mathrm{NAF}(\widehat{\delta b}))$ may be smaller or larger than $w(\mathrm{NAF}(\delta b))$. More precisely, $\widehat{\delta b} = \sum_{\lambda=0}^{31} e_\lambda l_\lambda 2^\lambda$, where $e_\lambda = 0$ if there exist indices i and j with $0 \leq j \leq \min(w, 31-i)$ such that $k_i = \pm 1$ and $\lambda = 21 + i + j \bmod 32$ and $e_\lambda = 1$ otherwise.

The bits in $\widehat{\delta b}$ can be eliminated as follows. Let $(\widehat{l}_i)_{i=0}^{31} = \mathrm{NAF}(\widehat{\delta b})$ and let j be such that $\widehat{l}_j = \pm 1$ and $j - 21 \bmod 32$ is minimal. Then the difference $\sum_{i=j}^{j+w'} \widehat{l}_i 2^i$ with $w' = \min(w, 31 - (j - 21 \bmod 32))$ can be eliminated from $\widehat{\delta b}$ using $\delta m_{11} = 2^{j-31 \bmod 32}$, which introduces a new difference $2^{j-21 \bmod 32}$ in δb, δc and δd. This latter difference is eliminated using $\delta m_{11} = -2^{j-31 \bmod 32}$, which then leads to a new difference vector $(0, \delta\bar{b}, 0, 0)$ with $w(\mathrm{NAF}(\delta\bar{b})) < w(\mathrm{NAF}(\widehat{\delta b}))$. The process is repeated until all differences have been eliminated.

Algorithm 6-3 summarizes the construction of pairs of near-collision blocks set forth above.

6.5.2 Birthday search

A birthday search on a search space V is generally performed as in [vOW99] by iterating a properly chosen deterministic function $f : V \to V$ and by assuming that the points of V thus visited form a 'random walk', also called a trail. After approximately $\sqrt{\pi |V|/2}$ iterations one may expect to have encountered a collision, i.e., different points x and y such that $f(x) = f(y)$. Because the entire trail can in practice not be stored and to take advantage of parallelism, different pseudo-random walks are generated, of which only the startpoints, lengths, and endpoints are kept. The endpoints are 'distinguished points', points with an easily recognizable bitpattern depending on $|V|$, available storage and other characteristics. The average length of a walk is inversely proportional to the fraction of distinguished points in V. Because intersecting walks share their endpoints, they can easily be detected. The collision point can then be recomputed given the startpoints and lengths of the two colliding walks. The expected cost (i.e., number of evaluations of f) to generate the walks is denoted by C_{tr} and the expected cost of the recomputation to determine collision points is denoted by C_{coll}.

In our case the search space V and iteration function f depend on an integer parameter $k \in \{0, 1, 2, \ldots, 32\}$ as explained in Section 6.5.1. The birthday collision that we try to find, however, needs to satisfy several additional conditions that cannot be captured by V, f, or k: the prefixes associated with x and y in a birthday collision $f(x) = f(y)$ must be different, and the required number of pairs of near-collision blocks may be at most r when allowing differential paths with parameter w. The probability that a collision satisfies all requirements depends not only on the choice of r and w, but also on the value for k, and is denoted by $p_{r,k,w}$. As a consequence, on average $1/p_{r,k,w}$ birthday collisions have to be found.

More precisely, for $k \in \{0, \ldots, 32\}$ let B and B' be the last $512 - 64 - k$ bits of

$P\|S_\mathrm{r}$ and $P'\|S'_\mathrm{r}$, respectively. Then we define V and f as follows:

$$V = \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^{32}} \times \mathbb{Z}_{2^k},$$
$$f(x,y,z) = (a, c-d, c-b \bmod 2^k) \text{ where}$$
$$(a,b,c,d) = \begin{cases} \mathrm{MD5Compress}(IHV_{n-1}, B\|x\|y\|z) & \text{if } x \bmod 2 = 0; \\ \mathrm{MD5Compress}(IHV'_{n-1}, B'\|x\|y\|z) & \text{if } x \bmod 2 = 1. \end{cases}$$

It should be noted that any function with inputs x, y and z that outputs either 0 or 1 can be used instead of x mod 2 above. To obtain the highest probability that a birthday collision involves both M and M', such a function should be balanced, that is the pre-image spaces of 0 and 1 should have the same size: $|f^{-1}(0)| = |f^{-1}(1)|$.

Table 6-6: *Expected birthday search costs for $k = 0$.*

$k=0$	$w=0$			$w=1$			$w=2$			$w=3$		
r	p	C_tr	M	p	C_tr	M	p	C_tr	M	p	C_tr	M
16	5.9	35.27	1MB	1.75	33.2	1MB	1.01	32.83	1MB	1.	32.83	1MB
15	7.2	35.92	1MB	2.39	33.52	1MB	1.06	32.86	1MB	1.	32.83	1MB
14	8.71	36.68	1MB	3.37	34.01	1MB	1.27	32.96	1MB	1.04	32.84	1MB
13	10.45	37.55	1MB	4.73	34.69	1MB	1.78	33.22	1MB	1.2	32.93	1MB
12	12.45	38.55	1MB	6.53	35.59	1MB	2.78	33.71	1MB	1.66	33.16	1MB
11	14.72	39.68	2MB	8.77	36.71	1MB	4.34	34.5	1MB	2.61	33.63	1MB
10	17.28	40.97	11MB	11.47	38.06	1MB	6.54	35.6	1MB	4.18	34.42	1MB
9	20.16	42.4	79MB	14.62	39.64	2MB	9.38	37.02	1MB	6.46	35.56	1MB
8	23.39	44.02	732MB	18.21	41.43	21MB	12.88	38.76	1MB	9.52	37.09	1MB
7	26.82	45.73	8GB	22.2	43.43	323MB	17.02	40.83	9MB	13.4	39.02	1MB
6	31.2	47.92	161GB	26.73	45.69	8GB	21.78	43.22	241MB	18.14	41.4	20MB
5	35.	49.83	3TB	31.2	47.92	161GB	27.13	45.89	10GB	23.74	44.2	938MB
4							34.	49.33	2TB	30.19	47.42	81GB

The columns p, C_tr and M denote the values of $-\log_2(p_{r,k,w})$, $\log_2(C_\mathrm{tr}(r,k,w))$ and the minimum required memory M such that $C_\mathrm{coll}(r,k,w,M) \le C_\mathrm{tr}(r,k,w)$, respectively. See Appendix D for more extensive tables. The values for $p_{r,k,w}$ were estimated based on Algorithm 6-3.

Assuming that M bytes of memory are available and that a single trail requires 28 bytes of storage (namely 96 bits for the start- and endpoint each, and 32 for the length), this leads to the following expressions for the birthday search costs:

$$C_\mathrm{tr}(r,k,w) = \sqrt{\frac{\pi \cdot |V|}{2 \cdot p_{r,k,w}}}, \quad C_\mathrm{coll}(r,k,w,M) = \frac{2.5 \cdot 28 \cdot C_\mathrm{tr}(r,k,w)}{p_{r,k,w} \cdot M},$$

where $|V| = 2^{64+k}$, and the factor of 2.5 is explained in Chapter 3 of [vOW99].

For $M = 70/p_{r,k,w}$ as given in the last column of Table 6-6 and in the more extensive tables in Appendix D, the two costs are equal, and the overall expected birthday search costs becomes $2C_\mathrm{tr}(r,k,w)$. However, if the cost at run time of finding the trails exceeds the expected cost by a factor of λ, then the cost to determine the

resulting birthday collisions can be expected to increase by a factor λ^2. Hence, in practice it is advisable to choose M considerably larger. For $\epsilon \leq 1$, using $M = 70/(p_{r,k,w} \cdot \epsilon)$ bytes of memory results in $C_{\text{coll}} \approx \epsilon \cdot C_{\text{tr}}$ and the expected overall birthday search cost is about $(1 + \epsilon) \cdot C_{\text{tr}}(r, k, w)$ MD5 compressions.

6.5.3 Complexity analysis

The overall complexity of the chosen-prefix collision attack depends on the parameters used for the birthday search and the construction of pairs of near-collision blocks. This involves various trade-offs and is described in this section.

The birthday search complexity depends on the parameter w (defining the family of differential paths), the upper bound on the number r of pairs of near-collision blocks, the size 2^{64+k} of the search space, and the amount of available memory M. For various choices of r, k and w we have tabulated the heuristically determined expected birthday search complexities and memory requirements in Appendix D (in practice it is advisable to use a small factor more memory than required to achieve that C_{coll} is significantly smaller than C_{tr}). Given r, w and M, the optimal value for k and the resulting birthday search complexity can thus easily be looked up. When there is no restriction on the value to be used for r, one can balance the birthday search complexity and the complexity of constructing r pairs of near-collision blocks.

Each pair of near-collision blocks requires construction of a set of full differential paths followed by the actual construction of the pair of near-collision blocks. The complexity of the former construction depends on several parameter choices, such as the size of the sets of lower and upper differential paths, and the restraints used when selecting BSDRs for a δQ_t. Naturally, a higher overall quality of the resulting complete differential paths, i.e., a low number of overall bitconditions and a high total tunnel strength, generally results when more effort is put into the construction. For practical purposes we have found parameters sufficient for almost all cases (as applicable to the chosen-prefix collision attack) that have an average total complexity equivalent to roughly 2^{35} MD5 compressions (see Section 6.2.6).

The complexity of the collision finding, i.e., the construction of a pair of near-collision blocks, depends on the parameter w, the total tunnel strength and the number of bitconditions in the last 2.5 rounds. For small $w = 0, 1, 2$ and paths based on Table 6-5, the construction requires on average roughly the equivalent of 2^{34} MD5 compressions. Combined with the construction of the differential paths, this leads to the rough overall estimate of about $2^{35.6}$ MD5 compressions to find a single pair of near-collision blocks for a chosen-prefix collision attack.

With $w = 2$ and optimizing for overall complexity this leads to the optimal parameter choices $r = 9$ and $k = 0$. For these choices, the birthday search cost is about 2^{37} MD5 compressions and constructing the $r = 9$ pairs of near-collision blocks costs about $2^{38.8}$ MD5 compressions. The overall complexity is thus estimated at roughly $2^{39.1}$ MD5 compressions, which takes about 35 hours on a single PC-core. For these parameter choices the memory requirements for the birthday search are very low, even negligible compared to the several hundreds of MBs required for the construction of

the differential paths.

With more specific demands, such as a small number r of near-collision blocks possibly in combination with a relatively low M, the overall complexity increases. As an example, in our rogue CA construction at most $r = 3$ near-collision blocks were allowed. Using $M = 5\text{TB}$ this results in an overall complexity of about 2^{49} MD5 compressions.

Implementations of our differential path construction algorithms for MD5, our collision finding algorithm and the birthday search [20] are published as part of project HashClash [HC]. Furthermore, we have also published a graphical user interface that allows one to automatically perform a chosen-prefix collision attack and tweak many of the parameter choices we have discussed in Chapter 6.

6.5.4 Single-block chosen-prefix collision

Using the same approach as in the proof of Theorem 3.6, it is even possible to construct a chosen-prefix collision using only a single pair of near-collision blocks. Together with 84 birthday bits, the chosen-prefix collision-causing appendages are only $84 + 512 = 596$ bits long.[21] This approach is based on an even richer family of differential paths that allows elimination using a single pair of near-collision blocks of a set of δIHVs. The set of δIHVs is small enough so that finding the near-collision blocks is still feasible, but large enough that such a δIHV can be found efficiently by a birthday search. Instead of using the family of differential paths based on $\delta m_{11} = \pm 2^i$, we use the fastest known collision attack for MD5 of Section 6.4 and vary the last few steps to find a large family of differential paths.

By properly tuning the birthday search, the same partial differential path of Section 6.4 leads to the construction of a single near-collision block chosen-prefix collision for MD5. By varying the last steps of the differential path and by allowing the collision finding complexity to grow by a factor of about 2^{26}, we have identified a set $\mathcal{S}$ of about $2^{23.3}$ different $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$ of the form $\delta a = -2^5$, $\delta d = -2^5 + 2^{25}$, $\delta c = -2^5 \bmod 2^{20}$ that can be eliminated. Such δIHVs can be found using an 84-bit birthday search with step function $f : \{0,1\}^{84} \to \{0,1\}^{84}$ of the form

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

where $\widehat{\delta IHV}$ is of the required form, $\tau : x \mapsto \{0,1\}$ is a balanced selector function and $\phi(a,b,c,d) \mapsto a\|d\|(c \bmod 2^{20})$. There are $2^{128-84} = 2^{44}$ possible δIHV values of this form, of which only about $2^{23.3}$ are in the allowed set $\mathcal{S}$. It follows that a birthday collision $f(x) = f(x')$ has probability $p = 2^{23.3}/(2^{44} \cdot 2) = 2^{-21.7}$ to be useful, where the additional factor 2 stems from the fact that different prefixes are required, i.e., $\tau(x) \neq \tau(x')$.

20. Supports both the CELL and CUDA architecture

21. The shortest collision attack for MD5 is the 512-bit single-block identical prefix collision attack presented in [XF10].

A useful birthday collision can be expected after $\sqrt{\pi 2^{84}/(2p)} \approx 2^{53.2}$ MD5 compressions, requires 400MB of storage and takes about three days on 215 PS3s. The average complexity of finding the actual near-collision blocks is bounded by about $2^{14.8+26} = 2^{40.8}$ MD5 compressions and negligible compared to the birthday complexity. Thus the overall complexity is approximately $2^{53.2}$ MD5 compressions.

In Table 6-7 two 128-byte messages are given both consisting of a 52-byte chosen prefix[KG07] and a 76-byte single-block chosen-prefix collision suffix and with colliding MD5 hash value:

$$\texttt{d320b6433d8ebc1ac65711705721c2e1}_{16}.$$

The differential path that is actually followed between these two messages is given in Table 6-8.

Table 6-7: *Example single-block chosen-prefix collision.*

Message 1

```
4f 64 65 64 20 47 6f 6c 64 72 65 69 63 68 0a 4f
64 65 64 20 47 6f 6c 64 72 65 69 63 68 0a 4f 64
65 64 20 47 6f 6c 64 72 65 69 63 68 0a 4f 64 65
64 20 47 6f d8 05 0d 00 19 bb 93 18 92 4c aa 96
dc e3 5c b8 35 b3 49 e1 44 e9 8c 50 c2 2c f4 61
24 4a 40 64 bf 1a fa ec c5 82 0d 42 8a d3 8d 6b
ec 89 a5 ad 51 e2 90 63 dd 79 b1 6c f6 7c 12 97
86 47 f5 af 12 3d e3 ac f8 44 08 5c d0 25 b9 56
```

Message 2

```
4e 65 61 6c 20 4b 6f 62 6c 69 74 7a 0a 4e 65 61
6c 20 4b 6f 62 6c 69 74 7a 0a 4e 65 61 6c 20 4b
6f 62 6c 69 74 7a 0a 4e 65 61 6c 20 4b 6f 62 6c
69 74 7a 0a 75 b8 0e 00 35 f3 d2 c9 09 af 1b ad
dc e3 5c b8 35 b3 49 e1 44 e8 8c 50 c2 2c f4 61
24 4a 40 e4 bf 1a fa ec c5 82 0d 42 8a d3 8d 6b
ec 89 a5 ad 51 e2 90 63 dd 79 b1 6c f6 fc 11 97
86 47 f5 af 12 3d e3 ac f8 44 08 dc d0 25 b9 56
```

Table 6-8: *Single-block chosen-prefix collision - differential path*

t	Bitconditions: $q_t[31] \dots q_t[0]$
-3	....+-- ---.....
-2	..10.0-0 .0.1..0. .1..0.+- ---.....
-1	.^11.-+1 --.0..1. .0..1... ..+..100
0	.-+-.+1+ 1+.+..+. .-..-.00 11+..100
1	.0+-.-+0 +1.-..-. .+.1+... ..+..---
2	.+.-.0-1 10.-..+. .+.0+... ..+..-+-
3	.+1-.01+ -.^-..0. ..1-.0.. ..+...--
4	.+.1..1- 0.++..1. .00100.. ..+..1+-
5	...-...- -.0-.^.. ..+0.-^. ..0...+-
6	.1.+.... 1.-+.+.. ..-..+-. ..-....1
7	...-...1 +.1-.0.. ..+..10. ..1...10
8	1.1-...0 +.01.1.. ..-.^+1. ..1.^..+
9	1.0+...1 +..+..^. ..-.+-.. ....-..0
10	+0--010- 00.-0.-1 01-.1+.. ...00..-
11	-0-0111- 10^-1100 11111+11 ..^101.-
12	0+0++++1 -----01+ ---00+00 ^^+-.10.
13	+++10--- --0-0-11 1+--+-++ ++10^+10
14	1011-+0- ..0.-++1 11.0-100 1011-++.
15	0+0.-+01 11.11-++ +101+100 11..1-+.
16	....10.- ....^-0. ....+... ....^00.
17	.^..^+.. .....0^^ ^..01... .....^+.
18	0......^ .....^.. ...10... .0......
19	1....^.. ....+.... .1....^.
20	+.....-- ....-......
21	.....0.. ....^.... ....
22	^....1.. ....^......
23	.....+.. ....
24	....0 ....
25	.....^.. ....1 ....
26	....- ....
27	....
28	....^ ....

$\delta m_2 = 2^8, \quad , \delta m_{11} = 2^{15}, \quad \delta m_4 = \delta m_{14} = 2^{31}$

Continued at p. 114.

Table 6-9: *Single-block chosen-prefix collision - differential path (cont.)*

t	Bitconditions: $\mathtt{q}_t[31] \ldots \mathtt{q}_t[0]$
29–32	....
33	1..... ....
34	0..... ....
35	-..... ....
36	-..... ....
37	+..... ....
38	+..... ....
39	-..... ....
40	+..... ....
41	-..... ....
42	-..... ....
43	+..... ....
44	+..... ....
45	+..... ....
46	-..... ....
47	+..... ....
48	1..... ....
49	0..... ....
50–58	....
59	....0.....
60	......0. ....01 101.....
61	101101.0 .1..... .....0 10-.....
62	0.01.1+. .1..... ......-+ +++.....
63	+-----?- .-..... ......?? 0-+.....
64	.+.-.--+ .-.-.-++ -.---... ..-.....

$$\delta m_2 = 2^8, \quad , \delta m_{11} = 2^{15}, \quad \delta m_4 = \delta m_{14} = 2^{31}$$

Note: steps 60–63 follows the differential path as found for the example collision in Table 6-7. These steps are not optimal and show only one of the allowed possibilities.

7 SHA-0 and SHA-1

Contents

7.1 Overview

Chapter 7 covers all results on SHA-0 and SHA-1. First we give a formal definition of SHA-0 and SHA-1 in Section 7.2 and provide a treatment on published collision attacks and techniques in Section 7.3. The remaining sections cover our contributions.

Similar to MD5, we apply and improve the differential path analysis and algorithms of Chapter 5 for SHA-0 and SHA-1 in Section 7.4. However in contrast to MD5, this differential path construction is only to be used in the first round, i.e., the first 20 steps.

For the remaining three rounds, i.e., the last 60 steps, the technique of combining *local collisions* is used. *Disturbance vectors* are used to describe combinations of local collisions that may be used for near-collision attacks. So far, for various reasons, it is assumed that the local collisions behave independently in the literature on the analysis of disturbance vectors. This assumption has been shown to be flawed by Stéphane Manuel [Man11].

In Section 7.5, we present a method to analyze the success probability of disturbance vectors over the last three rounds that does not assume independence of local collisions. Our method is based on constructing differential paths that follow the prescribed local collision differences and summing the success probabilities of these differential paths. Since the number of possible differential paths can grow exponentially in the number of steps, various techniques are used to reduce this growth. This method also allows one to divert from the prescribed local collision differences at the beginning of the second round and the last few steps in order to obtain higher success probabilities. Furthermore, it provides an easy way to determine message expansion conditions that are sufficient to obtain the (near-)optimal success probability, a topic which so far has only been treated thoroughly on individual local collisions instead of combinations thereof.

We have constructed and implemented a near-collision attack against full SHA-1 using the above mentioned tools. It has an estimated complexity equivalent to $2^{57.5}$ SHA-1 compressions which can be used directly in an identical-prefix collision attack. This improves upon the near-collision attack with complexity of about 2^{68} SHA-1 compressions presented by Wang et al. [WYY05b]. The construction of our near-collision attack is presented in Section 7.6. This near-collision attack is practically

achievable within a year using about 1300 pc-cores[22]. As no actual near-collision block have been found yet using this near-collision attack, we discuss the verification of both the correctness and the complexity of our near-collision attack in Section 7.6.9.

This near-collision attack results in an identical-prefix collision attack against SHA-1 with an average complexity between $2^{60.3}$ and $2^{65.3}$ calls to the compression function of SHA-1 as explained in Section 7.6.1. Finally, based on this near-collision attack, we present a chosen-prefix collision attack against SHA-1 with an average complexity of about $2^{77.1}$ SHA-1 compressions in Section 7.7.

Although we have not constructed any attacks against SHA-0 or its compression function, the differential path construction algorithm in Section 7.4 and the differential cryptanalysis in Section 7.5 can be used directly in the construction of collision attacks against SHA-0. In particular, the differential cryptanalysis may allow an improvement over current collision attacks due to a better selection of the disturbance vector, the target values for δIHV_{diff} and the message word bitrelations, since the exact joint success probabilities of local collisions can be used instead of approximations based on the individual success probabilities of local collisions.

7.2 Description of SHA-0 and SHA-1

7.2.1 Overview

SHA-0 and SHA-1 work as follows on a given bit string M of arbitrary bit length, cf. [NIS95]:

1. *Padding.* Pad the message: first append a '1'-bit, next the least number of '0' bits to make the resulting bit length equal to 448 mod 512, and finally the bit length of the original unpadded message M as a 64-bit big-endian[23] integer. As a result the total bit length of the padded message $\widehat{M}$ is $512N$ for a positive integer N.

2. *Partitioning.* Partition the padded message $\widehat{M}$ into N consecutive 512-bit blocks M_0, $M_1, \ldots, M_{N-1}$.

3. *Processing.* To hash a message consisting of N blocks, SHA-0 and SHA-1 go through $N+1$ states IHV_i, for $0 \leq i \leq N$, called the *intermediate hash values*. Each intermediate hash value IHV_i is a tuple of five 32-bit words (a, b, c, d, e). For $i = 0$ it has a fixed public value called the *initial value* (IV):

$$IV = (\texttt{67452301}_{16}, \texttt{efcdab89}_{16}, \texttt{98badcfe}_{16}, \texttt{10325476}_{16}, \texttt{c3d2e1f0}_{16}).$$

   For $i = 1, 2, \ldots, N$ intermediate hash value IHV_i is computed using the respective SHA-0 or SHA-1 compression function described below:

$$IHV_i = \text{SHA0Compress}(IHV_{i-1}, M_{i-1}) \quad \text{for SHA-0;}$$

22. Measured on a single core of a Intel Core2 Q9550 2.83Ghz processor.
23. SHA-0/SHA-1 uses big-endian to convert between words and bit strings, whereas MD5 uses little-endian.

$$IHV_i = \text{SHA1Compress}(IHV_{i-1}, M_{i-1}) \quad \text{for SHA-1.}$$

4. *Output.* The resulting hash value is the last intermediate hash value IHV_N, expressed as the concatenation of the hexadecimal byte strings of the five words a, b, c, d, e of $IHV_N = (a, b, c, d, e)$, converted back from their big-endian representation. As an example the IV would be expressed as

$$\texttt{67452301efcdab8998badcfe10325476c3d2e1f0}_{16}.$$

7.2.2 SHA-0 compression function

The input for the compression function SHA0Compress(IHV, B) consists of an intermediate hash value $IHV_{\text{in}} = (a, b, c, d, e)$ and a 512-bit message block B. The compression function consists of 80 *steps* (numbered 0 to 79), split into four consecutive *rounds* of 20 steps each. Each step t uses modular additions, left rotations, and a non-linear function f_t, and involves an *Addition Constant* AC_t. The addition constants are defined per round as follows:

$$AC_t = \begin{cases} \texttt{5a827999}_{16} & \text{for } 0 \le t < 20, \\ \texttt{6ed9eba1}_{16} & \text{for } 20 \le t < 40, \\ \texttt{8f1bbcdc}_{16} & \text{for } 40 \le t < 60, \\ \texttt{ca62c1d6}_{16} & \text{for } 60 \le t < 80. \end{cases}$$

The non-linear function f_t also depends on the round:

$$f_t(X, Y, Z) = \begin{cases} F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z) & \text{for } 0 \le t < 20, \\ G(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 20 \le t < 40, \\ H(X, Y, Z) = (X \wedge Y) \vee (Z \wedge (X \vee Y)) & \text{for } 40 \le t < 60, \\ I(X, Y, Z) = X \oplus Y \oplus Z & \text{for } 60 \le t < 80. \end{cases} \tag{7.1}$$

The 512-bit message block B is partitioned into sixteen consecutive 32-bit strings which are then interpreted as 32-bit words m_0, $m_1, \ldots, m_{15}$ (with big-endian byte ordering), and expanded to 80 words W_t, for $0 \le t < 80$,

$$W_t = \begin{cases} m_t & \text{for } 0 \le t < 16, \\ W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16} & \text{for } 16 \le t < 80. \end{cases} \tag{7.2}$$

Note that message expansion relation is reversible:

$$W_{t-16} = W_t \oplus W_{t-3} \oplus W_{t-8} \oplus W_{t-14}, \quad \text{for } 16 \le t < 80. \tag{7.3}$$

Similar to MD5 we describe SHA-0's compression function SHA0Compress in an 'unrolled' version such that SHA0Compress $\in \mathcal{F}_{\text{md4cf}}$. For each step t the compression function algorithm uses a working state consisting of five 32-bit words Q_t, Q_{t-1}, Q_{t-2},

Q_{t-3} and Q_{t-4} and calculates a new state word Q_{t+1}. The working state is initialized as

$$(Q_0, Q_{-1}, Q_{-2}, Q_{-3}, Q_{-4}) = (a, b, RR(c, 30), RR(d, 30), RR(e, 30)). \tag{7.4}$$

For $t = 0, 1, \ldots, 79$ in succession, Q_{t+1} is calculated as follows:

$$\begin{aligned} F_t &= f_t(Q_{t-1}, RL(Q_{t-2}, 30), RL(Q_{t-3}, 30)), \\ Q_{t+1} &= F_t + AC_t + W_t + RL(Q_t, 5) + RL(Q_{t-4}, 30). \end{aligned} \tag{7.5}$$

After all steps are computed, the resulting state words are added to the input intermediate hash value and returned as output:

$$\begin{aligned} &\text{SHA0Compress}(IHV_{\text{in}}, B) = \\ &\quad (a + Q_{80},\ b + Q_{79},\ c + RL(Q_{78}, 30),\ d + RL(Q_{77}, 30),\ e + RL(Q_{76}, 30)). \end{aligned} \tag{7.6}$$

7.2.3 SHA-1 compression function

The compression function SHA1Compress of SHA-1 is defined identically to the compression function SHA0Compress of SHA-0 except for the message expansion where a bitwise rotation is added:

$$W_t = \begin{cases} m_t & \text{for } 0 \le t < 16, \\ RL(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}, 1) & \text{for } 16 \le t < 80. \end{cases} \tag{7.7}$$

Note that SHA1Compress $\in \mathcal{F}_{\text{md4cf}}$ and also that SHA-1's message expansion relation is reversible:

$$W_{t-16} = RR(W_t, 1) \oplus W_{t-3} \oplus W_{t-8} \oplus W_{t-14}, \quad \text{for } 16 \le t < 80. \tag{7.8}$$

7.2.4 Expanded messages

A sequence $(W_t)_{t=0}^{79}$ is called an *expanded message* when it is the result of the message expansion from $W_0, \ldots, W_{15}$. An expanded message can be seen as an element of $\mathbb{F}_2^{80\times 32}$. Note that this "definition" differs between the contexts of SHA-0 and SHA-1 and has the following properties:

- for both SHA-0 and SHA-1, the message expansion is linear with respect to $\oplus$: $(W_t \oplus V_t)_{t=0}^{79}$ is an expanded message if $(W_t)_{t=0}^{79}$ and $(V_t)_{t=0}^{79}$ are expanded messages;
- any 16 consecutive words $W_t, \ldots, W_{t+15}$ uniquely determine the entire expanded message $W_0, \ldots, W_{79}$, since both SHA-0's and SHA-1's message expansion relations are reversible;
- the left rotation $(RL(W_t, r))_{t=0}^{79}$ of any expanded message $(W_t)_{t=0}^{79}$ by r bits is also an expanded message;

- the forward and backward shifts $(W_t)_{t=1}^{80}$ and $(W_t)_{t=-1}^{78}$ of any expanded message $(W_t)_{t=0}^{79}$ are also expanded messages, where for SHA-x (using Eq. 7.2 and 7.7):

$$\begin{aligned} W_{80} &= RL(W_{80-3} \oplus W_{80-8} \oplus W_{80-14} \oplus W_{80-16}, x); \\ W_{-1} &= RR(W_{15}, x) \oplus W_{15-3} \oplus W_{15-8} \oplus W_{15-14}. \end{aligned}$$

7.3 Techniques towards collision attacks

SHA-1 and its predecessor SHA-0 have a more complex message expansion compared to MD5. Changing any bit in the first 16 words $W_0, \ldots, W_{15}$ leads to many bit differences in the succeeding words $W_{16}, \ldots, W_{79}$. Constructing a collision attack requires thus a different approach to find a good differential path over the last three rounds.

7.3.1 Local collisions

In 1998, Chabaud and Joux [CJ98] constructed a collision attack against SHA-0 based on local collisions. The idea of a local collision is simple: in some step t a disturbance is created by some message word difference $\delta W_t = 2^b$ resulting in $\delta Q_{t+1} = 2^b$. This disturbance is corrected over the next five steps, so that after those five steps no differences occur in the five working state words.

Obvious corrections for step $t+1$ and $t+5$ are $\delta W_{t+1} = -2^{(b+5 \bmod 32)}$ and $\delta W_{t+5} = -2^{(b+30 \bmod 32)}$, since both corrections occur with probability at least $1/2$ and for many values of b this probability is close to 1. In steps $t+2$, $t+3$ and $t+4$, the disturbance $\delta Q_{t+1} = 2^b$ might cause δF_{t+2}, δF_{t+3} and δF_{t+4} to be non-zero, which can be corrected with $\delta W_{t+k} = -\delta F_{t+k}$ for $k \in \{2, 3, 4\}$. Possible corrections for steps $t+2$, $t+3$ and $t+4$ vary per round. Common non-zero values for δF_{t+2}, δF_{t+3} and δF_{t+4} are $\pm 2^b$, $\pm 2^{(b+30 \bmod 32)}$ and $\pm 2^{(b+30 \bmod 32)}$, respectively.

7.3.2 Disturbance vector

Due to the properties of the message expansion, Chabaud and Joux [CJ98] were able to interleave many of these local collisions such that the message word signed bit differences $(\Delta W_t)_{t=0}^{79}$ conform to the message expansion. For more convenient analysis, they consider the *disturbance vector* which is a non-zero expanded message $(DV_t)_{t=0}^{79}$ where every '1'-bit $DV_t[b]$ marks the start of a local collision based on the disturbance $\delta W_t[b] = \pm 1$.

We use $(DW_t)_{t=0}^{79}$ to denote all message word bit differences without sign: $W'_t = W_t \oplus DW_t$. Note that the vector $(DW_t)_{t=0}^{79}$ must be an expanded message, since $(W_t)_{t=0}^{79}$ and $(W'_t)_{t=0}^{79}$ are expanded messages. Chabaud and Joux use the same relative message word bit differences for all local collisions, as this implies that $(DW_t)_{t=0}^{79}$ forms an XOR($\oplus$) over rotated shifts of the disturbance vector. Hence, $(DW_t)_{t=0}^{79}$ is an expanded message.

Let $\mathcal{R} \subset \{0,\ldots,5\} \times \{0,\ldots,31\}$ describe the relative indexes of the changed bits over the six steps of a local collision, then $(DW_t)_{t=0}^{79}$ can be determined as:

$$DW_t = \bigoplus_{(i,r)\in\mathcal{R}} RL(DV_{t-i}, r), \quad t \in \{0,\ldots,79\}. \tag{7.9}$$

Here $DV_{-5},\ldots,DV_{-1}$ are the words resulting from shifting $(DV_t)_{t=0}^{79}$ forward five times for SHA-x:

$$DV_i = RR(DV_{i+16}, x) \oplus DV_{i+13} \oplus DV_{i+8} \oplus DV_{i+2}, \quad i = -1,\ldots,-5.$$

Table 7-1 provides a list of all high probability local collisions with a single bit disturbance in ΔQ_{t+1} (no carries) and for which steps these local collisions are valid. Although for some steps the local collision with the fewest differences is

$$(\delta W_i)_{i=t}^{t+5} = (2^b, -2^{b+5 \bmod 32}, 0, 0, 0, -2^{b+30 \bmod 32}),$$

thus $\mathcal{R} = \{(0,0),(1,5),(5,30)\}$, this local collision cannot be used in rounds two and four due to their boolean function.

The only local collision that is valid for all steps (not allowing carries in ΔQ_{t+1}) is

$$(\delta W_i)_{i=t}^{t+5} = (2^b, -2^{b+5 \bmod 32}, \pm 2^b, \pm 2^{b+30 \bmod 32}, \pm 2^{b+30 \bmod 32}, -2^{b+30 \bmod 32}),$$

thus $\mathcal{R} = \{(0,0),(1,5),(2,0),(3,30),(4,30),(5,30)\}$ which is used for all published attacks against either SHA-0 or SHA-1 and is used throughout the remainder of this thesis.

To show that there are no other possibilities, we show that the set

$$\mathcal{R} = \{(0,0),(1,5),(2,0),(3,30),(4,30),(5,30)\}$$

is the only set of relative bit differences that is valid for $t = 30$ and $b = 31$. Since there can be no carry, i.e., ΔQ_{t+1} is either $\{31\}$ or $\{\overline{31}\}$, it follows that here any local collision can only consist of relative bit differences that are in the set $\{(0,0),(1,5),(2,0),(3,30),(4,30),(5,30)\}$. For all such local collisions the first, second and sixth relative bit differences in this set are unavoidable. Whether the remaining relative bit differences $\{(2,0),(3,30),(4,30)\}$ either must, must not or may be used depends on the boolean function. For $t = 30$, this is the XOR boolean function that always has an output bit difference if there is a difference in only one of the corresponding input bits. This implies that all three remaining relative bit differences must be used. Hence, for $t = 30$ and $b = 31$ the only possible local collisions are described by the relative bit differences

$$\mathcal{R} = \{(0,0),(1,5),(2,0),(3,30),(4,30),(5,30)\}.$$

In the case of SHA-0, since its message expansion does not use bitwise rotation, all '1'-bits in the disturbance vector can be limited to a single bit position. Chabaud

Table 7-1: *Possible local collisions for SHA-0 and SHA-1*

$\mathcal{R}$	t
$\{(0,0),(1,5),(2,0),(3,30),(4,30),(5,30)\}$	$0-79$
$\{(0,0),(1,5),(5,30)\}$	$0-15, 38-55, 78-79$
$\{(0,0),(1,5),(4,30),(5,30)\}$	$0-16, 38-56, 78-79$
$\{(0,0),(1,5),(3,30),(5,30)\}$	$0-15, 38-55, 78-79$
$\{(0,0),(1,5),(3,30),(4,30),(5,30)\}$	$0-17, 38-57, 78-79$
$\{(0,0),(1,5),(2,0),(5,30)\}$	$0-15, 37-55, 77-79$
$\{(0,0),(1,5),(2,0),(4,30),(5,30)\}$	$0-16, 37-56, 77-79$
$\{(0,0),(1,5),(2,0),(3,30),(5,30)\}$	$0-15, 36-55, 76-79$

Note: See Section 7.3.2. Listed combinations of local collision relative bit differences $\mathcal{R}$ and starting step t together with any starting bit $b \in \{0,\ldots,31\}$ forms an exhaustive list of all local collisions with a single bit disturbance $\Delta Q_{t+1}[b] = \pm 1$ (no carries) with a success probability of at least 2^{-5}.

and Joux took advantage of this fact by placing all '1'-bits on bit position 1 where local collisions generally have a higher probability than at other bit positions. Due to the added bitwise rotation in the message expansion of SHA-1, disturbance vectors always have '1'-bits at different bit positions and generally more '1'-bits compared to those for SHA-0.

Consecutive '1'-bits within a word DV_t can be compressed to a single local collision [WYY05c], i.e., $DV_t[0] = DV_t[1] = 1$ is used as $\Delta W_t[0] = -1$ and $\Delta W_t[1] = +1$ which leads to the single bit disturbance $\delta W_t = 2^1 - 2^0 = +2^0$. Due to the rotations in the step function, the bits at bit positions 1 and 2 are not considered consecutive in this regard as they result in the non-consecutive bits at positions 31 and 0 after bitwise left rotation by 30. The same holds for the bit position pair (26,27) due to the bitwise left rotation by five.

7.3.3 Disturbance vector restrictions

Initially for the most straightforward collision attack, namely a single-block collision attack, three restrictions were placed on disturbance vectors:

1. $DV_{-5} = DV_{-4} = DV_{-3} = DV_{-2} = DV_{-1} = 0$: δIHV_{in} must be zero $(0,0,0,0,0)$ for a single-block collision attack. Using the disturbance vector for all steps, the '1'-bits in the words $DV_{-5},\ldots,DV_{-1}$ mark disturbances in $\delta Q_{-4},\ldots,\delta Q_0$ and thus in δIHV_{in}. Since $\delta IHV_{\text{in}} = (0,0,0,0,0)$, it follows that DV_{-5}, DV_{-4}, DV_{-3}, DV_{-2} and DV_{-1} must be zero as well;

2. $DV_{75} = DV_{76} = DV_{77} = DV_{78} = DV_{79} = 0$: this restriction is necessary to enforce that $\delta IHV_{\text{out}} = \delta IHV_{\text{in}}$. This implies that if $\delta IHV_{\text{in}} = (0,0,0,0,0)$ then also $\delta IHV_{\text{out}} = (0,0,0,0,0)$;

3. at most one of $DV_i[b]$ and $DV_{i+1}[b]$ is non-zero, for $b = 0,\ldots,31$ and $i = 0,\ldots,15$: the boolean function in the first round prevents having two consecutive local collisions in the same bit position starting in the first 16 steps.

The first and third restrictions can be alleviated by diverting from local collisions as prescribed by the disturbance vector in the first round. So instead one can construct a differential path over the first round and use local collisions for the remaining rounds for use in a (near-)collision attack. Wang et al. [WYY05b] were the first to do this.

Finally, given the use of differential paths in the first round, one can also alleviate the second restriction and construct a two-block collision attack [WYY05b]. Without these three restrictions significantly better disturbance vectors were found, thus a two-block collision attack can be more efficient than a single-block collision attack. Since the complexity of each near-collision attack contributes to the overall collision attack complexity, using more than two blocks does not offer a further advantage.

7.3.4 Disturbance vector selection

To choose the best disturbance vector, several cost functions of disturbance vectors can be used where the cost function is only applied over the last, say, 60 words that form the last three rounds.

- **Hamming weight** (e.g., [BC04, PRR05, RO05, MP05, JP05]): counts the total number of local collisions over the specified steps, since a lower number of local collisions is expected to yield a higher overall probability;
- **Bitcondition count** (e.g., [WYY05c, YIN+08]): sum of the number of bitconditions for each local collision independently (not allowing carries);
- **Probability** (e.g., [MPRR06, Man11]): product of the probabilities of all local collisions independently where carries are allowed;
- **Joint probability** (Section 7.5): the probability of fulfilling all local collisions simultaneously.

Stéphane Manuel [Man08, Man11] noticed that all interesting disturbance vectors, including all disturbance vectors used in attacks in the literature, belong to the two classes shown in Table 7-2. Within each class all disturbance vectors are forward or backward shifts and/or rotations of each other. Since a disturbance vector is an expanded message, it is uniquely determined by any 16 consecutive words. The first class named 'type I' consists of disturbance vectors $(DV_t)_{t=0}^{79}$ in which there are 15 consecutive zero words directly followed by a word DV_i with only a single bit position b set to '1', thus $DV_i = 2^b$. Such a disturbance vector is identified as disturbance vector $\mathrm{I}(i-15, b)$. The second class named 'type II' consists of disturbance vectors $(DV_t)_{t=0}^{79}$ identified as $\mathrm{II}(i, b)$ such that

$$DV_j = \begin{cases} 2^{b+31 \bmod 32} & j \in \{i+1, i+3\}; \\ 2^b & j = i+15; \\ 0 & j \in \{i, i+2, i+4, i+5, \ldots, i+14\}. \end{cases}$$

In the literature, a number of disturbance vectors reported as (near-)optimal with respect to some cost function are:

- [WYY05b]: DV I(49,2);
- [RO05]: DVs I(52,31) and DVs I(45,1), I(49,1), I(51,1)[24];
- [JP05]: DVs I(51,0), I(52,0), II(52,0);
- [PRR05]: DV I(50,2);
- [YIN+08]: DV II(56,2);
- [Man11]: DV II(52,0) (DV II(49,0) in an earlier version [Man08]).

Most of these disturbance vectors have many disturbances on bit position 31 and/or 1 as the local collisions starting at those bit positions generally have higher success probabilities. The disturbance vector used in the near-collision attack presented in Section 7.6 and [Ste10] is DV II(52,0). This choice is not based on the results in the above publications, but is entirely based on preliminary results of those presented in Section 7.5.11. In the course of writing this thesis, Manuel published an updated version [Man11] of [Man08] which supports our choice.

7.3.5 Differential path construction

As mentioned before, Wang et al. [WYY05b] were the first to construct a hand-made differential path. In 2006, De Cannière and Rechberger[CR06] introduced a more algorithmic solution to construct differential paths for SHA-0 and SHA-1 which, instead of working from two directions to each other, works similar to a probabilistic algorithm from coding theory that searches for low weight code words. Yajima et al. [YSN+07] also present a differential path construction algorithm which is similar to, but less efficient than, the one we present in Section 7.4 below.

7.3.6 History of published attacks

SHA-0 Attacks The first theoretical collision attack against SHA-0 was published by Chabaud and Joux [CJ98] with estimated attack complexity of 2^{61} SHA-0 compressions. Their results were achieved by composing local collisions such that message differences conform to the message expansion. Their work forms the basis for all subsequent attacks against SHA-0 and SHA-1.

The first practical attack against SHA-0 is the near-collision attack by Eli Biham and Rafi Chen [BC04] with an estimated complexity of 2^{40} SHA-0 compressions, which uses a message modification technique dubbed *neutral bits* by the authors. This is a technique that, given IHV, IHV' and messages M and M' that follow a differential path up to some step t, flips one or more message bit positions in M and M' simultaneously such that the altered M and M' also follow the

24. These results were obtained using a Hamming weight based cost function, thus rotated versions are considered equal. These specific rotations were chosen to avoid a situation where their analysis would always be incorrect, see section 6 of their paper.

Table 7-2: *SHA-1 disturbance vectors of type I and type II*

disturbance vector I$(K, 0)$, $K \in \mathbb{Z}$

i	DV_{K+i}	DW_{K+i}
...	...	...
-18	31	28, 31
-17	30, 31	4, 28, 29, 30, 31
-16	–	3, 4, 28, 31
-15	31	29, 30
-14	31	4, 28, 31
-13	–	4, 28, 31
-12	–	28, 31
-11	31	31
-10	–	4
-9	–	29, 31
-8	–	29
-7	31	29, 31
-6	–	4, 29
-5	31	–
-4	–	4, 29
-3	31	29
-2	–	4
-1	31	29
0	–	4
1	–	29, 31
2	–	–
3	–	29
4	–	29
$5 - 14$	–	–
15	0	0
16	–	5
17	–	0
18	1	1, 30
19	–	6, 30
20	–	1, 30
21	2	2, 31
22	–	7, 31
23	1	1, 2, 31
24	3	0, 3, 6
25	–	0, 1, 8
26	–	0, 3, 31
27	4	1, 4, 31
...	...	...

disturbance vector II$(K, 0)$, $K \in \mathbb{Z}$

i	DV_{K+i}	DW_{K+i}
...	...	...
-20	–	29
-19	31	31
-18	–	4
-17	31	–
-16	–	4, 29
-15	31	29
-14	–	4
-13	30, 31	29, 30
-12	–	3, 4
-11	–	29, 30, 31
-10	31	28, 31
-9	–	4, 28, 29
-8	–	28, 29, 31
-7	–	29
-6	–	29
-5	31	29, 31
-4	–	4
-3	–	31
-2	–	29
-1	–	29
0	–	29
1	31	31
2	–	4
3	31	–
4	–	4, 29
5	–	29, 31
6	–	–
7	–	29
8	–	29
$9 - 14$	–	–
15	0	0
16	–	5
17	–	0
18	1	1, 30
19	0	0, 6, 30
20	–	1, 5, 30
21	2	0, 2, 31
...	...	...

Note: we describe the bit-positions of all '1'-bits of the 32-bit words DV_{K+i} and DW_{K+i}. The SHA-1 (reverse) message expansion relation is used to extend the above tables forward (backward). Disturbance vectors I(K, b) and II(K, b) for $b \in \{0, \ldots, 31\}$ are obtained by left rotating all 80 words of disturbance vectors I$(K, 0)$ and II$(K, 0)$, respectively, by b bit positions [Man11].

differential path up to step t. In essence tunnels and neutral bits are very similar message modification techniques. Using their techniques they are able to find collisions for SHA-0 reduced to 65 steps.

The first (identical-prefix) collisions for full SHA-0 [BCJ+05][25], constructed using four near-collision blocks[26], were found by Biham et al. with an estimated complexity of 2^{51} SHA-0 compressions. Wang et al. [WYY05c] improved this to a collision attack consisting of two near-collision blocks with an estimated total complexity of 2^{39} SHA-0 compressions. This attack was further improved by Naito et al. [NSS+06] to approximately 2^{36} SHA-0 compressions and later by Manuel and Peyrin [MP08] to approximately $2^{33.6}$ SHA-0 compressions. These three improvements also all provide example collisions.

So far no chosen-prefix collision attacks against SHA-0 have been published.

SHA-1 Attacks The first theoretical (identical-prefix) collision attack against full SHA-1 was published by Wang et al. [WYY05b] with an estimated complexity of 2^{69} SHA-1 compressions. In their paper also a practical collision attack against SHA-1 reduced to 58 steps is presented with an example collision. They claimed have improved their theoretical attack against full SHA-1 to an estimated complexity of 2^{63} SHA-1 compressions [WYY05a, Wan06]. No publication has followed so far however, instead [Coc07] has reconstructed and confirmed parts of the analysis of their attack given the details presented at the CRYPTO2005 rump session.

The paper [BCJ+05], besides collision attacks against full SHA-0, also presented collisions for 40-step SHA-1 with an attack complexity of approximately 2^{57} SHA-1 compressions. De Cannière and Rechberger[CR06] were able to construct a practical collision attack against 64-step SHA-1 and provide example collisions. De Cannière et al. [CMR07] were able to find collisions for 70-step SHA-1 in 2007. Collisions for 73-step SHA-1 were presented in [Gre10]. So far no collisions have been presented for a larger number of steps of SHA-1. In particular, though anticipated since 2006, no actual collisions for SHA-1 have been found.

At the rump session of CRYPTO2007, Mendel et al. [MRR07] claimed to have a collision attack against full SHA-1 with an estimated complexity of $2^{60.x}$ SHA-1 compressions and started a distributed computing project. No further publication has followed and their distributed computing project was stopped mid 2009.

Rafael Chen claims an identical-prefix collision attack against full SHA-1 in his PhD thesis [Che11] with an estimated complexity of 2^{58}. However, in his complexity estimation he extends a single 72-step SHA-1 near-collision attack

25. Announced at the rump session of Crypto'04

26. It would have been more efficient to use only two near-collision blocks. However, lacking differential path construction algorithms, the authors used a linearized model of SHA-1 to deal with the first few steps of differential paths. At least four near-collision blocks were needed for a collision attack within this model and using an additional technique.

of complexity $2^{53.1}$ to a two-block identical-prefix collision attack against full SHA-1 with an additional factor of only $2^{4.9}$. It can be easily verified that the highest success probability over the last 8 steps that can be achieved for the second near-collision block (that targets one specific δIHV) is about $2^{-8.356}$, thus there is an error in the complexity estimation of at least a factor $2^{3.5}$.[27]

There are two other papers that have to be mentioned here, namely [MHP09] by McDonald et al. in which the authors claimed to have found a differential path leading to a collision attack against full SHA-1 with an estimated complexity of 2^{52} SHA-1 compressions. Their result was directly based on the disturbance vector analysis and claimed possible collision attack complexities of Stéphane Manuel [Man08]. McDonald et al. decided to withdraw their paper when later analysis indicated that the claimed possible collision attack complexities in [Man08] were inaccurate. In the journal version [Man11] of [Man08] a more detailed analysis of disturbance vectors is made using a more conservative cost function and any claims towards possible collision attack complexities have been removed.

The literature on SHA-1 does not represent the state-of-the-art cryptanalytic methods as several claims have not been substantiated by publications. Moreover, due to lack of details it is hard if not impossible to verify the correctness and accurateness of the above claimed attack complexities and/or compare them, thus it is unclear which attack should be considered as the best correct collision attack against SHA-1. RFC6194 [PCTH11] considers the first collision attack by Wang et al. [WYY05b] with estimated complexity 2^{69} SHA-1 compressions as the best (identical-prefix) collision attack against full SHA-1. This attack is based on two near-collisions attacks with a complexity of about 2^{68} SHA-1 compressions each.

So far no chosen-prefix collision attacks against SHA-1 have been published. No implementations of (near-)collision attacks against SHA-1 have been published to date, except the implementation of the near-collision attack described in this thesis [HC].[28]

27. Exact probability can be determined using the methods presented in Section 7.5, estimations can easily be obtained using a Monte Carlo simulation. The given error factor of $2^{3.5}$ does not take into account the complexity of the first near-collision block and the fact that the second near-collision block is slightly harder than $2^{53.1}$.

28. Necessary for public verification of correctness and the actual runtime complexity. Also aids in further understanding and allows further improvements by the cryptographic community.

7.4 Differential path construction

In this section we present differential path construction algorithms for SHA-0 and SHA-1 based on the algorithms presented in Chapter 5. Similar to MD5, the SHA-0 and SHA-1 differential path construction algorithms are improved versions that make use of bitconditions (see Section 6.2.2). Also, the final connect algorithm operates in a bit-wise manner similar to the algorithm in Section 6.2.5 instead of a word-wise manner as in Chapter 5.

Differential path construction algorithms for SHA-0 and SHA-1 have already been proposed by De Cannière and Rechberger [CR06] and Yajima et al. [YSN+07] each using a different methodology. The first uses an approach based on a probabilistic algorithm from coding theory for finding low weight codewords. The second is very similar to the algorithms proposed in this section where from two ends partial differential paths are constructed towards each other. Our algorithm improves over that of [YSN+07] on efficiency: our connection algorithm operates in a bit-wise manner and therefore it is able to stop earlier in case of impossible forward and backward differential path pairs.

Compared to MD5, the rotations of Q_i that are given as input to the boolean function in the step update $f_t(Q_{t-1}, RL(Q_{t-2}, 30), RL(Q_{t-3}, 30))$ (see Eq. 7.5) complicate matters, since for both SHA-0 and SHA-1 indirect bitconditions on $Q_t[i]$ can involve one of the following bits: $Q_{t-1}[i]$, $Q_{t+1}[i]$, $Q_{t-1}[i+2 \bmod 32]$, $Q_{t-2}[i+2 \bmod 32]$, $Q_{t+1}[i-2 \bmod 32]$ and $Q_{t+2}[i-2 \bmod 32]$.

However, this is simpler in the first round as the first round boolean function of both SHA-0 and SHA-1 (ignoring the input rotations) is identical to MD5's first round boolean function. As can be seen in Table C-1, indirect bitconditions are only applied to the second input involving the third input or vice versa. This implies for SHA-0 and SHA-1 that in the first round any indirect bitcondition on $Q_t[i]$ can involve only $Q_{t-1}[i]$ or $Q_{t+1}[i]$, thus never bit positions other than i. Therefore the improved algorithm for MD5 in Section 6.2.5 can be more easily adapted to SHA-0 and SHA-1 over the first round. This limitation to the first round does not pose a problem for constructing a near-collision attack, since for the remaining rounds the differential path is the result of interleaving local collisions.

7.4.1 Differential paths

Similar to MD5, a differential path for SHA-0 or SHA-1 is described using bitconditions $\mathfrak{q}_t = (\mathfrak{q}_t[i])_{i=0}^{31}$ on (Q_t, Q'_t), where each bitcondition $\mathfrak{q}_t[i]$ specifies a restriction on the bits $Q_t[i]$ and $Q'_t[i]$ possibly including values of other bits $Q_l[i]$ for $l \in \{t-1, t+1\}$. For the first round only the following bitconditions are necessary and used: '`.`', '`+`', '`-`', '`0`', '`1`', '`^`', '`v`', '`!`' and '`y`' (see Tables 6-1 and 6-2), where the last two bitconditions are only used by message modification techniques (see Section 7.6.8). Since the disturbance vector determines all bit differences $(DW_t)_{t=0}^{79}$ only up to their sign, the actual chosen differences $(\Delta\widehat{W}_t)_{t=0}^{19}$ are maintained besides the bitconditions in the differential path.

A partial differential path for SHA-0 or SHA-1 over steps $t = t_b, \ldots, t_e$ can be seen as a $(t_e - t_b + 6) \times 32$ matrix $(\mathfrak{q}_t)_{t=t_b-4}^{t_e+1}$ of bitconditions paired with message difference vector $(\Delta\widehat{W}_t)_{t=t_b}^{t_e}$. The bitconditions are used to specify the values of ΔQ_t and ΔF_t. As for each step t only $\Delta Q_{t-3}, \Delta Q_{t-2}, \Delta Q_{t-1}, \Delta Q_t$ are required in a differential step, in such a partial differential path $\mathfrak{q}_{t-4}$ and $\mathfrak{q}_{t_e+1}$ are used only to represent $\delta RL(Q_{t_b-4}, 30)$ and δQ_{t_e+1} instead of BSDRs.

7.4.2 Forward

Suppose we have a partial differential path consisting of at least bitconditions $\mathfrak{q}_{t-3}$, $\mathfrak{q}_{t-2}$ and the differences ΔQ_{t-4}, ΔQ_{t-1} and δQ_t are known. We want to extend this partial differential path forward with step t resulting in the differences δQ_{t+1}, ΔW_t, ΔQ_t, bitconditions $\mathfrak{q}_{t-1}$ and additional bitconditions $\mathfrak{q}_{t-2}$ and $\mathfrak{q}_{t-3}$. We use an adaptation of the algorithm from Section 5.6.1 to perform such a forward extension using bitconditions (see also Section 6.2.2).

If the BSDR ΔQ_{t-1} is only used in previous steps to determine δQ_{t-1} and $\delta RL(Q_{t-1}, 5)$ then one can replace ΔQ_{t-1} by any low weight BSDR $\delta\widehat{Q}_{t-1}$ of δQ_{t-1} such that $\delta RL(Q_{t-1}, 5) = \sigma(RL(\delta\widehat{Q}_{t-1}, 5))$. Otherwise $\Delta\widehat{Q}_{t-1} = \Delta Q_{t-1}$. The BSDR $\Delta\widehat{Q}_{t-1}$ directly translates to bitconditions $\mathfrak{q}_{t-1}$ as in Table 6-1.

We select ΔQ_t based on the value of δQ_t. Since upcoming steps can use the remaining freedom in ΔQ_t, we choose for each $Z \in dRL(\delta Q_t, 5)$ (see Lemma 5.4) at most one ΔQ_t such that $\sigma(RL(\Delta Q_t, 5)) = Z$. We continue with any one of these ΔQ_t, preferably one with low weight. We choose a ΔW_t such that $\Delta W_t[i] \in \{-DW_t[i], +DW_t[i]\}$ for $i = 0, \ldots, 31$.

We assume that all indirect bitconditions in $\mathfrak{q}_{t-3}$ are forward and involve only bits of Q_{t-2} and that $\mathfrak{q}_{t-2}$ consists of only direct bitconditions.[29] To determine the differences $\Delta F_t = (g_i)_{i=0}^{31}$ we proceed as follows. For $i = 0, \ldots, 31$ we assume that we have valid bitconditions

$$(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathfrak{q}_{t-1}[i],\ \mathfrak{q}_{t-2}[i + 2 \bmod 32],\ \mathfrak{q}_{t-3}[i + 2 \bmod 32]),$$

where only $\mathfrak{c}$ can be indirect and if so involves $Q_{t-2}[i + 2 \bmod 32]$ associated with $\mathfrak{b}$. Hence, in the notation of Section 6.2.2: $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$. If $|V_{t,\mathfrak{abc}}| = 1$ then there is no ambiguity and we set $g_i = V_{t,\mathfrak{abc}}$ and $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = (\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$. Otherwise, if $|V_{t,\mathfrak{abc}}| > 1$, then we choose g_i arbitrarily from $V_{t,\mathfrak{abc}}$ and we resolve the ambiguity in $\Delta F_t[i]$ by replacing bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ by $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = FC(t, \mathfrak{abc}, g_i)$. Note that in the next step $t + 1$ our assumptions hold again as both $\widehat{\mathfrak{a}}$ and $\widehat{\mathfrak{b}}$ are direct bitconditions.

29. This assumption is valid for the results of this algorithm for the previous step $t - 1$. However this algorithm requires valid inputs for the first step, e.g., $t = 0$. E.g, we use the values $\Delta Q_{-4} = \Delta Q_{-1} = (0)_{i=0}^{31}$, $\mathfrak{q}_{-3} = \mathfrak{q}_{-2} = (\text{'}\cdot\text{'})_{i=0}^{31}$ and $\delta Q_0 = 0$ to represent identical but unknown intermediate hash values $IHV_{\text{in}} = IHV'_{\text{in}}$. Another possibility is to use the values $\mathfrak{q}_{-4}, \ldots, \mathfrak{q}_0$ consisting of direct bitconditions '0', '1', '+', '-' that represent given values IHV_{in} and IHV'_{in}. In this case, the forward construction algorithm can skip choosing BSDRs ΔQ_{-1} and ΔQ_0 and translating them to $\mathfrak{q}_{-1}$ and $\mathfrak{q}_0$ in steps $t = 0$ and $t = 1$, respectively.

Once all g_i and thus ΔF_t have been determined, δQ_{t+1} is determined as

$$\delta Q_{t+1} = \sigma(\Delta F_t) + \sigma(\Delta W_t) + \sigma(RL(\Delta Q_t, 5)) + \sigma(RL(\Delta Q_{t-4}, 5)).$$

7.4.3 Backward

Similar to the forward extension, we now consider the backward extension of a partial differential path. Suppose we have a partial differential path consisting of at least bitconditions $\mathfrak{q}_{t-2}, \mathfrak{q}_{t-1}$ and differences $\delta RL(Q_{t-3}, 30)$, ΔQ_t and δQ_{t+1} are known. We want to extend this partial differential path backward with step t resulting in the differences $\delta RL(Q_{t-4}, 30)$, ΔW_t, bitconditions $\mathfrak{q}_{t-3}$ and additional bitconditions $\mathfrak{q}_{t-1}$ and $\mathfrak{q}_{t-2}$. We use an adaptation of the algorithm from Section 5.6.2 to perform such a backward extension using bitconditions.

We choose a ΔW_t such that $W_t[i] \in \{-DW_t[i], +DW_t[i]\}$ for $i = 0, \ldots, 31$. We select ΔQ_{t-3} based on the value of $\delta RL(Q_{t-3}, 30)$. We choose any low weight BSDR Z of $\delta RL(Q_{t-3}, 30)$, so that $\Delta Q_{t-3} = RR(Z, 30)$ which then translates into a possible $\mathfrak{q}_{t-3}$ as in Table 6-1.

The differences $\Delta F_t = (g_i)_{i=0}^{31}$ are determined by assuming for $i = 0, \ldots, 31$ that we have valid bitconditions

$$(\mathfrak{a}, \mathfrak{b}, \mathfrak{c}) = (\mathfrak{q}_{t-1}[i],\ \mathfrak{q}_{t-2}[i + 2 \bmod 32],\ \mathfrak{q}_{t-3}[i + 2 \bmod 32]),$$

where only $\mathfrak{a}$ can be indirect and if so it involves $Q_{t-2}[i]$.[30] Note that $Q_{t-2}[i]$ is *not* associated with $\mathfrak{b}$. To deal with this issue, we first ignore such indirect bitconditions and reapply them later on.

We set $\widetilde{\mathfrak{a}} = \mathfrak{a}$ if $\mathfrak{a}$ is a direct bitcondition, otherwise $\widetilde{\mathfrak{a}} =$ '.'. It follows that $(\widetilde{\mathfrak{a}}, \mathfrak{b}, \mathfrak{c}) \in \mathcal{L}$. If $|V_{t,\widetilde{\mathfrak{a}}\mathfrak{b}\mathfrak{c}}| = 1$ then there is no ambiguity and we set $\{g_i\} = V_{t,\widetilde{\mathfrak{a}}\mathfrak{b}\mathfrak{c}}$ and $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = (\widetilde{\mathfrak{a}}, \mathfrak{b}, \mathfrak{c})$. Otherwise, if $|V_{t,\widetilde{\mathfrak{a}}\mathfrak{b}\mathfrak{c}}| > 1$, then we choose g_i arbitrarily from $V_{t,\widetilde{\mathfrak{a}}\mathfrak{b}\mathfrak{c}}$ and we resolve the ambiguity by replacing bitconditions $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ by $(\widehat{\mathfrak{a}}, \widehat{\mathfrak{b}}, \widehat{\mathfrak{c}}) = BC(t, \widetilde{\mathfrak{a}}\mathfrak{b}\mathfrak{c}, g_i)$. Note that in the next step $t - 1$ our assumptions hold again as $\widehat{\mathfrak{c}}$ is a direct bitcondition and $\widehat{\mathfrak{b}}$ is either a direct bitcondition or an indirect backward bitcondition involving $\widehat{\mathfrak{c}}$.

Finally, for all i such that $\mathfrak{q}_{t-1}[i]$ was an indirect bitcondition, we reapply this bitcondition. This means that if the new $\widehat{\mathfrak{q}}_{t-1}[i]$='.' then we revert it to the old value of $\mathfrak{q}_{t-1}[i]$. Otherwise, it must be either '0' or '1', since $\widehat{\mathfrak{a}}$ cannot be an indirect bitcondition (see Table C-1). If $\widehat{\mathfrak{q}}_{t-2}[i] \in \{\text{'.'}, \widehat{\mathfrak{q}}_{t-1}[i]\}$ then we replace it by $\widehat{\mathfrak{q}}_{t-1}[i]$, otherwise a contradiction has arisen and other choices have to be tried.

Once all g_i and thus ΔF_t are determined, $\delta RL(Q_{t-4}, 30)$ is determined as

$$\delta RL(Q_{t-4}, 30) = \delta Q_{t+1} - \sigma(\Delta F_t) - \sigma(\Delta W_t) - \sigma(RL(\Delta Q_t, 5)).$$

30. Again this assumption is met by the previous step $t + 1$ of the algorithm. Nevertheless the first call to the algorithm for, e.g., $t = 19$ requires valid inputs.

7.4.4 Connect

Construction of a full differential path can be done as follows. Assume that the forward construction has been carried out up to some step t. Furthermore, assume that the backward construction has been carried out down to step $t+6$. For our near-collision attack we used a low value for t as explained in Section 7.6.4 (p. 168). For each combination of forward and backward partial differential paths thus found, this leads to bitconditions $\ldots$, $\mathfrak{q}_{t-3}$, $\mathfrak{q}_{t-2}$, $\mathfrak{q}_{t-1}$, and $\mathfrak{q}_{t+3}$, $\mathfrak{q}_{t+4}$, $\mathfrak{q}_{t+5}$, $\mathfrak{q}_{t+6}, \ldots$ and differences ΔQ_t, δQ_{t+1}, $\delta RL(Q_{t+2}, 30)$. As more thoroughly explained at the end of this section, we replace all backward indirect bitconditions '^' by '.' to ensure correctness. Later on, we reapply all such removed backward indirect bitconditions.

It remains to try and glue together each of these combinations by finishing steps $t+1, \ldots, t+5$ until a full differential path is found. We use an adaptation of the algorithm in Section 5.6.3 which uses bitconditions and operates in a bit-wise manner instead of a word-wise manner. Due to the bitwise left-rotations over 30 bit positions in the step function, the description of the algorithm for SHA-0 and SHA-1 is more complicated compared to MD5. We first present a sketch of our algorithm that deals with the core principles, followed by a more precise definition.

Similar to MD5, all values for δQ_i are known and the goal is to find bitconditions and differences $\delta W_{t+1}, \ldots, \delta W_{t+5}$ such that some target values $\delta F_{t+1}, \ldots, \delta F_{t+5}$ are obtained:

$$\delta Q_{i+1} = \sigma(RL(\Delta Q_i, 5)) + \sigma(RL(\Delta Q_{i-4}, 30)) + \delta F_i + \delta W_i, \quad i \in \{t+1, \ldots, t+5\}. \tag{7.10}$$

We have some amount of freedom in choosing ΔQ_t, ΔQ_{t+1} and ΔQ_{t+2} as long as they remain compatible with the known values of δQ_{t+i}, $\delta RL(Q_{t+i}, 5)$ and $\delta RL(Q_{t+i}, 30)$ as described later on.

Due to the bitwise left-rotation over 30 bit position it follows that bit position 0 of step i depends on bitcondition $\mathfrak{q}_t[2]$ which is treated at bit position 2 of step $i-1$ for $i \in \{t+2, \ldots, t+5\}$. This issue is dealt with by using 40 imaginary bit positions $b \in \{0, \ldots, 39\}$ and the connect algorithm first searches for correct values at bit position $b = 0$ and then iteratively extends to higher bit positions. For each successful extension to the last bit position $b = 39$, one finds at least one valid full differential path.

At each bit position $b \in \{0, \ldots, 39\}$, the algorithm considers step $t+1+j$ at bit position $b-2j$ for $j \in \{0, \ldots, 4\}$ if and only if $b-2j \in \{0, \ldots, 31\}$. Whenever the algorithm considers step $t+1+j$ for $j \in \{0, \ldots, 4\}$ at bit position $b-2j \in \{0, \ldots, 31\}$, it does the following:

1. If $j \in \{0, 1, 2\}$ then it first selects a value for $\Delta Q_{t+j}[b-2j]$ that is compatible with the three known differences δQ_{t+j}, $\delta RL(Q_{t+j}, 5)$ and $\delta RL(Q_{t+j}, 30)$.

2. Next (if $j \in \{0, \ldots, 4\}$), it searches for a value for $\Delta W_{t+j+1}[b-2j]$ and bitconditions $\mathfrak{q}_{t+j}[b-2j]$, $\mathfrak{q}_{t+j-1}[b-2j+2 \bmod 32]$ and $\mathfrak{q}_{t+j-2}[b-2j+2 \bmod 32]$. These bitconditions must be compatible with all bitconditions known up to this point.

Furthermore, they must unambiguously lead to some value $\Delta F_{t+j+1}[b-2j]$ that is 'compatible' with Equation 7.10 (using $i = t + j + 1$).

3. For each such resulting tuple of values $\Delta Q_{t+j}[b-2j]$, $\Delta W_{t+j+1}[b-2j]$, $\mathfrak{q}_{t+j}[b-2j]$, $\mathfrak{q}_{t+j-1}[b-2j+2 \bmod 32]$ and $\mathfrak{q}_{t+j-2}[b-2j+2 \bmod 32]$, the algorithm continues with the next step $t+2+j$ at bit position $b-2j-2$ if $j < 4$ and $b-2j-2 \in \{0,\ldots,31\}$.

Now we give a more precise definition of the connection algorithm. First, we choose a low weight BSDR $\Delta\widetilde{Q}_{t+1}$ of δQ_{t+1} and a low weight BSDR $\Delta\widetilde{Q}_{t+2}$ such that $\sigma(RL(\Delta\widetilde{Q}_{t+2}, 30)) = \delta RL(Q_{t+2}, 30)$. Then we determine target values for FW_k which can be seen as the target value for $\delta F_k + \delta W_k$ for $k = t+1,\ldots,t+5$:

$$FW_k = \sigma(\Delta\widetilde{Q}_{k+1}) - \sigma(RL(\Delta\widetilde{Q}_k, 5)) - \sigma(RL(\Delta\widetilde{Q}_{k-4}, 30)). \tag{7.11}$$

So far we can choose any ΔQ_t, ΔQ_{t+1} and ΔQ_{t+2} under the following requirements so that Equation 7.11 holds:

$$\sum_{b=b_l}^{b_h} 2^b \Delta Q_k[b] = \sum_{b=b_l}^{b_h} 2^b \Delta\widetilde{Q}_k[b] \quad (\text{in } \mathbb{Z}),$$

for $(b_l, b_h) \in \{(0,1),(2,26),(27,31)\}$ and $k \in \{t, t+1, t+2\}$.

We aim to complete the differential path by searching for new bitconditions $\mathfrak{q}_{t-3},\ldots,\mathfrak{q}_{t+6}$ that are compatible with the differential steps from the forward and backward construction, and by finding message word differences $\Delta W_{t+1},\ldots,\Delta W_{t+5}$ such that the following equation holds for $k = t+1,\ldots,t+5$:

$$\delta Q_{k+1} - \sigma(RL(\Delta Q_k, 5)) - \sigma(RL(\Delta Q_{k-4}, 30)) = FW_k = \delta F_k + \delta W_k.$$

An efficient way to find these new bitconditions is to first test if they exist, and if so to backtrack to actually construct them. For $i = 0, 1, \ldots, 40$ we attempt to construct a set $\mathcal{U}_i$ consisting of all tuples

$$(q_0, q_1, q_2, fw_1, fw_2, fw_3, fw_4, fw_5, (\mathfrak{q}_j[b])_{(j,b)\in A_i}),$$

where $q_0, q_1, q_2 \in \mathbb{Z}_{2^{32}}$ and $fw_1, fw_2, fw_3, fw_4, fw_5 \in \mathbb{Z}_{2^{32}}$ and A_i is a later to be defined constant set, such that:

1. $q_j \equiv 0 \bmod 2^{\min(32,\max(0,i-2j))}$ and $fw_j \equiv 0 \bmod 2^{\min(32,\max(0,i-2j-2))}$;

2. there exist bitconditions, compatible with the forward and backward differential paths and the bitconditions $(\mathfrak{q}_j[b])_{(j,b)\in A_i}$, that uniquely determine the $\Delta Q_j[b]$ and $\Delta F_j[b]$ below and BSDRs ΔW_k for which $W_k[i] \in \{-DW_k[i], +DW_k[i]\}$ for $k = t+1,\ldots,t+5$ and $i = 0,\ldots,31$ such that

$$\delta Q_{t+j} = q_j + \sum_{\ell=0}^{\theta(i-2j)} 2^\ell \Delta Q_{t+j}[\ell], \qquad j \in \{0,1,2\}; \tag{7.12}$$

$$FW_{t+j} = fw_j + \sum_{\ell=0}^{\theta(i-2j-2)} 2^\ell (\Delta F_{t+j}[\ell] + \Delta W_{t+j}[\ell]), \qquad j \in \{1,2,3,4,5\}; \tag{7.13}$$

where $\theta(j) = \min(32, \max(0, j))$.

The set A_i informally consists of all indices (j, b) for which $\mathfrak{q}_j[b]$ may have been modified by the construction of previous $\mathcal{U}_\ell$ for $\ell = 0, \ldots, i-1$ and for which the construction of upcoming $\mathcal{U}_\ell$ for $\ell = i+1, \ldots, 40$ depends on $\mathfrak{q}_j[b]$. This implies $A_0 = A_{40} = \emptyset$. The sets $A_1, \ldots, A_{39}$ are defined for $i = 1, \ldots, 39$ as:

$$\begin{aligned}
A_i &= \bigcup_{j \in \{1,2,3,4\}} \left(A_{i,j}^{(1)} \cup A_{i,j}^{(2)} \cup A_{i,j}^{(3)} \cup A_{i,j}^{(4)} \right), \\
A_{i,j}^{(1)} &= \left\{ (t+j-1, 0) \;\middle|\; i - 2j + 1 \in \{0, \ldots, 31\} \right\}, \\
A_{i,j}^{(2)} &= \left\{ (t+j-1, 1) \;\middle|\; i - 2j \in \{0, \ldots, 31\} \right\}, \\
A_{i,j}^{(3)} &= \left\{ (t+j-2, \ell + 2 \bmod 32) \;\middle|\; \ell = i - 2j + 1 \in \{0, \ldots, 31\} \right\}, \\
A_{i,j}^{(4)} &= \left\{ (t+j-2, \ell + 2 \bmod 32) \;\middle|\; \ell = i - 2j \in \{0, \ldots, 31\} \right\}.
\end{aligned}$$

From these conditions it follows that $\mathcal{U}_0$ must be chosen as

$$\{(\widetilde{q}_0, \widetilde{q}_1, \widetilde{q}_2, FW_{t+1}, FW_{t+2}, FW_{t+3}, FW_{t+4}, FW_{t+5}, \emptyset)\}, \tag{7.14}$$

where $\widetilde{q}_0 = \sigma(\Delta Q_t)$, $\widetilde{q}_1 = \sigma(\Delta Q_{t+1})$ and $\widetilde{q}_2 = \sigma(\Delta Q_{t+2})$. Algorithm 7-1 (p. 136–139) informally does the following to construct $\mathcal{U}_{i+1}$:

7-1. If $0 \le i < 32$ then step $t+1$ at bit i is processed (otherwise proceed directly to 7-1-a.): First a valid value for $\Delta W_{t+1}[i]$ and a valid differential bitcondition ('-','.' or '+') for $Q_t[i]$ are chosen such that Equation 7.12 holds. Next all possible boolean function differences $\Delta F_{t+1}[i]$ using $\mathfrak{q}_t[i]$, $\mathfrak{q}_{t-1}[i + 2 \bmod 32]$ and $\mathfrak{q}_{t-2}[i + 2 \bmod 32]$ such that Equation 7.13 holds are considered. Perform subroutine 7-1-a below for each possible choice.

7-1-a. If $0 \le i - 2 < 32$ then step $t+2$ at bit $i-2$ is processed (otherwise directly proceed to 7-1-b.): First a valid value for $\Delta W_{t+2}[i-2]$ and a valid differential bitcondition ('-','.' or '+') for $Q_{t+1}[i-2]$ are chosen such that Equation 7.12 holds. Next all possible boolean function differences $\Delta F_{t+2}[i-2]$ using $\mathfrak{q}_{t+1}[i-2]$, $\mathfrak{q}_t[i \bmod 32]$ and $\mathfrak{q}_{t-1}[i \bmod 32]$ such that Equation 7.13 holds are considered. Perform subroutine 7-1-b below for each possible choice.

7-1-b. If $0 \le i-4 < 32$ then step $t+3$ at bit $i-4$ is processed (otherwise directly proceed to 7-1-c.): First a valid value for $\Delta W_{t+3}[i-4]$ and a valid differential bitcondition ('-','.' or '+') for $Q_{t+2}[i-4]$ are chosen such that Eq. 7.12 holds. Next all possible boolean function differences $\Delta F_{t+3}[i-4]$ using $\mathfrak{q}_{t+2}[i-4]$, $\mathfrak{q}_{t+1}[i-2 \bmod 32]$

and $\mathfrak{q}_t[i-2 \bmod 32]$ such that Equation 7.13 holds are considered. Perform subroutine 7-1-c below for each possible choice.

7-1-c. If $0 \leq i-6 < 32$ then step $t+4$ at bit $i-6$ is processed (otherwise directly proceed to 7-1-d.): First a valid value for $\Delta W_{t+4}[i-6]$ is chosen. Next all possible boolean function differences $\Delta F_{t+4}[i-6]$ using $\mathfrak{q}_{t+3}[i-6]$, $\mathfrak{q}_{t+2}[i-4 \bmod 32]$ and $\mathfrak{q}_{t+1}[i-4 \bmod 32]$ such that Equation 7.13 holds are considered. Perform subroutine 7-1-d below for each possible choice.

7-1-d. If $0 \leq i-8 < 32$ then step $t+5$ at bit $i-8$ is processed (otherwise save the resulting new tuple in $\mathcal{U}_{i+1}$): First a valid value for $\Delta W_{t+5}[i-8]$ is chosen. Next all possible boolean function differences $\Delta F_{t+5}[i-8]$ using $\mathfrak{q}_{t+4}[i-8]$, $\mathfrak{q}_{t+3}[i-6 \bmod 32]$ and $\mathfrak{q}_{t+2}[i-6 \bmod 32]$ such that Equation 7.13 holds are considered and for each saves the resulting new tuple in $\mathcal{U}_{i+1}$.

For $i = 1, \ldots, 40$, we use Algorithm 7-1 (pp. 136–139) to construct $\mathcal{U}_i$ based on $\mathcal{U}_{i-1}$. As soon as we encounter an i for which $\mathcal{U}_i = \emptyset$, we know that the desired differential path cannot be constructed from this combination of forward and backward partial differential paths, and that we should try another combination. If, however, we find $\mathcal{U}_{40} \neq \emptyset$ then it must be the case that $\mathcal{U}_{40} = (0,0,0,0,0,0,0,0,\emptyset)$. Furthermore, in that case, every set of bitconditions that leads to this non-empty $\mathcal{U}_{40}$ gives rise to a full differential path. Thus if $\mathcal{U}_{40} \neq \emptyset$, there exists at least one valid trail $u_0, \ldots, u_{40}$ with $u_i \in \mathcal{U}_i$ and where u_{i+1} is a tuple resulting from u_i in Algorithm 7-1. For each valid trail, the desired new bitconditions $\mathfrak{q}_{t-2}, \ldots, \mathfrak{q}_{t+4}$ can be found as $\mathfrak{g}', \mathfrak{f}_0', \mathfrak{e}_0', \mathfrak{d}_0', \mathfrak{c}_0', \mathfrak{b}_2'', \mathfrak{a}'$ for bits $i+2, i, i-2, i-4, i-6, i-6, i-8$ (mod 32), respectively and if applicable, for $i = 0, \ldots, 39$ in Algorithm 7-1.[31]

Finally, there remains an issue that has not been dealt with so far, namely that $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ must be in $\mathcal{L}$ (see Section 6.2.2) for every occurrence of the form $FC(j, \mathfrak{abc}, z)$ in Algorithm 7-1. The connection algorithm works forward, which implies in the same way as in Section 7.4.2 that $\mathfrak{b}$ is a direct bitcondition and $\mathfrak{c}$ is either a direct bitcondition or a forward indirect bitcondition involving $\mathfrak{b}$. If the bitcondition $\mathfrak{a}$ is indirect then $(\mathfrak{a}, \mathfrak{b}, \mathfrak{c})$ cannot be in $\mathcal{L}$, since $\mathfrak{a}$ involves bitcondition other than $\mathfrak{b}$ and $\mathfrak{c}$. Thus $\mathfrak{a}$ must be a direct bitcondition. However, $\mathfrak{a}$ may come from the backward partial differential path and thus may be a backward indirect bitcondition. To resolve this issue, we replace all backward indirect bitconditions '^' by '.' before running Algorithm 7-1. We reapply all such removed backward indirect bitconditions '^' on $Q_j[b]$ to the differential paths resulting from the above procedure in the following manner. Note that $\mathfrak{q}_j[b]$ must be either '.', '0' or '1', since the only step that could have resulted in $\mathfrak{q}_j[b] =$'v' comes after the connection steps $t+1, \ldots, t+5$. If $\mathfrak{q}_j[b] =$'.' then we set $\mathfrak{q}_j[b] =$'^' to reapply the backward bitcondition. If $\mathfrak{q}_j[b] \in \{$'0','1'$\}$ and $\mathfrak{q}_{j-1}[b] \in \{$'.','v',$\mathfrak{q}_j[b]\}$ then we set $\mathfrak{q}_{j-1}[b] = \mathfrak{q}_j[b]$ to reapply the backward

31. Note that step $t+5$ at bit position $i-8$ determines three of these final bitconditions, namely $\mathfrak{q}_{t+2}[i-6 \bmod 32]$, $\mathfrak{q}_{t+3}[i-6 \bmod 32]$ and $\mathfrak{q}_{t+4}[i-8 \bmod 32]$. Furthermore, the lower steps $t+1$, $t+2$, $t+3$ and $t+4$ at bit position j determine only one final bitcondition, namely $\mathfrak{q}_{t-2}[j+2 \bmod 32]$, $\mathfrak{q}_{t-1}[j+2 \bmod 32]$, $\mathfrak{q}_t[j+2 \bmod 32]$ and $\mathfrak{q}_{t+1}[j+2 \bmod 32]$, respectively. This explains the double $i-6$.

bitcondition. If both options above do not hold then a contradiction has arisen and the full differential path cannot be valid.

For an example full differential path constructed with the above algorithm see Table 7-6 (p. 169).

7.4.5 Complexity

The complexity to construct valid differential paths for SHA-1 depends on many factors as is also explained in Sections 5.6.4 and 6.2.6. In the case of SHA-1, the complexity of the connection algorithm also depends on the number of possible message word differences on those five steps. However, the choice of which five steps to use for the connection algorithm depends mostly on the particular choice of the disturbance vector so as to leave maximal freedom for message modification techniques.

A rough approximation for the complexity to construct the differential path for our near-collision attack in Section 7.6 is the equivalent of 2^{43} SHA-1 compression function calls. This is significantly larger than the differential path construction for MD5. Nevertheless, it is also significantly smaller than the lowest complexity claimed (and withdrawn) so far for a SHA-1 collision attack. This complexity is based on our choices for the disturbance vector II(52,0), the five connecting steps, amount of freedom left for message modification techniques and the maximum number of bitconditions in the first round. It should be clear that for other choices the complexity of constructing differential paths can be smaller or larger.

Our implementations of our differential path construction algorithms for SHA-1 are published as part of project HashClash [HC].

Algorithm 7-1 Construction of $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$ for SHA-0 and SHA-1.

Assume $\mathcal{U}_i$ is constructed inductively by means of this algorithm. For each tuple $(q_0, q_1, q_2, fw_1, fw_2, fw_3, fw_4, fw_5, (\mathfrak{q}_j[b])_{(j,b)\in A_i}) \in \mathcal{U}_i$ do the following:†

1. Let $\mathcal{U}_{i+1} = \emptyset$ and $\widehat{\mathfrak{q}}_j[b] = \mathfrak{q}_j[b]$ for $(j, b) \in A_i$.‡
2. If $i \geq 32$ then
3. Let $\mathfrak{e}'_2 = \mathfrak{q}_{t+1-1}[i]$, $\widehat{q}_0 = q_0$ and $\widehat{fw}_1 = fw_1$.
4. Proceed with subroutine step2 (Algorithm 7-1-a, p. 137)
5. Else
6. For each different $q'_0 \in \{-q_0[i], +q_0[i]\}$ do
7. Let $\widehat{q}_0 = q_0 - 2^i q'_0$.
8. If $i \in \{1, 26, 31\}$ and $\widehat{q}_0 \neq \sum_{b=i+1}^{31} 2^b \Delta\widetilde{Q}_t[b]$ then skip steps 9-16.
9. Let $\mathfrak{e}_2 =$ '$\mathtt{-}$', '$\mathtt{.}$' or '$\mathtt{+}$' based on whether $q'_0 = -1$, 0 or $+1$.
10. Let $\mathfrak{f}_2 = \mathfrak{q}_{t+1-2}[i + 2 \bmod 32]$ and $\mathfrak{g} = \mathfrak{q}_{t+1-3}[i + 2 \bmod 32]$.
11. For each different $w'_1 \in \{-DW_{t+1}[i], +DW_{t+1}[i]\}$ do
12. Let $Z_1 = fw_1 - 2^i w'_1$
13. For each different $z'_1 \in \{-Z_1[i], Z_1[i]\} \cap V_{t+1,\mathfrak{e}_2\mathfrak{f}_2\mathfrak{g}}$ do
14. Let $(\mathfrak{e}'_2, \mathfrak{f}'_2, \mathfrak{g}') = FC(t + 1, \mathfrak{e}_2\mathfrak{f}_2\mathfrak{g}, z'_1)$ and $\widehat{fw}_1 = fw_1 - 2^i(w'_1 + z'_1)$
15. Let $\widehat{\mathfrak{q}}_{t+1-1}[i] = \mathfrak{e}'_2$ and $\widehat{\mathfrak{q}}_{t+1-2}[i + 2 \bmod 32] = \mathfrak{f}'_2$.
16. Proceed with subroutine step2 (Algorithm 7-1-a, p. 137)
17. End if
18. Return $\mathcal{U}_{i+1}$

† For any $\mathfrak{q}_j[b]$ above: if $(j, b) \in A_i$ this bitcondition is retrieved from the current tuple in $\mathcal{U}_i$, otherwise it is retrieved from the forward or backward differential path depending on j.
‡ This line provides default (previous) values $\widehat{\mathfrak{q}}_j[b]$ for Algorithm 7-1-d line 10.

Algorithm 7-1-a Construction of $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$ for SHA-0 and SHA-1 (continued).

Subroutine step2

1. If $i < 2$ or $i \geq 34$ then
2. Let $\mathfrak{d}'_2 = \mathfrak{q}_{t+2-1}[i-2]$, $\widehat{q}_1 = q_1$, $\widehat{fw}_2 = fw_2$.
3. Proceed with subroutine step3 (Algorithm 7-1-b, p. 138)
4. Else
5. For each different $q'_1 \in \{-q_1[i-2], +q_1[i-2]\}$ do
6. Let $\widehat{q}_1 = q_1 - 2^{i-2}q'_1$.
7. If $i-2 \in \{1, 26, 31\}$ and $\widehat{q}_1 \neq \sum_{b=i-2+1}^{31} 2^b \Delta\widetilde{Q}_t[b]$ then skip steps 8-15.
8. Let $\mathfrak{d}_2 =$ '-', '.' or '+' based on whether $q'_1 = -1$, 0 or $+1$.
9. Let $\mathfrak{f}_0 = \mathfrak{q}_{t+2-3}[i \bmod 32]$.
10. For each different $w'_2 \in \{-DW_{t+2}[i-2], +DW_{t+2}[i-2]\}$ do
11. Let $Z_2 = fw_2 - 2^{i-2}w'_2$
12. For each different $z'_2 \in \{-Z_2[i-2], Z_2[i-2]\} \cap V_{t+2,\mathfrak{d}_2\mathfrak{e}'_2\mathfrak{f}_0}$ do
13. Let $(\mathfrak{d}'_2, \mathfrak{e}''_2, \mathfrak{f}'_0) = FC(t+2, \mathfrak{d}_2\mathfrak{e}'_2\mathfrak{f}_0, z'_2)$ and $\widehat{fw}_2 = fw_2 - 2^{i-2}(w'_2 + z'_2)$
14. Let $\widehat{\mathfrak{q}}_{t+2-1}[i-2] = \mathfrak{d}'_2$ and $\widehat{\mathfrak{q}}_{t+2-2}[i \bmod 32] = \mathfrak{e}''_2$.
15. Proceed with subroutine step3 (Algorithm 7-1-b, p. 138)
16. End if
17. Return to main routine

Algorithm 7-1-b Construction of $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$ for SHA-0 and SHA-1 (continued).

Subroutine step3

1. If $i < 4$ or $i \geq 36$ then
2. Let $\mathfrak{c}'_2 = \mathfrak{q}_{t+3-1}[i-4]$, $\widehat{q_2} = q_2$, $\widehat{fw_3} = fw_3$.
3. Proceed with subroutine step4 (Algorithm 7-1-c, p. 139)
4. Else
5. For each different $q'_2 \in \{-q_2[i-4], +q_2[i-4]\}$ do
6. Let $\widehat{q_2} = q_2 - 2^{i-4}q'_2$.
7. If $i-4 \in \{1, 26, 31\}$ and $\widehat{q_2} \neq \sum_{b=i-4+1}^{31} 2^b \Delta\widetilde{Q}_t[b]$ then skip steps 8-15.
8. Let $\mathfrak{c}_2 =$ '$\mathtt{-}$', '$\mathtt{.}$' or '$\mathtt{+}$' based on whether $q'_2 = -1$, 0 or $+1$.
9. Let $\mathfrak{e}_0 = \mathfrak{q}_{t+3-3}[i - 2 \bmod 32]$.
10. For each different $w'_3 \in \{-DW_{t+3}[i-4], +DW_{t+3}[i-4]\}$ do
11. Let $Z_3 = fw_3 - 2^{i-4}w'_3$
12. For each different $z'_3 \in \{-Z_3[i-4], Z_3[i-4]\} \cap V_{t+3,\mathfrak{c}_2\mathfrak{d}'_2\mathfrak{e}_0}$ do
13. Let $(\mathfrak{c}'_2, \mathfrak{d}''_2, \mathfrak{e}'_0) = FC(t+3, \mathfrak{c}_2\mathfrak{d}'_2\mathfrak{e}_0, z'_3)$ and $\widehat{fw_3} = fw_3 - 2^{i-4}(w'_3 + z'_3)$
14. Let $\widehat{\mathfrak{q}}_{t+3-1}[i-4] = \mathfrak{c}'_2$ and $\widehat{\mathfrak{q}}_{t+3-2}[i - 2 \bmod 32] = \mathfrak{d}''_2$.
15. Proceed with subroutine step4 (Algorithm 7-1-c, p. 139)
16. End if
17. Return to subroutine step2

Algorithm 7-1-c Construction of $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$ for SHA-0 and SHA-1 (continued).
Subroutine step4

1. If $i < 6$ or $i \geq 38$ then
2. Let $\mathfrak{b}'_2 = \mathfrak{q}_{t+4-1}[i-6]$, $\widehat{fw}_4 = fw_4$.
3. Proceed with subroutine step5 (Algorithm 7-1-d, p. 139)
4. Else
5. Let $\mathfrak{b}_2 = \mathfrak{q}_{t+4-1}[i-6]$ and $\mathfrak{d}_0 = \mathfrak{q}_{t+4-3}[i-4 \bmod 32]$.
6. For each different $w'_4 \in \{-DW_{t+4}[i-6], +DW_{t+4}[i-6]\}$ do
7. Let $Z_4 = fw_4 - 2^{i-6}w'_4$
8. For each different $z'_4 \in \{-Z_4[i-6], Z_4[i-6]\} \cap V_{t+4,\mathfrak{b}_2\mathfrak{c}'_2\mathfrak{d}_0}$ do
9. Let $(\mathfrak{b}'_2, \mathfrak{c}''_2, \mathfrak{d}'_0) = FC(t+4, \mathfrak{b}_2\mathfrak{c}'_2\mathfrak{d}_0, z'_4)$ and $\widehat{fw}_4 = fw_4 - 2^{i-6}(w'_4 + z'_4)$
10. Let $\widehat{\mathfrak{q}}_{t+4-1}[i-6] = \mathfrak{b}'_2$ and $\widehat{\mathfrak{q}}_{t+4-2}[i-4 \bmod 32] = \mathfrak{c}''_2$.
11. Proceed with subroutine step5 (Algorithm 7-1-d, p. 139)
12. End if
13. Return to subroutine step3

Algorithm 7-1-d Construction of $\mathcal{U}_{i+1}$ from $\mathcal{U}_i$ for SHA-0 and SHA-1 (continued).
Subroutine step5

1. If $i < 8$ then
2. Let $\widehat{fw}_5 = fw_5$.
3. Insert $(\widehat{q}_0, \widehat{q}_1, \widehat{q}_2, \widehat{fw}_1, \widehat{fw}_2, \widehat{fw}_3, \widehat{fw}_4, \widehat{fw}_5, (\widehat{\mathfrak{q}}_j[b])_{(j,b)\in A_{i+1}})$ in $\mathcal{U}_{i+1}$.
4. Else
5. Let $\mathfrak{a} = \mathfrak{q}_{t+5-1}[i-8]$ and $\mathfrak{c}_0 = \mathfrak{q}_{t+5-3}[i-6 \bmod 32]$.
6. For each different $w'_5 \in \{-DW_{t+5}[i-8], +DW_{t+5}[i-8]\}$ do
7. Let $Z_5 = fw_5 - 2^{i-8}w'_5$
8. For each different $z'_5 \in \{-Z_5[i-8], Z_5[i-8]\} \cap V_{t+5,\mathfrak{a}\mathfrak{b}'_2\mathfrak{c}_0}$ do
9. Let $(\mathfrak{a}', \mathfrak{b}''_2, \mathfrak{c}'_0) = FC(t+5, \mathfrak{a}\mathfrak{b}'_2\mathfrak{c}_0, z'_5)$ and $\widehat{fw}_5 = fw_5 - 2^{i-8}(w'_5 + z'_5)$
10. Insert $(\widehat{q}_0, \widehat{q}_1, \widehat{q}_2, \widehat{fw}_1, \widehat{fw}_2, \widehat{fw}_3, \widehat{fw}_4, \widehat{fw}_5, (\widehat{\mathfrak{q}}_j[b])_{(j,b)\in A_{i+1}})$ in $\mathcal{U}_{i+1}$.
11. End if
12. Return to subroutine step4

7.5 Differential cryptanalysis

As laid out in Section 7.3.4, all publications so far assume independence of local collisions in their analysis of disturbance vectors. However, this assumption is flawed as shown in this section and for instance the updated version [Man11] of [Man08]. So far no disturbance vector cost function that treats the local collisions as dependent has been presented.

The differential cryptanalysis method presented in this section does exactly this, i.e., it allows one to determine the exact success probability of a specific local collision or a specific combination of local collisions. As such it can be used as a cost function to determine (near-)optimal disturbance vectors. It improves upon the cost function that takes the product of the exact success probability of each individual local collision (allowing additional carries of δQ_i) over a specified range of steps, since it determines the exact *joint* success probability over the specified range of steps.

Our results clearly show that the joint probability differs from the product of the individual probabilities. Also, our results show that using dependence between local collisions leads to significantly higher success probabilities *under the correct optimal message conditions* than when assuming independent local collisions. It should be noted that if message conditions are used that are incompatible with the correct optimal message conditions then the average success probability will be lower and in the extreme can be even 0. In particular this may be the case for message conditions derived using previous analysis methods.

Our method also allows a more detailed analysis of the beginning of the second round and the last few steps. At these steps it may be more advantageous to divert from a prescribed combination of local collisions, as even higher success probabilities may be achieved. Moreover, our method allows us to find the smallest set of message expansion conditions which still results in the highest joint probability of success over the last three rounds. However, these conditions may be more limiting than or even incompatible with the message expansion conditions as prescribed by local collisions.

Our method is based on constructing differential paths that follow the prescribed local collision differences and summing the success probabilities of such differential paths that share the same message differences, the first five working state differences and the last five working state differences. The message differences and first five working state differences are preconditions of a differential path, whereas the last five working state differences determine $\delta IHV_{\text{diff}} = \delta IHV_{\text{out}} - \delta IHV_{\text{in}}$. Here, the success probability of a differential path over steps $i, \ldots, j$ is defined informally as the probability that all differential steps are fulfilled simultaneously assuming that $W_i, \ldots, W_j$ and $Q_{i-4}, \ldots, Q_i$ are independent uniform random variables and that both the message differences $\delta W_i, \ldots, \delta W_j$ and the first five working state differences $\delta RL(Q_{i-4}, 30)$, $\delta Q_{i-3}, \ldots, \delta Q_i$ as described in the differential path hold.

To overcome the exponential growth in the number of differential paths and possible message difference vectors $(\delta W_t)_{t=i}^{j}$ over the number of steps considered, our method employs several techniques. We first sketch the two most important techniques and then we present our method in detail.

The main technique is differential path reduction which removes all information in the differential path that is not strictly required for a forward or backward extension with a differential step. For instance, consider all differential paths over steps 59 to 66 with a single local collision starting at step 60 and bit position 2. There are 50 different possible values for ΔQ_{61}, since $\sigma(\Delta Q_{61})$ can be either positive or negative and carries can extend from bit position 2 up to bit position 26. The number of possible differential paths can be even greater as there can be multiple values for ΔF_{62}, ΔF_{63} and ΔF_{64} for each value of ΔQ_{61}. Nevertheless, since $\delta Q_{55} = \ldots = \delta Q_{59} = 0$ and $\delta Q_{62} = \ldots = \delta Q_{67} = 0$, the information of this local collision is not strictly required for the forward and backward extension with differential step 67 and 58, respectively. It follows that all such differential paths reduce to the trivial differential path with no differences at all.

Together with each reduced differential path, we maintain intermediary probabilities that accumulate the success probabilities of the removed parts of all differential paths. Since in the definition of the success probability of a differential path we assume that the message differences hold, we maintain a separate intermediary probability for each combination of a reduced differential path and possible message difference vector. This reduction is performed whenever all differential paths have been extended with a certain step t, after which we continue by extending with the next step.

Since the number of possible message difference vectors also grows exponentially in the number of steps, we employ another technique that allows us to 'combine' message difference vectors. Consider for each possible message difference vector $w = (\delta W_t)_{t=i}^{j}$ the function that maps each possible reduced differential path $\mathcal{P}$ to the associated intermediary probability for that w and $\mathcal{P}$. Suppose there are several message difference vectors $w_1, \ldots, w_K$ (over steps $i, \ldots, j$) for which said functions are identical. Then these message difference vectors all lead to identical reduced differential paths and associated intermediary probabilities. Furthermore, any future extension, i.e., future reduced differential paths and intermediary probabilities, depend only on these identical reduced differential paths and intermediary probabilities and not on the message difference vector. This implies that we can combine these message difference vectors in the following manner. We remove all intermediary probabilities associated with $w_2, \ldots, w_K$ and keep only the intermediary probability of a single message difference vector w_1 for future extensions. Furthermore, we create a substitution rule such that in any future extended message difference vector w that has the same differences as w_1 over steps $i, \ldots, j$, we may replace this subvector w_1 of w by any of $w_2, \ldots, w_K$.

7.5.1 Definition

Our method is based on differential paths $\mathcal{P}$ over steps $t = t_b, \ldots, t_e$, which are not described by bitconditions as in Section 7.4, but described by:

$$\mathcal{P} = (\delta RL(Q_{t_b-4}, 30), (\Delta Q_t)_{t=t_b-3}^{t_e}, \delta Q_{t_e+1}, (\Delta F_t)_{t=t_b}^{t_e}, (\delta W_t)_{t=t_b}^{t_e}),$$

under the following restrictions:

- correct differential steps for $t = t_b, \ldots, t_e$:

$$\delta Q_{t+1} = \sigma(RL(\Delta Q_t, 5)) + \delta RL(Q_{t-4}, 30)) + \sigma(\Delta F_t) + \delta W_t, \tag{7.15}$$

  where $\delta Q_{t+1} = \sigma(\Delta Q_{t+1})$ if $t \neq t_e$ and $\delta RL(Q_{t-4}, 30) = \sigma(RL(\Delta Q_{t-4}, 30))$ if $t \neq t_b$;

- for $t = t_b, \ldots, t_e$, both values -1 and $+1$ of $\Delta F_t[31]$ result in the same contribution $2^{31} \in \mathbb{Z}_{2^{32}}$ in $\sigma(\Delta F_t)$. We restrict $\Delta F_t[31]$ to $\{0, 1\}$ and a non-zero value represents $\Delta F_t[31] = \pm 1$;

- each $\Delta F_t[b]$ is individually possible, i.e., $(2^b \Delta F_t[b] \bmod 2^{32}) \in V_{t,b}$, where

$$V_{t,b} = \left\{ \begin{array}{c} (f_t(q'_1,q'_2,q'_3) \wedge 2^b) \\ -(f_t(q_1,q_2,q_3) \wedge 2^b) \end{array} \middle| \begin{array}{c} q_i, q'_i \in \mathbb{Z}_{2^{32}} \text{ for } i=1,2,3, \\ \Delta q_1 = \Delta Q_{t-1} \\ \Delta q_2 = RL(\Delta Q_{t-2}, 30) \\ \Delta q_3 = RL(\Delta Q_{t-3}, 30) \end{array} \right\}; \tag{7.16}$$

The probability $\Pr[\mathcal{P}]$ of such a differential path $\mathcal{P}$ is defined as:[32]

$$\Pr \left[\begin{array}{c} \Delta \widehat{Q}_j = \Delta Q_j \text{ for } j \in \{t_b-3, \ldots, t_e\}, \\ \delta \widehat{Q}_{t_e+1} = \delta Q_{t_e+1}, \\ 2^b \Delta \widehat{F}_i[b] = 2^b \Delta F_i[b] \bmod 2^{32}, \\ \text{for } i \in \{t_b, \ldots, t_e\},\ b \in \{0, \ldots, 31\} \end{array} \middle| \begin{array}{c} \widehat{Q}_{t_b-4} \xleftarrow{R} \mathbb{Z}_{2^{32}}, \\ \widehat{Q}'_{t_b-4} = RR(RL(\widehat{Q}_{t_b-4}, 30) + \delta RL(Q_{t_b-4}, 30), 30), \\ \widehat{Q}_k \xleftarrow{R} \mathbb{Z}_{2^{32}},\ \widehat{Q}'_k = \widehat{Q}_k + \delta Q_k \text{ for } k \in \{t_b-3, \ldots, t_b\}; \\ \widehat{W}_t \xleftarrow{R} \mathbb{Z}_{2^{32}},\ \widehat{W}'_t = \widehat{W}_t + \delta W_t, \\ \widehat{F}_t = f_t(\widehat{Q}_{t-1}, RL(\widehat{Q}_{t-2}, 30), RL(\widehat{Q}_{t-3}, 30)), \\ \widehat{F}'_t = f_t(\widehat{Q}'_{t-1}, RL(\widehat{Q}'_{t-2}, 30), RL(\widehat{Q}'_{t-3}, 30)), \\ \widehat{Q}_{t+1} = RL(\widehat{Q}_t, 5) + RL(\widehat{Q}_{t-4}, 30) + \widehat{F}_t + \widehat{W}_t + AC_t, \\ \widehat{Q}'_{t+1} = RL(\widehat{Q}'_t, 5) + RL(\widehat{Q}'_{t-4}, 30) + \widehat{F}'_t + \widehat{W}'_t + AC_t, \\ \text{for } t \in \{t_b, \ldots, t_e\} \end{array} \right].$$

More informally, this is the success probability of the following experiment:

Experiment 7.1. *This experiment involves partial SHA-1 computations of two messages. For the first message, values for $\widehat{Q}_{t_b-4}, \ldots, \widehat{Q}_{t_b}$ and $\widehat{W}_{t_b}, \ldots, \widehat{W}_{t_e}$ are selected uniformly at random. The remaining values for $\widehat{Q}_{t_b+1}, \ldots, \widehat{Q}_{t_e+1}$ are computed using SHA-0's and SHA-1's step function for $t = t_b, \ldots, t_e$:*

$$\widehat{F}_t = f_t(\widehat{Q}_{t-1}, RL(\widehat{Q}_{t-2}, 30), RL(\widehat{Q}_{t-3}, 30)),$$
$$\widehat{Q}_{t+1} = RL(\widehat{Q}_t, 5) + RL(\widehat{Q}_{t-4}, 30) + \widehat{F}_t + \widehat{W}_t + AC_t.$$

For the second message, we apply the differential path differences to the randomly selected variables:

$$\begin{aligned} \widehat{Q}'_{t_b-4} &= RR(RL(\widehat{Q}_{t_b-4}, 30) + \delta RL(Q_{t_b-4}, 30), 30), \\ \widehat{Q}'_i &= \widehat{Q}_i + \delta Q_i && \text{for } i = t_b - 3, \ldots, t_b, \\ \widehat{W}'_j &= \widehat{W}_j + \delta W_j && \text{for } j = t_b, \ldots, t_e. \end{aligned}$$

32. Note that $2^b \Delta \widehat{F}_j[b] = 2^b \Delta F_j[b] \bmod 2^{32}$ for $b = 0, \ldots, 31$ does not imply $\Delta \widehat{F}_j = \Delta F_j$. At bit position 31, the first case distinguishes only between $\Delta F_j[31]$ being zero or non-zero, whereas the latter case also distinguishes $\Delta F_j[31]$ by sign.

The remaining values $\widehat{Q}'_{t_b+1}, \ldots, \widehat{Q}'_{t_e+1}$ *are computed using SHA-0's and SHA-1's step function for* $t = t_b, \ldots, t_e$*:*

$$\begin{aligned}\widehat{F}'_t &= f_t(\widehat{Q}'_{t-1}, RL(\widehat{Q}'_{t-2}, 30), RL(\widehat{Q}'_{t-3}, 30)),\\ \widehat{Q}'_{t+1} &= RL(\widehat{Q}'_t, 5) + RL(\widehat{Q}'_{t-4}, 30) + \widehat{F}'_t + \widehat{W}'_t + AC_t.\end{aligned}$$

The experiment has succeeded when the above step function computations follow the differential path $\mathcal{P}$*, thus when all the following equations hold:*

$$\begin{aligned}\delta\widehat{Q}_{t_e+1} &= \delta Q_{t_e+1},\\ \Delta\widehat{Q}_i &= \Delta Q_i && \text{for } i = t_b - 3, \ldots, t_e,\\ 2^b \Delta\widehat{F}_j[b] &= 2^b \Delta F_j[b] \bmod 2^{32} && \text{for } j = t_b, \ldots, t_e,\ b = 0, \ldots, 31.\end{aligned}$$

7.5.2 Probability analysis

In this section we present a method to efficiently determine the probability of a differential path $\mathcal{P}$. To this end, consider a slight change in Experiment 7.1:

Experiment 7.2. *This experiment is a modification of Experiment 7.1. Instead of randomly selecting values for* $\widehat{W}_{t_b}, \ldots, \widehat{W}_{t_e}$ *and computing values for* $\widehat{Q}_{t_b+1}, \ldots, \widehat{Q}_{t_e+1}$*, one randomly selects values for* $\widehat{Q}_{t_b+1}, \ldots, \widehat{Q}_{t_e+1}$ *and computes values for* $\widehat{W}_{t_b}, \ldots, \widehat{W}_{t_e}$ *using:*

$$\begin{aligned}\widehat{F}_t &= f_t(\widehat{Q}_{t-1}, RL(\widehat{Q}_{t-2}, 30), RL(\widehat{Q}_{t-3}, 30)),\\ \widehat{W}_t &= \widehat{Q}_{t+1} - RL(\widehat{Q}_t, 5) - RL(\widehat{Q}_{t-4}, 30) - \widehat{F}_t - AC_t.\end{aligned}$$

The success requirement is left unchanged.

Since there is a bijective relation between $(\widehat{W}_t)_{t=t_b}^{t_e}$ and $(\widehat{Q}_{t+1})_{t=t_b}^{t_e}$, this implies that $(\widehat{W}_t)_{t=t_b}^{t_e}$ is also uniformly distributed in Experiment 7.2. Hence, the success probabilities of both experiments are equal. Note that this second experiment is completely determined by the values of $(\widehat{Q}_t)_{t=t_b-4}^{t_e+1}$. Next, consider another experiment:

Experiment 7.3. *This experiment is a modification of Experiment 7.2. As above, we set*

$$\begin{aligned}\widehat{Q}'_{t_b-4} &= RR(RL(\widehat{Q}_{t_b-4}, 30) + \delta RL(Q_{t_b-4}, 30), 30),\\ \widehat{Q}'_i &= \widehat{Q}_i + \delta Q_i && \text{for } i = t_b - 3, \ldots, t_b.\end{aligned}$$

However, instead of setting $\widehat{W}'_t = \widehat{W}_t + \delta W_t$ *for* $t = t_b, \ldots, t_e$ *and computing values for* $\widehat{Q}'_{t_b+1}, \ldots, \widehat{Q}'_{t_e+1}$*, one sets* $\widehat{Q}'_{t+1} = \widehat{Q}_{t+1} + \delta Q_{t+1}$ *for* $t = t_b, \ldots, t_e$ *and computes values for* $\widehat{W}'_{t_b}, \ldots, \widehat{W}'_{t_e}$*:*

$$\begin{aligned}\widehat{F}'_t &= f_t(\widehat{Q}'_{t-1}, RL(\widehat{Q}'_{t-2}, 30), RL(\widehat{Q}'_{t-3}, 30)),\\ \widehat{W}'_t &= \widehat{Q}'_{t+1} - RL(\widehat{Q}'_t, 5) - RL(\widehat{Q}'_{t-4}, 30) - \widehat{F}'_t - AC_t.\end{aligned}$$

The success requirement is left unchanged. In particular, one does not need an additional check that $\delta\widehat{W}_t = \delta W_t$ as in case of success this is implied by Equation 7.15:

$$\begin{aligned}\delta\widehat{W}_t &= \delta\widehat{Q}_{t+1} - \sigma(RL(\Delta\widehat{Q}_t, 5)) - \delta RL(\widehat{Q}_{t-4}, 30)) - \sigma(\Delta\widehat{F}_t)\\ &= \delta Q_{t+1} - \sigma(RL(\Delta Q_t, 5)) - \delta RL(Q_{t-4}, 30)) - \sigma(\Delta F_t)\\ &= \delta W_t.\end{aligned}$$

Lemma 7.1. *For fixed values $(\widehat{Q}_t)_{t=t_b-4}^{t_e+1}$, Experiment 7.3 succeeds if and only if Experiment 7.2 succeeds.*

Proof. If Experiment 7.3 succeeds then Eq. 7.15 must hold resulting in $(\delta\widehat{W}_t)_{t=t_b}^{t_e} = (\delta W_t)_{t=t_b}^{t_e}$. This implies that Experiment 7.2 also succeeds, since it will obtain identical values for both $(\widehat{W}'_t)_{t=t_b}^{t_e}$ and $(\widehat{Q}'_{t+1})_{t=t_b}^{t_e}$ as Experiment 7.3.

Suppose Experiment 7.3 fails. If $(\delta\widehat{W}_t)_{t=t_b}^{t_e} = (\delta W_t)_{t=t_b}^{t_e}$ then again Experiment 7.2 will obtain identical values for both $(\widehat{W}'_t)_{t=t_b}^{t_e}$ and $(\widehat{Q}'_{t+1})_{t=t_b}^{t_e}$ as Experiment 7.3 and thus Experiment 7.2 also fails. Otherwise, let $\widetilde{t}$ be the smallest $t \in \{t_b, \ldots, t_e\}$ for which $\delta\widehat{W}_t \neq \delta W_t$. This implies that Experiment 7.2 obtains identical values for $(\widehat{W}'_t)_{t=t_b}^{\widetilde{t}-1}$, $(\widehat{Q}'_t)_{t=t_b+1}^{\widetilde{t}}$ and $\Delta\widehat{F}_{\widetilde{t}}$ as Experiment 7.3. Assume that $\Delta\widehat{Q}_t = \Delta Q_t$ holds for all $t = t_b-3, \ldots, \widetilde{t}$ and $2^b\Delta\widehat{F}_{\widetilde{t}}[b] = 2^b\Delta F_{\widetilde{t}}[b] \bmod 2^{32}$ holds for all $b \in \{0, \ldots, 31\}$. Then Equation 7.15 implies that $\delta\widehat{W}_{\widetilde{t}} = \delta W_{\widetilde{t}}$ which contradicts the choice of $\widetilde{t}$, therefore this assumption does not hold. This failed assumption together with the fact that Experiment 7.2 obtains identical values for $(\widehat{Q}'_t)_{t=t_b+1}^{\widetilde{t}}$ and $\Delta\widehat{F}_{\widetilde{t}}$ as Experiment 7.3 directly implies that Experiment 7.2 must also fail. □

We use these experiments to show that the probability $\Pr[\mathcal{P}]$ of such a differential path can be determined as the fraction $N_{\mathcal{P}}/2^{32(t_e-t_b+6)}$ where $N_{\mathcal{P}}$ is the number of possible values $(\widehat{Q}_t)_{t=t_b-4}^{t_e+1} \in \mathbb{Z}_{2^{32}}^{t_e-t_b+6}$ for which this third experiment succeeds. In other words, $N_{\mathcal{P}}$ is the number of possible values $(\widehat{Q}_t)_{t=t_b-4}^{t_e+1} \in \mathbb{Z}_{2^{32}}^{t_e-t_b+6}$ for which

- for $t = t_b - 3, \ldots, t_e$: $\Delta Q_t = \Delta\widehat{Q}_t$;
- for $t = t_b, \ldots, t_e$ and $b = 0, \ldots, 31$:

$$\begin{aligned}(2^b\Delta F_t[b] \bmod 2^{32}) \;=\; &(f_t(\widehat{Q}'_{t-1}, RL(\widehat{Q}'_{t-2}, 30), RL(\widehat{Q}'_{t-3}, 30)) \wedge 2^b)\\ &- (f_t(\widehat{Q}_{t-1}, RL(\widehat{Q}_{t-2}, 30), RL(\widehat{Q}_{t-3}, 30)) \wedge 2^b),\end{aligned}$$

where $\widehat{Q}'_t = \widehat{Q}_t + \delta Q_t$ for $t \in \{t_b - 3, \ldots, t_e\}$.

An efficient way to determine the probability $\Pr[\mathcal{P}]$ is based on the fact that we can partition the bits $\widehat{Q}_t[b]$ into parts $G_{\Delta Q}$, $G_0, \ldots, G_K$ for some $K \in \mathbb{N}$ that each contribute a factor to $\Pr[\mathcal{P}]$. One important part $G_{\Delta Q}$ consists of all indices (j, i) such that $\Delta Q_j[i] \neq 0$ where $j \in \{t_b - 3, \ldots, t_e\}$ and $i \in \{0, \ldots, 31\}$. Since the values $\widehat{Q}'_j[i]$ and $\widehat{Q}_j[i]$ are uniquely determined for all $(j, i) \in G_{\Delta Q}$, this partition contributes the factor of $p_{\Delta Q} = 1/2^{|G_{\Delta Q}|}$ to $\Pr[\mathcal{P}]$.

Consider the set $\mathcal{S}_F$ of all indices (t, b) where $t \in \{t_b, \ldots, t_e\}$ and $b \in \{0, \ldots, 31\}$ such that $|V_{t,b}| > 1$ and thus $\Delta F_t[b]$ is not trivially fulfilled. Let $\mathcal{S}_Q$ be the set of all indices (j, i) where $j \in \{t_b - 4, \ldots, t_e + 1\}$ and $i \in \{0, \ldots, 31\}$ such that $\Delta Q_j[i] = 0$ and $Q_j[i]$ is involved with some $\Delta F_t[b]$ with $(t, b) \in \mathcal{S}_F$:

$$\{(j+1, i), (j+2, i+2 \bmod 32), (j+3, i+2 \bmod 32)\} \cap \mathcal{S}_F \neq \emptyset.$$

All indices (j, i) of bits $Q_j[i]$ where $(j, i) \notin \mathcal{S}_Q \cup G_{\Delta Q}$ for $j \in \{t_b - 4, \ldots, t_e + 1\}$, $i \in \{0, \ldots, 31\}$ form part G_0. Part G_0 consists by construction of all indices of free bits $Q_j[i]$ whose values do not affect ΔQ_j or any of the non-trivially fulfilled ΔF_t and thus contributes a factor of $p_0 = 2^{|G_0|}/2^{|G_0|} = 1$ to $\Pr[\mathcal{P}]$.

The set of remaining indices $\mathcal{S}_Q$ is further partitioned by constructing a graph $\mathcal{G}$ consisting of vertices $F_t[b]$ for all $(t, b) \in \mathcal{S}_F$ and vertices $Q_j[i]$ for all $(j, i) \in \mathcal{S}_Q$. There is an edge between two nodes $F_t[b]$ and $Q_j[i]$ if and only if:

$$(t, b) \in \{(j+1, i), (j+2, i+2 \bmod 32), (j+3, i+2 \bmod 32)\}, \tag{7.17}$$

i.e., $Q_j[i]$ is involved with $F_t[b]$. The graph $\mathcal{G}$ can be uniquely partitioned into connected subgraphs $\mathcal{G}_1, \ldots, \mathcal{G}_K$. This partition $\mathcal{G}_1, \ldots, \mathcal{G}_K$ of $\mathcal{G}$ defines a partition $G_1, \ldots, G_K$ of $\mathcal{S}_Q$ as follows:

$$G_k = \{(j, i) \mid Q_j[i] \in \mathcal{G}_k\}, \quad k \in \{1, \ldots, K\}.$$

By construction, all bits $Q_j[i]$ with associated nodes in the partition G_k influence a non-trivially fulfilled $\Delta F_t[b]$ if and only if there is an associated node $F_t[b]$ in $\mathcal{G}_k$. The probability p_k can be determined as $N_{\mathcal{P},k} \cdot 2^{-|G_k|}$, where $N_{\mathcal{P},k}$ is the number of different values of $(Q_j[i])_{(j,i)\in G_k}$ that result in the correct value of all $\Delta F_t[b]$, where $F_t[b]$ is a node in $\mathcal{G}_k$, and assuming $Q'_j[i] = Q_j[i] + \Delta Q_j[i]$ for all $(j, i) \in G_{\Delta Q}$.

Lemma 7.2. *The probability* $\Pr[\mathcal{P}]$ *is the product of* $p_{\Delta Q}$, p_0, $p_1, \ldots,$ p_K*:*

$$\Pr[\mathcal{P}] = p_{\Delta Q} \cdot p_0 \cdot \prod_{k=1}^{K} p_k = 2^{-|G_{\Delta Q}|} \prod_{k=1}^{K} \frac{N_{\mathcal{P},k}}{2^{|G_k|}}.$$

Proof. This lemma follows directly from the above construction. □

As a simple example, a single local collision starting with $\delta W_t = 2^b$ (without carry in δQ_{t+1}) results in five parts: $G_{\Delta Q}, G_0, G_1, G_2, G_3$. Part $G_{\Delta Q}$ consists solely of the disturbance $(t+1, b)$. Parts G_1, G_2 and G_3 consist each of two bit indices namely the other two $Q_i[j]$ involved with $Q_{t+1}[b]$ in the boolean function in step $t+2$, $t+3$ and $t+4$, respectively.

7.5.3 Extending

Extending a differential path $\mathcal{P}$ forward or backward with step l is done as follows. First a δW_l consistent with DW_l is chosen:

$$\delta W_l \in \Big\{\sigma(\Delta W_l) \;\Big|\; \Delta W_l[i] \in \{-DW_l[i], +DW_l[i]\} \text{ for } i = 0, \ldots, 31\Big\}.$$

In the case of a forward extension choose any BSDR ΔQ_l of δQ_l and a valid ΔF_l: $\Delta F_l[i] \in V_{l,i}$ for $i = 0, \ldots, 31$ (see Equation 7.16). Now δQ_{l+1} is determined as

$$\delta Q_{l+1} = \sigma(RL(\Delta Q_l, 5)) + \sigma(RL(\Delta Q_{l-4}, 30)) + \sigma(\Delta F_l) + \delta W_l$$

and $\mathcal{P}$ is extended by appending ΔQ_l, δQ_{l+1}, ΔF_l and δW_l.

Otherwise for a backward extension choose any BSDR ΔQ_{l-3} of δQ_{l-3} and a valid ΔF_l: $\Delta F_l[i] \in V_{l,i}$ for $i = 0, \ldots, 31$. Then $\delta RL(Q_{l-4}, 30)$ is determined as

$$\delta RL(Q_{l-4}, 30) = \sigma(\Delta Q_{l+1}) - \sigma(RL(\Delta Q_l, 5)) - \sigma(\Delta F_l) - \delta W_l$$

and $\mathcal{P}$ is extended by prepending $\delta RL(Q_{l-4}, 30)$, ΔQ_{l-3} and ΔF_l and δW_l.

7.5.4 Reduction

As said before, we are interested in the sum of success probabilities of differential paths that share the same message differences, the first five working state differences and the last five working state differences. The differential paths themselves are of lesser interest.

To reduce the total number of differential paths after extending a given step, we try to sensibly remove differences $\Delta Q_t[b]$ and $\Delta F_i[j]$ whose fulfillment probabilities are independent of any possible forward or backward extension choices. By keeping a graph-like structure of all intermediary differential paths one can always reconstruct non-reduced differential paths over all prescribed steps by performing the same extension choices without the subsequent reductions. Furthermore, we show in the next section how to use intermediary probabilities that accumulate the probabilities of such removed differences.

Given a differential path $\mathcal{P}$ over steps $t = t_b, \ldots, t_e$, we determine which differences $\Delta Q_j[i]$ and $\Delta F_t[b]$ can safely be removed. This is done by constructing the following graph $\widetilde{\mathcal{G}}$:

1. For $t = t_b, \ldots, t_e$ and $b = 0, \ldots, 31$, we add a node $F_t[b]$ if and only if $\Delta Q_{t-1}[b] \neq 0$ or $\Delta Q_{t-2}[b + 2 \bmod 32] \neq 0$ or $\Delta Q_{t-3}[b + 2 \bmod 32] \neq 0$.

2. For $j = t_b - 4, \ldots, t_e + 1$ and $i = 0, \ldots, 31$, we add a node $Q_j[i]$ if and only if at least one of the following differences is present in $\mathcal{P}$ and non-zero: $\Delta Q_j[i]$, $\Delta Q_{j-1}[i + 2 \bmod 32]$, $\Delta Q_{j-2}[i + 2 \bmod 32]$, $\Delta Q_{j+1}[i - 2 \bmod 32]$, $\Delta Q_{j-1}[i]$, $\Delta Q_{j+2}[i - 2 \bmod 32]$, $\Delta Q_{j+1}[i]$.

3. We connect each node $F_t[b]$ in the graph with edges to the nodes $Q_{j-1}[i]$, $Q_{j-2}[i + 2 \bmod 32]$ and $Q_{j-3}[i + 2 \bmod 32]$.

Consider all connected subgraphs $\widetilde{\mathcal{G}}_1, \ldots, \widetilde{\mathcal{G}}_K$ of $\widetilde{\mathcal{G}}$. Let $k \in \{1, \ldots, K\}$, if for all nodes $Q_j[i]$ of the connected subgraph $\widetilde{\mathcal{G}}_k$ we have $\Delta Q_j[i] = 0$ or $j \in \{t_b + 1, \ldots, t_e - 4\}$ then all differences $\Delta Q_j[i]$ and $\Delta F_t[b]$ associated with the respective nodes $Q_j[i]$ and $F_t[b]$ in $\widetilde{\mathcal{G}}_k$ can be safely removed. Since Equation 7.15 must hold, the necessary corrections are made in δW_t. For all nodes $Q_i[j] \in \widetilde{\mathcal{G}}_k$ where $\Delta Q_i[j] \neq 0$ we do the following:

1. Replace the value of δW_{j+4} by $\delta W_{j+4} + \Delta Q_j[i] \cdot 2^{i-2 \bmod 32}$;
2. Replace the value of δW_j by $\delta W_j + \Delta Q_j[i] \cdot 2^{i+5 \bmod 32}$;
3. Replace the value of δW_{j-1} by $\delta W_{j-1} - \Delta Q_j[i] \cdot 2^i$;
4. Replace the value of $\Delta Q_j[i]$ by 0.

For all nodes $F_t[b] \in \widetilde{\mathcal{G}}_k$ we do the following:

1. Replace the value of δW_t by $\delta W_t + \Delta F_t[b] \cdot 2^b$;
2. Replace the value of $\delta F_t[b]$ by 0.

Note that $\Delta Q_{t_b-4}, \ldots, Q_{t_b}$ and $\Delta Q_{t_e-3}, \ldots, \Delta Q_{t_e+1}$ remain untouched. Also, it can be seen that the graph $\mathcal{G}$ from Section 7.5.2 is a subgraph of $\widetilde{\mathcal{G}}$.

Lemma 7.3. *Given a differential path $\mathcal{P}$ over steps $t_b, \ldots, t_e$ and its reduced version $\widehat{\mathcal{P}}$, let $\widetilde{\mathcal{P}}$ be defined over steps $t_b, \ldots, t_e$ by:*

$$\begin{aligned} \Delta\widetilde{Q}_j[i] &= \Delta Q_j[i] - \Delta\widehat{Q}_j[i] && \text{for } j = t_b - 4, \ldots, t_e + 1,\ i = 0, \ldots, 31; \\ \Delta\widetilde{F}_t[b] &= \Delta F_t[b] - \Delta\widehat{F}_t[b] && \text{for } t = t_b, \ldots, t_e,\ b = 0, \ldots, 31; \\ \delta\widetilde{W}_t &= \delta W_t - \delta\widehat{W}_t && \text{for } t = t_b, \ldots, t_e. \end{aligned}$$

(Thus $\widetilde{\mathcal{P}}$ is defined by the eliminated differences $\Delta Q_j[i]$ and $\Delta F_t[b]$ and the negative sum of corrections to δW_t.) Then $\widehat{\mathcal{P}}$ and $\widetilde{\mathcal{P}}$ are valid differential paths and

$$\Pr[\mathcal{P}] = \Pr[\widehat{\mathcal{P}}] \cdot \Pr[\widetilde{\mathcal{P}}].$$

Proof. Let $k \in \{1, \ldots, K\}$. Then for every $Q_j[i]$ in $\widetilde{\mathcal{G}}_k$ with $\Delta Q_j[i] \neq 0$, also all related $F_t[b]$ are in $\widetilde{\mathcal{G}}_k$ by construction. (As before a $Q_j[i]$ and $F_t[b]$ are related if Equation 7.17 holds.) Similarly, for every $F_t[b]$ in $\widetilde{\mathcal{G}}_k$ also all related $Q_j[i]$ are in $\widetilde{\mathcal{G}}_k$ by construction.

Let $\mathcal{K} \subset \{1, \ldots, K\}$ be the index set of all connected subgraphs $\widetilde{\mathcal{G}}_k$ such that $\Delta Q_j[i] = 0$ or $j \in \{t_b + 1, \ldots, t_e - 4\}$ for all nodes $Q_j[i]$ in $\widetilde{\mathcal{G}}_k$. Then the differences $\Delta Q_j[i]$ and $\Delta F_t[b]$ associated with the respective nodes $Q_j[i]$ and $F_t[b]$ in $\widetilde{\mathcal{G}}_k$ are either present in $\widetilde{\mathcal{P}}$ if $k \in \mathcal{K}$ or in $\widehat{\mathcal{P}}$ if $k \notin \mathcal{K}$. Furthermore, all other differences $\Delta Q_j[i]$ and $\Delta F_t[b]$ present in $\widehat{\mathcal{P}}$ and $\widetilde{\mathcal{P}}$ are zero.

The probability $\Pr[\mathcal{P}]$ is the product of $p_{\Delta Q}, p_1, \ldots, p_L$ as in Section 7.5.2 ($p_0 = 1$ is always trivial). First, $p_{\Delta Q}$ is determined by all differences $\Delta Q_j[i] \neq 0$ in $\mathcal{P}$ where each such difference $\Delta Q_j[i]$ contributes a factor of $1/2$ to $p_{\Delta Q}$. Since each such difference $\Delta Q_j[i]$ is either present in $\widehat{\mathcal{P}}$ or in $\widetilde{\mathcal{P}}$ it follows that $p_{\Delta Q} = \widehat{p}_{\Delta Q} \cdot \widetilde{p}_{\Delta Q}$. Since the graph $\mathcal{G}$ from Section 7.5.2 is a subgraph of $\widetilde{\mathcal{G}}$, it follows that each connected subgraph $\mathcal{G}_l$ for $l \in \{1, \ldots, L\}$ is a subgraph of some $\widetilde{\mathcal{G}}_k$. If $k \in \mathcal{K}$ then the probability p_l associated with $\mathcal{G}_l$ is a factor of $\Pr[\widetilde{\mathcal{P}}]$ and not of $\Pr[\widehat{\mathcal{P}}]$. Otherwise, $k \notin \mathcal{K}$ and p_l

is a factor of $\Pr[\widehat{\mathcal{P}}]$ and not of $\Pr[\widetilde{\mathcal{P}}]$. Since $p_{\Delta Q}$ is divided into two factors, one for $\widehat{\mathcal{P}}$ and one for $\widetilde{\mathcal{P}}$ and the probabilities $p_1, \ldots, p_L$ have been partitioned between $\Pr[\widehat{\mathcal{P}}]$ and $\Pr[\widetilde{\mathcal{P}}]$, one can conclude that $\Pr[\widehat{\mathcal{P}}] \cdot \Pr[\widetilde{\mathcal{P}}]$. □

For a given differential path $\mathcal{P}$ we denote by $\text{Reduce}(\mathcal{P})$ the differential path resulting from reducing $\mathcal{P}$ by the above method.

Observation 7.1. *Let $\mathcal{P}$ be a differential path over steps $t_b, \ldots, t_e$ for which $\sigma(\Delta Q_i) = 0$ for $i = t_b - 4, \ldots, t_b$ and for $i = t_e - 3, \ldots, t_e + 1$. Then $\widehat{\mathcal{P}} = \text{Reduce}(\mathcal{P})$ is trivial: $\widehat{\mathcal{P}} = (0, ((0)_{j=0}^{31})_{i=t_b-4}^{t_e+1}, 0, ((0)_{j=0}^{31})_{i=t_b}^{t_e}, (0)_{i=t_b}^{t_e})$.*

7.5.5 Single local collision analysis

In this section we analyze the probabilities of local collisions either with or without additional carries. In Section 7.5.7 we extend this analysis to combinations of local collisions as prescribed by a disturbance vector. For now, let $(DV_t)_{t=0}^{79} \in \mathbb{Z}_{2^{32}}^{80}$ be a disturbance vector consisting of a single disturbance starting between step 0 and 74, which implies that after step 79 all corrections for this single disturbance have been made. Although $(DV_t)_{t=0}^{79}$ is not a valid disturbance vector for either SHA-0 or SHA-1, it allows for differential cryptanalysis of a single local collision that is easily extended to valid disturbance vectors.

First we present a number of definitions that we need later on. For $t = 1, \ldots, 80$, we denote by $\mathcal{Q}_t$ the set of allowed values for ΔQ_t. We offer two choices for $\mathcal{Q}_t$. The first choice is to select one from the family of sets $\mathcal{Q}_{\text{c},u,t}$ for $u \in \{0, \ldots, 32\}$ of allowed values for ΔQ_t as prescribed by the disturbance(s) in $(DV_t)_{t=0}^{79}$ allowing carries and where the weight is bounded by u plus the NAF weight:

$$\mathcal{Q}_{\text{c},u,t} = \left\{ \text{BSDR } Y \;\middle|\; \begin{array}{l} \sigma(Y) = \sigma(Z), \\ Z[i] \in \{-DV_{t-1}[i], DV_{t-1}[i]\},\ i = 0, \ldots, 31, \\ w(Y) \leq w(\text{NAF}(\sigma(Y))) + u \end{array} \right\}.$$

For $u = 32$, the bound on the weight is trivially fulfilled and we denote $\mathcal{Q}_{\text{c},t}$ for $\mathcal{Q}_{\text{c},32,t}$. The set $\mathcal{Q}_{\text{c},t}$ is the preferred choice for $\mathcal{Q}_t$.

The second choice for $\mathcal{Q}_t$ is the set $\mathcal{Q}_{\text{nc},t}$ which is defined as the set of allowed values for ΔQ_t as prescribed by the disturbance(s) in $(DV_t)_{t=0}^{79}$ not allowing carries:

$$\mathcal{Q}_{\text{nc},t} = \left\{ Y \in \mathcal{Q}_{\text{c},0,t} \;\middle|\; Y[i] \in \{0, -DV_{t-1}[i], DV_{t-1}[i]\} \right\}.$$

No carries can be present as $Y \in \mathcal{Q}_{\text{c},0,t}$. Nevertheless, $\mathcal{Q}_{\text{c},0,t}$ still allows the disturbances $\Delta Q_t = \{2, \overline{0}\}$ and $\Delta Q_t = \{1, 0\}$ (using compact notation, see p. 18) in case of two serial local collisions starting at step $t-1$: $DV_{t-1} = 2^1 + 2^0$. The condition $Y[i] \in \{0, -DV_{t-1}[i], DV_{t-1}[i]\}$ prevents disturbances associated with '0'-bits in the disturbance vector.

We assume a choice has been made for $\mathcal{Q}_t$ and preferably this is $\mathcal{Q}_{\text{c},t}$, however we may need another choice for $\mathcal{Q}_t$ in the upcoming sections. To simplify notation, we

use $\sigma(\mathcal{Q}_t)$ and $\sigma(RL(\mathcal{Q}_t, n))$ to denote $\{\sigma(Y) \mid Y \in \mathcal{Q}_t\}$ and $\{\sigma(RL(Y, n)) \mid Y \in \mathcal{Q}_t\}$, respectively. The remaining definitions in this section may depend on the particular choice of $\mathcal{Q}_t$.

Let $(DW_t)_{t=0}^{79} \in \mathbb{Z}_{2^{32}}^{80}$ be the associated message expansion XOR difference vector of the disturbance vector (see Equation 7.9, p. 121). Then for $t = 0, \ldots, 79$, we define the set $\mathcal{W}_t$ as the set of possible δW_t given XOR difference DW_t:

$$\mathcal{W}_t = \Big\{\sigma(\Delta W_t) \;\Big|\; \Delta W_t[i] \in \{-DW_t[i], DW_t[i]\},\ i = 0, \ldots, 31\Big\}.$$

Let $\mathcal{D}_{[i,j]}$ be the set of all differential paths $\mathcal{P}$ over steps $i, \ldots, j$ with:

- $\delta W_t \in \mathcal{W}_t$ for $t = i, \ldots, j$;
- $\Delta Q_t \in \mathcal{Q}_t$ for $t = i-3, \ldots, j$;
- $\delta RL(Q_{i-4}, 30) \in \sigma(RL(\mathcal{Q}_{i-4}, 30))$;
- $\delta Q_{j+1} \in \sigma(\mathcal{Q}_{j+1})$;
- $\Pr[\mathcal{P}] > 0$.

Informally, $\mathcal{D}_{[i,j]}$ is the set of all possible differential paths over steps $i, \ldots, j$ that follow the local collision differences as prescribed by the disturbance vector where carries are limited through the choice for $\mathcal{Q}_t$. These are the differential paths that we are interested in. Let $\mathcal{R}_{[i,j]}$ denote the set $\{\text{Reduce}(\mathcal{P}) \mid \mathcal{P} \in \mathcal{D}_{[i,j]}\}$ of all reduced versions of the differential paths in $\mathcal{D}_{[i,j]}$. We like to point out that $|\mathcal{R}_{[i,j]}| \leq |\mathcal{D}_{[i,j]}|$ and that $|\mathcal{R}_{[i,j]}|$ can be significantly smaller than $|\mathcal{D}_{[i,j]}|$, especially for a large number of steps $j - i + 1$.

Now we can analyze the success probability of the local collision. The success probability $p_{w,[t_b,t_e]}$ of the single local collision in $(DV_t)_{t=0}^{79}$ which begins at step $t_b \geq 0$ and ends with step $t_e < 80$ using given message difference vector $w = (\delta W_t)_{t=t_b}^{t_e} \in (\mathcal{W}_t)_{t=t_b}^{t_e}$ is determined as:

$$p_{w,[t_b,t_e]} = \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,t_e]} \\ (\delta \widehat{W}_t)_{t=t_b}^{t_e} = w}} \Pr[\widehat{\mathcal{P}}].$$

Thus to analyze the local collision we generate differential paths $\mathcal{P} \in \mathcal{D}_{[i,j]}$. Instead of storing all possible differential paths, we keep only reduced differential paths along with intermediary probabilities. For a reduced differential path $\mathcal{P}_r$ over steps $i, \ldots, j$ and message difference vector $w \in (\mathcal{W}_t)_{t=i}^{j}$, we define the intermediary probability $p_{(w,\mathcal{P}_r,[i,j])}$ as:

$$p_{(w,\mathcal{P}_r,[i,j])} = \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[i,j]} \\ \text{Reduce}(\widehat{\mathcal{P}}) = \mathcal{P}_r \\ (\delta \widehat{W}_t)_{t=i}^{j} = w}} \Pr[\widehat{\mathcal{P}}] / \Pr[\mathcal{P}_r].$$

Algorithm 7-2 Local collision analysis (forward)

Let $(DV_t)_{t=0}^{79}$, $(DW_t)_{t=0}^{79}$, $(\mathcal{W}_t)_{t=0}^{79}$ and $(\mathcal{Q}_{t+1})_{t=0}^{79}$ be as defined in Section 7.5.11 and let $I = [t_b, t_e]$ be an interval of steps $t_b, \ldots, t_e$ such that $DV_i = 0$ for $i \in \{t_b{-}5, \ldots, t_b{-}1\}$.

1. We start with $\mathcal{A}_{t_b,t_b} = \emptyset$;
2. For all differential paths $\mathcal{P}$ over step t_b of the following form:

$$\delta RL(Q_{t_b-4}, 30) = 0, \quad \Delta Q_{t_b-3} = \ldots = \Delta Q_{t_b} = 0, \quad \Delta F_{t_b} = 0,$$

$$\delta Q_{t_b+1} = \delta W_{t_b} \in \mathcal{W}_{t_b} \cap \sigma(\mathcal{Q}_{t_b+1}),$$

   we insert the tuple $(\mathcal{P}, \{((\delta W_i)_{i=t_b}^{t_b}, 1)\})$ in the set $\mathcal{A}_{t_b,t_b}$;
3. For steps $t = t_b + 1, \ldots, t_e$ in that order we do the following:
4. Let $\mathcal{A}_{t_b,t} = \emptyset$;
5. For all tuples $(\mathcal{P}, \mathcal{S}) \in \mathcal{A}_{t_b,t-1}$ we extend $\mathcal{P}$ forward:
6. For all BSDRs $\Delta Q_t \in \mathcal{Q}_t$ of δQ_t:
7. For all $\delta W_t \in \mathcal{W}_t$, $\delta Q_{t+1} \in \sigma(\mathcal{Q}_{t+1})$ and ΔF_t such that

$$(\Delta F_t[b])_{b=0}^{31} \in (V_{t,b})_{b=0}^{31} \quad \dagger \quad \text{and}$$

$$\delta Q_{t+1} = \sigma(RL(\Delta Q_t, 5)) + \sigma(RL(\Delta Q_{t-4}, 30)) + \sigma(\Delta F_t) + \delta W_t$$

   do the following:
8. Let $\mathcal{P}_e$ be $\mathcal{P}$ extended with step t using ΔQ_t, ΔF_t, δW_t and δQ_{t+1};
9. If $\Pr[\mathcal{P}_e] > 0$ then do:
10. Let $\mathcal{P}_r = \text{Reduce}(\mathcal{P}_e)$, $p_r = \Pr[\mathcal{P}_e] / \Pr[\mathcal{P}_r]$ and

$$\widehat{\mathcal{S}} = \left\{ ((\delta\widehat{W}_i)_{i=t_b}^{t}, p_r \cdot \widehat{p}) \;\middle|\; ((\delta\widehat{W}_i)_{i=t_b}^{t-1}, \widehat{p}) \in \mathcal{S} \right\},$$

    where $\delta\widehat{W}_t = \delta W_t$;
11. If there exists a tuple $(\mathcal{P}_r, \widetilde{\mathcal{S}}) \in \mathcal{A}_{t_b,t}$ for some $\widetilde{\mathcal{S}}$ then replace $(\mathcal{P}_r, \widetilde{\mathcal{S}})$ in $\mathcal{A}_{t_b,t}$ by the tuple $(\mathcal{P}_r, \mathcal{S}_{\mathcal{P}_r})$, where

$$\mathcal{S}_{\mathcal{P}_r} = \left\{ \left(w, \sum_{(w,p') \in \widehat{\mathcal{S}} \cup \widetilde{\mathcal{S}}} p' \right) \;\middle|\; w \in \left\{ w' \middle| (w', p') \in \widehat{\mathcal{S}} \cup \widetilde{\mathcal{S}} \right\} \right\},$$

12. otherwise insert $(\mathcal{P}_r, \widehat{\mathcal{S}})$ in $\mathcal{A}_{t_b,t}$.
13. Return $\mathcal{A}_{t_b,t_e}$.

† See Equation 7.16 (p. 142).

The set $\mathcal{S}_{\mathcal{P}_r}$ for a reduced differential path $\mathcal{P}_r \in \mathcal{R}_{[i,j]}$ is defined as the set of all possible tuples $(w, p_{(w,\mathcal{P}_r,[i,j])})$ with $w \in (\mathcal{W}_t)_{t=i}^{j}$ and $p_{(w,\mathcal{P}_r,[i,j])} > 0$. Then for $0 \le i \le j < 80$, the set $\mathcal{A}_{i,j}$ is defined as the set of all tuples $(\mathcal{P}_r, \mathcal{S}_{\mathcal{P}_r})$ of differential paths $\mathcal{P}_r \in \mathcal{R}_{[i,j]}$ with $\mathcal{S}_{\mathcal{P}_r} \neq \emptyset$. The set $\mathcal{A}_{t_b,t_e}$ thus consists of all reduced versions of differential paths in $\mathcal{D}_{[t_b,t_e]}$ together with the intermediary probabilities for each possible message difference vector. To compute the set $\mathcal{A}_{t_b,t_e}$, we use Algorithm 7-2

Algorithm 7-3 Local collision analysis (backward)

Let $(DV_t)_{t=0}^{79}$, $(DW_t)_{t=0}^{79}$, $(\mathcal{W}_t)_{t=0}^{79}$ and $(\mathcal{Q}_{t+1})_{t=0}^{79}$ be as defined in Section 7.5.11 and let $I = [t_b, t_e]$ be an interval of steps $t_b, \ldots, t_e$ such that $DV_i = 0$ for $i \in \{t_e{-}4, \ldots, t_e\}$.

1. We start with $\mathcal{A}_{t_e,t_e} = \emptyset$;
2. For all differential paths $\mathcal{P}$ over step t_e of the following form:

$$\Delta Q_{t_e-3} = \ldots = \Delta Q_{t_e} = 0, \quad \delta Q_{t_e+1} = 0, \quad \Delta F_{t_e} = 0,$$

$$\delta RL(Q_{t_e-4}, 30) = -\delta W_{t_e} \in \sigma(RL(\mathcal{Q}_{t_e-4}, 30)), \quad \delta W_{t_e} \in \mathcal{W}_{t_e}$$

   we insert the tuple $(\mathcal{P}, \{((\delta W_i)_{i=t_e}^{t_e}, 1)\})$ in the set $\mathcal{A}_{t_e,t_e}$;
3. For steps $t = t_e - 1, \ldots, t_b$ in that order we do the following:
4. Let $\mathcal{A}_{t,t_e} = \emptyset$;
5. For all tuples $(\mathcal{P}, \mathcal{S}) \in \mathcal{A}_{t+1,t_e}$ we extend $\mathcal{P}$ backwards:
6. For all BSDRs $Y \in \mathcal{Q}_{t-3}$ with $\sigma(RL(Y, 30)) = \delta RL(Q_{t-3}, 30)$:
7. Let $\Delta Q_{t-3} = Y$;
8. For all $\delta W_t \in \mathcal{W}_t$, $\delta RL(Q_{t-4}, 30) \in \sigma(RL(\mathcal{Q}_{t-4}, 30))$ and ΔF_t such that $(\Delta F_t[b])_{b=0}^{31} \in (V_{t,b})_{b=0}^{31}$ and

$$\delta RL(Q_{t-4}, 30) = \sigma(\Delta Q_{t+1}) - \sigma(RL(\Delta Q_t, 5)) - \sigma(\Delta F_t) - \delta W_t$$

   do the following:
9. Let $\mathcal{P}_e$ be $\mathcal{P}$ extended with step t using $\delta RL(Q_{t-4}, 30)$, ΔQ_{t-3}, ΔF_t and δW_t;
10. If $\Pr[\mathcal{P}_e] > 0$ then do:
11. Let $\mathcal{P}_r = \text{Reduce}(\mathcal{P}_e)$, $p_r = \Pr[\mathcal{P}_e] / \Pr[\mathcal{P}_r]$ and

$$\widehat{\mathcal{S}} = \left\{ ((\delta\widehat{W}_i)_{i=t}^{t_e}, p_r \cdot \widehat{p}) \;\middle|\; ((\delta\widehat{W}_i)_{i=t+1}^{t_e}, \widehat{p}) \in \mathcal{S} \right\},$$

    where $\delta\widehat{W}_t = \delta W_t$;
12. If there exists a tuple $(\mathcal{P}_r, \widetilde{\mathcal{S}}) \in \mathcal{A}_{t,t_e}$ for some $\widetilde{\mathcal{S}}$ then replace $(\mathcal{P}_r, \widetilde{\mathcal{S}})$ in $\mathcal{A}_{t,t_e}$ by the tuple $(\mathcal{P}_r, \mathcal{S}_{\mathcal{P}_r})$, where

$$\mathcal{S}_{\mathcal{P}_r} = \left\{ \left(w, \sum_{(w,p') \in \widehat{\mathcal{S}} \cup \widetilde{\mathcal{S}}} p' \right) \;\middle|\; w \in \left\{ w' \middle| (w', p') \in \widehat{\mathcal{S}} \cup \widetilde{\mathcal{S}} \right\} \right\},$$

13. otherwise insert $(\mathcal{P}_r, \widehat{\mathcal{S}})$ in $\mathcal{A}_{t,t_e}$.
14. Return $\mathcal{A}_{t_b,t_e}$.

(p. 150) to iteratively construct the sets $\mathcal{A}_{t_b,j}$ for $j = t_b, \ldots, t_e$.

Since $DV_t = 0$ for $t = t_b - 5, \ldots, t_b - 1$ and for $t = t_e - 4, \ldots, t_e$ for this single local collision, it follows that for all $(\mathcal{P}, \mathcal{S}) \in \mathcal{A}_{t_b,t_e}$ we have for $\mathcal{P}$ by definition $\delta RL(Q_{t_b-4}, 30) = 0$, $\Delta Q_i = (0)_{j=0}^{31}$ for $i = t_b - 3, \ldots, t_b$, $\Delta Q_i = (0)_{j=0}^{31}$ for $i =$

$t_e - 3, \ldots, t_e$ and $\delta Q_{t_e+1} = 0$. Observation 7.1 (p. 148) implies that $\mathcal{P}$ is trivial:

$$\mathcal{P} = (0, ((0)_{j=0}^{31})_{i=t_b-4}^{t_e+1}, 0, ((0)_{j=0}^{31})_{i=t_b}^{t_e}, (0)_{i=t_b}^{t_e}).$$

Hence, $\mathcal{A}_{t_b,t_e}$ consists of a single tuple $(\mathcal{P}, \mathcal{S})$ where $\mathcal{P}$ is trivial and the set $\mathcal{S}$ contains all tuples $((\delta W_t)_{t=t_b}^{t_e}, p)$ where $p > 0$ by definition is the desired success probability $p_{w,[t_b,t_e]}$ of $w = (\delta W_t)_{t=t_b}^{t_e}$. Most importantly we have determined the highest success probability $p_{[t_b,t_e]} = \max\{p \mid (w,p) \in \mathcal{S}\}$ and all message difference vectors that attain that probability: $\{w \mid (w, p_{[t_b,t_e]}) \in \mathcal{S}\}$.

A similar analysis is possible backwards by iteratively constructing sets $\mathcal{A}_{i,t_e}$ for $i = t_e, \ldots, t_b$ using Algorithm 7-3 (p. 151). Note that Algorithm 7-2 expects $DV_t = 0$ for $t \in \{t_b - 5, \ldots, t_b - 1\}$, whereas Algorithm 7-3 expects $DV_t = 0$ for $t \in \{t_e - 4, \ldots, t_e\}$. Hence, working backwards provides an alternative whenever $DV_t \neq 0$ for some $t \in \{t_b - 5, \ldots, t_b - 1\}$ and $DV_t = 0$ for $t \in \{t_e - 4, \ldots, t_e\}$.

7.5.6 Message difference compression

Since the number of possible message difference vectors grows exponentially in the number of steps, we employ another technique that allows us to 'combine' message difference vectors. This is nothing more than a smart representation of $\mathcal{A}_{i,j}$ by a pair $(\mathcal{SR}_{i,j}, \mathcal{B}_{i,j})$ that has significantly reduced memory footprint and also allows an optimization that reduces the runtime complexity. In the implementations of Algorithms 7-2 and 7-3, we use the pair $(\mathcal{SR}_{i,j}, \mathcal{B}_{i,j})$ representing $\mathcal{A}_{i,j}$ to improve runtime and memory complexity. Nevertheless, in the upcoming sections we still use $\mathcal{A}_{i,j}$ instead of the equivalent representation $(\mathcal{SR}_{i,j}, \mathcal{B}_{i,j})$ for ease of notation.

First we introduce some necessary notation. We denote a message difference vector substitution rule as $(w_n)_{n=i}^{j} \hookrightarrow (v_n)_{n=i}^{j}$. For a message difference vector $(x_k)_{k=l}^{m}$ and a message difference vector substitution rule $(w_n)_{n=i}^{j} \hookrightarrow (v_n)_{n=i}^{j}$ such that $l \leq i \leq j \leq m$, we define $\Pi((x_k)_{k=l}^{m}, (w_n)_{n=i}^{j} \hookrightarrow (v_n)_{n=i}^{j})$ as the message difference vector $(y_k)_{k=l}^{m}$, where for $k \in \{l, \ldots, m\}$:

$$y_k = \begin{cases} x_k & \text{if } (x_n)_{n=i}^{j} \neq (w_n)_{n=i}^{j} \vee k \notin \{i, \ldots, j\}; \\ v_k & \text{otherwise.} \end{cases}$$

This definition of Π is extended to a set $\mathcal{SR}$ of message difference vector substitution rules:

$$\Pi(X, \mathcal{SR}) = \left\{\Pi(\cdots \Pi(X, s_1), \cdots, s_n) \;\middle|\; n \in \{0, \ldots, |\mathcal{SR}|\}, s_1, \ldots, s_n \in \mathcal{SR}\right\},$$

which includes X itself by using $n = 0$.

Now we can represent $\mathcal{A}_{i,j}$ in Algorithms 7-2 and 7-3 as a pair $(\mathcal{SR}_{i,j}, \mathcal{B}_{i,j})$ of a set of substitution rules $\mathcal{SR}_{i,j}$ and a set $\mathcal{B}_{i,j}$ of the form

$$\mathcal{B}_{i,j} = \{(\mathcal{P}, \mathcal{S}'_{\mathcal{P}}) \mid (\mathcal{P}, \mathcal{S}_{\mathcal{P}}) \in \mathcal{A}_{i,j}\}$$

such that for all $\mathcal{P}$ the set $\mathcal{S}'_{\mathcal{P}}$ together with $\mathcal{SR}_{i,j}$ generates $\mathcal{S}_{\mathcal{P}}$:

$$\mathcal{S}_{\mathcal{P}} = \bigcup_{(w,p)\in\mathcal{S}'_{\mathcal{P}}} \{(v,p) \mid v \in \Pi(w, \mathcal{SR}_{i,j})\}.^{33}$$

To obtain a representation $(\mathcal{SR}_{i,j}, \mathcal{B}_{i,j})$ of $\mathcal{A}_{i,j}$, we do the following. We start with $\mathcal{B}_{t_b,t} = \mathcal{A}_{t_b,t}$ and $\mathcal{SR}_{t_b,t} = \emptyset$. Let

$$\mathcal{W} = \{w \mid (w,p) \in \mathcal{S}_{\mathcal{P}}, (\mathcal{P}, \mathcal{S}_{\mathcal{P}}) \in \mathcal{A}_{i,j}\}$$

be the set of all message difference vectors w under consideration. For each $w \in \mathcal{W}$ we define the function θ_w that maps differential paths $\mathcal{P}$ to the respective intermediary probability of w and $\mathcal{P}$:

$$\theta_w : \{\mathcal{P} \mid (\mathcal{P}, \mathcal{S}) \in \mathcal{A}_{i,j}\} \to [0,1]$$

$$\theta_w : \mathcal{P} \mapsto p, \quad \text{for which } (w,p) \in \mathcal{S}_{\mathcal{P}}$$

Now for all groups $w_1, \ldots, w_K \in \mathcal{W}$ of message difference vectors that have identical functions $\theta_{w_1} = \ldots = \theta_{w_K}$, we remove all occurrences of $w_2, \ldots, w_K$ in $\mathcal{B}_{i,j}$ and compensate by inserting message difference vectors substitution rules $w_1 \hookrightarrow w_2$, $w_1 \hookrightarrow w_3$, ..., $w_1 \hookrightarrow w_K$ into $\mathcal{SR}_{i,j}$.[34] We denote the resulting representation $(\mathcal{SR}_{i,j}, \mathcal{B}_{i,j})$ of $\mathcal{A}_{i,j}$ as $\text{Compact}(\mathcal{A}_{i,j})$.

Let $\mathcal{A}_{t_b,t} = \text{Extend}(\mathcal{A}_{t_b,t-1})$ denote the computation of $\mathcal{A}_{t_b,t}$ using $\mathcal{A}_{t_b,t-1}$ in steps 4–12 of Algorithm 7-2. Then $(\mathcal{SR}_{t_b,t}, \mathcal{B}_{t_b,t})$ can be computed from $(\mathcal{SR}_{t_b,t-1}, \mathcal{B}_{t_b,t-1})$ as

$$(\mathcal{SR}_{t_b,t}, \mathcal{B}_{t_b,t}) = \text{Compact}\big(\text{Extend}\big((\mathcal{SR}_{t_b,t-1}, \mathcal{B}_{t_b,t-1})\big)\big).$$

Although this implies that we still need to store the entire set $\mathcal{A}_{t_b,t}$ at some point.

Note that in the steps 4 to 12 of Algorithm 7-2 for any $\mathcal{P}$ the values $\mathcal{P}_r$ and p_r do not directly depend on values $w \in \{v \mid (v,p) \in \mathcal{S}\}$. One can thus observe that for any two message difference vectors w and v used in $\mathcal{A}_{i,j}$ for which $\theta_w = \theta_v$, it follows that also $\theta_{w'} = \theta_{v'}$ for any extensions $w' = w||x$ and $v' = v||x$ of w and v by the same message differences x (as in step 10 of Algorithm 7-2). This implies that in the reduction of $\mathcal{A}_{t_b,t}$ to $(\mathcal{SR}_{t_b,t}, \mathcal{B}_{t_b,t}) = \text{Compact}(\mathcal{A}_{t_b,t})$ all these message difference vectors v' can be removed by using the substitution rule $w \hookrightarrow v$.

More specifically, let $(\mathcal{SR}_{t_b,t-1}, \mathcal{B}_{t_b,t-1})$ be a representation of $\mathcal{A}_{t_b,t-1}$, then for step t in Algorithm 7-2 the above observation holds for all w and $v \in \Pi(w, \mathcal{SR}_{t_b,t-1})$ and thus all extensions of such v in $\mathcal{A}_{t_b,t}$ can be removed and compensated by the substitution rule $w \hookrightarrow v$. This implies that

$$(\mathcal{SR}_{t_b,t-1}, \text{Extend}(\mathcal{B}_{t_b,t-1}))$$

33. This directly implies that for $\mathcal{SR}_{i,j} = \emptyset$ we have that $\mathcal{B}_{i,j} = \mathcal{A}_{i,j}$. Note that for any two message difference vectors w and $v \in \Pi(w, \mathcal{SR}_{i,j})$ their respective probabilities for all differential paths $\mathcal{P}$ must be equal: $\forall \mathcal{P} : (w,p) \in \mathcal{S}_{\mathcal{P}} \Leftrightarrow (v,p) \in \mathcal{S}_{\mathcal{P}}$.

34. To efficiently determine these groups with identical functions θ_w, we compute and compare hash values of a representation of θ_w for all $w \in \mathcal{W}$ instead of these function θ_w themselves.

is a representation of $\mathcal{A}_{t_b,t}$, which naturally may be further reduced.

We can thus obtain a representation $(\mathcal{SR}_{t_b,t}, \mathcal{B}_{t_b,t})$ of $\mathcal{A}_{t_b,t}$ more efficiently than simply computing $\text{Compact}\big(\text{Extend}\big((\mathcal{SR}_{t_b,t-1}, \mathcal{B}_{t_b,t-1})\big)\big)$ as follows:

$$(\widetilde{\mathcal{SR}}_{t_b,t}, \widetilde{\mathcal{B}}_{t_b,t}) = \text{Compact}(\text{Extend}(\mathcal{B}_{t_b,t-1}));$$
$$(\mathcal{SR}_{t_b,t}, \mathcal{B}_{t_b,t}) = (\widetilde{\mathcal{SR}}_{t_b,t} \cup \mathcal{SR}_{t_b,t-1}, \widetilde{\mathcal{B}}_{t_b,t}).$$

In this manner we can obtain a representation of the set $\mathcal{A}_{t_b,t_e}$, but in a way that does not require anymore that the entire set $\mathcal{A}_{t_b,t_e}$ must be computed and stored in memory. Furthermore, only the message difference vectors present in $\mathcal{B}_{t_b,t-1}$ are processed instead of all message difference vectors present in $\mathcal{A}_{t_b,t-1}$ in the processing of step t in step 3 of Algorithm 7-2.

Algorithm 7-3 is treated analogously.

7.5.7 Single local collision analysis: δIHV_{diff}

Consider a single local collision starting at step $t_b \geq 75$ and ending with step $t_e = 79$. Since $79 = t_e < t_b + 5$, not all corrections have been made after step 79 and it follows that $\mathcal{A}_{t_b,t_e}$ does not consist of a single trivial reduced differential path. In this case we are interested in the most likely differences $\delta IHV_{\text{diff}} = \delta IHV_{\text{out}} - \delta IHV_{\text{in}}$ added to the IHV according to a differential path $\mathcal{P}$ over steps $t_b, \ldots, 79$:

$$\phi(\widehat{\mathcal{P}}) = (d_i)_{i=76}^{80}, \quad d_i = \begin{cases} \sigma(RL(\Delta\widehat{Q}_i, 30)), & i = 76, 77, 78; \\ \sigma(\Delta\widehat{Q}_i), & i = 79; \\ \delta\widehat{Q}_i, & i = 80. \end{cases}$$

Using Algorithm 7-2 we compute the set $\mathcal{A}_{t_b,79}$ and we determine the set $\mathcal{I}$ of possible δIHV_{diff}: $\mathcal{I} = \{\phi(\widehat{\mathcal{P}}) \mid (\widehat{\mathcal{P}}, \mathcal{S}) \in \mathcal{A}_{t_b,79}\}$.

We can determine the success probability $p_{(w,\delta IHV_{\text{diff}},[t_b,79])}$ for each message difference vector $w = (\delta W_t)_{t=t_b}^{79} \in (\mathcal{W}_t)_{t=t_b}^{79}$ and for each $\delta IHV_{\text{diff}} \in \mathcal{I}$:

$$\begin{aligned} p_{(w,\delta IHV_{\text{diff}},[t_b,79])} &= \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,79]} \\ \phi(\widehat{\mathcal{P}}) = \delta IHV_{\text{diff}} \\ (\delta\widehat{W}_t)_{t=t_b}^{79} = w}} \Pr[\widehat{\mathcal{P}}] \\ &= \sum_{\substack{(\widehat{\mathcal{P}}, \mathcal{S}) \in \mathcal{A}_{t_b,79} \\ \phi(\widehat{\mathcal{P}}) = \delta IHV_{\text{diff}}}} \sum_{(w, p_{(w,\widehat{\mathcal{P}},[t_b,79])}) \in \mathcal{S}} p_{(w,\widehat{\mathcal{P}},[t_b,79])} \cdot \Pr[\widehat{\mathcal{P}}]. \end{aligned}$$

Finally, we can determine the highest success probability:

$$p_{[t_b,79]} = \max\{p_{(w,\delta IHV_{\text{diff}},[t_b,79])} \mid \delta IHV_{\text{diff}} \in \mathcal{I},\ w \in (\mathcal{W}_t)_{t=t_b}^{79}\}$$

and all δIHV_{diff} that attain that probability (almost):

$$\{\delta IHV_{\text{diff}} \mid \delta IHV_{\text{diff}} \in \mathcal{I},\ \exists w : p_{(w,\delta IHV,[t_b,79])} \geq p_{[t_b,79]} \cdot \alpha\},$$

where $\alpha \in [0,1]$ is a given factor, i.e., $\alpha = 0.95$. For each such δIHV_{diff} we determine all w for which $p_{(w,\delta IHV,[t_b,79])} \geq p_{[t_b,79]} \cdot \alpha$:

$$\{w \mid w \in (\mathcal{W}_t)_{t=t_b}^{79},\ p_{(w,\delta IHV_{\text{diff}},[t_b,79])} \geq p_{[t_b,79]} \cdot \alpha\}.$$

7.5.8 Single local collision analysis: alternative δIHV_{diff}

Our analysis so far only allowed working state differences as prescribed by the disturbances in $(DV_t)_{t=0}^{79}$ either with or without carries. However, for the last steps it is not necessary anymore to restrict ΔQ_t. We are interested in δIHV_{diff} with high probability, not necessarily one exactly as prescribed by disturbances in $(DV_t)_{t=0}^{79}$. To that end we choose $\mathcal{Q}_t$ for $t \in \{76,\ldots,80\}$ as the set of all low weight BSDRs or even by the set of all BSDRs as desired. Then using Algorithm 7-4 we iteratively generate sets $\mathcal{Y}_{t_b,i}$ for $i = t_b,\ldots,79$ that are defined as the set of all tuples $(\mathcal{P}_r, s_{\mathcal{P}_r})$ of differential paths $\mathcal{P}_r \in \mathcal{R}_{[t_b,i]}$ and associated weights $s_{\mathcal{P}_r}$. The associated weight $s_{\mathcal{P}_r}$ of a reduced differential path $\mathcal{P}_r \in \mathcal{R}_{[t_b,i]}$ is defined as

$$s_{\mathcal{P}_r} = \sum_{w \in (\mathcal{W})_{t=t_b}^{i}} p_{(w,\mathcal{P}_r,[t_b,i])},$$

where $p_{(w,\mathcal{P}_r,[t_b,i])}$ is the probability as defined earlier (using our choice of $\mathcal{Q}_t$).

The last set $\mathcal{Y}_{t_b,79}$ is used to determine for possible δIHV_{diff} a weight $s_{\delta IHV_{\text{diff}}}$ which is the sum of success probabilities over allowed message difference vectors:

$$\begin{aligned}
s_{\delta IHV_{\text{diff}}} &= \sum_{\substack{(\mathcal{P},s_{\mathcal{P}}) \in \mathcal{Y}_{t_b,79} \\ \phi(\mathcal{P}) = \delta IHV_{\text{diff}}}} s_{\mathcal{P}} \cdot \Pr[\mathcal{P}] \\
&= \sum_{\substack{(\mathcal{P},s_{\mathcal{P}}) \in \mathcal{Y}_{t_b,79} \\ \phi(\mathcal{P}) = \delta IHV_{\text{diff}}}} \sum_{w \in (\mathcal{W}_t)_{t=t_b}^{79}} \Pr[\mathcal{P}] \cdot p_{(w,\mathcal{P},[t_b,79])} \\
&= \sum_{\substack{(\mathcal{P},s_{\mathcal{P}}) \in \mathcal{Y}_{t_b,79} \\ \phi(\mathcal{P}) = \delta IHV_{\text{diff}}}} \sum_{w \in (\mathcal{W}_t)_{t=t_b}^{79}} \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,79]} \\ \text{Reduce}(\widehat{\mathcal{P}}) = \mathcal{P} \\ (\delta \widehat{W}_t)_{t=t_b}^{79} = w}} \Pr[\widehat{\mathcal{P}}] \\
&= \sum_{w \in (\mathcal{W}_t)_{t=t_b}^{79}} \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,79]} \\ \phi(\widehat{\mathcal{P}}) = \delta IHV_{\text{diff}} \\ (\delta \widehat{W}_t)_{t=t_b}^{79} = w}} \Pr[\widehat{\mathcal{P}}] \\
&= \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,79]} \\ \phi(\widehat{\mathcal{P}}) = \delta IHV_{\text{diff}}}} \Pr[\widehat{\mathcal{P}}].
\end{aligned}$$

Algorithm 7-4 δIHV_{diff} analysis (forward)

Let $(DV_t)_{t=0}^{79}$, $(DW_t)_{t=0}^{79}$, $(\mathcal{W}_t)_{t=0}^{79}$ and $(\mathcal{Q}_{t+1})_{t=0}^{79}$ be as defined in Section 7.5.11 and let $I = [t_b, 79]$ be an interval of steps $t_b, \ldots, 79$ such that $DV_i = 0$ for $i \in \{t_b - 5, \ldots, t_b - 1\}$.

1. We start with $\mathcal{Y}_{t_b,t_b} = \emptyset$;
2. For all differential paths $\mathcal{P}$ over step t_b of the following form:

$$\delta RL(Q_{t_b-4}, 30) = 0, \quad \Delta Q_{t_b-3} = \ldots = \Delta Q_{t_b} = 0, \quad \Delta F_{t_b} = 0,$$

$$\delta Q_{t_b+1} = \delta W_{t_b} \in \mathcal{W}_{t_b} \cap \sigma(\mathcal{Q}_{t_b+1}),$$

   we insert the tuple $(\mathcal{P}, 1)$ in the set $\mathcal{Y}_{t_b,t_b}$;
3. For steps $t = t_b + 1, \ldots, 79$ in that order we do the following:
4. Let $\mathcal{Y}_{t_b,t} = \emptyset$;
5. For all tuples $(\mathcal{P}, s) \in \mathcal{Y}_{t_b,t-1}$ we extend $\mathcal{P}$ forward:
6. For all BSDRs $\Delta Q_t \in \mathcal{Q}_t$ of δQ_t:
7. For all $\delta W_t \in \mathcal{W}_t$, $\delta Q_{t+1} \in \sigma(\mathcal{Q}_{t+1})$ and ΔF_t such that

$$(\Delta F_t[b])_{b=0}^{31} \in (V_{t,b})_{b=0}^{31} \quad \dagger \quad \text{and}$$

$$\delta Q_{t+1} = \sigma(RL(\Delta Q_t, 5)) + \sigma(RL(\Delta Q_{t-4}, 30)) + \sigma(\Delta F_t) + \delta W_t$$

   do the following:
8. Let $\mathcal{P}_e$ be $\mathcal{P}$ extended with step t using ΔQ_t, ΔF_t, δW_t and δQ_{t+1};
9. If $\Pr[\mathcal{P}_e] > 0$ then do:
10. Let $\mathcal{P}_r = \text{Reduce}(\mathcal{P}_e)$ and $s_r = s \cdot \Pr[\mathcal{P}_e] / \Pr[\mathcal{P}_r]$;
11. If there exists a tuple $(\mathcal{P}_r, \widetilde{s}) \in \mathcal{Y}_{t_b,t}$ for some $\widetilde{s}$ then replace $(\mathcal{P}_r, \widetilde{s})$ in $\mathcal{Y}_{t_b,t}$ by the tuple $(\mathcal{P}_r, s_r + \widetilde{s})$,
12. otherwise insert $(\mathcal{P}_r, s_r)$ in $\mathcal{Y}_{t_b,t}$.
13. Return $\mathcal{Y}_{t_b,79}$.

† See Equation 7.16 (p. 142).

We select a set $\widehat{\mathcal{I}}$ of differences δIHV_{diff} with high weights $s_{\delta IHV_{\text{diff}}}$ and define the sets $\mathcal{Q}_{\text{ihv},t}$ for $t = 1, \ldots, 80$:

$$\mathcal{Q}_{\text{ihv},t} = \begin{cases} \left\{ \Delta Q_t \;\middle|\; \exists (X_i)_{i=76}^{80} \in \widehat{\mathcal{I}} : \sigma(RL(\Delta Q_t, 30)) = X_t \right\}, & t \in \{76, 77, 78\}; \\ \left\{ \Delta Q_t \;\middle|\; \exists (X_i)_{i=76}^{80} \in \widehat{\mathcal{I}} : \sigma(\Delta Q_t) = X_t \right\}, & t \in \{79, 80\}; \\ \mathcal{Q}_t, & t < 76. \end{cases}$$

We can define related sets $\mathcal{Q}_{\text{ihv},u,t}$ for $u \in \{0,\ldots,32\}$ and $t \in \{1,\ldots,80\}$ where the weights are bounded by u plus the NAF weight:

$$\mathcal{Q}_{\text{ihv},u,t} = \{Y \mid Y \in \mathcal{Q}_{\text{ihv},t} \wedge w(Y) \leq w(\text{NAF}(\sigma(Y))) + u\}.$$

By repeating the analysis in Section 7.5.7 using the sets $\mathcal{Q}_{\text{ihv},u,t}$ for $\mathcal{Q}_t$, we can determine the success probabilities for each $\delta IHV_{\text{diff}} \in \widehat{\mathcal{I}}$. In this way we also find the highest success probability among those δIHV_{diff} as well as the message difference vectors that enable this highest success probability.

7.5.9 Single local collision analysis: round 1

If $DV_t = 0$ for $t \in \{t_e - 4,\ldots,t_e\}$ but not for all $t \in \{t_b - 5,\ldots,t_b - 1\}$ (e.g., if $20 = t_b > t_e - 5$) then the local collision is truncated and $\mathcal{A}_{t_b,t_e}$ does not consist of a single trivial reduced differential path. Instead we assume that t_b is chosen as the first step whose success probability should be taken into account, i.e., steps before t_b are ignored in the cost of a local collision since they are assumed to be fulfilled by message modification techniques. Thus we are interested in the most likely working state differences $\Lambda = \psi(\widehat{\mathcal{P}})$ of $\widehat{\mathcal{P}}$:

$$\psi(\widehat{\mathcal{P}}) = (d_i)_{i=t_b-4}^{t_b}, \quad d_i = \begin{cases} RR(\text{NAF}(\delta RL(\widehat{Q}_i, 30)), 30), & i = t_b - 4; \\ \Delta\widehat{Q}_i, & i = t_b - 3,\ldots,t_b; \end{cases}$$

Here we use $\Delta\widehat{Q}_t$ instead of $\delta\widehat{Q}_t$ as we assume that the final bitconditions disallow additional carries. Nevertheless, it is only a minor modification to use $\delta\widehat{Q}_t$ in the analysis below. The following analysis is very similar to the previous section.

Using Algorithm 7-3 we compute the set $\mathcal{A}_{t_b,t_e}$ and we determine the set $\mathcal{J}$ of possible Λ: $\mathcal{J} = \{\psi(\widehat{\mathcal{P}}) \mid (\widehat{\mathcal{P}},\mathcal{S}) \in \mathcal{A}_{t_b,t_e}\}$. We can determine the success probability $p_{(w,\Lambda,[t_b,t_e])}$ given message difference vector $w = (\delta W_t)_{t=t_b}^{t_e} \in (\mathcal{W}_t)_{t=t_b}^{t_e}$ for each $\Lambda \in \mathcal{J}$:

$$\begin{aligned} p_{(w,\Lambda,[t_b,t_e])} &= \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,t_e]} \\ \psi(\widehat{\mathcal{P}}) = \Lambda \\ (\delta\widehat{W}_t)_{t=t_b}^{t_e} = w}} \Pr[\widehat{\mathcal{P}}] \\ &= \sum_{\substack{(\widehat{\mathcal{P}},\mathcal{S}) \in \mathcal{A}_{t_b,t_e} \\ \psi(\widehat{\mathcal{P}}) = \Lambda}} \sum_{(w,p_{(w,\widehat{\mathcal{P}},[t_b,t_e])}) \in \mathcal{S}} p_{(w,\widehat{\mathcal{P}},[t_b,t_e])} \cdot \Pr[\widehat{\mathcal{P}}]. \end{aligned}$$

Finally, we can determine the highest success probability:

$$p_{[t_b,t_e]} = \max\{p_{(w,\Lambda,[t_b,t_e])} \mid \Lambda \in \mathcal{J},\ w \in (\mathcal{W}_t)_{t=t_b}^{t_e}\}$$

and all Λ that attain that probability:

$$\{\Lambda \mid \Lambda \in \mathcal{J},\ \exists w : p_{(w,\Lambda,[t_b,t_e])} = p_{[t_b,t_e]}\}.$$

For each such Λ we determine all w for which $p_{(w,\Lambda,[t_b,t_e])} = p_{[t_b,t_e]}$:

$$\{w \mid w \in (\mathcal{W}_t)_{t=t_b}^{t_e},\ p_{(w,\delta IHV_{\text{diff}},[t_b,t_e])} = p_{[t_b,t_e]}\}.$$

7.5.10 Single local collision analysis: alternate round 1

In the first round the differential path construction algorithms from Section 7.4 deviate from the local collisions as prescribed by the disturbance vector. Therefore, similar to the analysis of the last few steps, one can also allow all differences in the first few steps if they lead to higher success probabilities. We are interested in Λ with high probability. To that end we choose $\mathcal{Q}_t$ for $t \in \{t_b - 4, \ldots, t_b\}$ as the set of all low weight BSDRs. Then using Algorithm 7-5 we iteratively generate sets $\mathcal{Z}_{i,t_e}$ for $i = t_e - 1, \ldots, t_b$ that are defined as the set of all tuples $(\mathcal{P}_r, s_{\mathcal{P}_r})$ of differential paths $\mathcal{P}_r \in \mathcal{R}_{[t_b,t_e]}$ and associated weights $s_{\mathcal{P}_r}$. For each $\mathcal{P}_r$, the associated weight $s_{\mathcal{P}_r}$ is defined as

$$s_{\mathcal{P}_r} = \sum_{w \in (\mathcal{W})_{t=i}^{t_e}} p_{(w,\mathcal{P}_r,[i,t_e])},$$

where $p_{(w,\mathcal{P}_r,[i,t_e])}$ is the probability as defined earlier (using our choice of $\mathcal{Q}_t$).[35]

The last set $\mathcal{Z}_{t_b,t_e}$ is used to determine for possible Λ a weight s_Λ which is the sum of success probabilities over allowed message difference vectors:

$$\begin{aligned}
s_\Lambda &= \sum_{\substack{(\mathcal{P},s_\mathcal{P}) \in \mathcal{Z}_{t_b,t_e} \\ \psi(\mathcal{P}) = \Lambda}} s_\mathcal{P} \cdot \Pr[\mathcal{P}] \\
&= \sum_{\substack{(\mathcal{P},s_\mathcal{P}) \in \mathcal{Z}_{t_b,t_e} \\ \psi(\mathcal{P}) = \Lambda}} \sum_{w \in (\mathcal{W}_t)_{t=t_b}^{t_e}} \Pr[\mathcal{P}] \cdot p_{(w,\mathcal{P},[t_b,t_e])} \\
&= \sum_{\substack{(\mathcal{P},s_\mathcal{P}) \in \mathcal{Z}_{t_b,t_e} \\ \psi(\mathcal{P}) = \Lambda}} \sum_{w \in (\mathcal{W}_t)_{t=t_b}^{t_e}} \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,t_e]} \\ \mathrm{Reduce}(\widehat{\mathcal{P}}) = \mathcal{P} \\ (\delta \widehat{W}_t)_{t=t_b}^{t_e} = w}} \Pr[\widehat{\mathcal{P}}] \\
&= \sum_{w \in (\mathcal{W}_t)_{t=t_b}^{t_e}} \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,t_e]} \\ \psi(\widehat{\mathcal{P}}) = \Lambda \\ (\delta \widehat{W}_t)_{t=t_b}^{t_e} = w}} \Pr[\widehat{\mathcal{P}}] \\
&= \sum_{\substack{\widehat{\mathcal{P}} \in \mathcal{D}_{[t_b,t_e]} \\ \psi(\widehat{\mathcal{P}}) = \Lambda}} \Pr[\widehat{\mathcal{P}}].
\end{aligned}$$

We select a set $\widehat{\mathcal{J}}$ of differences Λ with high weights s_Λ and define the sets $\mathcal{Q}_{\mathrm{rnd1},t}$ for $t = 1, \ldots, 80$:

$$\mathcal{Q}_{\mathrm{rnd1},t} = \begin{cases} \left\{ \Delta Q_t \;\middle|\; (\Delta Q_i)_{i=t_b-4}^{t_b} \in \widehat{\mathcal{J}} \right\}, & t_b - 4 \le t \le t_b; \\ \mathcal{Q}_t, & \text{otherwise.} \end{cases}$$

35. This definition of $s_{\mathcal{P}_r}$ matches the definition from Section 7.5.8.

Algorithm 7-5 Λ analysis (backward)

Let $(DV_t)_{t=0}^{79}$, $(DW_t)_{t=0}^{79}$, $(\mathcal{W}_t)_{t=0}^{79}$ and $(\mathcal{Q}_{t+1})_{t=0}^{79}$ be as defined in Section 7.5.11 and let $I = [t_b, t_e]$ be an interval of steps $t_b, \ldots, t_e$ such that $DV_i = 0$ for $i \in \{t_e{-}4, \ldots, t_e\}$.

1. We start with $\mathcal{Z}_{t_e,t_e} = \emptyset$;
2. For all differential paths $\mathcal{P}$ over step t_e of the following form:

$$\Delta Q_{t_e-3} = \ldots = \Delta Q_{t_e} = 0, \quad \delta Q_{t_e+1} = 0, \quad \Delta F_{t_e} = 0,$$

$$\delta RL(Q_{t_e-4}, 30) = -\delta W_{t_e} \in \sigma(RL(\mathcal{Q}_{t_e-4}, 30)), \quad \delta W_{t_e} \in \mathcal{W}_{t_e}$$

   we insert the tuple $(\mathcal{P}, 1)$ in the set $\mathcal{Z}_{t_e,t_e}$;
3. For steps $t = t_e - 1, \ldots, t_b$ in that order we do the following:
4. Let $\mathcal{Z}_{t,t_e} = \emptyset$;
5. For all tuples $(\mathcal{P}, s) \in \mathcal{Z}_{t+1,t_e}$ we extend $\mathcal{P}$ backwards:
6. For all BSDRs $Y \in \mathcal{Q}_{t-3}$ with $\sigma(RL(Y, 30)) = \delta RL(Q_{t-3}, 30)$:
7. Let $\Delta Q_{t-3} = Y$;
8. For all $\delta W_t \in \mathcal{W}_t$, $\delta RL(Q_{t-4}, 30) \in \sigma(RL(\mathcal{Q}_{t-4}, 30))$ and ΔF_t such that $(\Delta F_t[b])_{b=0}^{31} \in (V_{t,b})_{b=0}^{31}$ and

$$\delta RL(Q_{t-4}, 30) = \sigma(\Delta Q_{t+1}) - \sigma(RL(\Delta Q_t, 5)) - \sigma(\Delta F_t) - \delta W_t$$

   do the following:
9. Let $\mathcal{P}_e$ be $\mathcal{P}$ extended with step t using $\delta RL(Q_{t-4}, 30)$, ΔQ_{t-3}, ΔF_t and δW_t;
10. If $\Pr[\mathcal{P}_e] > 0$ then do:
11. Not let $\mathcal{P}_r = \text{Reduce}(\mathcal{P}_e)$ and $s_r = s \cdot \Pr[\mathcal{P}_e] / \Pr[\mathcal{P}_r]$;
12. If there exists a tuple $(\mathcal{P}_r, \widetilde{s}) \in \mathcal{Z}_{t,t_e}$ then replace $(\mathcal{P}_r, \widetilde{s})$ in $\mathcal{Z}_{t,t_e}$ by the tuple $(\mathcal{P}_r, s_r + \widetilde{s})$,
13. otherwise insert $(\mathcal{P}_r, s_r)$ in $\mathcal{Z}_{t,t_e}$.
14. Return $\mathcal{Z}_{t_b,t_e}$.

We can define related sets $\mathcal{Q}_{\text{rnd1},u,t}$ for $u \in \{0, \ldots, 32\}$ and $t \in \{1, \ldots, 80\}$ where the weights are bounded by u plus the NAF weight:

$$\mathcal{Q}_{\text{rnd1},u,t} = \{Y \mid Y \in \mathcal{Q}_{\text{rnd1},t} \wedge w(Y) \leq w(\text{NAF}(\sigma(Y))) + u\}.$$

By repeating the analysis at the beginning of this section using sets $\mathcal{Q}_{\text{rnd1},u,t}$ for $\mathcal{Q}_t$, we can determine the success probabilities for each $\Lambda \in \widehat{\mathcal{J}}$. In this way we also

find the highest success probability among those Λ as well as the message expansion differences that enable this highest success probability.

7.5.11 Disturbance vector analysis

Analyzing a disturbance vector $(DV_t)_{t=0}^{79}$ and associated message expansion XOR difference vector $(DW_t)_{t=0}^{79}$ for SHA-1 (or SHA-0) can now be done using the methods described in the previous sections. In this section we define several cost functions and compare them on local collision (in)dependence, (dis)allowed carries and on both full and truncated disturbance vectors.

First we define the sets $\mathcal{D}_{u,[i,j]}$ and $\mathcal{D}_{\mathrm{nc},[i,j]}$ as the set $\mathcal{D}_{[i,j]}$ when using the underlying sets $\mathcal{Q}_{\mathrm{c},u,t}$ and $\mathcal{Q}_{\mathrm{nc},t}$, respectively. We define the following disturbance vector cost functions for $u \in \{0,\ldots,32\}$:

$$\mathrm{FDC}_{u,t_b}\left((DV_t)_{t=0}^{79}\right) = \max_{w,\delta IHV_{\mathrm{diff}},\Lambda} \sum_{\substack{\widehat{\mathcal{P}}\in\mathcal{D}_{u,[t_b,79]} \\ (\delta\widehat{W}_t)_{t=t_b}^{79}=w \\ \phi(\widehat{\mathcal{P}})=\delta IHV_{\mathrm{diff}} \\ \psi(\widehat{\mathcal{P}})=\Lambda}} \Pr[\widehat{\mathcal{P}}]\cdot 2^{w(\Delta\widehat{Q}_{t_b-3})+w(\Delta\widehat{Q}_{t_b-2})};$$

$$\mathrm{FDN}_{t_b}\left((DV_t)_{t=0}^{79}\right) = \max_{w,\delta IHV_{\mathrm{diff}},\Lambda} \sum_{\substack{\widehat{\mathcal{P}}\in\mathcal{D}_{\mathrm{nc},[t_b,79]} \\ (\delta\widehat{W}_t)_{t=t_b}^{79}=w \\ \phi(\widehat{\mathcal{P}})=\delta IHV_{\mathrm{diff}} \\ \psi(\widehat{\mathcal{P}})=\Lambda}} \Pr[\widehat{\mathcal{P}}]\cdot 2^{w(\Delta\widehat{Q}_{t_b-3})+w(\Delta\widehat{Q}_{t_b-2})}.$$

The first disturbance vector cost function FDC_{u,t_b} is defined as the maximal success probability (with a correction) over steps $t_b,\ldots,79$ where carries are allowed (bounded by u plus NAF weight) taken over all combinations of message difference vector w, starting working state Λ and IHV differences $\delta IHV_{\mathrm{diff}}$. The second disturbance vector cost function FDN_{t_b} is defined identically except carries are not allowed. The correction $2^{w(\Delta\widehat{Q}_{t_b-3})+w(\Delta\widehat{Q}_{t_b-2})}$ (which is determined by Λ) is chosen so that FDC and FDN are more closely related to the near-collision search. In the near-collision search we use bitconditions $\mathfrak{q}_i$ to search for valid (Q_i, Q'_i) and only then we proceed to (Q_{i+1}, Q'_{i+1}). This correction models that the differential bitconditions in $\mathfrak{q}_{t_b-3}$ and $\mathfrak{q}_{t_b-2}$ have been pre-fulfilled in this manner.[36]

These two disturbance vector cost functions can be efficiently determined using the methods of the previous sections. To that end, we split the range of steps $t_b,\ldots,79$ into independent intervals. As the disturbance of a local collision at step t is corrected within the next five steps, if no other local collision is started within those five steps then we can split the interval $[t_b, t_e]$ into two independent intervals: $[t_b, t+5]$ and $[t+6, t_e]$. Using this rule we split the range of steps $[t_b, 79]$ into intervals $I_1 =$

36. More ideally, we would have liked to have a correction that also includes the boolean function bitconditions with respect to ΔF_{t_b} in $\mathfrak{q}_{t_b-3}$ and $\mathfrak{q}_{t_b-2}$. However, such a correction would no longer be completely determined by Λ.

$[t_{b,1}, t_{e,1}] = [t_b, t_{e,1}]$, $I_2 = [t_{b,2}, t_{e,2}], \ldots, I_K = [t_{b,K}, 79]$ until no interval can be split further. The disturbance vectors in consideration (see Table 7-2) always result in at least two intervals as is required by our analysis. Intervals $I_2, \ldots, I_{K-1}$ are analyzed as in Section 7.5.5. The first and last interval can be analyzed as in Sections 7.5.9 and 7.5.7, respectively.

These cost functions can also be applied when the disturbance vector consists of a single local collision. For an arbitrary disturbance vector we can treat local collisions as independent (even though they are not) by applying the cost function to each individual local collision in said disturbance vector and taking the product over the resulting probabilities. This allows us to compare our cost function which takes into account the dependence of local collisions with similar cost functions that are based on the assumption that local collisions are independent. More specifically, we define the function Ω that compresses consecutive '1'-bits[37] in DV_i and the function Ψ that splits a disturbance vector into separate disturbance vectors each containing a single disturbance:

$$\Omega\left((DV_i)_{i=0}^{79}\right) = \left(DV_i \wedge \left(\neg RL(DV_i, 1) \vee \left(1 + 2^2 + 2^{27}\right)\right)\right)_{i=0}^{79},$$

$$\Psi\left((DV_i)_{i=0}^{79}\right) = \left\{ \begin{array}{c} (Y_i)_{i=0}^{79} \text{ where} \\ Y_t[b]{=}1, \\ Y_i[j]{=}0,\ i{\neq}t \vee j{\neq}b \end{array} \middle| \begin{array}{c} t{\in}\{0,\ldots,79\} \\ b{\in}\{0,\ldots,31\} \\ \text{such that } DV_t[b]{=}1 \end{array} \right\}.$$

Now we can define the following variants on FDC and FDN that assume independence of local collisions:

$$\begin{aligned} \mathrm{FIC}_{u,t_b}\left((DV_t)_{t=0}^{79}\right) &= \prod_{(Y_i)_{i=0}^{79} \in \Psi(\Omega((DV_t)_{t=0}^{79}))} \mathrm{FDC}_{u,t_b}((Y_i)_{i=0}^{79}); \\ \mathrm{FIN}_{t_b}\left((DV_t)_{t=0}^{79}\right) &= \prod_{(Y_i)_{i=0}^{79} \in \Psi(\Omega((DV_t)_{t=0}^{79}))} \mathrm{FIC}_{t_b}((Y_i)_{i=0}^{79}). \end{aligned}$$

Most cost functions in the literature only consider local collisions starting at step t_b or thereafter, all other local collisions are ignored even if some of their corrections take place at step t_b or later. This can be applied by 'truncating' the disturbance vector:

$$\Gamma_{t_b}\left((DV_i)_{i=0}^{79}\right) = (Y_i)_{i=0}^{79}, \quad Y_i = \begin{cases} DV_i, & i \geq t_b; \\ 0, & i < t_b. \end{cases}$$

Using Γ_{t_b} we can define variants of FDC, FIC, FDN and FIN:

$$\begin{aligned} \mathrm{TDC}_{u,t_b}\left((DV_t)_{t=0}^{79}\right) &= \mathrm{FDC}_{u,t_b}\left(\Gamma_{t_b}\left((DV_t)_{t=0}^{79}\right)\right); \\ \mathrm{TIC}_{u,t_b}\left((DV_t)_{t=0}^{79}\right) &= \mathrm{FIC}_{u,t_b}\left(\Gamma_{t_b}\left((DV_t)_{t=0}^{79}\right)\right); \\ \mathrm{TDN}_{t_b}\left((DV_t)_{t=0}^{79}\right) &= \mathrm{FDN}_{t_b}\left(\Gamma_{t_b}\left((DV_t)_{t=0}^{79}\right)\right); \\ \mathrm{TIN}_{t_b}\left((DV_t)_{t=0}^{79}\right) &= \mathrm{FIN}_{t_b}\left(\Gamma_{t_b}\left((DV_t)_{t=0}^{79}\right)\right). \end{aligned}$$

37. Bits 0, 2 and 27 are always kept, since bit positions pairs (31,0), (1,2) and (26,27) are not considered consecutive. See also Section 7.3.2

Applying Γ_{t_b} to a disturbance vector implies that there are no differences in Λ. This in turn implies that the correction $2^{w(\Delta\widehat{Q}_{t_b-3})+w(\Delta\widehat{Q}_{t_b-2})}$ is always one.

Our cost functions consist of three letters that indicate the following properties:

F Full D.V.: uses all local collisions that influence steps $t \geq t_b$;

T Truncated D.V.: uses only local collisions starting at steps $t \geq t_b$;

D Use dependence of local collisions;

I Assume independence of local collisions;

C Allow additional carries (total weight bounded by u plus NAF weight);

N Disallow carries.

The last cost function we define here is $\mathrm{HW}_{t_b}\left((DV_t)_{t=0}^{79}\right) = \sum_{i=t_b}^{79} w(DV_i)$ which is the most crude cost function as it counts the number of '1'-bits in $DV_{t_b}, \ldots, DV_{79}$.

Below we provide selected results in Table 7-3 to compare cost functions for SHA-1. For the cases in Table 7-3, the results of the cost function $\mathrm{FDC}_{7,20}$ are a factor between $2^{0.3}$ and $2^{12.5}$ higher than $\mathrm{FIC}_{32,20}$. This clearly shows that using the joint probability instead of using the product of individual probabilities leads to higher maximum success probabilities.

To compare FDC with varying starting step t_b for SHA-1, selected results are shown in Table 7-4. A more complete analysis of disturbance vectors for SHA-1 using FDC can be found in Appendix F.

Section 7.6 uses this analysis to obtain target values for $\delta IHV_{\mathrm{diff}}$ and message bitrelations for disturbance vector II(52,0). The maximum success probabilities for intervals $[33, 52]$, $[53, 60]$ and $[67, 79]$ as determined using this analysis have been confirmed by experiments as described in Section 7.6.9.

Table 7-3: *SHA-1 disturbance vector analysis - cost function comparison*

DV	FDC	FIC	FDN	FIN	TDC	TIC	TDN	TIN	HW
I(48, 0)	71.4	80.5	77	83	65.4	74.5	71	77	27
I(49, 0)	72.2	79.6	77	83	67.2	74.6	72	78	27
I(50, 0)	71.9	81.4	75	83	65.9	73.4	69	75	26
I(51, 0)	73.3	85.8	77	88	67.3	74.8	71	77	25
I(48, 2)	73.8	75.7	79	79	69.8	71.7	75	75	27
I(49, 2)†	73.8	74.1	78	78	70.8	71.1	75	75	27
II(50, 0)	73.0	77.4	78	80	68.0	70.4	73	73	27
II(51, 0)	71.9	77.7	77	81	67.6	69.7	73	73	26
II(52, 0)‡	71.8	79.4	75	81	65.4	67.4	69	69	25

The eight columns FDC to TIN show the negative $\log_2$ results from the cost functions $\mathrm{FDC}_{7,20}$, FDN_{20}, $\mathrm{FIC}_{32,20}$, FIN_{20}, $\mathrm{TDC}_{7,20}$, FDN_{20}, $\mathrm{TIC}_{32,20}$, TIN_{20}, respectively. The last column shows the result from the cost function HW_{20}. The disturbance vectors marked by † and ‡ are used in Wang et al.'s collision attack [WYY05b] and our near-collision attack in Section 7.6, respectively.

Table 7-4: *SHA-1 disturbance vector analysis - FDC with varying starting step*

	t_b								
DV	18	19	20	21	22	23	24	25	26
I$(48, 0)$	78.3	75.3	71.4	70.4	67.4	66.4	65.0	63.0	61.0
I$(49, 0)$	80.2	75.2	72.2	69.3	68.3	65.3	64.3	62.9	60.9
I$(50, 0)$	79.9	75.9	71.9	70.9	68.1	67.1	64.1	63.1	61.7
II$(51, 0)$	78.7	74.9	71.9	68.9	67.9	66.5	64.5	58.5	56.5
II$(52, 0)$	78.6	75.6	71.8	69.8	66.8	65.8	64.3	62.3	56.3

The columns are the negative $\log_2$ results from the cost function FDC_{8,t_b}.

7.6 SHA-1 near-collision attack

7.6.1 Overview

In this section we present the construction of a near-collision attack against SHA-1 using our methods from Sections 7.4 and 7.5. The construction consists of the following steps that each are discussed in the following sections:

1. We discuss the selection of the disturbance vector and our choice for II(52,0) in Section 7.6.2.

2. In Section 7.6.3 we perform a precise analysis of disturbance vector II(52,0) to obtain sets of optimal values for δIHV_{diff} and Λ.

3. In Section 7.6.4 we construct valid first round differential paths where we use the found set of optimal values for Λ to derive the first round upper partial differential paths.

4. The bitconditions given by these first round differential paths are extended into round two in Section 7.6.5 for the purposes of the early-stopping and message modification techniques.

5. Section 7.6.6 details the derivation of the smallest set of message bitrelations over all four rounds that lead (almost) to the highest success probability under the restrictions given by the bitconditions over round one and two.

6. In Section 7.6.7 we present our basic collision searching algorithm that efficiently searches for message blocks that fulfill all bitconditions up to Q_{17} and all message bitrelations.

7. Lastly, we significantly speed up our basic collision searching algorithm using tunnels in Section 7.6.8.

Our implementation of this near-collision attack is published as part of project HashClash [HC]. The attack has a complexity equivalent to about $2^{57.5}$ calls to the SHA-1 compression function. This improves upon the near-collision attack by Wang et al. with a complexity of about 2^{68} SHA-1 compressions. So far no near-collision blocks have been found using our near-collision attack that provide explicit proof of the correctness and complexity of our attack. For that reason we discuss verification of the correctness and the complexity of our near-collision attack in Section 7.6.9.

Our near-collision attack can directly be used in a two-block identical-prefix collision attack against SHA-1. It should be noted that such a two-block identical-prefix collision attack actually consists of three blocks where the first block is part of the identical-prefix part and is used to gain 160-bits of freedom and to satisfy the bitconditions $\mathfrak{q}_{-4}, \ldots, \mathfrak{q}_0$ of our near-collision attack. The remaining two blocks are two sequential near-collision blocks where the second block cancels the δIHV_{out} resulting from the first block.

A lower-bound for the complexity of a complete two-block identical-prefix collision attack based on our current near-collision implementation is approximately $(1 + 6) \cdot 2^{57.5} \approx 2^{60.3}$ SHA-1 compressions. This follows from the fact that the first near-collision attack has the luxury of six allowed values for δIHV_{diff} for each possible vector of message differences $(\delta W_t)_{t=0}^{79}$, whereas the second near-collision attack must target one specific δIHV_{diff}.

The second near-collision attack has a lesser amount of freedom to exploit compared to the first near-collision attack, i.e., it cannot use a prior block to gain 160-bits of freedom and it requires more message bitrelations. This may further restrict the number of tunnels that can be used. Hence, the complexity of the identical-prefix collision attack as directly based on our near-collision attack will in all likelihood be larger than $2^{60.3}$ SHA-1 compressions. Nevertheless, we expect it will be only a small factor larger than this lower bound. Taking into account the extra message bitrelations and up to four fewer tunnels, one arrives at the conservative upper bound of $2^{65.3}$ SHA-1 compressions.

There is a wide gap between the lower bound $2^{60.3}$ and the conservative upper bound $2^{65.3}$ wherein the actual complexity of the second near-collision attack will lie. The complexity of a complete two-block identical-prefix collision attack may be more accurately estimated when the first near-collision block(s) has been found and thus the second near-collision attack can be constructed.

7.6.2 Disturbance vector selection

The selection of which disturbance vector to use is the most important choice in constructing a near-collision attack. This choice directly determines the message bit differences up to sign DW_t, the set of optimal δIHV_{diff} to target and the set of optimal Λ. It also determines most of the message bitrelations (some of which are only determined up to parity) and thereby also limits which tunnels may be used as is discussed in Section 7.6.8.

In Appendix F we present the results of our analysis of many disturbance vectors, see also Section 7.5. Table F-1 shows the results of the seven disturbance vectors that (almost) result in a success probability of 2^{-73} or higher over the last 60 steps: I(48,0), I(49,0), I(50,0), II(46,0), II(50,0), II(51,0) and II(52,0).

We have chosen for disturbance vector II(52,0) as it is the second best under our cost function $\text{FDC}_{u,20}$ *and* it showed the most promising results in a preliminary analysis from Section 7.5.8. We do not claim that disturbance vector II(52,0) is the best, since our choice is only based on the analysis of the last 60 steps and not on the other factors that the choice of disturbance vector influences such as number of message bitrelations and which tunnels can be used. In fact we encourage further analysis of the remaining disturbance vectors and construction of near-collision attacks under these disturbance vectors.

7.6.3 Finding optimal δIHV_{diff} and Λ

Disturbance vector II(52,0) can be analyzed over the last 60 steps using the following independent intervals of steps: $I_1 = [20, 32]$, $I_2 = [33, 52]$, $I_3 = [53, 60]$ and $I_4 = [67, 79]$. The steps $t = 61, \ldots, 66$ are trivial differential steps with no differences in the working state or message words with success probability 1.

First we analyze the last interval using the methods presented in Section 7.5. We optimize for the first near-collision attack in a two-block collision attack. Below we provide a speedup only for the first near-collision attack, thus the second near-collision attack complexity is the most important term in the two-block collision attack complexity. For that reason we desire that the contribution of steps $67, \ldots, 79$ to the second near-collision attack complexity is as low as possible. This can be achieved by considering only δIHV_{diff} for which there exist some message difference vector w such that the success probability $p_{(w,\delta IHV_{\text{diff}},[67,79])}$ is (almost) the highest success probability called $p_{[67,79]}$.[38]

For each possible message difference vector $w \in (\mathcal{W})_{t=67}^{79}$ we count the number N_w of δIHV_{diff} for which the success probability $p_{(w,\delta IHV_{\text{diff}},[67,79])}$ is (almost) $p_{[67,79]}$. To speed up the first near-collision attack by a factor of $N_{\max} = \max_{w \in (\mathcal{W})_{t=67}^{79}} N_w$, we use only those message difference vectors $w \in (\mathcal{W})_{t=67}^{79}$ for which $N_w = N_{\max}$. The speed up by the factor of $N_{\max}$ follows from the fact that the first near-collision attack always has $N_{\max}$ chances of finding a target δIHV_{diff}.

More precisely, we do the following:

1. We apply the analysis in Section 7.5.8 over $I_4 = [67, 79]$ to determine a set $\widehat{\mathcal{I}}$ of differences δIHV_{diff} with high weights. The set $\widehat{\mathcal{I}}$ defines the sets $\mathcal{Q}_{\text{ihv},u,t}$.

2. Using the sets $\mathcal{Q}_{\text{ihv},u,t}$ for the analysis in Section 7.5.7 we determine success probabilities $p_{(w,\delta IHV_{\text{diff}},[67,79])}$ for message difference vectors $w \in (\mathcal{W}_t)_{t=67}^{79}$ and IHV differences $\delta IHV_{\text{diff}} \in \widehat{\mathcal{I}}$.

3. Let $p_{[67,79]} = \max\{p_{(w,\delta IHV_{\text{diff}},[67,79])} \mid w \in (\mathcal{W}_t)_{t=67}^{79}, \delta IHV_{\text{diff}} \in \widehat{\mathcal{I}}\}$ be the maximum success probability of all $p_{(w,\delta IHV_{\text{diff}},[67,79])}$.

4. For each $w \in (\mathcal{W}_t)_{t=67}^{79}$, let N_w be the number of target δIHV_{diff} with high success probability (where $\alpha = 0.90$):

$$N_w = \left| \left\{ \delta IHV_{\text{diff}} \in \widehat{\mathcal{I}} \;\middle|\; p_{(w,\delta IHV_{\text{diff}},[67,79])} \geq p_{[67,79]} \cdot \alpha \right\} \right|.$$

5. Let $N_{\max} = \max_{w \in (\mathcal{W}_t)_{t=67}^{79}} N_w$ and

$$\mathfrak{W}_{[67,79]} = \left\{ w \in (\mathcal{W}_t)_{t=67}^{79} \;\middle|\; N_w = N_{\max} \right\}$$

be the set of all w for which $N_w = N_{\max}$.

38. See Section 7.5.7, p. 154.

6. Now we determine the set of target δIHV_{diff} values:

$$\widetilde{\mathcal{I}} = \left\{ \delta IHV_{\text{diff}} \in \widehat{\mathcal{I}} \;\middle|\; \exists w \in \mathfrak{W}_{[67,79]} : p_{(w,\delta IHV_{\text{diff}},[67,79]} \geq p_{[67,79]} \cdot \alpha \right\}.$$

The value α influences the obtained average success probability over steps $67, \ldots, 79$ and the obtained $N_{\max}$. A smaller value for α can increase $N_{\max}$ (thus providing a speedup for the first near-collision attack) at the cost of a lower average success probability (thus increasing the complexity of both the first and second near-collision attacks).

For our near-collision attack we have found the highest success probability over $I_4 = [67, 79]$ to be $p_{[67,79]} \approx 2^{-19.2}$. This proves the value of Section 7.5.8 as $2^{-20.4}$ approximates the highest success probability over $I_4 = [67, 79]$ allowing only working state differences as prescribed by the disturbances in the disturbance vector. Thus Section 7.5.8 provided a speed up of a factor of about $2^{1.2}$. We have found a set of message difference vectors such that $N_{\max} = 6$ and $\widetilde{\mathcal{I}}$ presented in Table 7-5 consists of 192 values for δIHV_{diff}. This implies that the success probability over I_4 of our first near-collision attack is $6 \cdot p_{[67,79]} \approx 2^{-16.6}$.

We analyze the first interval to determine the set of optimal values for Λ with (almost) the highest success probability. After constructing differential paths using this set, we are limited to one specific Λ-value and only then we can determine

$$\begin{aligned}
\mathcal{I}_0 = \big\{ & (2^{11} + 2^4 - 2^2, 2^6, 2^{31}, 2^1, 2^{31}), \\
& (2^{12} + 2^3 + 2^1, 2^7, 0, 2^1, 2^{31}), \\
& (2^{12} + 2^4 - 2^1, 2^7, 0, 2^1, 2^{31}), \\
& (2^{11} + 2^9 + 2^4 - 2^2, 2^6 + 2^4, 2^{31}, 2^1, 2^{31}), \\
& (2^{12} + 2^9 + 2^3 + 2^1, 2^7 + 2^4, 0, 2^1, 2^{31}), \\
& (2^{12} + 2^9 + 2^4 - 2^1, 2^7 + 2^4, 0, 2^1, 2^{31}) \big\}; \\
\mathcal{I}_1 = \mathcal{I}_0 \cup \big\{ & (2^{12} + 2^{11} + 2^4 - 2^2, 2^7 + 2^6, 2^{31}, 2^1, 2^{31}), \\
& (2^{12} + 2^{11} + 2^9 + 2^4 - 2^2, 2^7 + 2^6 + 2^4, 2^{31}, 2^1, 2^{31}) \big\}; \\
\mathcal{I}_2 = \big\{ & (v_1 - c \cdot 2^5, v_2, v_3, v_4, v_5) \;\big|\; (v_i)_{i=1}^5 \in \mathcal{I}_1,\ c \in \{0, 1\} \big\}; \\
\mathcal{I}_3 = \big\{ & (v_1 + c \cdot 2^3, v_2, v_3, v_4, v_5) \;\big|\; (v_i)_{i=1}^5 \in \mathcal{I}_2,\ c \in \{0, 1\} \big\}; \\
\mathcal{I}_4 = \big\{ & (v_1 - c \cdot 2^{13}, v_2 - c \cdot 2^8, v_3, v_4, v_5) \;\big|\; (v_i)_{i=1}^5 \in \mathcal{I}_3,\ c \in \{0, 1\} \big\}; \\
\mathcal{I}_5 = \big\{ & (v_1 - c \cdot 2^9, v_2 - c \cdot 2^4, v_3, v_4, v_5) \;\big|\; (v_i)_{i=1}^5 \in \mathcal{I}_4,\ c \in \{0, 1\} \big\}; \\
\widetilde{\mathcal{I}} = \big\{ & (v_1, v_2, v_3, v_4 - c \cdot 2^2, v_5) \;\big|\; (v_i)_{i=1}^5 \in \mathcal{I}_5,\ c \in \{0, 1\} \big\};
\end{aligned}$$

The resulting set $\widetilde{\mathcal{I}}$ is the set of 192 target δIHV_{diff} values. Note that some of the target δIHV_{diff} values can be constructed in several manners in the above sets, otherwise the cardinality of $\widetilde{\mathcal{I}}$ would be $(6 + 2) \cdot 2^5 = 256$. Furthermore, for any $\delta IHV_{\text{diff}} \in \widetilde{\mathcal{I}}$ also $-\delta IHV_{\text{diff}} \in \widetilde{\mathcal{I}}$.

Table 7-5: *SHA-1 near-collision attack target δIHV_{diff} values*

the corresponding optimal set of message difference vectors. Although it would be possible to use the analysis in Section 7.5.10 to possibly obtain even higher success probabilities, it diverts from the differences prescribed by local collisions which may lead to a higher number of bitconditions in the first round and thus a lower amount of freedom. Cf. our differential path in Table 7-6 (p. 169) uses local collisions over steps $10, \ldots, 19$ and has very few bitconditions over these steps, even though it has been constructed algorithmically as in Section 7.4.3. We leave it to future research to determine whether the analysis in Section 7.5.10 can provide an improvement. For now, we directly use the analysis in Section 7.5.9 and do the following:

1. We apply the analysis in Section 7.5.9 over $I_1 = [20, 32]$ to determine the set $\mathcal{J}$ of possible Λ together with the success probabilities $p_{(w,\Lambda,[20,32])}$ for $w \in (\mathcal{W}_t)_{t=20}^{32}$ and $\Lambda \in \mathcal{J}$.

2. Let $p_{[20,32]} = \max\{p_{(w,\Lambda,[20,32])} \mid w \in (\mathcal{W}_t)_{t=20}^{32}, \Lambda \in \mathcal{J}\}$ be the maximum success probability of all $p_{(w,\Lambda,[20,32])}$.

3. Let $\widetilde{\mathcal{J}} = \{\Lambda \mid \exists w \in (\mathcal{W}_t)_{t=20}^{32} : p_{(w,\Lambda,[20,32])} = p_{[20,32]}\}$ be the set of optimal values for Λ.

7.6.4 Constructing differential paths

We use our method from Section 7.4 to construct valid differential paths over the first round. The five connecting steps as in Section 7.4.4 are 3, 4, 5, 6 and 7. The forward differential path construction in Section 7.4.2 does not have a sufficient amount of freedom when lower connecting steps are chosen. For higher connecting steps, the many bitconditions around the connecting steps easily conflict with message modification techniques and the fulfillment of the message bitrelations over the last 64 steps.

In order to construct a valid first round differential path for our near-collision attack, we need sets of forward and backward partial differential paths. The forward partial differential paths are forward extensions up to step $t = 2$ of the trivial differential path defined by $\delta IHV_{\text{in}} = 0$. The backward partial differential paths are created using the set of optimal values for Λ and are then extended backwards. For each value $\Lambda = (\Delta Q_i)_{i=16}^{20}$ we directly obtain a partial differential path consisting of bitconditions $\mathfrak{q}_{16}, \ldots, \mathfrak{q}_{20}$ derived from $\Delta Q_{16}, \ldots, \Delta Q_{20}$ as in Table 6-1. These partial differential paths are extended backwards down to step $t = 8$.

Finally, we used Algorithm 7-1 and these sets of forward and backward partial differential paths to search for valid first round differential paths. The full differential path that we selected for our near-collision attack is shown in Table 7-6.

7.6.5 Second round bitconditions

Now that we have a full differential path we are bound to a specific value for Λ. Moreover, the bitconditions of this differential path may also affect steps in round two. For the purpose of early stopping techniques and tunnels, we also desire sufficient

Table 7-6: *SHA-1 near-collision differential path - round 1*

t	Bitconditions: $\mathtt{q}_t[31]\ldots\mathtt{q}_t[0]$	ΔW_t
$-4,-3,-2$	`........ ........ ........ ........`	
-1	`...1.... ........ ........ ....0...`	
0	`.^.0.1.. .....0.1 ...00.10 .1..1..1`	$\{1, 26, 27\}$
1	`.0.+^-^^ ^^^^^1^0 ^^^11^10 .0..1.+0`	$\{\overline{4}, \overline{30}, 31\}$
2	`1-...+-- -------- -------- --.-1.+0`	$\{\overline{2}, 3, \overline{4}, \overline{26}, 28, 29, 31\}$
3	`.-.-.0.1 11111111 11110++1 +-1-00-0`	$\{\overline{2}, 26, 27, 28, \overline{29}\}$
4	`.-...1.0 11111111 1111-+++ ++0.1.+1`	$\{1, \overline{3}, 4, 26, \overline{27}, \overline{28}, \overline{29}, 31\}$
5	`.-...0.. ........ ......0. .+.+10+0`	$\{\overline{4}, \overline{29}\}$
6	`.-.+.... ........ ......01 100-.0+.`	$\{2, 3, 4, 26, \overline{29}\}$
7	`-1...1.. ........ ........ ...0.0..`	$\{2, 4, \overline{26}, \overline{27}, 29, 30, 31\}$
8	`1.1-.1.. ........ ........ .....1..`	$\{\overline{1}, 26, \overline{27}\}$
9	`..-..0.. ........ ........ ........`	$\{\overline{4}, 30, 31\}$
10	`^...00.. ........ ........ .......1`	$\{\overline{2}, \overline{3}, 4, \overline{26}, \overline{28}, 29, 31\}$
11	`..-.1... ........ ........ .......0`	$\{2, \overline{26}, 27, \overline{29}\}$
12	`0-..1... ........ ........ ......!.`	$\{\overline{3}, 4, \overline{26}, \overline{27}, \overline{28}, 29, 31\}$
13	`+..01... ........ ........ ........`	$\{\overline{4}, 28, 29, 31\}$
14	`..-1.... ........ ........ ......!.`	$\{\overline{2}, 3\}$
15	`+.0.1... ........ ........ ......!^`	$\{\overline{4}, \overline{27}, \overline{28}, \overline{29}, 31\}$
16	`+-0.0... ........ ........ ......!.`	$\{3, \overline{4}, \overline{27}\}$
17	`+..1.... ........ ........ ......^.`	$\{\overline{4}, \overline{27}, \overline{28}, \overline{29}, 30\}$
18	`-.+0.... ........ ........ ........`	$\{\overline{2}, 4, \overline{27}\}$
19	`-....... ........ ........ ........`	$\{4, 28, 29, \overline{30}\}$
20	`..+..... ........ ........ ........`	

Note that we use the compact notation for the BSDRs ΔW_t (see Section 2.1.3) and the bitconditions from Table B-1.

bitconditions up to the last working state variable Q_i that may be corrected by tunnels. In our case we desire sufficient bitconditions up to Q_{25}.

For this purpose we define extra bitconditions in Table 7-7 for the second round boolean function of SHA-1.

Table 7-7: *Round two bitconditions for SHA-1.*

$\mathtt{q_t[i]}$	**condition on** $(\mathbf{Q_t[i]}, \mathbf{Q'_t[i]})$	**direct/indirect**	**direction**
`r`	$Q_t[i] = Q'_t[i] = RL(Q_{t-1}, 30)[i]$	indirect	backward
`u`	$Q_t[i] = Q'_t[i] = RR(Q_{t+1}, 30)[i]$	indirect	forward
`R`	$Q_t[i] = Q'_t[i] = \overline{RL(Q_{t-1}, 30)[i]}$	indirect	backward
`U`	$Q_t[i] = Q'_t[i] = \overline{RR(Q_{t+1}, 30)[i]}$	indirect	forward
`s`	$Q_t[i] = Q'_t[i] = RL(Q_{t-2}, 30)[i]$	indirect	backward
`c`	$Q_t[i] = Q'_t[i] = RR(Q_{t+2}, 30)[i]$	indirect	forward
`S`	$Q_t[i] = Q'_t[i] = \overline{RL(Q_{t-2}, 30)[i]}$	indirect	backward
`C`	$Q_t[i] = Q'_t[i] = \overline{RR(Q_{t+2}, 30)[i]}$	indirect	forward

To obtain the desired sufficient bitconditions for our near-collision attack we first

implemented our near-collision attack without second round bitconditions and tunnels that can correct Q_{19} and up. We used this preliminary attack to find and store many message block pairs that follow our first round differential path and the disturbance vector up to step 32, that is, all message block pairs that result in $\delta Q_{29} = \delta Q_{30} = \delta Q_{31} = \delta Q_{32} = \delta Q_{33} = 0$. For each message block pair we can derive the differential path it follows up to step 32 and thereby bitconditions up to Q_{33}. We are interested in the set of bitconditions up to Q_{25} that occurred most frequently over all found message block pairs found. Our resulting set of bitconditions $\mathfrak{q}_{19}, \ldots, \mathfrak{q}_{25}$ is shown in Table 7-8.

Table 7-8: *SHA-1 near-collision differential path - round two bitconditions*

t	Bitconditions: $\mathfrak{q}_t[31] \ldots \mathfrak{q}_t[0]$	ΔW_t
19	`-...s... ........ ........ ........`	
20	`..+.r... ........ ........ ........`	$\{2, 3, \overline{4}, \overline{27}, 28, \overline{29}, 31\}$
21	`^.r.s... ........ ........ ........`	$\{\overline{27}, 29, 30, 31\}$
22	`..+.r... ........ ........ ........`	$\{\overline{2}, 28, 29, 31\}$
23	`-...s... ........ ........ .......!`	$\{4, 27, 28, \overline{30}\}$
24	`.-+.R... ........ ........ ........`	$\{\overline{2}, 3, 28, \overline{29}, 31\}$
25	`..rSS... ........ ........ ........`	

Note that we use the compact notation for the BSDRs ΔW_t (see Section 2.1.3) and the bitconditions from Table B-1.

7.6.6 Finding optimal message bitrelations

Finding the optimal message bitrelations is split over five intervals: $I_0 = [0, 19]$, I_1, I_2, I_3 and I_4. For the first round we define $\mathfrak{W}_{[0,19]} = \{(\sigma(\Delta\widehat{W}_i))_{i=0}^{19}\}$ where as before $\Delta\widehat{W}_i$ is the message difference in step i from the differential paths in Table 7-6 and Table 7-8. For the last interval we already have found the optimal set $\mathfrak{W}_{[67,79]}$.

For the interval $I_2 = [33, 52]$ we simply apply the analysis of Section 7.5.5 resulting in the sets $\mathcal{A}_{33,52} = \{(\mathcal{P}, \mathcal{S})\}$ where $\mathcal{P}$ is trivial and $\mathcal{S}$ consists of pairs $(w, p_{w,[33,52]})$. Let $p_{[33,52]} = \max_{(w,p)\in\mathcal{S}} p$ be the maximum over all such $p_{w,[33,52]}$. Now we have found the optimal set of message difference vectors over the interval I_2:

$$\mathfrak{W}_{[33,52]} = \{w \mid (w, p) \in \mathcal{S},\ p = p_{[33,52]}\}.$$

The interval $I_3 = \{53, \ldots, 60\}$ is treated analogously to I_2 resulting in the set $\mathfrak{W}_{[53,60]}$.

To determine the optimal set of message difference vectors over the interval $I_1 = [20, 32]$ we apply the analysis of Section 7.5.9 with the following restrictions due to the bitconditions and message differences found in Tables 7-6 and 7-8:

- Let $\mathcal{Q}_{\mathrm{bc},i} = \{\Delta\widehat{Q}_i\}$ for $i \in \{-4, \ldots, 25\}$ where $\Delta\widehat{Q}_i$ follows from the differential bitconditions in $\mathfrak{q}_i$. Use $\mathcal{Q}_{\mathrm{bc},i}$ for the sets $\mathcal{Q}_i$ for $i \in \{-4, \ldots, 25\}$ in Section 7.5.9. For $i \in \{26, 33\}$, use the sets $\mathcal{Q}_{\mathrm{c},u,i}$ for $\mathcal{Q}_i$ for some value of u. In our case $u = 7$ suffices.

- Let $\Delta\widehat{F}_i$ be the boolean function differences that are implied by the bitconditions $\mathfrak{q}_{-4}, \ldots, \mathfrak{q}_{25}$ for $i \in \{0, \ldots, 26\}$. We add the restriction $\Delta F_t = \Delta\widehat{F}_t$ in step 8 of Algorithm 7-3 for $t \in \{20, \ldots, 26\}$.
- For $i \in \{0, \ldots, 24\}$, we restrict $\mathcal{W}_i$ to the single value $\{\sigma(\Delta\widehat{W}_i)\}$ where $\Delta\widehat{W}_i$ is the message difference from the differential paths in Table 7-6 and Table 7-8.

Let $\widehat{\mathcal{A}}_{20,32}$ be the resulting set of Algorithm 7-3 under these three added restrictions. For each $w \in (\mathcal{W}_t)_{t=20}^{32}$ we define its success probability under the restriction of our bitconditions as follows:

$$p_{\mathrm{bc},w,[20,32]} = \sum_{(\mathcal{P},\mathcal{S}) \in \widehat{\mathcal{A}}_{20,32}} \sum_{\substack{(w',p) \in \mathcal{S} \\ w = w'}} p.$$

This naturally leads to the maximum probability $p_{[20,32]} = \max_w p_{\mathrm{bc},w,[20,32]}$ and the optimal set of message difference vectors $\mathfrak{W}_{[20,32]}$ over interval I_1:

$$\mathfrak{W}_{[20,32]} = \left\{ w \in (\mathcal{W}_t)_{t=20}^{32} \;\middle|\; p_{\mathrm{bc},w,[20,32]} = p_{[20,32]} \right\}.$$

Now we translate each of the sets $\mathfrak{W}_{[0,19]}$, $\mathfrak{W}_{[20,32]}$, $\mathfrak{W}_{[33,52]}$, $\mathfrak{W}_{[53,60]}$ and $\mathfrak{W}_{[67,79]}$ to message bitrelations on the message words $(W_t)_{t=0}^{79}$ associated with M of the message block pair (M, M'). These message bitrelations are of the form

$$\sum_{t=0}^{79} \sum_{b=0}^{31} a_{i,t,b} \cdot W_t[b] = c_i \bmod 2$$

where $a_{i,t,b}, c_i \in \{0,1\}$. Together they can also be written as a matrix equation $A \cdot x = c$ over $\mathbb{F}_2$ where $x \in \mathbb{F}_2^{32 \cdot 80}$ represents the bits $W_t[b]$ for $t \in \{0, \ldots, 79\}$ and $b \in \{0, \ldots, 31\}$.

For $(t_b, t_e) \in \{(0,19), (20,32), (33,52), (53,60), (67,79)\}$, we do the following. For each $\widehat{w} = (\delta\widehat{W}_i)_{i=t_b}^{t_e} \in \mathfrak{W}_{[t_b,t_e]}$ we define the set $\mathcal{V}_{\widehat{w}}$ as the set of all $(W_i)_{i=0}^{79} \in \mathbb{Z}_{2^{32}}^{80}$ that are compatible with $\widehat{w}$:

$$(W_i \oplus DW_i) - W_i = \delta\widehat{W}_i, \quad \text{for } i \in \{t_b, \ldots, t_e\}.^{39}$$

Let the set $\mathcal{V} = \bigcup_{w \in \mathfrak{W}_{[t_b,t_e]}} \mathcal{V}_w$ consist of all $(W_t)_{t=0}^{79} \in \mathbb{Z}_{2^{32}}^{80}$ that are compatible with some $w \in \mathfrak{W}_{[t_b,t_e]}$.

Choose natural mappings from $\mathbb{Z}_{2^{32}}^{16}$ to $\mathbb{F}_2^{32 \cdot 80}$ and from $\mathbb{Z}_{2^{32}}^{80}$ to $\mathbb{F}_2^{32 \cdot 80}$. Let $\mathcal{V}'$ be the set consisting of all elements of $\mathcal{V}$ mapped to $\mathbb{F}_2^{32 \cdot 80}$. We search for an affine

39. Note that for $t \in \{0, \ldots, 79\}$ and $b \in \{0, \ldots, 31\}$ if $t \notin \{t_b, \ldots, t_e\}$ or $DW_t[b] = 0$ or $b = 31$ then the bit $W_t[b]$ is a free bit in $\mathcal{V}_{\widehat{w}}$, i.e., for $(W_i)_{i=0}^{79} \in \mathcal{V}$ also $(\widetilde{W}_i)_{i=0}^{79} \in \mathcal{V}$ where $\widetilde{W}_t = W_t \oplus 2^b$ and $\widetilde{W}_i = W_i$ for $i \neq t$. This implies that in practice we only need to consider those bits $W_t[b]$ for which $t \in \{t_b, \ldots, t_e\}$ and $b \neq 31$ and $DW_t[b] = 1$.

subspace $y + \mathcal{U} \subseteq \mathcal{V}'$ which is as large as possible. For our near-collision attack this affine subspace $y + \mathcal{U}$ is simply found by random trials where a random $y \in \mathcal{V}'$ is selected and beginning with $\mathcal{U} = \emptyset$. For randomly selected v and w from $\mathcal{V}'$ for which $v - w \notin \mathcal{U}$, we add $v - w$ to the span of $\mathcal{U}$ if and only if $y + \mathrm{span}(\mathcal{U}, v - w) \subseteq \mathcal{V}'$. This is repeated until we can no longer add elements to $\mathcal{U}$ in this manner. If $|\mathcal{U}| \geq |\mathcal{V}'|/2$ for the resulting y and $\mathcal{U}$ then this affine subspace is optimal. Otherwise, we repeat this construction several times in the hope of finding a larger affine subspace.

Having found an affine subspace $y + \mathcal{U} \subseteq \mathcal{V}'$, we determine the orthogonal complement $\mathcal{U}^\perp$ of the subspace $\mathcal{U}$. Choose any basis of $\mathcal{U}^\perp$ of size k and let the k rows of the matrix $A_{[t_b,t_e]} \in \mathbb{F}_2^{k \times (32 \cdot 80)}$ consist of the k basis vectors of $\mathcal{U}^\perp$. It follows that $x \in \mathcal{U} \Leftrightarrow A_{[t_b,t_e]} \cdot x = 0$ and thus

$$x \in y + \mathcal{U} \quad \Leftrightarrow \quad A_{[t_b,t_e]} \cdot x = A_{[t_b,t_e]} \cdot y.$$

Hence, the matrix equation $A_{[t_b,t_e]} \cdot x = c_{[t_b,t_e]}$ where $c_{[t_b,t_e]} = A_{[t_b,t_e]} \cdot y$ describes the desired sufficient message bitrelations over the interval $[t_b, t_e]$.

We present the message bitrelations of our near-collision attack for the last three rounds in Table 7-9. The message bitrelations for the first round can directly be read from Table 7-6: if $b \neq 31$ then $\Delta W_t[b] = -1$ and $\Delta W_t[b] = +1$ imply the message bitrelations $W_t[b] = 1$ and $W_t[b] = 0$, respectively. As $\Delta W_t[31] = -1$ and $\Delta W_t[31] = +1$ both result in $\delta m_t = 2^{31}$, no message bitrelations on bit position 31 are necessary.

The matrix equations found as above for $\mathfrak{W}_{[0,19]}$, $\mathfrak{W}_{[20,32]}$, $\mathfrak{W}_{[33,52]}$, $\mathfrak{W}_{[53,60]}$ and $\mathfrak{W}_{[67,79]}$ can be combined into a single matrix equation $A_{[0,79]} \cdot x = c_{[0,79]}$ that defines our message search space. It remains to reduce this matrix equation over the $32 \cdot 80$ message words bits to a matrix equation over the 512 message block bits using the message expansion relation. Let the message expansion be described by the matrix ME such that $ME \cdot m = w$ where $w \in \mathbb{F}_2^{80 \cdot 32}$ is the expanded message generated by $m \in \mathbb{F}_2^{16 \cdot 32}$ under the chosen natural mappings from $\mathbb{Z}_{2^{32}}^{16}$ to $\mathbb{F}_2^{32 \cdot 16}$ and from $\mathbb{Z}_{2^{32}}^{80}$ to $\mathbb{F}_2^{32 \cdot 80}$. Then the message bitrelations over the 512 message block bits is described by the matrix equation:

$$(A_{[0,79]} \cdot ME) \cdot x = c_{[0,79]}, \quad x \in \mathbb{F}_2^{32 \cdot 16}.$$

We use Gaussian elimination to obtain message bitrelations such that bitrelations on $m_t[b]$ are expressed in terms of the bits of $m_0, \ldots, m_{t-1}$ and $m_t[i]$ for $i < b$.

It may happen that these message bitrelations conflict and that there are no solutions to the above matrix equation. Since the first round has almost as many message bitrelations as the other three rounds together, one can try to use a different first round differential path with other message differences in Section 7.6.4.

Table 7-9: *SHA-1 near-collision rounds 2-4 message expansion conditions*

$W_{20}[2] = 0$	$W_{20}[3] = 0$	$W_{20}[4] = 1$	
$W_{20}[27] = 1$	$W_{20}[28] = 0$	$W_{20}[29] = 1$	
$W_{21}[27] = 1$	$W_{21}[29] = 0$	$W_{21}[30] = 0$	
$W_{22}[2] = 1$	$W_{22}[28] = 0$	$W_{22}[29] = 0$	
$W_{23}[4] = 0$	$W_{23}[27] = 0$	$W_{23}[28] = 0$	$W_{23}[30] = 1$
$W_{24}[2] = 1$	$W_{24}[3] = 0$	$W_{24}[28] = 0$	$W_{24}[29] = 1$
$W_{25}[27] = 0$	$W_{25}[30] = 1$		
$W_{26}[28] = 1$	$W_{26}[29] = 0$		
$W_{27}[4] + W_{29}[29] = 1$	$W_{27}[27] + W_{27}[28] = 1$		$W_{27}[29] = 0$
$W_{28}[4] + W_{32}[29] = 0$	$W_{28}[27] = 1$		$W_{28}[28] = 0$
$W_{36}[4] + W_{44}[29] = 0$	$W_{38}[4] + W_{44}[29] = 1$	$W_{39}[30] + W_{44}[29] = 0$	
$W_{40}[3] + W_{44}[29] = 1$	$W_{40}[4] + W_{44}[29] = 0$	$W_{41}[29] + W_{41}[30] = 0$	
$W_{42}[28] + W_{44}[29] = 1$	$W_{43}[4] + W_{47}[29] = 0$	$W_{43}[28] + W_{44}[29] = 1$	
$W_{43}[29] + W_{44}[29] = 0$	$W_{44}[28] + W_{44}[29] = 1$	$W_{45}[29] + W_{47}[29] = 0$	
$W_{46}[29] + W_{47}[29] = 0$	$W_{48}[4] + W_{52}[29] = 0$	$W_{50}[29] + W_{52}[29] = 0$	
$W_{51}[29] + W_{52}[29] = 0$			
$W_{54}[4] + W_{60}[29] = 1$	$W_{56}[4] + W_{60}[29] = 0$	$W_{56}[29] + W_{60}[29] = 1$	
$W_{57}[29] + W_{60}[29] = 1$	$W_{59}[29] + W_{60}[29] = 0$		
$W_{67}[0] + W_{72}[30] = 1$	$W_{68}[5] + W_{72}[30] = 0$	$W_{70}[1] + W_{71}[6] = 1$	
$W_{71}[0] + W_{76}[30] = 1$	$W_{72}[5] + W_{76}[30] = 0$	$W_{73}[2] + W_{78}[0] = 1$	
$W_{74}[1] + W_{75}[6] = 1$	$W_{74}[7] + W_{78}[0] = 0$	$W_{75}[1] + W_{76}[6] = 1$	
$W_{76}[0] + W_{76}[1] = 1$	$W_{76}[3] + W_{77}[8] = 1$	$W_{77}[1] + W_{77}[2] = 1$	

7.6.7 Basic collision search

The first step is to find an identical-prefix block such that IHV bitconditions $\mathfrak{q}_{-4}, \ldots, \mathfrak{q}_0$ for our near-collision attack are satisfied. This is done by simply trying random blocks until one is found that satisfies these bitconditions. Since there are 14 such bitconditions, this step has average complexity 2^{14} SHA-1 compressions.

Our near-collision algorithm can roughly be divided into three parts. The first part searches for message blocks that fulfill all bitconditions up to $\mathfrak{q}_{16}$ and all message bitrelations. The second part exploits message modification techniques, in our case tunnels, to find message blocks that fulfill all bitconditions up to $\mathfrak{q}_{25}$. The third part simply applies the message block difference, computes IHV_{out} and IHV'_{out} and checks whether the resulting δIHV_{out} is one of the target δIHV_{diff} values. The first part is discussed below, the second part is discussed in Section 7.6.8 and the third part needs no further explanation.

The first part consists of 16 steps $t = 0, \ldots, 15$. The working state $Q_{-4} = Q'_{-4}, \ldots, Q_0 = Q'_0$ is initialized using the IHV_{in} resulting from the identical-prefix block. Each step $t = 0, \ldots, 15$ does the following given values $Q_{-4}, \ldots, Q_t$ and $m_0, \ldots, m_{t-1}$:

1. Let R be the set of message bitrelations that use multiple bits of m_t and let $B \subseteq \{0, \ldots, 31\}$ be the set of bit positions b such that $m_t[b]$ is used in some message bitrelation in R. If $R = \emptyset$ then continue at step 2. Otherwise, for each of the possible values $(\widehat{m}_t[b])_{b \in B}$ that satisfy all message bitrelations in R we perform steps 2 through 6.

2. The message bitrelations imply target bit values for m_t. Let $m_{\mathrm{mask}',t}$ and $m_{\mathrm{val}',t}$ be words such that:

$$(m_t \oplus m_{\mathrm{val}',t}) \wedge m_{\mathrm{mask}',t} = 0 \quad \Leftrightarrow \quad m_t \text{ satisfies all bitrelations not in } R.$$

   We define $m_{\mathrm{mask},t}$ and $m_{\mathrm{val},t}$ using $m_{\mathrm{mask}',t}$, $m_{\mathrm{val}',t}$, B and $(\widehat{m}_t[b])_{b \in B}$:

$$m_{\mathrm{mask},t}[b] = \begin{cases} m_{\mathrm{mask}',t}[b] & \text{if } b \notin B; \\ 1 & \text{if } b \in B; \end{cases}$$

$$m_{\mathrm{val},t}[b] = \begin{cases} m_{\mathrm{val}',t}[b] & \text{if } b \notin B; \\ \widehat{m}_t[b] & \text{if } b \in B. \end{cases}$$

   It follows that $(m_t \oplus m_{\mathrm{val},t}) \wedge m_{\mathrm{mask},t} = 0$ implies that m_t satisfies all bitrelations.

3. The bitconditions $\mathfrak{q}_{t+1}$ using the given values Q_t and Q_{t-1} imply target bit values for Q_{t+1}. Let $Q_{\mathrm{mask},t+1}$ and $Q_{\mathrm{val},t+1}$ be words such that

$$(Q_{t+1} \oplus Q_{\mathrm{val},t+1}) \wedge Q_{\mathrm{mask},t+1} = 0 \quad \Leftrightarrow \quad Q_{t+1} \text{ satisfies } \mathfrak{q}_{t+1}.$$

4. Let $C = m_{\mathrm{mask},t} \oplus Q_{\mathrm{mask},t+1}$ be the mask whose '1'-bits describe bit positions b where either $Q_{t+1}[b]$ can be used to correct $m_t[b]$ or vice-versa.

5. If $w(Q_{\mathrm{mask},t+1} \wedge \overline{C}) > w(m_{\mathrm{mask},t} \wedge \overline{C})$ then the number of uncorrectable bits in Q_{t+1} that have to be satisfied by trial is larger than the number of such bits in m_t. We therefore iterate over correct values for Q_{t+1} and test for correct values of m_t:

   (a) Let $Q_{\mathrm{fixed},t+1} = (Q_{\mathrm{val},t+1} \wedge Q_{\mathrm{mask},t+1}) \oplus (\overline{m_{\mathrm{val},t}} \wedge C \wedge \overline{Q_{\mathrm{mask},t+1}})$ consist of the target bit values in $Q_{\mathrm{val},t+1}$ and the complemented bit values in $m_{\mathrm{val},t}$ that can be corrected.

   (b) For all Q_{t+1} such that $(Q_{t+1} \oplus Q_{\mathrm{fixed},t+1}) \wedge (C \vee Q_{\mathrm{mask},t+1}) = 0$ do the following:

      i. Compute

$$F_t = f_t(Q_{t-1}, RL(Q_{t-2}, 30), RL(Q_{t-3}, 30)),$$

$$m_t = Q_{t+1} - RL(Q_t, 5) - RL(Q_{t-4}, 30) - F_t - AC_t.$$

      ii. If $(m_t \oplus m_{\mathrm{val},t}) \wedge m_{\mathrm{mask},t} \wedge \overline{C} \neq 0$ then m_t does not satisfy the message bitrelations on some bit b which is uncorrectable ($C[b] = 0$). We continue at (b).

iii. We correct bits in m_t if necessary. Let $Z = (m_t \oplus m_{\mathrm{val},t}) \wedge C$ be the mask of bits that need correction.
iv. Set $\widehat{Q}_{t+1} = Q_{t+1} \oplus Z$ and $\widehat{m}_t = m_t \oplus Z$.
v. Proceed to the next step $t+1$ using the corrected values $\widehat{Q}_{t+1}$ and $\widehat{m}_t$ and continue at (b) afterwards.

6. Otherwise $w(Q_{\mathrm{mask},t+1} \wedge \overline{C}) \leq w(m_{\mathrm{mask},t} \wedge \overline{C})$ and we iterate over correct values for m_t and test for correct values of Q_{t+1}:
   (a) Let $m_{\mathrm{fixed},t} = (m_{\mathrm{val},t} \wedge m_{\mathrm{mask},t}) \oplus (\overline{Q_{\mathrm{val},t+1}} \wedge C \wedge \overline{m_{\mathrm{mask},t}})$ consist of the target bit values in $m_{\mathrm{val},t}$ and the complemented bit values in $Q_{\mathrm{val},t+1}$ that can be corrected.
   (b) For all m_t such that $(m_t \oplus m_{\mathrm{fixed},t}) \wedge (C \vee m_{\mathrm{mask},t}) = 0$ do the following:
      i. Compute
$$F_t = f_t(Q_{t-1}, RL(Q_{t-2}, 30), RL(Q_{t-3}, 30)),$$
$$Q_{t+1} = RL(Q_t, 5) + RL(Q_{t-4}, 30) + F_t + AC_t + m_t.$$
      ii. If $(Q_{t+1} \oplus Q_{\mathrm{val},t+1}) \wedge Q_{\mathrm{mask},t+1} \wedge \overline{C} \neq 0$ then Q_{t+1} does not satisfy the bitconditions $\mathfrak{q}_{t+1}$ on some bit b which is uncorrectable ($C[b] = 0$). We continue at (b).
      iii. We correct bits in Q_{t+1} if necessary. Let
$$Z = (Q_{t+1} \oplus Q_{\mathrm{val},t+1}) \wedge C$$
      be the mask of bits that need correction.
      iv. Set $\widehat{Q}_{t+1} = Q_{t+1} \oplus Z$ and $\widehat{m}_t = m_t \oplus Z$.
      v. Proceed to the next step $t+1$ using the corrected values $\widehat{Q}_{t+1}$ and $\widehat{m}_t$ and continue at (b) afterwards.

For now the last step $t = 16$ sets $m'_i = m_i \oplus DW_i$ for $i \in \{0, \ldots, 15\}$ and tests whether $\delta IHV_{\mathrm{out}}$ given by

$$\delta IHV_{\mathrm{out}} = \mathrm{SHA1Compress}(IHV_{\mathrm{in}}, (m'_i)_{i=0}^{15}) - \mathrm{SHA1Compress}(IHV_{\mathrm{in}}, (m_i)_{i=0}^{15})$$

matches one of the target $\delta IHV_{\mathrm{diff}}$ values.

The correctness of the corrections made using C and Z above is shown here for the case handled in step 4. The case handled in step 5 works analogously, where the roles of m_t and Q_{t+1} are interchanged. For some $b \in \{0, \ldots, 31\}$, let $Q_{t+1}[b]$ be an otherwise free bit and let $m_t[b]$ be restricted: $Q_{\mathrm{mask},t+1}[b] = 0$ and $m_{\mathrm{mask},t+1}[b] = 1$. Using C in step 4(b), the value of $Q_{t+1}[b]$ is fixed to the value $Q_{\mathrm{fixed},t+1}[b] = \overline{m_{\mathrm{val},t}}[b]$.

Suppose $m_t[b] = 0$ does not satisfy $m_{\mathrm{val},t}[b] = 1$. As $Q_{t+1}[b] = \overline{m_{\mathrm{val},t}}[b] = 0$, this can easily be corrected without affecting other bits by adding 2^b to m_t and Q_{t+1}. This correction maintains the equation $m_t = Q_{t+1} - RL(Q_t, 5) - RL(Q_{t-4}, 30) - F_t - AC_t$ implied by the SHA-1 step function. Suppose $m_t[b] = 1$ does not satisfy $m_{\mathrm{val},t}[b] = 0$. Then similarly, since $Q_{t+1}[b] = 1$, we can correct this without affecting other bits by subtracting 2^b from m_t and Q_{t+1}. Note that both cases only flip bit b and no other bits, exactly what is done using Z in 4(b)iii and 4(b)iv.

Table 7-10: *SHA-1 near-collision tunnels*

r	Tunnels
17	$(Q_1[7], Q_{15}[12], Q_{16}[17])$
18	$(Q_1[7], Q_{13}[10]), (Q_{14}[7]), (Q_{14}[8]), (Q_{14}[9]),$ $(Q_{15}[9]), (Q_{15}[10]), (Q_{15}[12])$
19	$(Q_{15}[5]), (Q_{15}[6]), (Q_{15}[7]), (Q_{15}[8])$
21	$(Q_{10}[6])$
22	$(Q_7[7], Q_{15[6]})$
23	$(Q_7[6]), (Q_7[8])$

These are the tunnels that are used in our near-collision attack at step r. Note that each tuple describes a tunnel through the working state bits that are changed. In principle a working state bit $Q_i[j]$ is always changed as $\Delta Q_i[j] = +1$, except if that working state bit has been changed before by a tunnel (e.g., $Q_1[7]$ in step 18). Another exception is $(Q_{12}[6])$ for which $\Delta Q_{12}[6] = -1$ is used so that that tunnel's message conditions could be combined with those of other tunnels. The additional bitconditions and message conditions used for these tunnels are presented in Table 7-11 and Table 7-12, respectively.

7.6.8 Tunnels

Similar to the tunnels for MD5, we use tunnels to speed up our near-collision search by modifying a message block pair (M, M') that fulfills all bitconditions up to some $\mathfrak{q}_k$ and all message bitrelations of Section 7.6.6. This modification from (M, M') to $(\widehat{M}, \widehat{M}')$ is such that $(\widehat{M}, \widehat{M}')$ also fulfills all bitconditions up to $\mathfrak{q}_k$ and all message bitrelations of Section 7.6.6. In this section we use differences not between M and M', but rather between M and $\widehat{M}$. For only this section we denote $\widehat{X} - X$ and $(\widehat{X}[b] - X[b])_{b=0}^{31}$ by δX and ΔX, respectively, where X and $\widehat{X}$ are associated variables in the computation of SHA1Compress of M and $\widehat{M}$, respectively.

A tunnel consists of a controlled change in the first 16 steps which results in a change somewhere in the next 16 steps. The controlled change is in fact a local collision using message differences

$$\delta m_t = 2^b, \quad \delta m_{t+1} = -2^{b+5 \bmod 32},$$

$$\delta m_{t+2} = \delta m_{t+3} = \delta m_{t+4} = 0 \quad \text{and} \quad \delta m_{t+5} = -2^{b+30 \bmod 32}.$$

We allow no carries in the working state difference. The tunnel requires therefore the following bitconditions:

$$Q_{t+1}[b] = Q'_{t+1}[b] = 0, \quad Q_{t-1}[b + 2 \bmod 32] = Q_{t-2}[b + 2 \bmod 32],$$

$$Q_{t+2}[b - 2 \bmod 32] = 0 \quad \text{and} \quad Q_{t+3}[b - 2 \bmod 32] = 1.$$

Evidently, a tunnel must be compatible with our near-collision attack so far. This implies that the bitconditions required for a tunnel must be compatible with the bitconditions of the near-collision attack *with respect to the first message block*. For instance, the bitcondition $Q_t[b] = 0$ is compatible with $\mathfrak{q}_t[b] =$'.', $\mathfrak{q}_t[b] =$'0' and counter-intuitively also with $\mathfrak{q}_t[b] =$'+'.

Furthermore, the message differences may not break the message bitrelations of our near-collision attack. We search for values Δm_t, Δm_{t+1} and Δm_{t+5} such that $\sigma(\Delta m_i) = \delta m_i$ for $i \in \{t, t+1, t+5\}$ and for all message bitrelations

$$\sum_{i=0}^{15} \sum_{j=0}^{31} c_{i,j} \cdot m_i[j] = a \bmod 2$$

we have that these message differences do not break the message bitrelation:

$$\sum_{i \in \{t,t+1,t+5\}} \sum_{j=0}^{31} c_{i,j} \cdot \Delta m_i[j] = 0 \bmod 2.$$

Most of the time there is only a single value for each of these Δm_i that is interesting for practical use[40], which directly implies additional message bitrelations: $\Delta m_i[j] = +1$ implies $m_i[j] = 0$ and $\Delta m_i[j] = -1$ implies $m_i[j] = 1$, except for $\Delta m_i[31] \neq 0$ for which no additional message bitrelations are necessary. If for any of the Δm_i there are multiple interesting values then we do not use message bitrelations. Instead, we test at step i whether adding δm_i to m_i results in one of the interesting Δm_i.

Tunnel message bitrelations are only a precondition when applying the tunnel, after which they do not have to be fulfilled anymore. Thus a tunnel may not break the original message bitrelations from Section 7.6.6 but also may not break any message bitrelations from other tunnels that are used later on in our near-collision search algorithm.

Since the tunnel's message bitdifferences in the first 16 steps have been decided on, we can determine the possible message bitdifferences in the next steps $t = 16, \ldots$. Since Δm_t, Δm_{t+1} and Δm_{t+5} are not always uniquely determined, neither are the possible message bitdifferences in steps $16, \ldots$. An important aspect of a tunnel is the first step $s > 16$ for which the possible message differences δm_s are non-zero. Instead of using a tunnel at step s, i.e., after the verification whether $\mathfrak{q}_s$ is satisfied, we use a tunnel at the highest step $r \geq s$ such that with almost certainty the tunnel does not affect any bit $Q_i[j]$ with $\mathfrak{q}_i[j] \neq$ '.' for $i \in \{16, \ldots r\}$. The choice of r implies that the tunnel with probability not almost 0 affects at least one bit $Q_{r+1}[b]$ for which $\mathfrak{q}_{r+1}[b] \neq$'.'. Depending on the tunnel used there may be an exactly predictable change in Q_{r+1} or the change may be not so predictable. If there is an exactly predictable change in Q_{r+1} then as a speed up we can do a quick test whether the new Q_{r+1} satisfies $\mathfrak{q}_{r+1}$ before actually applying the tunnel and fully recomputing steps $s, \ldots, r$.

For some tunnels we use the following modifications of the description above:

40. In this case we only find a Δm_i 'interesting' if $w(\Delta m_i) = 1$ or if there is an 'interesting' $\Delta \widehat{m}_i$ such that $w(\Delta m_i) = w(\Delta \widehat{m}_i) + 1$. This choice allows a very efficient check for a given m_i whether adding δm_i to m_i results in one of the 'interesting' Δm_i: either using message bitrelations or a simple check of the form $(m_i \wedge X) \neq Y$. Naturally if this does not lead to any 'interesting' Δm_i, we use only that allowed Δm_i with the lowest weight.

Table 7-11: *SHA-1 near-collision tunnel bitconditions*

t	Bitconditions: $\mathfrak{q}_t[31] \ldots \mathfrak{q}_t[0]$
−1	...1.... ....1. ....0...
0	.^.0.1.. .....0.1 ...00.10 .1..1..1
1	.0.+^-^^ ^^^^^1^0 ^^^11^10 00..1.+0
2	1-...+-- -------- -------- --0-1.+0
3	.-.-.0.1 11111111 11110++1 +-1-00-0
4	.-...1.0 11111111 1111-+++ ++0.1.+1
5	.-...0.. ....01 1+.+10+0
6	.-.+.... ....^01 100-.0+.
7	-1...1.. ....0 0000.0..
8	1.1-.1.. .... .0000 1..
9	..-..0.. ....^^ .1111...
10	^...00.. .... 000....1
11	..-.1... ....^ ..00...0
12	0-..1... ....^.... .1110.!.
13	+..01... ....0.0 ...0..0.
14	..-1.... ....0 ...1..1.
15	+.0.1... .... .0....0^

The additional bitconditions used for the tunnels are underlined.

- Restricting the set of Δm_i for $i \in \{t, t+1, t+5\}$ to the ones with lowest weight if this leads to a higher value of r.

- Restricting the set of Δm_i for $i \in \{t, t+1, t+5\}$ to the ones with lowest weight may also have as result that the first two message differences after step 15 are of the form: $\Delta m_s[b] = \pm 1$ and $\Delta m_{s+1}[b + 5 \bmod 32] = \pm 1$ and $\Delta m_s[i] = \Delta m_{s+1}[j] = 0$ for all other bits $i \neq b$ and $j \neq b + 5 \bmod 32$. In that case, adding the message bitrelation $m_s[b] + m_{s+1}[b + 5 \bmod 32] = 1 \bmod 2$ may also lead to a higher value of r as then $m_{s+1}[b + 5 \bmod 32]$ forms a correction for the disturbance caused by $m_s[b]$.

- For $t \geq 13$, not all of the three bitconditions

$$Q_{t-1}[b + 2 \bmod 32] = Q_{t-2}[b + 2 \bmod 32],$$

$$Q_{t+2}[b - 2 \bmod 32] = 0 \quad \text{and} \quad Q_{t+3}[b - 2 \bmod 32] = 1$$

  are strictly necessary for the first 16 steps. We remove not strictly necessary bitconditions if the removal does not lead to a lower value of r.

- Even though two separate tunnels cannot be used when both break some of the message bitrelations, if they break exactly the same set of message bitrelations

Table 7-12: *SHA-1 near-collision tunnel message conditions*

$W_0 \wedge (2^9 - 2^7) \neq (2^9 - 2^7)$
$W_1[12] = 1$
$W_5 \wedge (2^{10} - 2^5) \neq 0$
$W_6[10] = 0$
$W_7[10] = W_7[11] = W_7[12] = W_7[13] = 1$
$W_9[6] = W_9[7] = 0$
$W_{10}[11] = W_{10}[12] = 1$
$W_{11}[4] = W_{11}[5] = W_{11}[6] = W_{11}[7]$
$W_{12} \wedge (2^{10} - 2^8) \neq 0, \quad W_{12}[10] = W_{12}[11] = 0$
$W_{13}[15] = 1$
$W_{14}[4] = W_{14}[5] = 1, \quad W_{14}[6] = 0$
$W_{15}[11] = 1$

Note that the expression $(2^x - 2^y)$ with $x > y$ denotes a word for which the only '1'-bits occur at bit positions $y, \ldots, x - 1$.

then they can be used simultaneously. Similarly, a set of $n > 2$ separate tunnels may be used simultaneously if each of the message bitrelations is broken by an even number of tunnels from this set.

- For $t < 14$, if a tunnel breaks only message bitrelations over m_{14} and m_{15} then that tunnel can be used to speed up steps 14 and 15 instead of higher steps. Such a tunnel can be used in our near-collision search algorithm at step 14 (or 15 if it does not break message bitrelations over m_{14}) before $m_{\text{val}',14}$ (or $m_{\text{val}',15}$) is determined. Since such tunnels were not considered at the time of implementing our near-collision attack, these tunnels may lead to an improvement of our near-collision attack.

The tunnels that we use in our near-collision are described in Table 7-10. The additional bitconditions and message conditions they required are presented in Table 7-11 and Table 7-12, respectively. For more details on how these tunnels are used in the implementation of our near-collision attack we refer to [HC]. We like to note that even with the added conditions from the tunnels, there are at least 40 bits of freedom left in the 512-bit message space (taking into account the IHV_{in} conditions, but ignoring the degrees of freedom from the identical IHV_{in}) so that many solutions should exist for any given IHV_{in}.

7.6.9 Verification of correctness and runtime complexity

Although so far we were unable to find actual near-collision blocks using our near-collision attack, we show how to verify the correctness of our implementation and its runtime complexity. The implementation of our near-collision attack can be retrieved by checking out a copy of the hashclash source repository at Google Code [HC] using

the Subversion client `svn`[41]:

```
svn checkout http://hashclash.googlecode.com/svn/trunk/ .
```

The C++ source code of our near-collision attack can be found in the following subdirectory:

```
src/diffpathcollfind_sha1/collfind.cpp .
```

After all tunnels have been exploited, we the function `step_16_33()` to compute all remaining steps up to step 32 for both messages M and M' and to verify whether $Q'_i = Q_i$ for $i \in \{29, \ldots, 33\}$. Although uncommented for performance reasons, this function can at this point call the function `check40()` for a secondary check that $Q'_i = Q_i$ for $i \in \{29, \ldots, 33\}$ and thus that our implementation works correctly up to this point. If $Q'_i = Q_i$ for $i \in \{29, \ldots, 33\}$ then `step_16_33()` increases a counter that allows us to determine the average complexity C_0 of searching for blocks M and M' that follow our disturbance vector up to step 32. Our implementation prints this counter divided by the number of seconds as `timeavg 40`. We have determined in this manner that C_0 is equivalent to about $2^{11.97}$ SHA-1 compressions on an Intel Core2 Duo Mobile T9400 operating on Windows 7. The runtime complexity is thus equal to C_0/p, where p is the probability that a message block pair (M, M') leads to one of the target δIHV_{out} values, assuming that $Q'_i = Q_i$ for $i \in \{29, \ldots, 33\}$.

To check whether the current message block pair (M, M') leads to one of the target δIHV_{out} values, the function `check_nc()` is called. There are four independent intervals $I_1 = [0, 32]$, $I_2 = [33, 52]$, $I_3 = [53, 60]$ and $I_4 = [67, 79]$. The first interval is always successful whenever `check_nc()` is called. The other remaining intervals $[t_b, t_e]$ require that $Q'_i = Q_i$ for $i \in \{t_b - 4, \ldots, t_b\}$, i.e., that the previous intervals were successful. For the second interval this condition is thus guaranteed when `check_nc()` is called. We determine the probability p as the product of the success probabilities over the last three intervals.

For each of the last three intervals we can verify their success probabilities experimentally as follows. For interval $[t_b, t_e]$ let $(W_t)_{t=0}^{79}$ and $(W'_t)_{t=0}^{79}$ be the expanded messages from M and M', respectively. Set $Q_{-4}, \ldots, Q_0$ to the IHV_{in} resulting from the identical-prefix block using Equation 7.4 (see p. 119). Compute steps $t = 0, \ldots, t_e$ for the message block M:

$$\begin{aligned} F_t &= f_t(Q_{t-1}, RL(Q_{t-2}, 30), RL(Q_{t-3}, 30)), \\ Q_{t+1} &= F_t + AC_t + W_t + RL(Q_t, 5) + RL(Q_{t-4}, 30). \end{aligned}$$

To more quickly determine the success probability, we assume that the previous intervals were successful, i.e., we set Q'_i to Q_i for $i \in \{t_b - 4, \ldots, t_b\}$. Then we compute steps $t = t_b, \ldots, t_e$ for the message block M':

$$\begin{aligned} F'_t &= f_t(Q'_{t-1}, RL(Q'_{t-2}, 30), RL(Q'_{t-3}, 30)), \\ Q'_{t+1} &= F'_t + AC_t + W'_t + RL(Q'_t, 5) + RL(Q'_{t-4}, 30). \end{aligned}$$

41. Subversion is a version control system: `http://subversion.apache.org/`

For I_2 and I_3 the success probability of these intervals can thus be computed as the probability that $Q'_i = Q_i$ for $i \in \{t_e-3, \ldots, t_e+1\}$. This probability is experimentally approximated as y/x by counting the number of times x that `check_nc()` is called and the number of times y that after the computations above $Q'_i = Q_i$ for $i \in \{t_e - 3, \ldots, t_e + 1\}$. For I_4 we do the same except instead of using $Q'_i = Q_i$ for $i \in \{t_e-3, \ldots, t_e+1\}$ as the success condition, we use the condition whether $(RL(Q'_i, r_i) - RL(Q_i, r_i))_{i=76}^{80}$ is one of the target δIHV_{diff} values where $r_i = 30$ for $i \leq 78$ and $r_i = 0$ otherwise (see also Equation 7.6, p. 119).

The experimentally approximated success probabilities of the intervals I_2, I_3 and I_4 are printed by our near-collision attack as the numbers `avg 53 stats`, `avg 61 stats` and `avg 80 stats`, respectively. The success probabilities are in this manner estimated as $\Pr[I_2] = 2^{-20.91}$, $\Pr[I_3] = 2^{-8.00}$ and $\Pr[I_4] = 2^{-16.65}$ which accurately match the theoretical maximum success probabilities[42] as determined by the differential cryptanalysis of Section 7.5. Since these success probabilities are non-zero, our implementation also works correctly over steps $t > 32$. The runtime complexity of our near-collision attack is hereby estimated in the number of SHA-1 compressions as

$$\frac{C_0}{\Pr[I_2] \cdot \Pr[I_3] \cdot \Pr[I_4]} \approx 2^{11.97} \cdot 2^{20.91} \cdot 2^{8.00} \cdot 2^{16.65} = 2^{57.53}.$$

We have found an example message pair shown in Table 7-13 that satisfies our differential path up to I_4 (thus up to step 66), such message pairs can be found with an average complexity of about $2^{11.97} \cdot 2^{20.91} \cdot 2^{8.00} = 2^{40.9}$ SHA-1 compressions.

42. Taking into account the speed-up factor $N_{\max} = 6$ for I_4.

Table 7-13: *Example message pair each consisting of an identical-prefix block and a near-collision block satisfying our differential path up to step 66.*

First message
bc 7e 39 3a 04 70 f6 84 e0 a4 84 de a5 56 87 5a
cd df f9 c8 2d 02 01 6b 86 0e e7 f9 11 e1 84 18
71 bf bf f1 06 70 95 c9 ed 44 af ee 78 12 24 09
a3 b2 eb 2e 16 c0 cf c2 06 c5 20 28 10 38 3c 2b
73 e6 e2 c8 43 7f b1 3e 4e 4d 5d b6 e3 83 e0 1d
7b ea 24 2c 2b b6 30 54 68 45 b1 43 0c 21 94 ab
fb 52 36 be 2b c9 1e 19 1d 11 bf 8f 66 5e f9 ab
9f 8f e3 6a 40 2c bf 39 d7 7c 1f b4 3c b0 08 72
Second message
bc 7e 39 3a 04 70 f6 84 e0 a4 84 de a5 56 87 5a
cd df f9 c8 2d 02 01 6b 86 0e e7 f9 11 e1 84 18
71 bf bf f1 06 70 95 c9 ed 44 af ee 78 12 24 09
a3 b2 eb 2e 16 c0 cf c2 06 c5 20 28 10 38 3c 2b
7f e6 e2 ca 83 7f b1 2e fa 4d 5d aa df 83 e0 19
c7 ea 24 36 0b b6 30 44 4c 45 b1 5f e0 21 94 bf
f7 52 36 bc eb c9 1e 09 a9 11 bf 93 4a 5e f9 af
23 8f e3 72 f0 2c bf 29 d7 7c 1f b8 84 b0 08 62

7.7 Chosen-prefix collision attack

Theorem 3.6 (p. 36) allows us to construct a chosen-prefix collision attack against SHA-1 using the near-collision attack presented in Section 7.6. Given chosen prefixes P and P', we append padding bit strings S_r and S'_r such that the bit lengths of $P||S_\mathrm{r}$ and $P'||S'_\mathrm{r}$ are both equal to $N \cdot 512 - K$, where $N, K \in \mathbb{N}^+$ and K is a later to be defined constant value. Let IHV_{N-1} and IHV'_{N-1} be the intermediate hash values after processing the first $(N-1) \cdot 512$ bits of $P||S_\mathrm{r}$ and $P'||S'_\mathrm{r}$, respectively. Furthermore, let B and B' be the last $512 - K$ bits of $P||S_\mathrm{r}$ and $P'||S'_\mathrm{r}$, respectively.

7.7.1 Birthday search

Section 6.5.2 and [vOW99] explain how to perform a birthday search. We need to choose the birthday search space V and the birthday step function $f : V \to V$. Based on the 192 target δIHV_diff-values given in Table 7-5 (p. 167), we have chosen V and f as follows:

$$V = \mathbb{Z}_{2^{13}} \times \mathbb{Z}_{2^{18}} \times \mathbb{Z}_{2^{31}} \times \mathbb{Z}_{2^{25}} \times \mathbb{Z}_{2^{32}};$$

$$f(v) = \begin{cases} \phi\big(\mathrm{SHA1Compress}(IHV_{N-1}, B||v)\big) & \text{if } \tau(v) = 0; \\ \phi\big(\mathrm{SHA1Compress}(IHV'_{N-1}, B'||v) - (0,0,0,0,2^{31})\big) & \text{if } \tau(v) = 1, \end{cases}$$

where $\phi : \mathbb{Z}_{2^{32}}^5 \to V$ and $\tau : V \to \{0,1\}$ are defined as

$$\phi(a,b,c,d,e) = \big((a[i])_{i=19}^{31}, (b[i])_{i=14}^{31}, (c[i])_{i=0}^{30}, (d[i])_{i=7}^{31}, e\big);$$
$$\tau(a,b,c,d,e) = w(a) \bmod 2.$$

These choices were made with the following considerations:

- The 192 target δIHV_diff-values are all of the form $(a, b, \mu \cdot 2^{31}, \nu \cdot 2^1, 2^{31})$, where $\mu \in \{0,1\}$, $\nu \in \{-1,1\}$, and $a, b \in \mathbb{Z}_{2^{32}}$.
- For all δIHV_diff-values (a,b,c,d,e), we have that $a \in \{-2^{13}, \ldots, 2^{13}\}$. This implies that with low probability adding a to a randomly chosen $x \in \mathbb{Z}_{2^{32}}$ affects bit position 19 and higher.
- For all δIHV_diff-values (a,b,c,d,e), we have that $b \in \{-2^8, \ldots, 2^8\}$. This implies that with low probability adding b to a randomly chosen $x \in \mathbb{Z}_{2^{32}}$ affects bit position 14 and higher.
- For all δIHV_diff-values (a,b,c,d,e), we have that $d \in \{-2^1, 2^1\}$. This implies that with low probability adding d to a randomly chosen $x \in \mathbb{Z}_{2^{32}}$ affects bit position 7 and higher.

For a birthday search collision $f(v) = f(w)$ with $\tau(v) \neq \tau(w)$, let $(x,y) = (v,w)$ if $\tau(v) = 1$ and $(x,y) = (w,v)$ otherwise. Then

$$IHV'_N = (a',b',c',d',e') = \mathrm{SHA1Compress}(IHV'_{N-1}, B'||x),$$
$$IHV_N = (a,b,c,d,e) = \mathrm{SHA1Compress}(IHV_{N-1}, B||y).$$

The resulting $\delta IHV_N = (\delta a, \delta b, \delta c, \delta d, \delta e) = IHV'_N - IHV_N$ is of the form

- $\delta a \in \left\{ l - m \;\middle|\; l, m \in \mathbb{Z}_{2^{32}}, (l[i])_{i=19}^{31} = (m[i])_{i=19}^{31} \right\}$;
- $\delta b \in \left\{ l - m \;\middle|\; l, m \in \mathbb{Z}_{2^{32}}, (l[i])_{i=14}^{31} = (m[i])_{i=14}^{31} \right\}$;
- $\delta c \in \left\{ l - m \;\middle|\; l, m \in \mathbb{Z}_{2^{32}}, (l[i])_{i=0}^{30} = (m[i])_{i=0}^{30} \right\} = \{0, 2^{31}\}$;
- $\delta d \in \left\{ l - m \;\middle|\; l, m \in \mathbb{Z}_{2^{32}}, (l[i])_{i=7}^{31} = (m[i])_{i=7}^{31} \right\}$;
- $\delta e = 2^{31}$, since $e' - 2^{31} = e$ by definition of f and $f(x) = f(y)$.

For each of the 192 target δIHV_{diff} we can determine the probability $p_{\delta IHV_{\text{diff}}}$ that $\delta IHV_N = \delta IHV_{\text{diff}}$. The sum of these 192 probabilities $p_{\delta IHV_{\text{diff}}}$ is approximately $2^{-33.46}$.

Therefore, a birthday search collision pair v, w with $f(v) = f(w)$ has a probability of $q = 2^{-33.46-1}$ that $\tau(v) \neq \tau(w)$ and δIHV_N is one of the 192 target δIHV_{diff}-values. This implies that the expected birthday search complexity in SHA-1 compressions (ignoring the cost of computing collision points) is

$$\sqrt{\frac{\pi \cdot |V|}{2 \cdot q}} \approx 2^{77.06}.$$

Storing a single trail (beginpoint, endpoint, length) costs about 36 bytes. When using $2.5 \cdot 36/q \approx 2^{40.95}$ bytes (about 2TB) then the expected complexity of generating trails equals the expected complexity of computing the collision points. The expected complexity of computing collision points can be made significantly lower by using more memory. Hence, the overall expected birthday search complexity is approximately $2^{77.1}$ SHA-1 compressions.

7.7.2 Near-collision block

Assume we have found bit strings S_{b} and S'_{b} using the above birthday search such that δIHV_N is one of the 192 δIHV_{diff}-values, where IHV_N and IHV'_N are the intermediate hash values after processing the first $512 \cdot N$ bits of $P||S_{\text{r}}||S_{\text{b}}$ and $P'||S'_{\text{r}}||S'_{\text{b}}$, respectively. The remaining step is to execute a near-collision attack identical to the second near-collision attack in a two-block identical-prefix collision attack as described in Section 7.6.1. To construct this near-collision attack we follow the steps as described in Section 7.6 with the following modifications:

- In Section 7.6.3 for I_4, we set $\widehat{\mathcal{I}}$ to $-\delta IHV_N$ in step 1. This leads to $N_{\max} = 1$ and a smaller set of optimal message difference vectors $\mathfrak{W}_{[67,79]}$.
- Instead of using the trivial differential path defined by $\delta IHV_{\text{in}} = 0$ in Section 7.6.4, we use the differential path $\mathfrak{q}_{-4}, \ldots, \mathfrak{q}_0$ consisting of bitconditions '0', '1', '-' and '+' that match the values $Q_{-4}, \ldots, Q_0$ and $Q'_{-4}, \ldots, Q'_0$ as initialized by definition from IHV_N and IHV'_N.

Executing the constructed near-collision attack results in message blocks S_c and S'_c such that

$$\text{SHA-1}\big(P||S_r||S_b||S_c\big) = \text{SHA-1}\big(P'||S'_r||S'_b||S'_c\big).$$

7.7.3 Complexity

As mentioned in Section 7.6.1, an upper bound for the second near-collision attack in a two-block identical-prefix collision attack is about $2^{65.3}$ SHA-1 compressions. This same upper bound also holds for the above near-collision attack. The near-collision attack complexity is thus a factor of $2^{11.8}$ smaller than the expected birthday search cost of $2^{77.1}$ SHA-1 compressions. Hence, the overall cost of a chosen-prefix collision attack against SHA-1 is dominated by the expected $2^{77.1}$ SHA-1 compressions required to generate the birthday search trails. This complexity is currently infeasible and this chosen-prefix collision attack against SHA-1 remains a theoretical attack.

8 Detecting collision attacks

Contents

8.1 Extra line of defense

It has been known since 2004 and 2005 that MD5 and SHA-1 have weaknesses. In Chapters 6 and 7 we have improved attacks against MD5 and SHA-1 and the future will tell how far these attacks can be further improved. So it is clear that MD5 and SHA-1 should be replaced by more secure hash functions such as SHA-2 or the upcoming SHA-3 standard for all security purposes that depend on the collision resistance of MD5 and SHA-1. Completely banishing all such occurrences of MD5 and SHA-1 in software and hardware might be desirable in light of this, but might very well be impossible due to the high costs and effort required. Even replacing MD5 and SHA-1 in the security-wise most important software and hardware may be difficult due to for instance compatibility issues. Therefore, the most likely scenario is the one that we currently see with respect to MD5, namely that more secure hash functions such as SHA-2 are used where possible however that MD5 and SHA-1 remain supported.

Except when MD5 and SHA-1 have been explicitly disallowed, this scenario still leaves users vulnerable to collision attacks. The technique presented in this section allows to add an extra line of defense in security applications by detecting collision attacks. Obviously when given two messages that result in the same hash value, it is almost certain that a collision attack is involved. The first MD5 collision attack [WY05] could easily be detected by applying its specific message differences to a message and checking whether a collision has occurred. Since other message differences that lead to fast near-collision attacks were found and our differential path construction algorithm was published, there is an infinite number of ways to construct a collision attack and thereby making it impossible to check for all possible collision attacks in this manner. Moreover, our chosen-prefix collision attack is completely undetectable in this manner since the message differences that alter the first chosen-prefix into the second chosen-prefix cannot be guessed in general.

Our technique allows to efficiently detect collisions attacks for MD5 and SHA-1, both identical-prefix collision attacks and chosen-prefix collision attacks, given only one of the two colliding messages. It has been tested successfully in a simple proof of concept plug-in for Microsoft's Internet Explorer. For this test this plug-in detects only chosen-prefix collisions in the certificate-chain of secure websites when connecting to a secure web server before any sensitive information has been sent. It has been

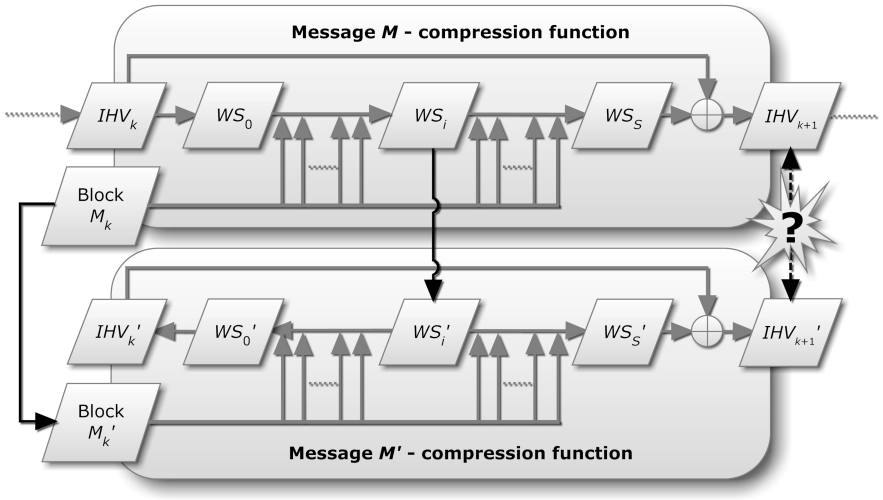

Figure 9: *Principle of detecting near-collisions*

successfully used to detect man-in-the-middle attacks using our rogue Certification Authority described in Section 4.2.

8.2 Core principle

The core principle of our technique of detecting collision attacks is to detect the last near-collision block of a collision attack and uses two key observations:

- There are only a small number of possible message block differences that may lead to feasible near-collision attacks.
- All published MD5 and SHA-1 collision attacks use a differential path that at some step has no differences at all in the working state, or in the case of MD5 it also can use differences $(2^{31}, 2^{31}, 2^{31}, 2^{31})$ (see [dBB93]).[43]

Due to these observations it is possible to check for collision attacks given only one message of a colliding pair of messages.

For two colliding messages M and M', let (M_k, M'_k) be the last near-collision block pair of a collision attack. Let IHV_{k+1} and IHV'_{k+1} be the intermediate hash values just after applying the compression function to M_k and M'_k in the hash value computation of M and M', respectively. It follows that

$$IHV_{k+1} = IHV'_{k+1}.$$

43. The reason for this is simple: these working state differences can be maintained at every step of the 64 steps of MD5Compress with probability at least 1/2 if not 1.

Suppose we are only given the message M, the message block differences δM_k and a step i after which the working state difference in the near-collision attack should be either zero or, in the case of MD5, $(2^{31}, 2^{31}, 2^{31}, 2^{31})$. Then as illustrated in Figure 9, we can easily compute IHV_{k+1} and IHV'_{k+1} and test for the telltale condition $IHV_{k+1} = IHV'_{k+1}$, even though both the second message M' and the message block M'_k were not given.

The hash value computation of M gives us values for IHV_k, IHV_{k+1} and WS_i which is the working state before step i of the compression function applied to IHV_k and M_k. Since we know the message block differences and the working state differences, we can determine the message block M'_k and the working state WS'_i associated with the message M' that collides with M. Computing steps $i+1, \ldots, S$ of the compression function using M'_k and WS'_i, we obtain working states $WS'_{i+1}, \ldots, WS'_S$. As the step function of MD5 and SHA-1 is reversible (see Section 5.4.1) we can also compute working states $WS'_{i-1}, \ldots, WS'_0$. The value of IHV'_k can be derived from WS'_0 and the value of IHV'_{k+1} can be computed from IHV'_k and WS'_S. Finally, if $IHV'_{k+1} = IHV_{k+1}$ then we have confirmed that (M_k, M'_k) is a near-collision block pair.

This principle leads to our collision detection algorithm presented in Algorithm 8-1 that works for any Merkle-Damgård hash function that uses a compression function in $\mathcal{F}_{\text{md4cf}}$ (see Section 5.3, p. 65). Let C be the number of possible combinations of values for message block differences δB, step i and working state differences δWS_i belonging to a feasible near-collision attack. Then the runtime-complexity of Algorithm 8-1 for a message M is approximately $C+1$ times the runtime-complexity of computing the hash value of M.

The probability of a false positive is at least $C \cdot 2^{-L}$ where L is the bit length of the hash value, since $IHV'_{k+1} = IHV_{k+1}$ holds with probability 2^{-L} for randomly selected IHV'_{k+1} and IHV_{k+1}. The false positive probability may be higher when there exists a differential path compatible with one of the combinations of δB, i and δWS_i that holds with probability less than 2^{-L}.

8.3 Application to MD5

Algorithm 8-1 can be directly applied to MD5. What remains is to determine possible combinations of values for message block differences δB, step i and working state differences δWS_i that belong to a feasible near-collision attack. The message block differences are additive in $\mathbb{Z}_{2^{32}}$ and for each message block M_k the message block M'_k can be either $M_k + \delta B$ or $M_k - \delta B$. There are two different working state differences δWS_i that can be used for MD5, namely $(0,0,0,0)$ and $(2^{31}, 2^{31}, 2^{31}, 2^{31})$, written more compactly as $\underline{0}$ and $\underline{2^{31}}$.

Used non-zero message block differences in published near-collision attacks are (together with selected values for i and δWS_i):

- $\delta B = \pm(\delta m_{11} = 2^{15}, \delta m_4 = \delta m_{14} = 2^{31})$ [WY05]: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_2 = 2^8, \delta m_{11} = 2^{15}, \delta m_4 = \delta m_{14} = 2^{31})$ [SSA+09b]: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;

Algorithm 8-1 Last near-collision block detection

This algorithm returns `True` if a near-collision attack is detected and `False` otherwise. For a given message M, let $M_0, \ldots, M_{N-1}$ be the N message blocks that result from the padding and the splitting of M by the hash function. For $k \in \{0, \ldots, N-1\}$ do the following:

1. Let IHV_k be the intermediate hash value before the message block M_k is processed.
2. Initialize the working state WS_0 with IHV_k, compute all S working states $WS_1, \ldots, WS_S$ and determine IHV_{k+1} using IHV_k and WS_S.
3. For each possible combination of values for message block differences δB, step i and working state differences δWS_i belonging to a feasible near-collision attack do the following:
   (a) Apply the message block differences δB to M_k to obtain M'_k.
   (b) Apply the working state differences δWS_i to WS_i to obtain WS'_i.
   (c) Compute the working states $WS'_{i-1}, \ldots, WS'_0$ backwards.
   (d) Compute the working states $WS'_{i+1}, \ldots, WS'_S$ forward.
   (e) Determine IHV'_k from WS'_0 and IHV'_{k+1} from IHV'_k and WS'_S.
   (f) If $IHV'_{k+1} = IHV_{k+1}$ then (M_k, M'_k) is a near-collision block pair: return `True`
4. Return `False`

- $\delta B = \pm(\delta m_{11} = 2^b)$ for $b \in \{0, \ldots, 30\}$ [SLdW07c]: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_{11} = 2^{31})$ [SLdW07c]: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_{10} = 2^{31})$ [XF10]: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_8 = 2^{31})$ [XLF08]: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_6 = 2^8, \delta m_9 = \delta m_{15} = 2^{31})$ [XFL08]: $i = 37$, $\delta WS_{37} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_9 = 2^{27}, \delta m_2 = \delta m_{12} = 2^{31})$ [VJBT08]: $i = 37$, $\delta WS_{37} \in \{\underline{0}, \underline{2^{31}}\}$.

Other possible non-zero message block differences taken from [XF09] and [XLF08] are:

- $\delta B = \pm(\delta m_4 = 2^{20}, \delta m_7 = \delta m_{13} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_2 = 2^8)$: $i = 37$, $\delta WS_{37} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_{11} = 2^{21})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_{11} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_5 = 2^{31}, \delta m_8 = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;

- $\delta B = \pm(\delta m_2 = 2^8, \delta m_{14} = 2^{31})$: $i = 37$, $\delta WS_{37} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_4 = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_5 = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_{14} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_4 = 2^{25})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_8 = 2^{25})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_{11} = 2^{21})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_{14} = 2^{16})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_4 = 2^{20})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_6 = 2^8)$: $i = 50$, $\delta WS_{50} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_9 = 2^{27})$: $i = 50$, $\delta WS_{50} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_5 = 2^{10}, \delta m_9 = 2^{27})$: $i = 37$, $\delta WS_{37} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = (\delta m_5 = 2^{31}, \delta m_{11} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_8 = 2^{31}, \delta m_{11} = 2^{21})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$;
- $\delta B = \pm(\delta m_8 = 2^{25}, \delta m_{13} = 2^{31})$: $i = 44$, $\delta WS_{44} \in \{\underline{0}, \underline{2^{31}}\}$.

The number of all combinations given in the two lists above is $(36 \cdot 4 + 2 \cdot 2) + (16 \cdot 4 + 5 \cdot 2) = 222$. We do not guarantee that the lists of combinations given above form the exhaustive list of all combinations that lead to feasible near-collision attacks. Nevertheless, other combinations arising from future collision attacks can easily be added to the above lists.

All published near-collision attacks require complex differential steps in the first round, thereby requiring a high number of bitconditions, say at least 200. E.g., the differential paths by Wang et al. require roughly 300 bitconditions (see Table 2-4 and Table 2-6 on pages 28 and 30). This implies that the probability of a false positive is dominated by the general $C \cdot 2^{-L}$ term explained earlier. Hence, the probability of a false positive is estimated as $222 \cdot 2^{-128}$ and thus negligible.

Due to the *pseudo-collision* attack against MD5's compression function by den Boer and Bosselaers [dBB93], there is also a special near-collision attack with zero message block differences and with $\delta WS_i = \underline{2^{31}}$ for all $i \in \{0, \ldots, 64\}$. One can test for this pseudo-collision attack using $\delta B = 0$, $i = 32$ and $\delta WS_{32} = \underline{2^{31}}$. The probability of a false positive is 2^{-48} which is not negligible. However, since it requires $\delta WS_0 = \underline{2^{31}}$ and thus $IHV_{\text{in}} = \underline{2^{31}}$, this pseudo-collision attack requires at least one other preceding near-collision block to form a collision attack against MD5.

This observation calls for the following modification of Algorithm 8-1 for MD5 to reduce the chance of a false positive to $222 \cdot 2^{-128} \cdot 2^{-48}$ for $\delta B = 0$. Whenever a near-collision block is detected in step 3.(f) for the combination $\delta B = 0$, $i = 32$ and $\delta WS_{32} = \underline{2^{31}}$ and before returning `True`, perform steps 1–4 of Algorithm 8-1 on the previous message block M_{k-1} using all combinations that have $\delta B \neq 0$ and using

the condition $IHV'_k = IHV_k + \underline{2^{31}}$ instead of the condition $IHV'_k = IHV_k$. If this sub-instance of Algorithm 8-1 returns `False` then the main instance continues with the next combination of δB, i and δWS_i. Otherwise, the main instance returns `True`.

Given a message M, the average complexity to detect whether M is constructed by a collision attack against MD5 using one of the given message differences is about $222 + 1 + 1 = 224$ times the complexity of computing the MD5 hash of M.

This technique has been tested successfully in a simple proof of concept plug-in for Microsoft's Internet Explorer. For this test this plug-in detects only chosen-prefix collisions in the certificate-chain of secure websites when connecting to a secure web server before any sensitive information has been sent. It has been successfully used to detect man-in-the-middle attacks using our rogue Certification Authority described in Section 4.2.

8.4 Application to SHA-1

Algorithm 8-1 can be directly applied to SHA-1. Note that this is possible even though no practical collision attacks against SHA-1 or actual colliding messages are known yet. What remains is to determine possible combinations of values for message block differences δB, step i and working state differences δWS_i that belong to a feasible near-collision attack. For SHA-1 the message block differences δB are given as $M_k \oplus M'_k$ which is the bitwise XOR of the message block pair (M_k, M'_k) and can be derived from a disturbance vector as $(DW_t)_{t=0}^{15}$ (see Section 7.3.2). For disturbance vector $\mathrm{I}(j, b)$ or $\mathrm{II}(j, b)$ there are no differences at step $j + 8$, hence to test for near-collision block pair using this disturbance vector we use Algorithm 8-1 with the combination $(DW_t)_{t=0}^{15}$, $i = j + 8$ and $\delta WS_i = (0, 0, 0, 0, 0)$.

We arbitrarily choose all disturbance vectors $(DV_t)_{t=0}^{79}$ in Appendix F where for some $u \in \{0, \ldots, 7\}$ and $\epsilon \leq 0.5$ it holds that $\mathrm{FDC}_{u,20,\epsilon}((DV_t)_{t=0}^{79}) \geq 2^{-74}$:

I(46,0),	I(48,0),	I(49,0),	I(50,0),	I(51,0),	I(48,2),	I(49,2),
II(46,0),	II(50,0),	II(51,0),	II(52,0),	II(53,0),	II(54,0),	II(56,0).

Similar to the case of MD5, it is always possible to add extra disturbance vectors to the above list in the future whenever it is believed it can lead to a feasible collision attack. Ignoring the first round, each disturbance vector has a probability in the order of 2^{-70} that a false positive occurs. Taking into account the complex differential steps necessary in the first round, we can safely assume that the probability of a false positive is negligible.

Given a message M, the average complexity to detect whether M is constructed by one of the above possibly feasible collision attacks against SHA-1 is about 15 times the complexity of computing the SHA-1 hash of M.

A MD5 compression function constants

Table A-1: *MD5 Addition and Rotation Constants and message block expansion.*

t	AC_t	RC_t	W_t
0	$\texttt{d76aa478}_{16}$	7	m_0
1	$\texttt{e8c7b756}_{16}$	12	m_1
2	$\texttt{242070db}_{16}$	17	m_2
3	$\texttt{c1bdceee}_{16}$	22	m_3
4	$\texttt{f57c0faf}_{16}$	7	m_4
5	$\texttt{4787c62a}_{16}$	12	m_5
6	$\texttt{a8304613}_{16}$	17	m_6
7	$\texttt{fd469501}_{16}$	22	m_7
8	$\texttt{698098d8}_{16}$	7	m_8
9	$\texttt{8b44f7af}_{16}$	12	m_9
10	$\texttt{ffff5bb1}_{16}$	17	m_{10}
11	$\texttt{895cd7be}_{16}$	22	m_{11}
12	$\texttt{6b901122}_{16}$	7	m_{12}
13	$\texttt{fd987193}_{16}$	12	m_{13}
14	$\texttt{a679438e}_{16}$	17	m_{14}
15	$\texttt{49b40821}_{16}$	22	m_{15}

t	AC_t	RC_t	W_t
16	$\texttt{f61e2562}_{16}$	5	m_1
17	$\texttt{c040b340}_{16}$	9	m_6
18	$\texttt{265e5a51}_{16}$	14	m_{11}
19	$\texttt{e9b6c7aa}_{16}$	20	m_0
20	$\texttt{d62f105d}_{16}$	5	m_5
21	$\texttt{02441453}_{16}$	9	m_{10}
22	$\texttt{d8a1e681}_{16}$	14	m_{15}
23	$\texttt{e7d3fbc8}_{16}$	20	m_4
24	$\texttt{21e1cde6}_{16}$	5	m_9
25	$\texttt{c33707d6}_{16}$	9	m_{14}
26	$\texttt{f4d50d87}_{16}$	14	m_3
27	$\texttt{455a14ed}_{16}$	20	m_8
28	$\texttt{a9e3e905}_{16}$	5	m_{13}
29	$\texttt{fcefa3f8}_{16}$	9	m_2
30	$\texttt{676f02d9}_{16}$	14	m_7
31	$\texttt{8d2a4c8a}_{16}$	20	m_{12}

t	AC_t	RC_t	W_t
32	$\texttt{fffa3942}_{16}$	4	m_5
33	$\texttt{8771f681}_{16}$	11	m_8
34	$\texttt{6d9d6122}_{16}$	16	m_{11}
35	$\texttt{fde5380c}_{16}$	23	m_{14}
36	$\texttt{a4beea44}_{16}$	4	m_1
37	$\texttt{4bdecfa9}_{16}$	11	m_4
38	$\texttt{f6bb4b60}_{16}$	16	m_7
39	$\texttt{bebfbc70}_{16}$	23	m_{10}
40	$\texttt{289b7ec6}_{16}$	4	m_{13}
41	$\texttt{eaa127fa}_{16}$	11	m_0
42	$\texttt{d4ef3085}_{16}$	16	m_3
43	$\texttt{04881d05}_{16}$	23	m_6
44	$\texttt{d9d4d039}_{16}$	4	m_9
45	$\texttt{e6db99e5}_{16}$	11	m_{12}
46	$\texttt{1fa27cf8}_{16}$	16	m_{15}
47	$\texttt{c4ac5665}_{16}$	23	m_2

t	AC_t	RC_t	W_t
48	$\texttt{f4292244}_{16}$	6	m_0
49	$\texttt{432aff97}_{16}$	10	m_7
50	$\texttt{ab9423a7}_{16}$	15	m_{14}
51	$\texttt{fc93a039}_{16}$	21	m_5
52	$\texttt{655b59c3}_{16}$	6	m_{12}
53	$\texttt{8f0ccc92}_{16}$	10	m_3
54	$\texttt{ffeff47d}_{16}$	15	m_{10}
55	$\texttt{85845dd1}_{16}$	21	m_1
56	$\texttt{6fa87e4f}_{16}$	6	m_8
57	$\texttt{fe2ce6e0}_{16}$	10	m_{15}
58	$\texttt{a3014314}_{16}$	15	m_6
59	$\texttt{4e0811a1}_{16}$	21	m_{13}
60	$\texttt{f7537e82}_{16}$	6	m_4
61	$\texttt{bd3af235}_{16}$	10	m_{11}
62	$\texttt{2ad7d2bb}_{16}$	15	m_2
63	$\texttt{eb86d391}_{16}$	21	m_9

B MD5 and SHA-1 bitconditions

Table B-1: *Bitconditions for MD5 and SHA-1.*

$\mathtt{q_t[i]}$	condition on $(\mathbf{Q_t[i]}, \mathbf{Q'_t[i]})$	direct/indirect	direction
`.`	$Q_t[i] = Q'_t[i]$	direct	
`+`	$Q_t[i] = 0, \quad Q'_t[i] = 1$	direct	
`-`	$Q_t[i] = 1, \quad Q'_t[i] = 0$	direct	
`0`	$Q_t[i] = Q'_t[i] = 0$	direct	
`1`	$Q_t[i] = Q'_t[i] = 1$	direct	
`^`	$Q_t[i] = Q'_t[i] = Q_{t-1}[i]$	indirect	backward
`v`	$Q_t[i] = Q'_t[i] = Q_{t+1}[i]$	indirect	forward
`!`	$Q_t[i] = Q'_t[i] = \overline{Q_{t-1}[i]}$	indirect	backward
`y`	$Q_t[i] = Q'_t[i] = \overline{Q_{t+1}[i]}$	indirect	forward
`m`	$Q_t[i] = Q'_t[i] = Q_{t-2}[i]$	indirect	backward
`w`	$Q_t[i] = Q'_t[i] = Q_{t+2}[i]$	indirect	forward
`#`	$Q_t[i] = Q'_t[i] = \overline{Q_{t-2}[i]}$	indirect	backward
`h`	$Q_t[i] = Q'_t[i] = \overline{Q_{t+2}[i]}$	indirect	forward
`?`	$Q_t[i] = Q'_t[i] \wedge (Q_t[i] = 1 \vee Q_{t-2}[i] = 0)$	indirect	backward
`q`	$Q_t[i] = Q'_t[i] \wedge (Q_{t+2}[i] = 1 \vee Q_t[i] = 0)$	indirect	forward
`r`	$Q_t[i] = Q'_t[i] = RL(Q_{t-1}, 30)[i]$	indirect	backward
`u`	$Q_t[i] = Q'_t[i] = RR(Q_{t+1}, 30)[i]$	indirect	forward
`R`	$Q_t[i] = Q'_t[i] = \overline{RL(Q_{t-1}, 30)[i]}$	indirect	backward
`U`	$Q_t[i] = Q'_t[i] = \overline{RR(Q_{t+1}, 30)[i]}$	indirect	forward
`s`	$Q_t[i] = Q'_t[i] = RL(Q_{t-2}, 30)[i]$	indirect	backward
`c`	$Q_t[i] = Q'_t[i] = RR(Q_{t+2}, 30)[i]$	indirect	forward
`S`	$Q_t[i] = Q'_t[i] = \overline{RL(Q_{t-2}, 30)[i]}$	indirect	backward
`C`	$Q_t[i] = Q'_t[i] = \overline{RR(Q_{t+2}, 30)[i]}$	indirect	forward

Note: these conditions are tailored for the boolean functions of MD5 and the first and second round boolean function of SHA-1.

C MD5 boolean function bitconditions

The four tables in this appendix correspond to the four rounds of MD5, i.e., $0 \leq t < 16$, $16 \leq t < 32$, $32 \leq t < 48$ and $48 \leq t < 64$. The '$\mathfrak{abc}$' in each of the first columns denotes the three differential bitconditions $(\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ for the relevant t and $0 \leq i \leq 31$, with each table containing all 27 possible triples. Columns $2, 3, 4$ contain forward bitconditions $FC(t, \mathfrak{abc}, g)$ for $g = 0, +1, -1$, respectively, and columns $5, 6, 7$ contain backward bitconditions $BC(t, \mathfrak{abc}, g)$ for those same g-values, respectively. The parenthesized number next to a triple $\mathfrak{def}$ is $|U_{\mathfrak{def}}|$, the amount of freedom left. An entry is left empty if $g \notin V_{t,\mathfrak{abc}}$. We refer to Section 6.2.2 for more details.

C.1 Bitconditions applied to boolean function F

Table C-1: *Round 1 ($0 \leq t < 16$) bitconditions applied to boolean function F:*

$$F(X, Y, Z) = (X \wedge Y) \oplus (\overline{X} \wedge Z)$$

D.B.	Forward bitconditions			Backward bitconditions		
abc	$g = 0$	$g = +1$	$g = -1$	$g = 0$	$g = +1$	$g = -1$
... (8)	... (8)			... (8)		
..+ (4)	1.+ (2)	0.+ (2)		1.+ (2)	0.+ (2)	
..- (4)	1.- (2)		0.- (2)	1.- (2)		0.- (2)
.+. (4)	0+. (2)	1+. (2)		0+. (2)	1+. (2)	
.++ (2)		.++ (2)			.++ (2)	
.+- (2)		1+- (1)	0+- (1)		1+- (1)	0+- (1)
.-. (4)	0-. (2)		1-. (2)	0-. (2)		1-. (2)
.-+ (2)		0-+ (1)	1-+ (1)		0-+ (1)	1-+ (1)
.-- (2)			.-- (2)			.-- (2)
+.. (4)	+.v (2)	+10 (1)	+01 (1)	+^. (2)	+10 (1)	+01 (1)
+.+ (2)	+0+ (1)	+1+ (1)		+0+ (1)	+1+ (1)	
+.- (2)	+1- (1)		+0- (1)	+1- (1)		+0- (1)
++. (2)	++1 (1)	++0 (1)		++1 (1)	++0 (1)	
+++ (1)		+++ (1)			+++ (1)	
++- (1)	++- (1)			++- (1)		
+-. (2)	+-0 (1)		+-1 (1)	+-0 (1)		+-1 (1)
+-+ (1)	+-+ (1)			+-+ (1)		
+-- (1)			+-- (1)			+-- (1)
-.. (4)	-.v (2)	-01 (1)	-10 (1)	-^. (2)	-01 (1)	-10 (1)
-.+ (2)	-1+ (1)	-0+ (1)		-1+ (1)	-0+ (1)	
-.- (2)	-0- (1)		-1- (1)	-0- (1)		-1- (1)
-+. (2)	-+0 (1)	-+1 (1)		-+0 (1)	-+1 (1)	
-++ (1)		-++ (1)			-++ (1)	
-+- (1)	-+- (1)			-+- (1)		
--. (2)	--1 (1)		--0 (1)	--1 (1)		--0 (1)
--+ (1)	--+ (1)			--+ (1)		
--- (1)			--- (1)			--- (1)

C.2 Bitconditions applied to boolean function G

Table C-2: *Round 2 ($16 \leq t < 32$) bitconditions applied to boolean function G:*

$$G(X, Y, Z) = (Z \wedge X) \oplus (\overline{Z} \wedge Y)$$

D.B.		Forward bitconditions						Backward bitconditions					
abc		$g = 0$		$g = +1$		$g = -1$		$g = 0$		$g = +1$		$g = -1$	
...	(8)	...	(8)					...	(8)				
..+	(4)	.v+	(2)	10+	(1)	01+	(1)	^.+	(2)	10+	(1)	01+	(1)
..-	(4)	.v-	(2)	01-	(1)	10-	(1)	^.-	(2)	01-	(1)	10-	(1)
.+.	(4)	.+1	(2)	.+0	(2)			.+1	(2)	.+0	(2)		
.++	(2)	0++	(1)	1++	(1)			0++	(1)	1++	(1)		
.+-	(2)	1+-	(1)	0+-	(1)			1+-	(1)	0+-	(1)		
.-.	(4)	.-1	(2)			.-0	(2)	.-1	(2)			.-0	(2)
.-+	(2)	1-+	(1)			0-+	(1)	1-+	(1)			0-+	(1)
.--	(2)	0--	(1)			1--	(1)	0--	(1)			1--	(1)
+..	(4)	+.0	(2)	+.1	(2)			+.0	(2)	+.1	(2)		
+.+	(2)	+1+	(1)	+0+	(1)			+1+	(1)	+0+	(1)		
+.-	(2)	+0-	(1)	+1-	(1)			+0-	(1)	+1-	(1)		
++.	(2)			++.	(2)					++.	(2)		
+++	(1)			+++	(1)					+++	(1)		
++-	(1)			++-	(1)					++-	(1)		
+-.	(2)			+-1	(1)	+-0	(1)			+-1	(1)	+-0	(1)
+-+	(1)	+-+	(1)					+-+	(1)				
+--	(1)	+--	(1)					+--	(1)				
-..	(4)	-.0	(2)			-.1	(2)	-.0	(2)			-.1	(2)
-.+	(2)	-0+	(1)			-1+	(1)	-0+	(1)			-1+	(1)
-.-	(2)	-1-	(1)			-0-	(1)	-1-	(1)			-0-	(1)
-+.	(2)			-+0	(1)	-+1	(1)			-+0	(1)	-+1	(1)
-++	(1)	-++	(1)					-++	(1)				
-+-	(1)	-+-	(1)					-+-	(1)				
--.	(2)					--.	(2)					--.	(2)
--+	(1)					--+	(1)					--+	(1)
---	(1)					---	(1)					---	(1)

C.3 Bitconditions applied to boolean function H

Table C-3: *Round 3 ($32 \leq t < 48$) bitconditions applied to boolean function H:*

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

D.B.	Forward bitconditions			Backward bitconditions		
abc	$g = 0$	$g = +1$	$g = -1$	$g = 0$	$g = +1$	$g = -1$
... (8)	... (8)			... (8)		
..+ (4)		.v+ (2)	.y+ (2)		^.+ (2)	!.+ (2)
..- (4)		.y- (2)	.v- (2)		!.- (2)	^.- (2)
.+. (4)		.+w (2)	.+h (2)		m+. (2)	#+. (2)
.++ (2)	.++ (2)			.++ (2)		
.+- (2)	.+- (2)			.+- (2)		
.-. (4)		.-h (2)	.-w (2)		#-. (2)	m-. (2)
.-+ (2)	.-+ (2)			.-+ (2)		
.-- (2)	.-- (2)			.-- (2)		
+.. (4)		+.v (2)	+.y (2)		+^. (2)	+!. (2)
+.+ (2)	+.+ (2)			+.+ (2)		
+.- (2)	+.- (2)			+.- (2)		
++. (2)	++. (2)			++. (2)		
+++ (1)		+++ (1)			+++ (1)	
++- (1)			++- (1)			++- (1)
+-. (2)	+-. (2)			+-. (2)		
+-+ (1)			+-+ (1)			+-+ (1)
+-- (1)		+-- (1)			+-- (1)	
-.. (4)		-.y (2)	-.v (2)		-!. (2)	-^. (2)
-.+ (2)	-.+ (2)			-.+ (2)		
-.- (2)	-.- (2)			-.- (2)		
-+. (2)	-+. (2)			-+. (2)		
-++ (1)			-++ (1)			-++ (1)
-+- (1)		-+- (1)			-+- (1)	
--. (2)	--. (2)			--. (2)		
--+ (1)		--+ (1)			--+ (1)	
--- (1)			--- (1)			--- (1)

C.4 Bitconditions applied to boolean function I

Table C-4: *Round 4 ($48 \leq t < 64$) bitconditions applied to boolean function I:*

$$I(X, Y, Z) = Y \oplus (X \vee \overline{Z})$$

D.B.		Forward bitconditions						Backward bitconditions					
abc		$g = 0$		$g = +1$		$g = -1$		$g = 0$		$g = +1$		$g = -1$	
...	(8)	...	(8)					...	(8)				
..+	(4)	1.+	(2)	01+	(1)	00+	(1)	1.+	(2)	01+	(1)	00+	(1)
..-	(4)	1.-	(2)	00-	(1)	01-	(1)	1.-	(2)	00-	(1)	01-	(1)
.+.	(4)			0+1	(1)	.+q	(3)			0+1	(1)	?+.	(3)
.++	(2)	0++	(1)			1++	(1)	0++	(1)			1++	(1)
.+-	(2)	0+-	(1)			1+-	(1)	0+-	(1)			1+-	(1)
.-.	(4)			.-q	(3)	0-1	(1)			?-.	(3)	0-1	(1)
.-+	(2)	0-+	(1)	1-+	(1)			0-+	(1)	1-+	(1)		
.--	(2)	0--	(1)	1--	(1)			0--	(1)	1--	(1)		
+..	(4)	+.0	(2)	+01	(1)	+11	(1)	+.0	(2)	+01	(1)	+11	(1)
+.+	(2)	+.+	(2)					+.+	(2)				
+.-	(2)			+0-	(1)	+1-	(1)			+0-	(1)	+1-	(1)
++.	(2)	++1	(1)			++0	(1)	++1	(1)			++0	(1)
+++	(1)					+++	(1)					+++	(1)
++-	(1)	++-	(1)					++-	(1)				
+-.	(2)	+-1	(1)	+-0	(1)			+-1	(1)	+-0	(1)		
+-+	(1)			+-+	(1)					+-+	(1)		
+--	(1)	+--	(1)					+--	(1)				
-..	(4)	-.0	(2)	-11	(1)	-01	(1)	-.0	(2)	-11	(1)	-01	(1)
-.+	(2)			-1+	(1)	-0+	(1)			-1+	(1)	-0+	(1)
-.-	(2)	-.-	(2)					-.-	(2)				
-+.	(2)	-+1	(1)			-+0	(1)	-+1	(1)			-+0	(1)
-++	(1)	-++	(1)					-++	(1)				
-+-	(1)					-+-	(1)					-+-	(1)
--.	(2)	--1	(1)	--0	(1)			--1	(1)	--0	(1)		
--+	(1)	--+	(1)					--+	(1)				
---	(1)			---	(1)					---	(1)		

D MD5 chosen-prefix collision birthday search cost

In this appendix notation and variables are as in Section 6.5.2. The columns p, C_{tr} and M denote the values $-\log_2(p_{r,k,w})$, $\log_2(C_{\text{tr}}(r,k,w))$ and the minimum required memory such that $C_{\text{coll}}(r,k,w,M) \leq C_{\text{tr}}(r,k,w)$, respectively.

$r=3$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0												
4										34.01	51.33	2TB
8							33.42	53.03	748GB	31.31	51.98	174GB
12	34.01	55.33	2TB	32.42	54.53	374GB	30.55	53.6	103GB	28.24	52.44	21GB
16	31.	55.83	141GB	29.65	55.15	55GB	27.36	54.01	12GB	25.6	53.13	4GB
20	27.51	56.08	13GB	26.18	55.42	5GB	24.53	54.59	2GB	23.26	53.96	673MB
24	24.33	56.49	2GB	23.35	56.	714MB	22.17	55.41	315MB	21.19	54.92	160MB
28	21.11	56.88	152MB	20.56	56.6	103MB	19.98	56.32	70MB	19.57	56.11	52MB
32	17.88	57.26	17MB	17.88	57.27	17MB	17.89	57.27	17MB	17.88	57.27	17MB

$r=3$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0				31.68	48.17	225GB	30.25	47.45	84GB	28.01	46.33	18GB
4	32.2	50.43	323GB	29.92	49.29	67GB	28.06	48.36	19GB	26.2	47.43	6GB
8	28.83	50.74	32GB	27.33	49.99	11GB	25.88	49.26	5GB	24.47	48.56	2GB
12	26.63	51.64	7GB	25.14	50.9	3GB	23.96	50.3	2GB	22.94	49.8	537MB
16	24.31	52.48	2GB	23.27	51.96	675MB	22.49	51.57	394MB	21.86	51.26	255MB
20	22.28	53.46	340MB	21.62	53.13	215MB	21.14	52.9	155MB	20.73	52.69	117MB
24	20.53	54.59	102MB	20.01	54.33	71MB	19.65	54.15	55MB	19.38	54.01	46MB
28	19.25	55.95	42MB	19.02	55.83	36MB	18.82	55.74	31MB	18.65	55.65	28MB
32	17.88	57.27	17MB	17.88	57.27	17MB	17.88	57.27	17MB	17.88	57.27	17MB

$r=4$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0							34.	49.33	2TB	30.19	47.42	81GB
4				33.42	51.04	749GB	30.36	49.51	90GB	27.59	48.12	14GB
8	35.	53.83	3TB	30.3	51.48	87GB	27.21	49.93	11GB	24.87	48.76	2GB
12	29.58	53.12	53GB	27.53	52.09	13GB	24.59	50.62	2GB	22.47	49.56	388MB
16	26.26	53.45	6GB	24.36	52.51	2GB	22.06	51.36	292MB	20.38	50.51	91MB
20	23.16	53.91	628MB	21.5	53.08	199MB	19.72	52.19	58MB	18.54	51.6	26MB
24	20.25	54.45	84MB	19.09	53.87	38MB	17.8	53.23	16MB	16.86	52.76	8MB
28	17.26	54.95	11MB	16.63	54.64	7MB	16.02	54.34	5MB	15.6	54.13	4MB
32	14.29	55.47	2MB	14.29	55.47	2MB	14.29	55.47	2MB	14.29	55.47	2MB

$r=4$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	26.98	45.81	9GB	24.45	44.55	2GB	22.14	43.4	310MB	20.33	42.49	88MB
4	24.95	46.8	3GB	22.82	45.73	493MB	21.04	44.84	144MB	19.55	44.1	52MB
8	22.63	47.64	432MB	20.92	46.79	133MB	19.58	46.12	53MB	18.56	45.61	26MB
12	20.67	48.66	112MB	19.41	48.03	47MB	18.45	47.55	24MB	17.71	47.18	15MB
16	19.08	49.86	37MB	18.19	49.42	21MB	17.56	49.1	13MB	17.08	48.86	10MB
20	17.66	51.16	14MB	17.09	50.87	10MB	16.7	50.67	8MB	16.39	50.52	6MB
24	16.25	52.45	6MB	15.82	52.24	4MB	15.54	52.09	4MB	15.33	51.99	3MB
28	15.31	53.98	3MB	15.09	53.87	3MB	14.93	53.79	3MB	14.78	53.72	2MB
32	14.29	55.47	2MB	14.29	55.47	2MB	14.29	55.47	2MB	14.29	55.47	2MB

$r=5$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	35.	49.83	3TB	31.2	47.92	161GB	27.13	45.89	10GB	23.74	44.2	938MB
4	33.42	51.04	749GB	28.47	48.56	25GB	24.63	46.64	2GB	21.58	45.12	210MB
8	28.61	50.63	27GB	25.61	49.13	4GB	22.	47.33	280MB	19.39	46.02	46MB
12	25.43	51.04	3GB	22.74	49.7	468MB	19.66	48.15	56MB	17.53	47.09	13MB
16	22.36	51.51	360MB	20.02	50.34	72MB	17.59	49.12	14MB	15.95	48.3	5MB
20	19.38	52.01	46MB	17.48	51.07	13MB	15.67	50.16	4MB	14.55	49.6	2MB
24	16.68	52.66	7MB	15.35	52.	3MB	14.06	51.36	2MB	13.17	50.91	1MB
28	13.92	53.29	2MB	13.22	52.93	1MB	12.61	52.63	1MB	12.21	52.43	1MB
32	11.2	53.92	1MB	11.2	53.93	1MB	11.2	53.92	1MB	11.2	53.93	1MB

$r=5$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	20.53	42.59	102MB	18.03	41.34	18MB	16.17	40.41	5MB	14.92	39.79	3MB
4	18.94	43.79	34MB	17.	42.82	9MB	15.57	42.11	4MB	14.53	41.59	2MB
8	17.27	44.96	11MB	15.79	44.22	4MB	14.75	43.7	2MB	14.01	43.33	2MB
12	15.92	46.28	5MB	14.84	45.75	2MB	14.09	45.37	2MB	13.56	45.11	1MB
16	14.8	47.73	2MB	14.06	47.35	2MB	13.55	47.1	1MB	13.18	46.92	1MB
20	13.79	49.22	1MB	13.31	48.98	1MB	12.99	48.82	1MB	12.76	48.7	1MB
24	12.64	50.64	1MB	12.29	50.47	1MB	12.07	50.36	1MB	11.91	50.28	1MB
28	11.95	52.3	1MB	11.76	52.2	1MB	11.62	52.14	1MB	11.5	52.07	1MB
32	11.2	53.92	1MB	11.2	53.93	1MB	11.2	53.92	1MB	11.2	53.93	1MB

$r=6$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	31.2	47.92	161GB	26.73	45.69	8GB	21.78	43.22	241MB	18.14	41.4	20MB
4	28.18	48.42	20GB	23.89	46.27	2GB	19.56	44.11	52MB	16.46	42.55	6MB
8	24.66	48.66	2GB	21.17	46.91	158MB	17.37	45.01	12MB	14.79	43.72	2MB
12	21.67	49.16	224MB	18.6	47.62	27MB	15.43	46.04	3MB	13.4	45.03	1MB
16	18.82	49.74	31MB	16.21	48.43	6MB	13.74	47.2	1MB	12.23	46.44	1MB
20	16.03	50.34	5MB	13.97	49.31	2MB	12.2	48.43	1MB	11.18	47.92	1MB
24	13.54	51.1	1MB	12.11	50.38	1MB	10.86	49.75	1MB	10.04	49.35	1MB
28	11.03	51.84	1MB	10.28	51.47	1MB	9.69	51.17	1MB	9.33	50.99	1MB
32	8.56	52.6	1MB	8.56	52.6	1MB	8.56	52.6	1MB	8.56	52.6	1MB

$r=6$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	15.12	39.88	3MB	13.05	38.85	1MB	11.73	38.19	1MB	10.91	37.78	1MB
4	14.05	41.35	2MB	12.44	40.55	1MB	11.39	40.02	1MB	10.7	39.68	1MB
8	12.92	42.79	1MB	11.73	42.19	1MB	10.95	41.8	1MB	10.44	41.54	1MB
12	12.01	44.33	1MB	11.14	43.9	1MB	10.57	43.61	1MB	10.2	43.42	1MB
16	11.25	45.95	1MB	10.64	45.64	1MB	10.24	45.45	1MB	9.98	45.32	1MB
20	10.53	47.59	1MB	10.14	47.39	1MB	9.89	47.27	1MB	9.72	47.19	1MB
24	9.59	49.12	1MB	9.31	48.98	1MB	9.14	48.9	1MB	9.04	48.85	1MB
28	9.09	50.87	1MB	8.93	50.79	1MB	8.82	50.74	1MB	8.73	50.69	1MB
32	8.56	52.6	1MB	8.56	52.6	1MB	8.56	52.6	1MB	8.56	52.6	1MB

$r=7$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	26.82	45.73	8GB	22.2	43.43	323MB	17.02	40.83	9MB	13.4	39.02	1MB
4	24.02	46.34	2GB	19.68	44.16	56MB	15.16	41.9	3MB	12.18	40.41	1MB
8	21.1	46.88	151MB	17.23	44.94	11MB	13.37	43.01	1MB	10.97	41.81	1MB
12	18.32	47.49	22MB	14.96	45.8	3MB	11.82	44.24	1MB	9.98	43.31	1MB
16	15.67	48.16	4MB	12.87	46.76	1MB	10.48	45.56	1MB	9.13	44.89	1MB
20	13.1	48.88	1MB	10.93	47.79	1MB	9.26	46.95	1MB	8.35	46.5	1MB
24	10.82	49.74	1MB	9.32	48.99	1MB	8.15	48.4	1MB	7.43	48.04	1MB
28	8.56	50.6	1MB	7.78	50.22	1MB	7.23	49.94	1MB	6.91	49.78	1MB
32	6.34	51.5	1MB	6.34	51.5	1MB	6.34	51.5	1MB	6.34	51.5	1MB

$r=7$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	10.8	37.73	1MB	9.25	36.95	1MB	8.35	36.5	1MB	7.84	36.25	1MB
4	10.13	39.39	1MB	8.9	38.78	1MB	8.17	38.41	1MB	7.74	38.19	1MB
8	9.42	41.03	1MB	8.5	40.57	1MB	7.94	40.3	1MB	7.61	40.13	1MB
12	8.82	42.74	1MB	8.15	42.4	1MB	7.74	42.19	1MB	7.48	42.07	1MB
16	8.31	44.48	1MB	7.84	44.24	1MB	7.55	44.1	1MB	7.37	44.01	1MB
20	7.82	46.23	1MB	7.51	46.08	1MB	7.32	45.99	1MB	7.21	45.93	1MB
24	7.06	47.86	1MB	6.84	47.75	1MB	6.72	47.69	1MB	6.66	47.65	1MB
28	6.71	49.68	1MB	6.58	49.62	1MB	6.5	49.58	1MB	6.43	49.54	1MB
32	6.34	51.5	1MB	6.34	51.5	1MB	6.34	51.5	1MB	6.34	51.5	1MB

$r=8$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	23.39	44.02	732MB	18.21	41.43	21MB	12.88	38.76	1MB	9.52	37.09	1MB
4	20.57	44.61	105MB	15.94	42.29	5MB	11.39	40.02	1MB	8.69	38.67	1MB
8	17.91	45.28	17MB	13.77	43.21	1MB	9.99	41.32	1MB	7.86	40.26	1MB
12	15.35	46.	3MB	11.78	44.22	1MB	8.79	42.72	1MB	7.17	41.91	1MB
16	12.91	46.78	1MB	10.	45.32	1MB	7.75	44.2	1MB	6.59	43.62	1MB
20	10.56	47.61	1MB	8.35	46.5	1MB	6.81	45.73	1MB	6.03	45.34	1MB
24	8.49	48.57	1MB	6.97	47.81	1MB	5.91	47.28	1MB	5.29	46.97	1MB
28	6.48	49.56	1MB	5.71	49.18	1MB	5.21	48.93	1MB	4.93	48.79	1MB
32	4.54	50.6	1MB	4.54	50.6	1MB	4.54	50.6	1MB	4.54	50.6	1MB

$r=8$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	7.45	36.05	1MB	6.37	35.51	1MB	5.8	35.23	1MB	5.5	35.08	1MB
4	7.06	37.85	1MB	6.18	37.42	1MB	5.71	37.18	1MB	5.45	37.05	1MB
8	6.63	39.64	1MB	5.96	39.31	1MB	5.6	39.12	1MB	5.39	39.02	1MB
12	6.26	41.46	1MB	5.77	41.21	1MB	5.49	41.07	1MB	5.34	40.99	1MB
16	5.94	43.29	1MB	5.59	43.12	1MB	5.39	43.02	1MB	5.28	42.97	1MB
20	5.61	45.13	1MB	5.38	45.01	1MB	5.25	44.95	1MB	5.18	44.92	1MB
24	5.01	46.83	1MB	4.85	46.75	1MB	4.77	46.71	1MB	4.73	46.69	1MB
28	4.78	48.71	1MB	4.68	48.67	1MB	4.62	48.64	1MB	4.58	48.62	1MB
32	4.54	50.6	1MB	4.54	50.6	1MB	4.54	50.6	1MB	4.54	50.6	1MB

$r=9$	$w=0$			$w=1$			$w=2$			$w=3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	20.16	42.4	79MB	14.62	39.64	2MB	9.38	37.02	1MB	6.46	35.56	1MB
4	17.56	43.1	13MB	12.63	40.64	1MB	8.26	38.45	1MB	5.93	37.29	1MB
8	15.09	43.87	3MB	10.75	41.7	1MB	7.2	39.92	1MB	5.41	39.03	1MB
12	12.73	44.69	1MB	9.06	42.86	1MB	6.3	41.47	1MB	4.96	40.81	1MB
16	10.51	45.58	1MB	7.57	44.11	1MB	5.53	43.09	1MB	4.57	42.61	1MB
20	8.39	46.52	1MB	6.2	45.43	1MB	4.83	44.74	1MB	4.2	44.42	1MB
24	6.53	47.59	1MB	5.05	46.85	1MB	4.12	46.39	1MB	3.63	46.14	1MB
28	4.77	48.71	1MB	4.05	48.35	1MB	3.61	48.13	1MB	3.4	48.02	1MB
32	3.14	49.9	1MB	3.14	49.9	1MB	3.14	49.9	1MB	3.14	49.9	1MB

$r=9$	$w=4$			$w=5$			$w=6$			$w=7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	4.95	34.8	1MB	4.25	34.45	1MB	3.92	34.28	1MB	3.76	34.21	1MB
4	4.73	36.69	1MB	4.15	36.4	1MB	3.87	36.26	1MB	3.74	36.2	1MB
8	4.49	38.57	1MB	4.04	38.35	1MB	3.82	38.24	1MB	3.72	38.18	1MB
12	4.28	40.47	1MB	3.94	40.3	1MB	3.78	40.21	1MB	3.69	40.17	1MB
16	4.09	42.37	1MB	3.85	42.25	1MB	3.73	42.19	1MB	3.67	42.16	1MB
20	3.88	44.27	1MB	3.72	44.19	1MB	3.64	44.15	1MB	3.6	44.13	1MB
24	3.42	46.04	1MB	3.32	45.99	1MB	3.27	45.96	1MB	3.25	45.95	1MB
28	3.28	47.97	1MB	3.21	47.93	1MB	3.18	47.92	1MB	3.16	47.9	1MB
32	3.14	49.9	1MB	3.14	49.9	1MB	3.14	49.9	1MB	3.14	49.9	1MB

$r = 10$	$w = 0$			$w = 1$			$w = 2$			$w = 3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	17.28	40.97	11MB	11.47	38.06	1MB	6.54	35.6	1MB	4.18	34.42	1MB
4	14.87	41.76	3MB	9.77	39.21	1MB	5.73	37.19	1MB	3.87	36.26	1MB
8	12.6	42.63	1MB	8.18	40.42	1MB	4.97	38.81	1MB	3.56	38.11	1MB
12	10.45	43.55	1MB	6.79	41.72	1MB	4.34	40.49	1MB	3.3	39.97	1MB
16	8.45	44.55	1MB	5.57	43.11	1MB	3.8	42.22	1MB	3.06	41.86	1MB
20	6.56	45.61	1MB	4.48	44.56	1MB	3.31	43.98	1MB	2.83	43.74	1MB
24	4.92	46.79	1MB	3.53	46.09	1MB	2.78	45.71	1MB	2.42	45.53	1MB
28	3.44	48.04	1MB	2.78	47.72	1MB	2.44	47.54	1MB	2.28	47.47	1MB
32	2.13	49.39	1MB	2.13	49.39	1MB	2.13	49.39	1MB	2.13	49.39	1MB
$r = 10$	$w = 4$			$w = 5$			$w = 6$			$w = 7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	3.17	33.91	1MB	2.77	33.71	1MB	2.6	33.62	1MB	2.53	33.59	1MB
4	3.06	35.85	1MB	2.72	35.69	1MB	2.58	35.62	1MB	2.52	35.59	1MB
8	2.94	37.8	1MB	2.68	37.66	1MB	2.56	37.61	1MB	2.51	37.58	1MB
12	2.83	39.74	1MB	2.63	39.64	1MB	2.54	39.6	1MB	2.5	39.58	1MB
16	2.73	41.69	1MB	2.59	41.62	1MB	2.52	41.59	1MB	2.49	41.57	1MB
20	2.61	43.63	1MB	2.51	43.58	1MB	2.47	43.56	1MB	2.45	43.55	1MB
24	2.28	45.47	1MB	2.22	45.44	1MB	2.2	45.42	1MB	2.19	45.42	1MB
28	2.2	47.43	1MB	2.16	47.41	1MB	2.15	47.4	1MB	2.14	47.39	1MB
32	2.13	49.39	1MB	2.13	49.39	1MB	2.13	49.39	1MB	2.13	49.39	1MB

$r = 11$	$w = 0$			$w = 1$			$w = 2$			$w = 3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	14.72	39.68	2MB	8.77	36.71	1MB	4.34	34.5	1MB	2.61	33.63	1MB
4	12.5	40.58	1MB	7.36	38.	1MB	3.8	36.23	1MB	2.45	35.55	1MB
8	10.43	41.54	1MB	6.06	39.36	1MB	3.3	37.98	1MB	2.29	37.47	1MB
12	8.49	42.57	1MB	4.94	40.8	1MB	2.89	39.77	1MB	2.15	39.4	1MB
16	6.7	43.68	1MB	3.98	42.32	1MB	2.54	41.59	1MB	2.02	41.34	1MB
20	5.06	44.86	1MB	3.15	43.9	1MB	2.22	43.44	1MB	1.89	43.27	1MB
24	3.64	46.15	1MB	2.42	45.54	1MB	1.86	45.25	1MB	1.63	45.14	1MB
28	2.44	47.54	1MB	1.91	47.28	1MB	1.66	47.16	1MB	1.56	47.11	1MB
32	1.49	49.07	1MB	1.49	49.07	1MB	1.49	49.07	1MB	1.49	49.07	1MB
$r = 11$	$w = 4$			$w = 5$			$w = 6$			$w = 7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	2.02	33.33	1MB	1.82	33.24	1MB	1.75	33.2	1MB	1.73	33.19	1MB
4	1.97	35.31	1MB	1.81	35.23	1MB	1.75	35.2	1MB	1.72	35.19	1MB
8	1.92	37.29	1MB	1.79	37.22	1MB	1.74	37.2	1MB	1.72	37.19	1MB
12	1.87	39.26	1MB	1.77	39.21	1MB	1.73	39.19	1MB	1.72	39.19	1MB
16	1.83	41.24	1MB	1.75	41.2	1MB	1.73	41.19	1MB	1.72	41.18	1MB
20	1.76	43.21	1MB	1.71	43.18	1MB	1.7	43.17	1MB	1.69	43.17	1MB
24	1.55	45.1	1MB	1.52	45.09	1MB	1.52	45.08	1MB	1.51	45.08	1MB
28	1.52	47.09	1MB	1.5	47.08	1MB	1.49	47.07	1MB	1.49	47.07	1MB
32	1.49	49.07	1MB	1.49	49.07	1MB	1.49	49.07	1MB	1.49	49.07	1MB

$r = 12$	$w = 0$			$w = 1$			$w = 2$			$w = 3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	12.45	38.55	1MB	6.53	35.59	1MB	2.78	33.71	1MB	1.66	33.16	1MB
4	10.43	39.54	1MB	5.39	37.02	1MB	2.45	35.55	1MB	1.59	35.12	1MB
8	8.55	40.6	1MB	4.37	38.51	1MB	2.15	37.4	1MB	1.52	37.09	1MB
12	6.81	41.73	1MB	3.51	40.08	1MB	1.91	39.28	1MB	1.46	39.06	1MB
16	5.24	42.95	1MB	2.8	41.72	1MB	1.71	41.18	1MB	1.41	41.03	1MB
20	3.85	44.25	1MB	2.2	43.43	1MB	1.54	43.09	1MB	1.35	43.	1MB
24	2.66	45.66	1MB	1.69	45.17	1MB	1.32	44.98	1MB	1.2	44.93	1MB
28	1.75	47.2	1MB	1.37	47.01	1MB	1.23	46.94	1MB	1.18	46.92	1MB
32	1.15	48.9	1MB	1.15	48.9	1MB	1.15	48.9	1MB	1.15	48.9	1MB
$r = 12$	$w = 4$			$w = 5$			$w = 6$			$w = 7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	1.38	33.02	1MB	1.3	32.98	1MB	1.28	32.97	1MB	1.28	32.96	1MB
4	1.36	35.01	1MB	1.3	34.98	1MB	1.28	34.97	1MB	1.28	34.96	1MB
8	1.35	37.	1MB	1.3	36.97	1MB	1.28	36.97	1MB	1.28	36.96	1MB
12	1.33	38.99	1MB	1.29	38.97	1MB	1.28	38.96	1MB	1.27	38.96	1MB
16	1.31	40.98	1MB	1.29	40.97	1MB	1.28	40.96	1MB	1.27	40.96	1MB
20	1.29	42.97	1MB	1.27	42.96	1MB	1.26	42.96	1MB	1.26	42.95	1MB
24	1.17	44.91	1MB	1.16	44.91	1MB	1.16	44.91	1MB	1.16	44.91	1MB
28	1.16	46.91	1MB	1.16	46.9	1MB	1.15	46.9	1MB	1.15	46.9	1MB
32	1.15	48.9	1MB	1.15	48.9	1MB	1.15	48.9	1MB	1.15	48.9	1MB

$r = 13$	$w = 0$			$w = 1$			$w = 2$			$w = 3$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	10.45	37.55	1MB	4.73	34.69	1MB	1.78	33.22	1MB	1.2	32.93	1MB
4	8.62	38.64	1MB	3.86	36.26	1MB	1.62	35.13	1MB	1.18	34.91	1MB
8	6.93	39.79	1MB	3.09	37.87	1MB	1.47	37.06	1MB	1.15	36.9	1MB
12	5.41	41.03	1MB	2.47	39.56	1MB	1.35	39.	1MB	1.13	38.89	1MB
16	4.06	42.35	1MB	1.98	41.31	1MB	1.26	40.95	1MB	1.12	40.88	1MB
20	2.91	43.78	1MB	1.59	43.12	1MB	1.18	42.92	1MB	1.1	42.87	1MB
24	1.96	45.31	1MB	1.27	44.96	1MB	1.08	44.87	1MB	1.04	44.85	1MB
28	1.33	46.99	1MB	1.11	46.88	1MB	1.05	46.85	1MB	1.03	46.84	1MB
32	1.03	48.84	1MB	1.03	48.84	1MB	1.03	48.84	1MB	1.03	48.84	1MB
$r = 13$	$w = 4$			$w = 5$			$w = 6$			$w = 7$		
k	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M	p	C_{tr}	M
0	1.1	32.88	1MB	1.08	32.87	1MB	1.08	32.86	1MB	1.08	32.86	1MB
4	1.1	34.87	1MB	1.08	34.87	1MB	1.08	34.86	1MB	1.08	34.86	1MB
8	1.09	36.87	1MB	1.08	36.87	1MB	1.08	36.86	1MB	1.08	36.86	1MB
12	1.09	38.87	1MB	1.08	38.87	1MB	1.08	38.86	1MB	1.08	38.86	1MB
16	1.09	40.87	1MB	1.08	40.86	1MB	1.08	40.86	1MB	1.08	40.86	1MB
20	1.08	42.86	1MB	1.07	42.86	1MB	1.07	42.86	1MB	1.07	42.86	1MB
24	1.03	44.84	1MB	1.03	44.84	1MB	1.03	44.84	1MB	1.03	44.84	1MB
28	1.03	46.84	1MB	1.03	46.84	1MB	1.03	46.84	1MB	1.03	46.84	1MB
32	1.03	48.84	1MB	1.03	48.84	1MB	1.03	48.84	1MB	1.03	48.84	1MB

E Rogue CA Construction

Table E-1: *The to-be-signed part of our end-user X.509-certificate. Available from:* `http://www.win.tue.nl/hashclash/rogue-ca/downloads/real.cert.tbs.bin`.

Bytes (hex)	offset	description
30 82 03 9b	0–3	header
a0 03 02 01 02	4–8	version number
02 03 09 cf c7	9–13	serial number (‘*643015*’)
30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00	14–28	signature algorithm (‘*md5withRSAEncryption*’)
30 5a 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 1c 30 1a 06 03 55 04 0a 13 13 45 71 75 69 66 61 78 20 53 65 63 75 72 65 20 49 6e 63 2e 31 2d 30 2b 06 03 55 04 03 13 24 45 71 75 69 66 61 78 20 53 65 63 75 72 65 20 47 6c 6f 62 61 6c 20 65 42 75 73 69 6e 65 73 73 20 43 41 2d 31	29 – 120	issuer Distinguished Name (countryName: ‘*US*’, organizationName: ‘*Equifax Secure Inc.*’, commonName: ‘*Equifax Secure Global eBusiness CA*’)
30 1e 17 0d 30 38 31 31 30 33 30 37 35 32 30 32 5a 17 0d 30 39 31 31 30 34 30 37 35 32 30 32 5a	121–152	validity (‘*3 November 2008 07:52:02 UTC*’ –’*3 November 2009 07:52:02 UTC*’)
30 82 01 1c 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 49 30 47 06 03 55 04 0a 13 40 69 2e 62 72 6f 6b 65 2e 74 68 65 2e 69 6e 74 65 72 6e 65 74 2e 61 6e 64 2e 61 6c 6c 2e 69 2e 67 6f 74 2e 77 61 73 2e 74 68 69 73 2e 74 2d 73 68 69 72 74 2e 70 68 72 65 65 64 6f 6d 2e 6f 72 67 31 13 30 11 06 03 55 04 0b 13 0a 47 54 31 31 30 32 39 30 30 31 31 31 30 2f 06 03 55 04 0b 13 28 53 65 65 20 77 77 77 2e 72 61 70 69	153–440	subject Distinguished Name (countryName: ‘*US*’, organizationName: ‘*i.broke.the.internet.and.all .i.got.was.this.t-shirt .phreedom.org*’, organizationalUnitName: ‘*GT11029001*’, organizationalUnitName: ‘*See www.rapidssl.com/resources/cps (c)08*’, organizationalUnitName: ‘*Domain Control Validated - RapidSSL(R)*’, commonName: ‘*i.broke.the.internet.and.all .i.got.was.this.t-shirt*

Table E-1: *The to-be-signed part of our rogue CA X.509-certificate. (cont.)*

Bytes (hex)	offset	description
64 73 73 6c 2e 63 6f 6d 2f 72 65 73 6f 75 72 63 65 73 2f 63 70 73 20 28 63 29 30 38 31 2f 30 2d 06 03 55 04 0b 13 26 44 6f 6d 61 69 6e 20 43 6f 6e 74 72 6f 6c 20 56 61 6c 69 64 61 74 65 64 20 2d 20 52 61 70 69 64 53 53 4c 28 52 29 31 49 30 47 06 03 55 04 03 13 40 69 2e 62 72 6f 6b 65 2e 74 68 65 2e 69 6e 74 65 72 6e 65 74 2e 61 6e 64 2e 61 6c 6c 2e 69 2e 67 6f 74 2e 77 61 73 2e 74 68 69 73 2e 74 2d 73 68 69 72 74 2e 70 68 72 65 65 64 6f 6d 2e 6f 72 67		*.phreedom.org*')
30 82 01 22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 82 01 0f 00 30 82 01 0a 02 82 01 01 00 b2 d3 25 81 aa 28 e8 78 b1 e5 0a d5 3c 0f 36 57 6e a9 5f 06 41 0e 6b b4 cb 07	441 – 734	subject Public Key Info ('*rsaEncryption*', 2048-bit RSA modulus, RSA public exponent '*65537*'):
17 00 00 00 5b fd 6b 1c 7b 9c e8 a9	500 – 511	96-bit birthdaystring
a3 c5 45 0b 36 bb 01 d1 53 aa c3 08 8f 6f f8 4f 3e 87 87 44 11 dc 60 e0 df 92 55 f9 b8 73 1b 54 93 c5 9f d0 46 c4 60 b6 35 62 cd b9 af 1c a8 6b 1a c9 5b 3c 96 37 c0 ed 67 ef bb fe c0 8b 9c 50	512 – 575	1st near-collision block
2f 29 bd 83 22 9e 8e 08 fa ac 13 70 a2 58 7f 62 62 8a 11 f7 89 f6 df b6 67 59 73 16 fb 63 16 8a	576–639	2nd near-collision block

Table E-1: *The to-be-signed part of our rogue CA X.509-certificate. (cont.)*

Bytes (hex)	offset	description
b4 91 38 ce 2e f5 b6 be 4c a4 94 49 e4 65 51 0a 42 15 c9 c1 30 e2 69 d5 45 7d a5 26 bb b9 61 ec		
62 64 f0 39 e1 e7 bc 68 d8 50 51 9e 1d 60 d3 d1 a3 a7 0a f8 03 20 a1 70 01 17 91 36 4f 02 70 31 86 83 dd f7 0f d8 07 1d 11 b3 13 04 a5 da f0 ae 50 b1 28 0e 63 69 2a 0c 82 6f 8f 47 33 df 6c a2	640 – 703	3rd near-collision block
06 92 f1 4f 45 be d9 30 36 a3 2b 8c d6 77 ae 35 63 7f 4e 4c 9a 93 48 36 d9 9f 02 03 01 00 01	704–734	subject Public Key Info (continued)
	735 – 926	extensions:
a3 81 bd 30 81 ba	735 – 740	extensions header
30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 04 f0	741 – 756	keyUsage (‘*digital signature*’, ‘*nonrepudiation*’, ‘*key encipherment*’, ‘*data encipherment*’)
30 1d 06 03 55 1d 0e 04 16 04 14 cd a6 83 fa a5 60 37 f7 96 37 17 29 de 41 78 f1 87 89 55 e7	757 – 787	subjectKeyIdentifier
30 3b 06 03 55 1d 1f 04 34 30 32 30 30 a0 2e a0 2c 86 2a 68 74 74 70 3a 2f 2f 63 72 6c 2e 67 65 6f 74 72 75 73 74 2e 63 6f 6d 2f 63 72 6c 73 2f 67 6c 6f 62 61 6c 63 61 31 2e 63 72 6c	788 – 848	cRLDistributionPoints (‘*http://crl.geotrust.com* */crls/globalca1.crl*’)
30 1f 06 03 55 1d 23 04 18 30 16 80 14 be a8 a0 74 72 50 6b 44 b7 c9 23 d8 fb a8 ff b3 57 6b 68 6c	849 – 881	authorityKeyIdentifier
30 1d 06 03 55 1d 25 04 16 30 14 06 08 2b 06 01	882 – 912	extKeyUsage (‘*server authentication*’,

Table E-1: *The to-be-signed part of our rogue CA X.509-certificate. (cont.)*

Bytes (hex)	offset	description
05 05 07 03 01 06 08 2b 06 01 05 05 07 03 02		'*client authentication*')
30 0c 06 03 55 1d 13 01 01 ff 04 02 30 00	912 – 926	basicConstraints (CA: '*false*', path length: '*none*')

Table E-2: *The to-be-signed part of our rogue CA X.509-certificate. Available from:* `http://www.win.tue.nl/hashclash/rogue-ca/downloads/rogue_ca.cert.tbs.bin`.

Bytes (hex)	offset	description
30 82 03 9b	0–3	header
a0 03 02 01 02	4–8	version number
02 01 41	9–11	serial number ('*65*')
30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00	12–26	signature algorithm ('*md5withRSAEncryption*')
30 5a 31 0b 30 09 06 03 55 04 06 13 02 55 53 31 1c 30 1a 06 03 55 04 0a 13 13 45 71 75 69 66 61 78 20 53 65 63 75 72 65 20 49 6e 63 2e 31 2d 30 2b 06 03 55 04 03 13 24 45 71 75 69 66 61 78 20 53 65 63 75 72 65 20 47 6c 6f 62 61 6c 20 65 42 75 73 69 6e 65 73 73 20 43 41 2d 31	27 – 118	issuer Distinguished Name (countryName: '*US*', organizationName: '*Equifax Secure Inc.*', commonName: '*Equifax Secure Global eBusiness CA*')
30 1e 17 0d 30 34 30 37 33 31 30 30 30 30 30 30 5a 17 0d 30 34 30 39 30 32 30 30 30 30 30 30 5a	119–150	validity ('*31 July 2004 00:00:00 UTC*' –'*2 September 2004 00:00:00 UTC*')
30 3c 31 3a 30 38 06 03 55 04 03 13 31 4d 44 35 20 43 6f 6c 6c 69 73 69 6f 6e 73 20 49 6e 63 2e 20 28 68 74 74 70 3a 2f 2f 77 77 77 2e 70 68 72 65 65 64 6f 6d 2e 6f 72 67 2f 6d 64 35 29	153 – 212	subject Distinguished Name (commonName: '*MD5 Collisions Inc.* *(http://www.phreedom.org/md5)*')
30 81 9f 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 03 81 8d 00 30 81	213 – 374	subject Public Key Info ('*rsaEncryption*', 1024-bit RSA modulus,

Table E-2: *The to-be-signed part of our rogue CA X.509-certificate. (cont.)*

Bytes (hex)	**offset**	**description**
89 02 81 81 00 ba a6 59 c9 2c 28 d6 2a b0 f8 ed 9f 46 a4 a4 37 ee 0e 19 68 59 d1 b3 03 99 51 d6 16 9a 5e 37 6b 15 e0 0e 4b f5 84 64 f8 a3 db 41 6f 35 d5 9b 15 1f db c4 38 52 70 81 97 5e 8f a0 b5 f7 7e 39 f0 32 ac 1e ad 44 d2 b3 fa 48 c3 ce 91 9b ec f4 9c 7c e1 5a f5 c8 37 6b 9a 83 de e7 ca 20 97 31 42 73 15 91 68 f4 88 af f9 28 28 c5 e9 0f 73 b0 17 4b 13 4c 99 75 d0 44 e6 7e 08 6c 1a f2 4f 1b 41 02 03 01 00 01		RSA public exponent '*65537*')
	375 – 926	extensions:
a3 82 02 24 30 82 02 20	375 – 378	extensions header
30 0b 06 03 55 1d 0f 04 04 03 02 01 c6	383 – 395	keyUsage ('*digital signature*', '*nonrepudiation*', '*certificate signing*', '*CRL signing*' '*offline CRL signing*')
30 0f 06 03 55 1d 13 01 01 ff 04 05 30 03 01 01 ff	396 – 412	basicConstraints (CA: '*true*', path length: '*none*')
30 1d 06 03 55 1d 0e 04 16 04 14 a7 04 60 1f ab 72 43 08 c5 7f 08 90 55 56 1c d6 ce e6 38 eb	413 – 443	subjectKeyIdentifier
30 1f 06 03 55 1d 23 04 18 30 16 80 14 be a8 a0 74 72 50 6b 44 b7 c9 23 d8 fb a8 ff b3 57 6b 68 6c	444 – 476	authorityKeyIdentifier
	477 – 926	netscape comment:
30 82 01 be 06 09 60 86 48 01 86 f8 42 01 0d 04 82 01 af	477 – 495	comment header
16 82 01 ab	496–499	random padding

Table E-2: *The to-be-signed part of our rogue CA X.509-certificate. (cont.)*

Bytes (hex)	offset	description
33 00 00 00 27 5e 39 e0 89 61 0f 4e	500–511	96-bit birthday string
a3 c5 45 0b 36 bb 01 d1 53 aa c3 08 8f 6f f8 4f 3e 87 87 44 11 dc 60 e0 df 92 55 f9 b8 73 1b 54 93 c5 9f d0 46 c4 60 b6 35 62 cd b9 af 1c a8 69 1a c9 5b 3c 96 37 c0 ed 67 ef bb fe c0 8b 9c 50	512–575	1st near-collision block
2f 29 bd 83 22 9e 8e 08 fa ac 13 70 a2 58 7f 62 62 8a 11 f7 89 f6 df b6 67 59 73 16 fb 63 16 8a b4 91 38 ce 2e f5 b6 be 4c a4 94 49 e4 65 11 0a 42 15 c9 c1 30 e2 69 d5 45 7d a5 26 bb b9 61 ec	576–639	2nd near-collision block
62 64 f0 39 e1 e7 bc 68 d8 50 51 9e 1d 60 d3 d1 a3 a7 0a f8 03 20 a1 70 01 17 91 36 4f 02 70 31 86 83 dd f7 0f d8 07 1d 11 b3 13 04 a5 dc f0 ae 50 b1 28 0e 63 69 2a 0c 82 6f 8f 47 33 df 6c a2	640–703	3rd near-collision block
06 92 f1 4f 45 be d9 30 36 a3 2b 8c d6 77 ae 35 63 7f 4e 4c 9a 93 48 36 d9 9f 02 03 01 00 01 a3 81 bd 30 81 ba 30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 04 f0 30 1d 06 03 55 1d 0e 04 16 04 14 cd a6 83 fa a5 60 37 f7 96 37 17 29 de 41 78 f1 87 89 55 e7 30 3b 06 03 55 1d 1f 04 34 30 32 30 30 a0 2e a0 2c 86 2a 68 74 74 70 3a 2f 2f 63 72 6c 2e 67 65 6f 74 72 75	704–926	identical suffix: copied from end-user certificate

Table E-2: *The to-be-signed part of our rogue CA X.509-certificate. (cont.)*

Bytes (hex)	offset	description
73 74 2e 63 6f 6d 2f 63		
72 6c 73 2f 67 6c 6f 62		
61 6c 63 61 31 2e 63 72		
6c 30 1f 06 03 55 1d 23		
04 18 30 16 80 14 be a8		
a0 74 72 50 6b 44 b7 c9		
23 d8 fb a8 ff b3 57 6b		
68 6c 30 1d 06 03 55 1d		
25 04 16 30 14 06 08 2b		
06 01 05 05 07 03 01 06		
08 2b 06 01 05 05 07 03		
02 30 0c 06 03 55 1d 13		
01 01 ff 04 02 30 00		

Table E-3: *Rogue CA - first differential path*

t	Bitconditions: $\mathfrak{q}_t[31]\ldots\mathfrak{q}_t[0]$
-3	....
-2	.....1. .0.0-... .1.10.0. .+---.+-
-1	..1.1.0. .1-++..0 .010101. .+--1---
0	..1.0.-. .+1++^.0 .-1++1-. ^..0+1+-
1	..+.-.-. .+++.+.- .0-11-0. -101+0+0
2	0.+.-... .-0-11.+ .1+00-1. 0..-.-.-
3	0.1.+.0. .-01.0.- ..1.+0.. +..10+0-
4	-.-.+1.. .+.1.+.0 ..+.0+.. -..01-.+
5	+.1..0.. .+..^0.0 ..-.10.. 0...+1.1
6	+.1.1+.. ....+1.- ..-.00.. 0...+-.-
7	+.+..-.. .11.1+.+ ..-.+0.. 1....1.+
8	-.0.0-.. ..1.10.- ....0-.. 1...01.0
9	-.+.1-.. ..-..1.- .^0.1+.. -...0..1
10	0.00+.10 01-00..- ^+.0.-.1 -.101101
11	1.-1-010 11-011.+ +1.10-01 -101+010
12	-^+++1+- ++1+-1^+ 01^-0+1- 00+-+-++
13	1-+0--+- +-+----- --+-+++- ++-+.1--
14	10-1000+ .10+.-10 0-1.-.-. 1-+0010-
15	.11.01++ 000+1000 0-00-010 111+..00
16	..0...11 ...-.+.. .+..0.^. .++..0.+
17	..10+.00 ...1.0.. .0..^.0. .0.^.1..
18	..+.+.1. ...-.-.. .-....0. .-^..-.^
19	..1++.-. ...-.... 0-....+. .....
20	..0++... 0..0.^.. 0+..0... 0^...^..
21	...++.^. 1..^.... +1..1.^. 1....0..
22	...-1... -..... .^..-... +....1..
23	...00... ...0.... ^..... .....-.0
24	...1.... ^..1...0 ....^... ^.0....1
25	....+...1 ....^.+
26	......0. .....+ ..... .0-.....
27	......1. ...^.... ..... .1-....^
28	......-. .....^ ..... .-+.....
29	......0. ....0.....
30	......+. ....^1.....
31	......+. ....
32	......1. ....
33	......0. ....
34–59	....
60	....00...
61	....1...
62	....-+...
63	....?-...
64	......+. ....-...

$$\delta m_{11} = -2^{25}$$

Table E-4: *Rogue CA - second differential path*

t	Bitconditions: $\mathfrak{q}_t[31]\ldots\mathfrak{q}_t[0]$
-3	....
-2	....1.0- +++++... .0.00... ......+-
-1	.0..1^10 0.11-1.. .1.01.^1 .....^0+
0	.1..+--- -1-0-0.. .-.++.-1 .....+--
1	.+..+000 0.++++.. .-.++.1- 111
2	.0..+01+ 11+01-0. ..1.-.++ .....-00
3	.-..-..- .1+0-10. .0011.0+ ..^..-..
4	.+^.-..- .++.+0+. ..+.0.+. ..-.11..
5	.++...0- .+..1.+. ..+...00 ..0.1+..
6	.-1.0.0+ ..0.1.+. ..-...11 ..1.+1..
7	.11...++ .1....+. ..1.^.-- ....-+..
8	.-1...-. ....-.-.00 ....0-..
9	.1-.1.11 ....1.0. ..-00.11 ....++..
10	1+-.01-. .11010.. 10-1+.0. ..0.0-0.
11	0-0.+00. 1001-10^ 110--0+. .01.111^
12	--0^0++^ 0-++-+1+ +-+1-0+0 ^0-^+1+-
13	0++-++++ +1++-1-0 +0-1--+1 ------+1
14	10101.-0 0-1.+0-0 +011.+.- 1+.1-+01
15	.10101+1 0++10.+. 1.0-010- 1100+1-.
16	....0.11 .+..1.-. -...0^.1 .^..--..
17	....1.-. .1^.-.0. 0..^...^ .0..1.^.
18	0...+.-+ .1....-. +...-... .0..-^..
19	......++ .-..^... ...0-... .+......
20	+...^.+0 ...0..^. ^..0+..0 ...0^...
21	+.....0+ .^.1.... ...+1..1 .^.1....
22	-.....-. ...-.... ....^..- ...+....
23	-.0....^ ......0. ...^.... ....
24	1.1...^. ...^..1. ..0....^ ...^.0..
25	0.+..... ......+. ..1..... ....
26	.... .0...... ..+..... ....0-..
27	..^..... .1....^. ....1-..
28	....-...... ..^..... ....-+..
29	.... .0...... ....0..
30	....+...... ....^1..
31	....+...... ....
32	.... .0...... ....
33	.... .1...... ....
34–59	....
60	000
61	100
62	....-++
63	....-++
64	.... -+-..... ....-

$$\delta m_{11} = -2^{22}$$

Table E-5: *Rogue CA - third differential path*

t	Bitconditions: $\mathfrak{q}_t[31]\ldots\mathfrak{q}_t[0]$
-3	....
-2	....1-+... ..01.00. .....
-1	.....0.. ..-++1.0 .010011. .....
0	.....1.. ..-++1.1 .1--0++. .1.....0
1	...^.+.. ..-+.+.- .-0++0-. .1.....1
2	...-.1.. ...+0+.- .-1++1-. .+^....-
3	...0.1.. ..1+...+ ..-...-^ .+-...^-
4	...0.^.0 ^.1+.0.+ .1-10..+ .-0..^+.
5	.^...+.0 -.0....- ......10 .+0..-10
6	.+...0.- 0.-0...+ ..1....- .+..10-.
7	.0...-1+ 1.+....- .....^.1 .-..110.
8	.1...-1+ +.+....0 ..0..+.1 .1..-.0.
9	......-- +.+^...0 ..0..0.+ .0..-.0.
10	10..11-- +.0+0000 .1-0.+.+ .-.1-.01
11	11..11-0 -.101111 01-1^110 ^+.1-1-0
12	-+.^-0+- +.-+++++ 1--+-101 +-1-10+-
13	+0.+---- 1.11-001 ---00.+- 0-0+-+++
14	11.1+-0+ 0.+-011+ 1.-1-.1+ 0.-010.0
15	00.0+.10 ..1+0.-0 11-01.1- .01-1100
16	....0^.+ .0+1.0.- .01.0..1 ..^+.0..
17	.+..^.0. .0.0.1^. .1^++.00 ...1.1..
18	.+....0^ .-^..+.^ .-.++.1. ...+.-..
19	0-....-. ....--.+. ...+....
20	1-..0... 0^...^.. .^.0-... 0..0.^..
21	+0..1.^. 1....0.. ...1-.^. 1..^....
22	.^..+... -....1.. ...+1... +.......
23	^....... .....+.0 ....0... ...0....
24	....^... ^.0....1 ...^.... ^..1...0
25	....^.- ....-...1
26	..... .0+..... ......0. ....-
27	..... .1+....^ ......1. ...^....
28	..... .+-..... ......+. ....^
29	....0..... ......0. .....
30	..... .^1..... ......-. .....
31	....-. .....
32	....1. .....
33	....0. .....
34–59	....
60	....0... ....
61	.......1 01011... ....
62	.......0 .1.1+... ....
63	.......+ -----... ....
64	....+-..+ --.-.--. .....

$$\delta m_{11} = 2^9$$

F SHA-1 disturbance vector analysis

The tables shown in this appendix are based on the disturbance vector cost function $\mathrm{FDC}_{u,t_b,\epsilon}$ which is a modification of the cost function FDC_{u,t_b} presented in Section 7.5.11. The cost function FDC_{u,t_b} uses Algorithms 7-2 and 7-3 (p. 150 and 151) to determine sets $\mathcal{A}_{i,j}$. The cost function $\mathrm{FDC}_{u,t_b,\epsilon}$ uses a simple modification of these algorithms that works instead of sets $\mathcal{A}_{i,j}$ with sets $\mathcal{A}'_{i,j}$ of the form:

$$\mathcal{A}'_{i,j} \subseteq \{(\mathcal{P}, g(\mathcal{S}_\mathcal{P})) \mid (\mathcal{P}, \mathcal{S}_\mathcal{P}) \in \mathcal{A}_{i,j}\}, \quad \text{where } g(\mathcal{S}_\mathcal{P}) \subseteq \mathcal{S}_\mathcal{P} \text{ for all } \mathcal{P}. \tag{F.1}$$

We add in both algorithms a step 3.5 between step 3 and step 4 that, for each value of t in step 3, removes all message difference vectors in the previous $\mathcal{A}'_{i,j}$ (thus either $(i,j) = (t_b, t-1)$ or $(i,j) = (t+1, t_e)$) with a too low total success probability. Let $\mathcal{W} = \{w \mid (w,p) \in \mathcal{S}, (\mathcal{P}, \mathcal{S}) \in \mathcal{A}'_{i,j}\}$ and for $w \in \mathcal{W}$ let

$$p_w = \sum_{(\mathcal{P},\mathcal{S}) \in \mathcal{A}'_{i,j}} \sum_{\substack{(w,p) \in \mathcal{S} \\ w'=w}} p$$

be the total success probability of w. Let $p_{\max} = \max_{w \in \mathcal{W}} p_w$ be the maximum of these total success probabilities. Then after $p_{\max}$ has been determined we remove from $\mathcal{A}'_{i,j}$ all occurrences of message difference vectors w for which $p_w < \epsilon \cdot p_{\max}$, where $\epsilon \in [0,1]$ is some chosen fraction. Next we remove all pairs of the form $(\mathcal{P}, \emptyset)$ from $\mathcal{A}'_{i,j}$. More precisely, let $\widetilde{\mathcal{A}}'_{i,j}$ denote the contents of the set $\mathcal{A}'_{i,j}$ in these algorithms before step 3.5, then the contents of $\mathcal{A}'_{i,j}$ after step 3.5 is determined as:

$$\mathcal{A}'_{i,j} = \left\{ \left(\mathcal{P}, \mathrm{Trim}\left(\widetilde{\mathcal{S}}'_\mathcal{P}\right)\right) \;\middle|\; \left(\mathcal{P}, \widetilde{\mathcal{S}}'_\mathcal{P}\right) \in \widetilde{\mathcal{A}}'_{i,j} \wedge \mathrm{Trim}\left(\widetilde{\mathcal{S}}'_\mathcal{P}\right) \neq \emptyset \right\},$$
$$\mathrm{Trim}\left(\widetilde{\mathcal{S}}'_\mathcal{P}\right) = \left\{ (w,p) \in \widetilde{\mathcal{S}}'_\mathcal{P} \;\middle|\; p_w \geq \epsilon \cdot p_{\max} \right\}.$$

Informally, $\mathcal{A}'_{i,j}$ contains a subset of the information present in $\mathcal{A}_{i,j}$ (Equation F.1), thus it follows that $\mathrm{FDC}_{u,t_b,\epsilon}((DV_t)_{t=0}^{79}) \leq \mathrm{FDC}_{u,t_b}((DV_t)_{t=0}^{79})$ for all disturbance vectors $(DV_t)_{t=0}^{79}$. For $\epsilon = 0$, it is clear that nothing will we removed in the added procedure and thus $\mathrm{FDC}_{u,t_b,\epsilon}$ is equivalent to FDC_{u,t_b}. Moreover, the outcome of $\mathrm{FDC}_{u,t_b,\epsilon}$ will also be the same as that of FDC_{u,t_b} if ϵ is small enough so that no optimal message difference vectors are removed in this manner. For $\epsilon \leq 0.5$ we have not noticed a difference in outcomes so far, but we have noticed differences for values of ϵ that were even slightly bigger than 0.5.

The numbers presented in the following tables are the negative $\log_2$ results of the cost function $\mathrm{FDC}_{u,20,\epsilon}$ for $u \in \{0,\ldots,7\}$. For each disturbance vector and each value of u, we try the values $0, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}$ for ϵ in that order and keep the first result (and thus lowest value for ϵ) for which the computation succeeds within 24 hours using the 8GB of RAM available. An empty result in the following tables reflects the fact that the computations failed for all these values for ϵ either due a too long runtime or a memory allocation failure.

Table F-1: *Most interesting disturbance vectors*

	u							
DV	0	1	2	3	4	5	6	7
I(48, 0)	75.00 ϵ=0	71.84 ϵ=0	71.61 ϵ=0	71.51 ϵ=0	71.46 ϵ=0	71.44 ϵ=0	71.43 ϵ=0	71.42 ϵ=0
I(49, 0)	76.00 ϵ=0	72.59 ϵ=0	72.34 ϵ=0	72.24 ϵ=0	72.19 ϵ=0	72.17 ϵ=0	72.16 ϵ=0	72.15 ϵ=0
I(50, 0)	75.00 ϵ=0	72.02 ϵ=0	71.95 ϵ=0	71.93 ϵ=0	71.92 ϵ=0	71.92 ϵ=0	71.92 ϵ=0	71.92 ϵ=0
II(46, 0)	76.00 ϵ=0	71.85 ϵ=0	71.83 ϵ=1/2					
II(50, 0)	78.00 ϵ=0	73.52 ϵ=0	73.23 ϵ=0	73.12 ϵ=0	73.06 ϵ=0	73.04 ϵ=0	73.03 ϵ=0	73.02 ϵ=0
II(51, 0)	77.00 ϵ=0	72.55 ϵ=0	72.18 ϵ=0	72.02 ϵ=0	71.95 ϵ=0	71.91 ϵ=0	71.89 ϵ=0	71.88 ϵ=0
II(52, 0)	75.00 ϵ=0	71.88 ϵ=0	71.87 ϵ=0	71.76 ϵ=0	71.76 ϵ=0	71.75 ϵ=0	71.75 ϵ=0	71.75 ϵ=0

The columns are the negative $\log_2$ results of the cost function $\mathrm{FDC}_{u,20,\epsilon}$.

Table F-2: *Overview of disturbance vectors* $I(K, 0)$

	u							
DV	0	1	2	3	4	5	6	7
I(42, 0)	82.68 ϵ=0	78.67 ϵ=0	78.36 ϵ=1/4					
I(43, 0)	82.00 ϵ=0	77.65 ϵ=0	77.31 ϵ=1/8					
I(44, 0)	81.00 ϵ=0	77.41 ϵ=0	77.1 ϵ=0	76.98 ϵ=0	76.93 ϵ=1/8	76.90 ϵ=1/8	76.89 ϵ=1/8	76.89 ϵ=1/8
I(45, 0)	81.00 ϵ=0	76.91 ϵ=0	76.66 ϵ=0	76.54 ϵ=0	76.49 ϵ=0	76.47 ϵ=1/8	76.46 ϵ=1/8	76.45 ϵ=1/8
I(46, 0)	79.00 ϵ=0	75.02 ϵ=0	74.92 ϵ=0	74.84 ϵ=0	74.83 ϵ=0	74.83 ϵ=0	74.83 ϵ=0	74.83 ϵ=1/8
I(47, 0)	79.00 ϵ=0	75.15 ϵ=0	74.83 ϵ=0	74.71 ϵ=0	74.65 ϵ=0	74.63 ϵ=0	74.62 ϵ=0	74.61 ϵ=0
I(48, 0)	75.00 ϵ=0	71.84 ϵ=0	71.61 ϵ=0	71.51 ϵ=0	71.46 ϵ=0	71.44 ϵ=0	71.43 ϵ=0	71.42 ϵ=0
I(49, 0)	76.00 ϵ=0	72.59 ϵ=0	72.34 ϵ=0	72.24 ϵ=0	72.19 ϵ=0	72.17 ϵ=0	72.16 ϵ=0	72.15 ϵ=0
I(50, 0)	75.00 ϵ=0	72.02 ϵ=0	71.95 ϵ=0	71.93 ϵ=0	71.92 ϵ=0	71.92 ϵ=0	71.92 ϵ=0	71.92 ϵ=0
I(51, 0)	77.00 ϵ=0	73.76 ϵ=0	73.53 ϵ=0	73.43 ϵ=0	73.38 ϵ=0	73.36 ϵ=0	73.35 ϵ=0	73.34 ϵ=0
I(52, 0)	79.00 ϵ=0	76.26 ϵ=0	76.24 ϵ=0	76.24 ϵ=0	76.24 ϵ=0	76.24 ϵ=0	76.24 ϵ=0	76.24 ϵ=0
I(53, 0)	82.83 ϵ=0	78.86 ϵ=0	78.79 ϵ=0	78.77 ϵ=0	78.77 ϵ=0	78.77 ϵ=0	78.77 ϵ=0	78.77 ϵ=0
I(54, 0)	82.83 ϵ=0	79.60 ϵ=0	79.38 ϵ=0	79.28 ϵ=0	79.23 ϵ=0	79.21 ϵ=0	79.19 ϵ=0	79.19 ϵ=0
I(55, 0)	81.54 ϵ=0	78.67 ϵ=0	78.42 ϵ=0	78.32 ϵ=0	78.27 ϵ=0	78.25 ϵ=0	78.24 ϵ=0	78.23 ϵ=0
I(56, 0)	81.54 ϵ=0	79.10 ϵ=0	79.03 ϵ=0	79.01 ϵ=0	79.01 ϵ=0	79.01 ϵ=0	79.01 ϵ=0	79.01 ϵ=0

The columns are the negative $\log_2$ results of the cost function $\mathrm{FDC}_{u,20,\epsilon}$.

Table F-3: *Overview of disturbance vectors* $I(K, 2)$

	u							
DV	0	1	2	3	4	5	6	7
I(42, 2)	85.09 ϵ=0	82.17 ϵ=1/4	81.84 ϵ=1/2	81.72 ϵ=1/2				
I(43, 2)	84.42 ϵ=0	81.15 ϵ=1/4	80.78 ϵ=1/2					
I(44, 2)	84.42 ϵ=0	81.92 ϵ=0	81.57 ϵ=1/4	81.45 ϵ=1/2	81.40 ϵ=1/2	81.38 ϵ=1/2	81.37 ϵ=1/2	81.36 ϵ=1/2
I(45, 2)	83.42 ϵ=0	80.80 ϵ=0	80.52 ϵ=0	80.41 ϵ=1/4	80.36 ϵ=1/2	80.34 ϵ=1/2	80.33 ϵ=1/2	80.32 ϵ=1/2
I(46, 2)	80.42 ϵ=0	78.10 ϵ=0	78.00 ϵ=0	77.99 ϵ=1/8	77.99 ϵ=1/8	77.99 ϵ=1/8	77.99 ϵ=1/8	77.99 ϵ=1/4
I(47, 2)	79.68 ϵ=0	77.01 ϵ=0	76.68 ϵ=0	76.56 ϵ=0	76.51 ϵ=1/8	76.48 ϵ=1/8	76.47 ϵ=1/8	76.47 ϵ=1/8
I(48, 2)	76.68 ϵ=0	74.27 ϵ=0	73.99 ϵ=0	73.88 ϵ=0	73.83 ϵ=0	73.81 ϵ=0	73.80 ϵ=0	73.79 ϵ=0
I(49, 2)	77.00 ϵ=0	74.30 ϵ=0	74.02 ϵ=0	73.92 ϵ=0	73.87 ϵ=0	73.85 ϵ=0	73.84 ϵ=0	73.83 ϵ=0
I(50, 2)	77.00 ϵ=0	74.74 ϵ=0	74.63 ϵ=0	74.61 ϵ=0	74.61 ϵ=0	74.60 ϵ=0	74.60 ϵ=0	74.60 ϵ=0
I(51, 2)	80.00 ϵ=0	77.47 ϵ=0	77.21 ϵ=0	77.11 ϵ=0	77.07 ϵ=0	77.04 ϵ=0	77.03 ϵ=0	77.03 ϵ=0
I(52, 2)	82.00 ϵ=0	79.98 ϵ=0	79.93 ϵ=0	79.92 ϵ=0	79.92 ϵ=0	79.92 ϵ=0	79.92 ϵ=0	79.92 ϵ=0
I(53, 2)	84.00 ϵ=0	81.91 ϵ=0	81.80 ϵ=0	81.78 ϵ=0	81.78 ϵ=0	81.78 ϵ=0	81.78 ϵ=0	81.78 ϵ=0
I(54, 2)	84.00 ϵ=0	81.37 ϵ=0	81.06 ϵ=0	80.95 ϵ=0	80.90 ϵ=0	80.87 ϵ=0	80.86 ϵ=0	80.85 ϵ=0
I(55, 2)	84.00 ϵ=0	81.78 ϵ=0	81.53 ϵ=0	81.43 ϵ=0	81.38 ϵ=0	81.36 ϵ=0	81.34 ϵ=0	81.34 ϵ=0
I(56, 2)	82.00 ϵ=0	80.22 ϵ=0	80.13 ϵ=0	80.12 ϵ=0	80.11 ϵ=0	80.11 ϵ=0	80.11 ϵ=0	80.11 ϵ=0

The columns are the negative $\log_2$ results of the cost function $\mathrm{FDC}_{u,20,\epsilon}$.

Table F-4: *Overview of disturbance vectors* $II(K, 0)$

DV	u							
	0	1	2	3	4	5	6	7
$II(44,0)$	87.00 ϵ=0	79.51 ϵ=1/2						
$II(45,0)$	83.00 ϵ=0	75.45 ϵ=1/8	74.82 ϵ=1/2					
$II(46,0)$	76.00 ϵ=0	71.85 ϵ=0	71.83 ϵ=1/2					
$II(47,0)$	81.42 ϵ=0	76.23 ϵ=0	75.87 ϵ=1/2					
$II(48,0)$	80.00 ϵ=0	76.11 ϵ=0	75.89 ϵ=0	75.79 ϵ=1/8	75.74 ϵ=1/2			
$II(49,0)$	80.00 ϵ=0	75.04 ϵ=0	74.72 ϵ=0	74.60 ϵ=0	74.55 ϵ=1/8	74.52 ϵ=1/8	74.51 ϵ=1/2	74.51 ϵ=1/2
$II(50,0)$	78.00 ϵ=0	73.52 ϵ=0	73.23 ϵ=0	73.12 ϵ=0	73.06 ϵ=0	73.04 ϵ=0	73.03 ϵ=0	73.02 ϵ=0
$II(51,0)$	77.00 ϵ=0	72.55 ϵ=0	72.18 ϵ=0	72.02 ϵ=0	71.95 ϵ=0	71.91 ϵ=0	71.89 ϵ=0	71.88 ϵ=0
$II(52,0)$	75.00 ϵ=0	71.88 ϵ=0	71.87 ϵ=0	71.76 ϵ=0	71.76 ϵ=0	71.75 ϵ=0	71.75 ϵ=0	71.75 ϵ=0
$II(53,0)$	76.96 ϵ=0	73.65 ϵ=0	73.34 ϵ=1/8	73.23 ϵ=1/8	73.17 ϵ=1/8	73.15 ϵ=1/8	73.14 ϵ=1/8	73.14 ϵ=1/8
$II(54,0)$	77.96 ϵ=0	73.97 ϵ=0	73.74 ϵ=1/8	73.64 ϵ=1/8	73.59 ϵ=1/8	73.57 ϵ=1/8	73.56 ϵ=1/8	73.55 ϵ=1/8
$II(55,0)$	77.96 ϵ=0	75.22 ϵ=1/8	74.99 ϵ=1/2	74.89 ϵ=1/2	74.84 ϵ=1/2	74.82 ϵ=1/2	74.81 ϵ=1/2	74.80 ϵ=1/2
$II(56,0)$	76.96 ϵ=0	74.48 ϵ=1/2	74.18 ϵ=1/2	74.07 ϵ=1/2	74.01 ϵ=1/2	73.99 ϵ=1/2	73.98 ϵ=1/2	73.97 ϵ=1/2

The columns are the negative $\log_2$ results of the cost function $\mathrm{FDC}_{u,20,\epsilon}$.

Table F-5: *Overview of disturbance vectors* $II(K, 2)$

	u							
DV	0	1	2	3	4	5	6	7
$II(45, 2)$	85.00 ϵ=0	78.64 ϵ=1/2						
$II(46, 2)$	82.00 ϵ=0	77.51 ϵ=1/2						
$II(47, 2)$	85.42 ϵ=0	79.83 ϵ=1/2						
$II(48, 2)$	83.00 ϵ=0	78.81 ϵ=1/2	78.46 ϵ=1/2					
$II(49, 2)$	83.00 ϵ=0	78.09 ϵ=0	77.74 ϵ=1/2					
$II(50, 2)$	81.00 ϵ=0	76.51 ϵ=0	76.16 ϵ=1/8	76.03 ϵ=1/8				
$II(51, 2)$	82.00 ϵ=0	77.74 ϵ=0	77.36 ϵ=1/8	77.20 ϵ=1/8	77.13 ϵ=1/2			
$II(52, 2)$	82.00 ϵ=0	79.07 ϵ=0	78.96 ϵ=0	78.94 ϵ=0	78.94 ϵ=1/8	78.93 ϵ=1/8	78.93 ϵ=1/8	78.93 ϵ=1/2
$II(53, 2)$	83.00 ϵ=0	79.60 ϵ=0	79.30 ϵ=0	79.18 ϵ=0	79.13 ϵ=1/8	79.11 ϵ=1/8	79.09 ϵ=1/8	79.09 ϵ=1/8
$II(54, 2)$	84.00 ϵ=0	80.49 ϵ=0	80.21 ϵ=0	80.10 ϵ=0	80.04 ϵ=1/8	80.02 ϵ=1/8	80.01 ϵ=1/8	80.00 ϵ=1/8
$II(55, 2)$	84.00 ϵ=0	81.20 ϵ=0	80.88 ϵ=0	80.76 ϵ=0	80.71 ϵ=0	80.68 ϵ=0	80.67 ϵ=0	80.67 ϵ=1/8
$II(56, 2)$	85.00 ϵ=0	82.69 ϵ=1/4	82.39 ϵ=1/4	82.27 ϵ=1/4	82.22 ϵ=1/4	82.20 ϵ=1/4	82.19 ϵ=1/4	82.18 ϵ=1/4

The columns are the negative $\log_2$ results of the cost function $\mathrm{FDC}_{u,20,\epsilon}$.

References

[AHMP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan, *SHA-3 proposal BLAKE*, Submission to NIST (Round 3), 2010.

[ANPS07] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton, *Seven-Property-Preserving Iterated Hashing: ROX*, ASIACRYPT (Kaoru Kurosawa, ed.), Lecture Notes in Computer Science, vol. 4833, Springer, 2007, pp. 130–146.

[BC04] Eli Biham and Rafi Chen, *Near-Collisions of SHA-0*, CRYPTO (Matthew K. Franklin, ed.), Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 290–305.

[BCJ+05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby, *Collisions of SHA-0 and Reduced SHA-1*, EUROCRYPT (Ronald Cramer, ed.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 36–57.

[BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *The Keccak SHA-3 submission*, Submission to NIST (Round 3), 2011.

[Ber92] Thomas A. Berson, *Differential Cryptanalysis Mod* 2^{32} *with Applications to MD5*, EUROCRYPT, 1992, pp. 71–80.

[BP95] Antoon Bosselaers and Bart Preneel (eds.), *Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation RIPE-RACE 1040*, Lecture Notes in Computer Science, vol. 1007, Springer, 1995.

[BR06a] Mihir Bellare and Thomas Ristenpart, *Multi-Property-Preserving Hash Domain Extension and the EMD Transform*, ASIACRYPT (Xuejia Lai and Kefei Chen, eds.), Lecture Notes in Computer Science, vol. 4284, Springer, 2006, pp. 299–314.

[BR06b] Steven M. Bellovin and Eric Rescorla, *Deploying a New Hash Algorithm*, NDSS, The Internet Society, 2006.

[BR07] Mihir Bellare and Thomas Ristenpart, *Hash Functions in the Dedicated-Key Setting: Design Choices and MPP Transforms*, ICALP (Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, eds.), Lecture Notes in Computer Science, vol. 4596, Springer, 2007, pp. 399–410.

[BS90] Eli Biham and Adi Shamir, *Differential Cryptanalysis of DES-like Cryptosystems*, CRYPTO (Alfred Menezes and Scott A. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 2–21.

[BS91] Eli Biham and Adi Shamir, *Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer*, CRYPTO (Joan Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1991, pp. 156–171.

[BS92] Eli Biham and Adi Shamir, *Differential Cryptanalysis of the Full 16-Round DES*, CRYPTO (Ernest F. Brickell, ed.), Lecture Notes in Computer Science, vol. 740, Springer, 1992, pp. 487–496.

[Bun08] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen, *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen)*, 2008, p. 376.

[Che11] Rafael Chen, *New Techniques for Cryptanalysis of Cryptographic Hash Functions*, Ph.D. thesis, Technion, Aug 2011.

[CJ98] Florent Chabaud and Antoine Joux, *Differential Collisions in SHA-0*, CRYPTO (Hugo Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1462, Springer, 1998, pp. 56–71.

[CMR07] Christophe De Cannière, Florian Mendel, and Christian Rechberger, *Collisions for 70-Step SHA-1: On the Full Cost of Collision Search*, Selected Areas in Cryptography (Carlisle M. Adams, Ali Miri, and Michael J. Wiener, eds.), Lecture Notes in Computer Science, vol. 4876, Springer, 2007, pp. 56–73.

[Coc07] Martin Cochran, *Notes on the Wang et al.* 2^{63} *SHA-1 Differential Path*, Cryptology ePrint Archive, Report 2007/474, 2007.

[CR06] Christophe De Cannière and Christian Rechberger, *Finding SHA-1 Characteristics: General Results and Applications*, ASIACRYPT (Xuejia Lai and Kefei Chen, eds.), Lecture Notes in Computer Science, vol. 4284, Springer, 2006, pp. 1–20.

[CSF+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Internet Request for Comments, May 2008, RFC 5280.

[Dam89] Ivan Damgård, *A Design Principle for Hash Functions*, CRYPTO (Gilles Brassard, ed.), Lecture Notes in Computer Science, vol. 435, Springer, 1989, pp. 416–427.

[Dau05] Magnus Daum, *Cryptanalysis of Hash Functions of the MD4-Family*, Ph.D. thesis, Ruhr-Universität Bochum, May 2005.

[dBB91] Bert den Boer and Antoon Bosselaers, *An Attack on the Last Two Rounds of MD4*, CRYPTO (Joan Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1991, pp. 194–203.

[dBB93] Bert den Boer and Antoon Bosselaers, *Collisions for the Compression Function of MD5*, EUROCRYPT, 1993, pp. 293–304.

[DBP96] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, *RIPEMD-160: A Strengthened Version of RIPEMD*, FSE (Dieter Gollmann, ed.), Lecture Notes in Computer Science, vol. 1039, Springer, 1996, pp. 71–82.

[DH76] Whitfield Diffie and Martin E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory **IT-22** (1976), no. 6, 644–654.

[DL05] Magnus Daum and Stefan Lucks, *Attacking Hash Functions by Poisoned Messages, "The Story of Alice and her Boss"*, June 2005, `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/`.

[Dob96] Hans Dobbertin, *Cryptanalysis of MD5 Compress*, 1996, In Rump Session of EuroCrypt '96.

[Dob98] Hans Dobbertin, *Cryptanalysis of MD4*, J. Cryptology **11** (1998), no. 4, 253–271.

[DP80] D. W. Davies and W. L. Price, *The Application of Digital Signatures Based on Public-Key Cryptosystems*, Proc. Fifth Intl. Computer Communications Conference, October 1980, pp. 525–530.

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker, *The Skein Hash Function Family*, Submission to NIST (Round 3), 2010.

[GIS05] M. Gebhardt, G. Illies, and W. Schindler, *A Note on Practical Value of Single Hash Collisions for Special File Formats*, NIST First Cryptographic Hash Workshop, October 2005.

[GKM+11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen, *Grøstl – a SHA-3 candidate*, Submission to NIST (Round 3), 2011.

[Gre10] E. A. Grechnikov, *Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics*, Cryptology ePrint Archive, Report 2010/413, 2010.

[HC] *HashClash project webpage*, `http://code.google.com/p/hashclash`.

[HK06] Shai Halevi and Hugo Krawczyk, *Strengthening Digital Signatures Via Randomized Hashing*, CRYPTO (Cynthia Dwork, ed.), Lecture Notes in Computer Science, vol. 4117, Springer, 2006, pp. 41–59.

[HPR04] Philip Hawkes, Michael Paddon, and Gregory G. Rose, *Musings on the Wang et al. MD5 Collision*, Cryptology ePrint Archive, Report 2004/264, 2004.

[HS05] Paul Hoffman and Bruce Schneier, *Attacks on Cryptographic Hashes in Internet Protocols*, Internet Request for Comments, November 2005, RFC 4270.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby, *Pseudo-random Generation from one-way functions (Extended Abstracts)*, STOC, ACM, 1989, pp. 12–24.

[Jou04] Antoine Joux, *Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions*, CRYPTO (Matthew K. Franklin, ed.), Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 306–316.

[JP05] Charanjit S. Jutla and Anindya C. Patthak, *A Matching Lower Bound on the Minimum Weight of SHA-1 Expansion Code*, Cryptology ePrint Archive, Report 2005/266, 2005.

[Kam04] Dan Kaminsky, *MD5 To Be Considered Harmful Someday*, Cryptology ePrint Archive, Report 2004/357, 2004.

[KG07] Neil Koblitz and Oded Goldreich, 2007, See `http://www.sigcrap.org/2007/09/14/the-koblitz-controversy/`.

[KK06] John Kelsey and Tadayoshi Kohno, *Herding Hash Functions and the Nostradamus Attack*, EUROCRYPT (Serge Vaudenay, ed.), Lecture Notes in Computer Science, vol. 4004, Springer, 2006, pp. 183–200.

[Kli05] Vlastimil Klima, *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*, Cryptology ePrint Archive, Report 2005/102, 2005.

[Kli06] Vlastimil Klima, *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, Cryptology ePrint Archive, Report 2006/105, 2006.

[KS05] John Kelsey and Bruce Schneier, *Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work*, EUROCRYPT (Ronald Cramer, ed.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 474–490.

[LdW05] Arjen K. Lenstra and Benne de Weger, *On the Possibility of Constructing Meaningful Hash Collisions for Public Keys*, ACISP (Colin Boyd and Juan Manuel González Nieto, eds.), Lecture Notes in Computer Science, vol. 3574, Springer, 2005, pp. 267–279.

[Lin98] J. H. Van Lint, *Introduction to Coding Theory*, 3rd ed., Graduate Texts in Mathematics, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[LL05] Jie Liang and Xuejia Lai, *Improved Collision Attack on Hash Function MD5*, Cryptology ePrint Archive, Report 2005/425, 2005.

[Man08] Stephane Manuel, *Classification and Generation of Disturbance Vectors for Collision Attacks against SHA-1*, Cryptology ePrint Archive, Report 2008/469, 2008.

[Man11] Stéphane Manuel, *Classification and generation of disturbance vectors for collision attacks against SHA-1*, Des. Codes Cryptography **59** (2011), no. 1-3, 247–263.

[Mer78] Ralph C. Merkle, *Secure Communications Over Insecure Channels*, Commun. ACM **21** (1978), no. 4, 294–299.

[Mer82] Ralph C. Merkle, *Secrecy, Authentication, and Public Key Systems*, UMI Research press, 1982.

[Mer89] Ralph C. Merkle, *One Way Hash Functions and DES*, CRYPTO (Gilles Brassard, ed.), Lecture Notes in Computer Science, vol. 435, Springer, 1989, pp. 428–446.

[MHP09] Cameron McDonald, Philip Hawkes, and Josef Pieprzyk, *Differential Path for SHA-1 with complexity $O(2^{52})$*, Cryptology ePrint Archive, Report 2009/259, 2009.

[Mik04] Ondrej Mikle, *Practical Attacks on Digital Signatures Using MD5 Message Digest*, Cryptology ePrint Archive, Report 2004/356, 2004.

[MP05] Krystian Matusiewicz and Josef Pieprzyk, *Finding Good Differential Patterns for Attacks on SHA-1*, WCC (Øyvind Ytrehus, ed.), Lecture Notes in Computer Science, vol. 3969, Springer, 2005, pp. 164–177.

[MP08] Stéphane Manuel and Thomas Peyrin, *Collisions on SHA-0 in One Hour*, FSE (Kaisa Nyberg, ed.), Lecture Notes in Computer Science, vol. 5086, Springer, 2008, pp. 16–35.

[MPRR06] Florian Mendel, Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, *The Impact of Carries on the Complexity of Collision Attacks on SHA-1*, FSE (Matthew J. B. Robshaw, ed.), Lecture Notes in Computer Science, vol. 4047, Springer, 2006, pp. 278–292.

[MRR07] Florian Mendel, Christian Rechberger, and Vincent Rijmen, *Update on SHA-1*, Rump session of CRYPTO 2007, 2007.

[MS06] James A. Muir and Douglas R. Stinson, *Minimality and other properties of the width- nonadjacent form*, Math. Comput. **75** (2006), no. 253, 369–384.

[NIS93] National Institute of Standards and Technology NIST, *FIPS PUB 180: Secure Hash Standard*, 1993.

[NIS95] National Institute of Standards and Technology NIST, *FIPS PUB 180-1: Secure Hash Standard*, 1995.

[NIS02] National Institute of Standards and Technology NIST, *FIPS PUB 180-2: Secure Hash Standard*, 2002.

[NIS07] National Institute of Standards and Technology NIST, *Announcing Request for Candidate Algorithm Nominations for a new Cryptographic Hash Algorithm (SHA-3) Family*, Federal Register Vol. 72, No. 212, November 2007, Available at `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf`.

[NIS08] National Institute of Standards and Technology NIST, *FIPS PUB 180-3: Secure Hash Standard*, 2008.

[NIS11] National Institute of Standards and Technology NIST, *Draft FIPS PUB 180-4: Secure Hash Standard*, 2011.

[NSS+06] Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Jun Yajima, Noboru Kunihiro, and Kazuo Ohta, *Improved Collision Search for SHA-0*, ASIACRYPT (Xuejia Lai and Kefei Chen, eds.), Lecture Notes in Computer Science, vol. 4284, Springer, 2006, pp. 21–36.

[PCTH11] T. Polk, L. Chen, S. Turner, and P. Hoffman, *Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms*, Internet Request for Comments, March 2011, RFC 6194.

[PD05] Robert Primmer and Carl D'Halluin, *Collision and Preimage Resistance of the Centera Content Address*, Technical Report, 2005, EMC Corporation.

[PRR05] Norbert Pramstaller, Christian Rechberger, and Vincent Rijmen, *Exploiting Coding Theory for Collision Attacks on SHA-1*, IMA Int. Conf. (Nigel P. Smart, ed.), Lecture Notes in Computer Science, vol. 3796, Springer, 2005, pp. 78–95.

[Rab78] M. O. Rabin, *Digitalized Signatures*, Foundations of Secure Computation (Richard A. DeMillo, David P. Dobkin, Anita K. Jones, and Richard J. Lipton, eds.), Academic Press, 1978, pp. 155–168.

[Riv90a] Ronald L. Rivest, *The MD4 Message Digest Algorithm*, CRYPTO (Alfred Menezes and Scott A. Vanstone, eds.), Lecture Notes in Computer Science, vol. 537, Springer, 1990, pp. 303–311.

[Riv90b] Ronald L. Rivest, *The MD4 Message-Digest Algorithm*, Internet Request for Comments, October 1990, RFC 1186; obsoleted by RFC 1320.

[Riv92] Ronald L. Rivest, *The MD5 Message-Digest Algorithm*, Internet Request for Comments, April 1992, RFC 1321.

[RO05] Vincent Rijmen and Elisabeth Oswald, *Update on SHA-1*, CT-RSA (Alfred Menezes, ed.), Lecture Notes in Computer Science, vol. 3376, Springer, 2005, pp. 58–71.

[Rog06] Phillip Rogaway, *Formalizing Human Ignorance*, VIETCRYPT (Phong Q. Nguyen, ed.), Lecture Notes in Computer Science, vol. 4341, Springer, 2006, pp. 211–228.

[Rom90] John Rompel, *One-Way Functions are Necessary and Sufficient for Secure Signatures*, STOC, ACM, 1990, pp. 387–394.

[RS04] Phillip Rogaway and Thomas Shrimpton, *Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance*, FSE (Bimal K. Roy and Willi Meier, eds.), Lecture Notes in Computer Science, vol. 3017, Springer, 2004, pp. 371–388.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Commun. ACM **21** (1978), no. 2, 120–126.

[Sel06] Peter Selinger, 2006, `http://www.mathstat.dal.ca/~selinger/md5collision/`.

[Sha48] Claude E. Shannon, *A Mathematical Theory of Communication*, Bell System Technical Journal **27** (1948), 379–423, 623–656.

[Sha49] Claude E. Shannon, *Communication Theory of Secrecy Systems*, Bell System Technical Journal **28** (1949), 657–715.

[SLdW07a] Marc Stevens, Arjen Lenstra, and Benne de Weger, 2007, `http://www.win.tue.nl/hashclash/Nostradamus/`.

[SLdW07b] Marc Stevens, Arjen Lenstra, and Benne de Weger, 2007, `http://www.win.tue.nl/hashclash/SoftIntCodeSign/`.

[SLdW07c] Marc Stevens, Arjen K. Lenstra, and Benne de Weger, *Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*, EUROCRYPT (Moni Naor, ed.), Lecture Notes in Computer Science, vol. 4515, Springer, 2007, pp. 1–22.

[SLdW12] Marc Stevens, Arjen Lenstra, and Benne de Weger, *Chosen-Prefix Collisions for MD5 and Applications*, To appear in International Journal of Applied Cryptology (IJACT), 2012.

[SNKO05] Yu Sasaki, Yusuke Naito, Noboru Kunihiro, and Kazuo Ohta, *Improved Collision Attack on MD5*, Cryptology ePrint Archive, Report 2005/400, 2005.

[SSA+09a] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger, 2009, `http://www.win.tue.nl/hashclash/rogue-ca/`.

[SSA+09b] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger, *Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate*, CRYPTO (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 55–69.

[Ste06] Marc Stevens, *Fast Collision Attack on MD5*, Cryptology ePrint Archive, Report 2006/104, 2006.

[Ste09] Didier Stevens, 2009, `http://blog.didierstevens.com/2009/01/17/`.

[Ste10] Marc Stevens, *SHA-1 near collision attack source code*, Nov 2010, `http://code.google.com/p/hashclash`.

[VJBT08] Jirí Vábek, Daniel Joscák, Milan Bohácek, and Jirí Tuma, *A New Type of 2-Block Collisions in MD5*, INDOCRYPT (Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, eds.), Lecture Notes in Computer Science, vol. 5365, Springer, 2008, pp. 78–90.

[vOW99] Paul C. van Oorschot and Michael J. Wiener, *Parallel Collision Search with Cryptanalytic Applications*, J. Cryptology **12** (1999), no. 1, 1–28.

[Wan06] Xiaoyun Wang, *Cryptanalysis of Hash Functions and Potential Dangers*, RSA Conference 2006 invited talk, 2006.

[WFLY04] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, Cryptology ePrint Archive, Report 2004/199, 2004.

[Wu11] Hongjun Wu, *The Hash Function JH*, Submission to NIST (Round 3), 2011.

[WY05] Xiaoyun Wang and Hongbo Yu, *How to Break MD5 and Other Hash Functions*, EUROCRYPT (Ronald Cramer, ed.), Lecture Notes in Computer Science, vol. 3494, Springer, 2005, pp. 19–35.

[WYY05a] Xiaoyun Wang, Andrew C. Yao, and Frances Yao, *Cryptanalysis on SHA-1*, NIST Cryptographic Hash Workshop, 2005, `http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf`.

[WYY05b] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, *Finding Collisions in the Full SHA-1*, CRYPTO (Victor Shoup, ed.), Lecture Notes in Computer Science, vol. 3621, Springer, 2005, pp. 17–36.

[WYY05c] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin, *Efficient Collision Search Attacks on SHA-0*, CRYPTO (Victor Shoup, ed.), Lecture Notes in Computer Science, vol. 3621, Springer, 2005, pp. 1–16.

[XF09] Tao Xie and Dengguo Feng, *How To Find Weak Input Differences For MD5 Collision Attacks*, Cryptology ePrint Archive, Report 2009/223, 2009.

[XF10] Tao Xie and Deng Guo Feng, *Construct MD5 Collisions Using Just A Single Block Of Message*, Cryptology ePrint Archive, Report 2010/643, 2010.

[XFL08] Tao Xie, DengGuo Feng, and FanBao Liu, *A New Collision Differential For MD5 With Its Full Differential Path*, Cryptology ePrint Archive, Report 2008/230, 2008.

[XLF08] Tao Xie, Fan Bao Liu, and Deng Guo Feng, *Could The 1-MSB Input Difference Be The Fastest Collision Attack For MD5?*, Cryptology ePrint Archive, Report 2008/391, 2008.

[YIN+08] Jun Yajima, Terutoshi Iwasaki, Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Noboru Kunihiro, and Kazuo Ohta, *A strict evaluation method on the number of conditions for the SHA-1 collision search*, ASIACCS (Masayuki Abe and Virgil D. Gligor, eds.), ACM, 2008, pp. 10–20.

[YS05] Jun Yajima and Takeshi Shimoyama, *Wang's sufficient conditions of MD5 are not sufficient*, Cryptology ePrint Archive, Report 2005/263, 2005.

[YSN+07] Jun Yajima, Yu Sasaki, Yusuke Naito, Terutoshi Iwasaki, Takeshi Shimoyama, Noboru Kunihiro, and Kazuo Ohta, *A New Strategy for Finding a Differential Path of SHA-1*, ACISP (Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, eds.), Lecture Notes in Computer Science, vol. 4586, Springer, 2007, pp. 45–58.

[Yuv79] G. Yuval, *How to Swindle Rabin*, Cryptologia **3** (1979), 187–189.

Index

Notation

$\texttt{01}_2$	Binary notation	17
$\texttt{0f}_{16}$	Hexadecimal notation	17
$\overline{b}$	Bit at position b is -1 in BSDR notation	19
$X \wedge Y$	Bitwise AND	17, 63
$\overline{X}$	Bitwise negation	17, 63
$X \vee Y$	Bitwise OR	17, 63
$X \oplus Y$	Bitwise XOR	17, 63
$RL(X, n)$	Bitwise cyclic left rotation	18, 19, 63, 64
$RR(X, n)$	Bitwise cyclic right rotation	18, 19, 63, 64
$dRL(X, n)$	Bitwise cyclic left rotation of differences	74
ΔX	Bitwise difference	19, 74, 176
δX	Modular difference	19, 74, 176
$X[i]$	Bit selection	18, 19, 63, 64
$X + Y$	Modular addition	18, 63
$X - Y$	Modular subtraction	18, 63
X'	Related variable	19, 74
$\sigma(X)$	Mapping from $\{-1, 0, 1\}^N$ to $\mathbb{Z}_{2^N}$	19, 64
$w \hookrightarrow v$	Substitution rule	152
$w(X)$	Weight of (signed-)bit string	18, 19, 63, 64

List of Algorithms

List of Figures

List of Tables

List of Theorems

Nederlandse samenvatting

Cryptografische hashfuncties, of simpelweg hashfuncties, zijn een van de belangrijkste bouwstenen binnen de cryptografie. Een hashfunctie is een algoritme dat voor elk willekeurig bericht een korte hash-waarde van een vast aantal bits (bv. 256 bits) berekent. Zulke hash-waarden worden gebruikt als een soort van vingerafdruk om het originele bericht mee te identificeren. Een belangrijke toepassing van hashfuncties is in digitale handtekeningen, waardoor documenten en software ondertekend kunnen worden en digitale communicatie beveiligd wordt.

Een belangrijke veiligheidseis is dat het praktisch onmogelijk moet zijn om collisions (twee berichten met dezelfde hash-waarde) te vinden. Een hashfunctie is gebroken wanneer er een efficiëntere aanval om collisions te vinden bestaat dan de generieke aanval gebaseerd op de birthday paradox.

Rond 2005 hebben Wang en coauteurs MD5 en SHA-1, twee belangrijke internationale cryptografische hashfunctie standaarden, gebroken met zogenaamde identical-prefix collision aanvallen. Niettemin, was de software industrie initieel sceptisch over de gevaren in de praktijk van deze eerste aanvallen op MD5 en SHA-1 omdat deze aanvallen technisch beperkt waren.

Dit proefschrift omhelst de volgende vijftal bijdragen op het gebied van de analyse van zwakheden van internationale cryptografische hashfunctie standaarden.

Ten eerste het verfijnen van de exacte differentiële analyse die ten grondslag ligt van bovenstaande collision aanvallen. En dan met name het introduceren van algoritmische methoden voor het vinden van de benodigde zogenaamde differentiële paden voor een grote klasse van hashfuncties die MD5 en SHA-1 omhelst, met specifieke verbeteringen toegespitst op MD5 en SHA-1.

Ten tweede het construeren van efficiëntere en met name flexibelere aanvallen op MD5. In het bijzonder introduceren we de zogenaamde chosen-prefix collision aanval op MD5, waarin een grote technische beperking van identical-prefix collision aanvallen geëlimineerd wordt: namelijk de eis dat de twee botsende bestanden volledig identiek zijn tot aan de 128 bytes die gegenereerd worden in de identical-prefix collision aanval en die tot de botsing lijden. Chosen-prefix collision aanvallen staan daarmee significant meer praktische aanvallen toe op cryptografische systemen.

De derde bijdrage is dan ook het aantonen van het gevaar van collision aanvallen op huidige cryptografische systemen met het doel de industrie zodanig te informeren dat het tijdig en op gepaste wijze zwakke hashfuncties kan vervangen door veiligere hashfuncties. Hoewel destijds het gevaar van deze verbeterde en efficiëntere aanvallen door de industrie nog enigszins onderschat werd, hebben wij in 2009 bewezen dat MD5's zwakheden wel degelijk een groot gevaar opleveren door een MD5-gebaseerde handtekening van een wereldwijd vertrouwde Certification Authority te gebruiken om zelf een vertrouwde Certification Authority te creëren. Met de geconstrueerde ver-

trouwde Certification Authority zouden we in principe – indien we deze niet bij voorbaat nietig hadden geconstrueerd – beveiligde digitale communicatie met beveiligde websites kunnen corrumperen op zodanige wijze dat webbrowsers de gecorrumpeerde digitale communicatie alsnog als veilig bestempelen ten overstaan van de gebruiker. Ons beoogde doel hiermee werd snel behaald: MD5 werd een verboden hashfunctie voor Certification Authorities.

De vierde bijdrage is het introduceren van nieuwe exacte differentiële analytische methoden voor SHA-1 en het construeren van identical-prefix en chosen-prefix collision aanvallen op SHA-1. Echter in tegenstelling tot MD5 zijn deze aanvallen niet praktisch uitvoerbaar vanwege de belemmerende hoge computationele kosten. Sinds de eerste theoretische aanval op SHA-1 door Xiaoyun Wang en coauteurs zijn er meerdere efficiëntere aanvallen geclaimd. Echter de claims die later wel gesubstantieerd zijn, blijken niet volledig correct te zijn wat zelfs geleid heeft tot terugtrekking van een zekere publicatie. Ten einde de correctheid van onze aanvallen op SHA-1 aan te tonen, alsmede een beter begrip en verdere verbeteringen van onze aanvallen mogelijk te maken, hebben wij als eerste ook de daadwerkelijke implementatie van onze aanvallen en onze analytische algoritmen gepubliceerd.

Voor een collision aanval zijn bepaalde vergelijkingen nodig die de zoekruimte voor een collision beperkt tot een specifieke ruimte waarin een bekende succeskans behaald kan worden, huidige methoden bepalen dit benodigde systeem van vergelijkingen en een schatting van de bijbehorende succeskans op basis van heuristieken. Onze aanvallen op SHA-1 zijn gebaseerd op een nieuwe exacte analyse welke sterk leunt op computationele combinatoriek en die in tegenstelling tot huidige methoden de interactie tussen zogenaamde local collisions op een exacte en uitputtende manier kan analyseren. Met behulp van onze nieuwe analyse is het mogelijk om voor alle geldige systemen van vergelijkingen exact te bepalen wat de bijbehorende succeskans is en daarmee hét systeem van vergelijkingen te kiezen dat leidt tot de hoogste succeskans en tevens de meeste vrijheidsgraden toelaat.

De laatste bijdrage is het introduceren van een efficiënte methode die cryptografische systemen kan beschermen tegen potentiële kwaadwillenden die gebruik maken van bovenstaande collision aanvallen. Hoewel zwakke hashfuncties zoals MD5 en SHA-1 dienen vervangen te worden door veiligere hashfuncties, lijken er een aantal praktische belemmeringen te zijn. Ten einde cryptografische systemen die gebruik maken van MD5 en/of SHA-1 veilig te houden tot aan die tijd dat MD5 en SHA-1 vervangen zijn, hebben wij een efficiënte methode geïntroduceerd die kan detecteren of dat bestanden en/of digitale communicatie met bekende collision aanvallen geconstrueerd zijn en daardoor het verder verwerken van boosaardige bestanden en/of digitale communicatie kan blokkeren.

Acknowledgments

This PhD thesis would not be the same without the influence of many people whom I would like to thank here.

First of all, I would like to thank my supervisor Ronald Cramer for our many interactions and discussions that have improved my skills as a researcher and for letting me pursue a line of research different from his own that has led to this thesis.

Many thanks go to Arjen Lenstra and Benne de Weger for our many discussions, their many helpful suggestions and in writing our papers.

Also, I would like to thank the other members of my defense committee for taking the time to read this thesis and for their insightful comments: Eli Biham, Berry Schoenmakers, Peter Stevenhagen and Xiaoyun Wang. Special thanks go to Eli Biham and Xiaoyun Wang who have taken extra efforts to be able to attend the defense in Leiden and their contributions to the preceding RISC seminar.

Thanks to all the people of CWI who have helped with public relations and legal counseling concerning the publication of our rogue Certification Authority certificate.

I am very grateful to all my colleagues, family and friends for their continuing support and confidence in me during these years.

And finally Lisanne: thank you for all your support and understanding and enriching my life together with our future child.

Marc Stevens
Curriculum Vitae

Personal details

First name, surname:	Marc Stevens
Date and place of birth:	April 7, 1981, Hellevoetsluis
Nationality:	Netherlands
E-mail:	marc@marc-stevens.nl

Degrees

PhD

Title of thesis:	'Attacks on hash functions and applications'
University:	Mathematical Institute, Leiden University
Advisors:	prof. dr. Ronald Cramer, prof. dr. Arjen Lenstra (EPFL) *&* dr. Benne de Weger (TU/e)
Date of defense:	*expected June 19, 2012*

MSc

Title of thesis:	'On collisions for MD5'
University:	Faculty of Mathematics and Computer Science, Eindhoven University of Technology
Advisors:	prof. dr. Henk van Tilborg, dr. Benne de Weger *&* Gido Schmitz MSc (NBV, Dutch national communications security agency)
Date of defense:	August 28, 2007

Honors, awards and prizes

- Winner of the $10,000-challenge to construct a single-block collision for MD5 posted by Tao Xie and Dengguo Feng: `http://eprint.iacr.org/2010/643`
- Nominated for *Discoverer of the Year 2011*, Faculty of Science, Leiden University, January 2012
- *CRYPTO 2009 – Best Paper Award*

- *Eindhoven University of Technology – Afstudeerprijs 2008*
  (best master's thesis university-wide)
- Nominated for *Joop Bautz Information Security Award 2007* (best contribution to dutch information security branch, PvIB, ISACA, NOREA, `http://www.jbisa.nl`)
- Graduated Applied Mathematics *cum laude* (2007)

Publications

- Marc Stevens, *Exact joint local-collision analysis & new collision attacks for SHA-1*, submitted to CRYPTO 2012.
- Marc Stevens, *Single-block collision attack on MD5*, Cryptology ePrint Archive, Report 2012/040, 2012. Winner of the $10,000-challenge posted by Tao Xie and Dengguo Feng: `http://eprint.iacr.org/2010/643`.
- Marc Stevens, Arjen K. Lenstra and Benne de Weger, *Chosen-prefix Collisions for MD5 and Applications*, to appear in: International Journal of Applied Cryptology (IJACT).
- Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger, *Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate*, CRYPTO (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 55–69.
- Marc Stevens, Arjen K. Lenstra, and Benne de Weger, *Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities*, EUROCRYPT (Moni Naor, ed.), Lecture Notes in Computer Science, vol. 4515, Springer, 2007, pp. 1–22.
- Marc Stevens, *Fast Collision Attack on MD5*, Cryptology ePrint Archive, 2006, Report 2006/104.
- Tanja Lange and Marc Stevens, *Efficient Doubling on Genus Two Curves over Binary Fields*, Selected Areas in Cryptography (Helena Handschuh and M. Anwar Hasan, eds.), Lecture Notes in Computer Science, vol. 3357, Springer, 2004, pp. 170–181.

Invited Talks (selected)

- Keynote address, SHARCS 2012, Washington DC, March 2012, *Cryptanalysis of MD5 and SHA-1*.

Software

- Marc Stevens, *HashClash project*, an open-source C++ framework for MD5 & SHA-1 differential path construction and chosen-prefix collisions for MD5, 2009–2011, `http://code.google.com/p/hashclash`.