Reverse-Engineering of the Cryptanalytic Attack Used in the Flame Super-Malware

Max Fillinger(✉) and Marc Stevens

CWI, Amsterdam, The Netherlands
max.fillinger@cwi.nl, marc@marc-stevens.nl

Abstract. In May 2012, a highly advanced malware for espionage dubbed Flame was found targeting the Middle-East. As it turned out, it used a forged signature to infect Windows machines by MITM-ing Windows Update. Using counter-cryptanalysis, Stevens found that the forged signature was made possible by a chosen-prefix attack on MD5 [25]. He uncovered some details that prove that this attack differs from collision attacks in the public literature, yet many questions about techniques and complexity remained unanswered.

In this paper, we demonstrate that significantly more information can be deduced from the example collision. Namely, that these details are actually sufficient to reconstruct the collision attack to a great extent using some weak logical assumptions. In particular, we contribute an analysis of the differential path family for each of the four near-collision blocks, the chaining value differences elimination procedure and a complexity analysis of the near-collision block attacks and the associated birthday search for various parameter choices. Furthermore, we were able to prove a lower-bound for the attack's complexity.

This reverse-engineering of a non-academic cryptanalytic attack exploited in the real world seems to be without precedent. As it allegedly was developed by some nation-state(s) [11,12,19], we discuss potential insights to their cryptanalytic knowledge and capabilities.

Keywords: MD5 · Hash function · Cryptanalysis · Reverse engineering · Signature forgery

1 Introduction

1.1 End-of-Life of a Cryptographic Primitive

The end-of-life of a widely-used cryptographic primitive is an uncommon event, preferably orchestrated in an organized fashion by replacing it with a next generation primitive as a precaution as soon as any kind of weakness has been exposed. Occasionally such idealistic precautions are thrown to the wind for various reasons. Unfortunately, the sudden introduction of practical attacks may then seriously reduce the security of systems protected by the cryptographic primitive. The ensuing forced mitigation efforts need to overcome important hurdles in a

T. Iwata and J.H. Cheon (Eds.): ASIACRYPT 2015, Part II, LNCS 9453, pp. 586–611, 2015.
DOI: 10.1007/978-3-662-48800-3_24

short amount of time and thus prove to be less successful than precautionary mitigation efforts. The topic of this paper, namely an exposed cryptanalytic attack on the hash function MD5 exploited in the real-world eight years after the first practical break of MD5, is a recent example of the above.

1.2 Collisions for MD5

The cryptographic hash function MD5 found widespread use for many years since its inception in 1991 by Ron Rivest [21]. It became the de facto industry standard in combination with RSA to generate digital signatures upon which our Internet's Public Key Infrastructure (PKI) for TLS/SSL has been build. This despite early collision attacks on the compression function of MD5 by den Boer and Bosselaers [2] and Dobbertin [6].

That changed after in 2004 the first real MD5 collision attack, as well as example collisions, were presented by Wang et al. in a major breakthrough in hash function cryptanalysis [28,29]. Improvements to their attack were published in a series of papers (e.g., see [9,10,13,22,24,27,30,31]). Unfortunately, no convincing threatening scenarios arose due to the important restriction that colliding message pairs $M = P||C||S$, $M' = P||C'||S$ can only differ in the random-looking C, C'.

This restriction was lifted with the introduction of the first chosen-prefix collision attack on MD5 [26] that for any two equal-length prefixes P and P' constructs short random-looking C and C' such that $P||C||S$ and $P'||C'||S$ collide for any common suffix S. Chosen-prefix collisions make it significantly easier to construct collisions with meaningful differences, i.e., often it suffices to choose M and M' appropriately and to hide C and C' somewhere within the messages. It enabled the first truly convincing attack scenario using MD5 collisions, namely the construction of a rogue Certificate Authority (CA) certificate presented in 2009 [27]. As it turned out, many CAs had voluntarily stopped using MD5. Nevertheless, the remaining few MD5-using CAs endangered the entire PKI as any PKI is only as strong as its weakest link, i.e., CA.

Based on these developments, various authorities explicitly disallowed MD5 in digital signatures (e.g., The CA/Browser Forum adopted Baseline Requirements for CAs in 2011[1], Microsoft updated its Root CA Program in 2009[2]).

1.2.1 Counter-Cryptanalysis

Due to its widespread and pervasive use, MD5 remains supported to accommodate old signatures even up to the time of this writing. Any party world-wide still signing with an MD5-based digital signature scheme – against all advice – may be attacked using a chosen-prefix collision attack. Furthermore, a resulting digital signature forgery can be exploited against nearly everyone due to the near-ubiquitous support of MD5-based signatures. Stevens recently proposed

[1] https://cabforum.org/wp-content/uploads/Baseline_Requirements_V1.pdf
[2] http://technet.microsoft.com/en-us/library/cc751157.aspx

to counter these threats using *counter-cryptanalysis* [25], specifically a collision detection algorithm, i.e., an algorithm that asserts whether any given *single* message belongs to a colliding message pair that was constructed using a MD5 and/or SHA-1 collision attack. The main idea is to guess the colliding part (i.e., the C') of the assumed sibling colliding message and to verify whether an internal collision occurs. Once a collision has been verified, one knows the near-collision blocks for both messages, however, one cannot reconstruct earlier parts of the missing message with counter-cryptanalysis.

Collision detection can strengthen digital signatures by invalidating forged digital signatures, thereby allowing the continued secure use of MD5-based signatures. However, collision detection is clearly not a permanent solution and cannot replace proper migration to the more secure SHA-2 and SHA-3.

1.3 The Super-Malware 'Flame'

1.3.1 Background

Flame is a highly advanced malware for espionage and was discovered in May 2012 by the Iranian CERT, Kaspersky Lab and CrySyS Lab [11,12]. It seemed to have targeted the Middle-East, with the most infections in Iran. Among the targets were government-related organizations, private companies, educational institutions as well as specific individuals. According to these reports by malware experts Flame was developed by some nation-state(s) with near-certainty. It seems the best report so far on the origin is a Washington Post article reporting that – according to unnamed officials and experts – Flame was a joint U.S.-Israel classified effort [19].

For espionage, Flame collected keyboard inputs, Skype conversations and local documents of potential interest. It could also record screen contents, microphone audio, webcam video as well as network traffic, sometimes triggered by the use of specific applications of interest like Instant Messaging applications.

According to Kaspersky [12], Flame was active since at least 2010. However, CrySyS Lab reports Flame or a preliminary version thereof may have been active since 2007 due to an observed file in the security enterprise webroot in 2007. Infections seem to have occurred with surgical precision with each target carefully selected instead of wildly spreading, which may be one of the reasons why it has evaded discovery for several years.

We refer to the analyses by Kaspersky Lab and CrySyS Lab [11,12] for more details on the functionality, purpose and origin of Flame. Here we focus on the variant chosen-prefix collision attack that enabled its propagation.

1.3.2 Propagation

As described by Sotirov [23], Flame used WPAD (Web Proxy Auto-Discovery Protocol) to register itself as a proxy for the domain `update.windows.com` to launch Man-In-The-Middle attacks for Windows Update on other computers on the local network. By forcing a fall-back from the secure HTTPS protocol to the insecure HTTP protocol, Flame was able to push validly signed Windows

Update patches of its choice. This included a properly, but illegitimately, signed Windows Update patch by which Flame could spread to other machines. Flame's code-signing certificate for this security patch was obtained by fooling a certain part of Microsoft's PKI into signing a colliding – innocuous-looking – sibling certificate using an MD5-based signature algorithm. As the to-be-signed parts of both certificates were carefully crafted to result in the same MD5-hash using a chosen-prefix collision attack, the MD5-based signature was valid for both certificates.

Even though Microsoft was fully aware of the severe weaknesses of MD5 and spent great effort on migrating to more secure hash functions for *new* digital signatures at least since 2008, their software continued to accept (old) MD5-based digital signatures. Unfortunately, the use of MD5-based signatures for licensing purposes in their Terminal Server Licensing Service was overlooked in their efforts.[3] This, together with other unforeseen circumstances, allowed the creation of Flame's properly, but illegitimately, signed security patch that was trusted by *all* versions of Windows [16].[4]

1.3.3 Unknown Variant Chosen-Prefix Collision Attack

On the 3rd of June 2012, Microsoft blogged that in their initial analysis of Flame they "*identified that an older cryptography algorithm could be exploited and then be used to sign code as if it originated from Microsoft*" [17]. An immediate guess was that this cryptically worded statement refers to the construction of a rogue code-signing certificate using a chosen-prefix collision attack on MD5 similar to [27]. Only the certificates in the chain leading to the forged signature on Flame's executable were circulating on the Internet [20], its sibling colliding certificate remains lost. Using his collision detection technique, Stevens was able to reconstruct the collision part of the missing sibling colliding certificate [25].

Having both colliding parts one can observe the differential paths used for this attack which Stevens uses to provide a preliminary analysis of Flame's attack:

Flame's differential paths clearly show a chosen-prefix collision attack that starts with a chaining-value difference containing many bit differences that is gradually reduced to zero by the four sequential "near-collision" block pairs. However, these differential paths do not match any family of published chosen-prefix collision attacks [27], but instead were variants based on the first differential paths for MD5 by Wang et al.[29]. Also, they show characteristics that do not match those from known differential path construction methods for MD5. The author provides arguments indicating an unnecessary costly differential path construction method was used. Furthermore, experimental results were given constructing replacement paths with significantly fewer bitconditions in only about 15 s on average on a single Intel i7-2600 CPU (equivalent to about 2^{29} MD5 compressions).

[3] Microsoft invalidated this part of their PKI after the discovery of Flame in 2012.

[4] Any license certificate produced by the Terminal Server Licensing Service could directly be used to attack Windows Vista and earlier versions, but not later versions.

Based on the differential paths and the observation that the best known message modification technique was used, for each block a lower bound for the average complexity to find the near-collision blocks is given. Note the implicit assumption that the differential path including the target output chaining value difference is fixed before the near-collision block search.

Based on the weight of the observed chaining value difference after the birthday search that need to be eliminated by the four near-collision attacks, an indicative complexity estimate of about 2^{42} MD5 compressions is given. Although further constraints make it more likely to be even higher instead of lower. Lacking a more detailed analysis of the chaining value difference elimination strategy, no more accurate prediction could be given.

Although Stevens was able to show a yet unknown variant attack was used, so far, no reconstruction of Flame's attack has been presented and many questions regarding techniques and complexities remained unanswered. Specifically there is no analysis so far for the possible differential path family for each block, and therefore for the chaining value reduction procedure that selects which chaining value differences (the tail of the differential path) to eliminate in each block. This in turn makes it hard to provide accurate complexity estimates for each of the four near-collision attacks as well as for the associated birthday attack. Furthermore, the work in this paper makes it clear that Stevens' assumption that each near-collision block targets a specific chaining value difference is inaccurate, making his preliminary comments on the attack complexity incorrect.

2 Our Contributions

In [25] Stevens presented proof that Flame uses a yet unknown chosen-prefix collision attack and made indications of the complexity to find solutions for the recovered differential paths. No attack reconstruction or more accurate complexity estimates were given.

Our paper is entirely based on the four near-collision block pairs shown in Appendix B that can be recovered from the single available certificate in Flame's attack using counter-cryptanalysis. This paper significantly improves upon Stevens' preliminary reconstruction and we demonstrate for the first time that a single example of a collision pair is actually sufficient to reconstruct the used collision attack to a great extent under some weak logical assumptions. Furthermore, the high level of detail of our reconstruction even admits concrete conclusions under a complexity analysis, specifically we prove a lower-bound for the estimated attack complexity and provide a cost figure for the closest fit of attack parameters. Our work shows that Stevens' indications of the near-collision costs are not the real expected costs. In particular the attack does not use fixed differential paths, but allows some random chaining value differences to occur in the first two blocks that can be efficiently negated in the last two blocks. Lacking more information about the near-collision attack procedures, Stevens was also unable to give real indications of the birthday search complexity of Flame's chosen-prefix collision attack. However, our reconstruction as well as our complexity analysis includes the birthday search and shows there is a trade-off between the birthday search cost and the total cost of the near-collision attacks.

At a high level we can draw some insights from our analysis into the cryptanalytic capabilities and the available resources of Flame's creators. In particular, the complexity for the closest fit of attack parameters is equivalent to $2^{49.3}$ MD5 compressions which takes roughly 40,000 CPUcore hours. That means for say 3-day attempts to succeed in reasonable time given the large number of required attempts, one needs about 560 CPUcores, which is large but not unreasonable even for academic research groups. With an estimated complexity of $2^{44.55}$ MD5 compressions from [27], this seems to be suboptimal. Not only the overall complexity seems to be suboptimal, also the differential path construction method and the near-collision speed-up techniques seem to be suboptimal. Overall we can report that it is clear that significant expertize in cryptanalysis was required, yet there are no indications at all of superior techniques, but instead that various parts are sub-optimal. It seems a working attack that succeeds in reasonable time was more important than optimizing the overall attack using all of the state of the art techniques[5].

Noteworthy is the following thought by an anonymous reviewer: developing a new variant attack required significant human effort which would have been unnecessary if its creators had enough computational power to do a general birthday search of complexity $2^{64.85}$ MD5 compressions in reasonable time. This may indicate a reasonable upper bound on available resources. Although, given the public availability of the Hashclash tools [8] since mid 2009, it might have been unnecessary in the first place which would imply they explicitly chose to build their attack or use their already built attack for Flame for some reasons.

At a more detailed level, our analysis revealed that a central idea behind the attack seems to be that the near-collision blocks operate in *pairs*: The first two blocks together eliminate one part of the intermediate hash value differential, allowing mostly random changes to other parts. The remaining differences (including the random changes from the first pair) are eliminated by the second two blocks. This idea allows a significant reduction in the expected complexity compared to the previous estimate by Stevens [25], where each near-collision pair was assumed to target specific intermediate hash value differences.

We have deduced the most likely parametrized family of differential paths for each near-collision block from which one is selected to eliminate specific intermediate hash value differences, as well as the complementary parametrized birthday search procedure that results in an intermediate hash value difference that can be eliminated using the 4 families of differential paths. We provide a complexity analysis for plausible parameter choices. Furthermore, we prove Theorem 6 stating a lower-bound complexity independent of parameter choices to be $2^{46.6}$ calls to the compression function in Sect. 4.3.3, and provide parameter

[5] At the time there seems to have been no reason to hold back more advanced techniques, given that counter-cryptanalysis was not publicly known then. Also, if there was a concern about revealing their knowledge then they could have easily used the publicly available Hashclash tools [8] instead.

choices that achieve this cost. Sotirov estimated that obtaining their forgery was significantly more difficult than the original Rogue CA construction, thus requiring many collisions in order to succeed [23]. This indicates that significant computational resources need to have been brought to bear to execute each chosen-prefix collision attack in a relatively short amount of time in order to succeed in their overall aim to obtain a forgery.

Lacking more examples or other hints about the actual attack procedure, it seems to be very hard to determine more specifics of Flame's chosen-prefix collision attack with any significant level of certainty. This includes the differential path construction algorithm and the collision search algorithm. For more details and analysis of less important aspects to our complexity analysis we refer to the full version of this paper.

The remainder of this paper is as follows. We start in Sect. 3 with an exposition of the main known techniques for chosen-prefix collision attacks. In Sect. 4.1, we break down the data from the recovered near-collision block pairs. We present our reconstruction in Sect. 4.2 and its complexity analysis in Sect. 4.3.

3 MD5 Chosen-Prefix Collision Attacks

3.1 MD5

The hash function MD5 maps an arbitrarily long input message M to a 128-bit output string. Its design follows the Merkle-Damgård construction [5,15], using a *compression function* which we call `MD5Compress` and a chaining value denoted IHV.

1. Unambiguously pad M to a length that is a multiple of 512.
2. For $i = 0, \ldots, N-1$, let M_i denote the ith 512-bit block of M. Let

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

3. For $i = 1, \ldots, N$, let $IHV_i = \texttt{MD5Compress}(IHV_{i-1}, M_{i-1})$.
4. Output IHV_N converted back from little-endian representation.

The description of `MD5Compress` we give here is not the standard one but an equivalent "unrolled" formulation [7] that is better suited for cryptanalysis. The compression function has 64 steps and computes a sequence of *working states* Q_t for inputs $IHV_{\text{in}} \in \{0,1\}^{128}$, $M \in \{0,1\}^{512}$:

1. Split IHV_{in} and M into 32-bit words; $IHV_{\text{in}} = a \| b \| c \| d$, $M = m_0 \| \ldots \| m_{15}$
2. Let $Q_{-3} = a$, $Q_{-2} = d$, $Q_{-1} = c$ and $Q_0 = b$.
3. For $t = 0, \ldots, 63$, compute

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

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

where $AC_t = \lfloor 2^{32} \cdot |\sin(t+1)| \rfloor$ and W_t, $f_t(X, Y, Z)$ and RC_t are given by:

Step	W_t	$f_t(X,Y,Z)$	RC_t
$0 \leq t < 16$	m_t	$(X \wedge Y) \oplus (\overline{X} \wedge Z)$	$(7,12,17,22)_{[t \bmod 4]}$
$16 \leq t < 32$	$m_{(1+5t) \bmod 16}$	$(Z \wedge X) \oplus (\overline{Z} \wedge Y)$	$(5,9,14,20)_{[t \bmod 4]}$
$32 \leq t < 48$	$m_{(5+3t) \bmod 16}$	$X \oplus Y \oplus Z$	$(4,11,16,23)_{[t \bmod 4]}$
$48 \leq t < 64$	$m_{(7t) \bmod 16}$	$Y \oplus (X \vee \overline{Z})$	$(6,10,15,21)_{[t \bmod 4]}$

3.2 General Approach

When constructing a chosen-prefix collision pair $P||C||S_{any}$ and $P'||C'||S_{any}$ for given prefixes P and P' and arbitrary suffix S_{any}, we may assume without loss of generality that P and P' are of equal length and that their length is a multiple of the MD5 message block size. (Otherwise, one can just add padding.) A chosen-prefix collision attack consists of two stages. The first is the Birthday Search where one searches for equal-length suffixes S_b and S'_b such that the difference in the intermediate hash value after processing $P\|S_b$ and $P'\|S'_b$ has a particular form necessary for the second stage. In the second stage, one constructs near-collision block pairs $(S_1,S'_1),(S_2,S'_2),\ldots,(S_n,S'_n)$ such that after processing $P\|S_b\|S_1\|\ldots\|S_n$ and $P'\|S'_b\|S'_1\|\ldots\|S'_n$ the intermediate hash values are equal. Thus one has found the desired $C = S_b\|S_1\|\ldots\|S_n$ and $C' = S'_b\|S'_1\|\ldots\|S'_n$ for which the pair $P||C||S_{any}$ and $P'||C'||S_{any}$ form a collision for any suffix S_{any}. We explain the construction of the near-collision block pairs below.

3.3 Differential Cryptanalysis

Differential cryptanalysis is based on the analysis of the propagation of input differences throughout a cryptosystem. This technique was publicly introduced in 1993 by Eli Biham and Adi Shamir who first applied it to block ciphers [1]. Differential cryptanalysis of hash functions has been very successful. One of the key techniques introduced by Wang et al. against MD5 was the simultaneous use of the difference modulo 2^{32} and the bitwise XOR difference resulting in a bitwise signed difference.

Let I and I' be two different inputs, for any variable X involved in the computation for input I, we denote the respective variable for input I' as X'. For $X, X' \in \mathbb{Z}_{2^{32}}$, we denote by $\delta X = X' - X \bmod 2^{32}$ the *arithmetic* differential. When it is necessary to keep track of the bitwise differences as well, we use the *Binary Signed Digit Representation* (BSDR). The BSDR differential is $(\Delta X[i])_{i=0,\ldots,31}$ where $\Delta X[i] = X'[i] - X[i] \in \{-1,0,1\}$. We can easily calculate the arithmetic difference from the BSDR: $\delta X = \sum_i \Delta X[i] \cdot 2^i \bmod 2^{32}$.

For a BSDR ΔX, we define the *weight* $w(\Delta X)$ as the number of indices i where $\Delta X[i] \neq 0$. For $\delta X \neq 0$, there are multiple BSDRs ΔX such that $\delta X = \sum_{i=0}^{31} \Delta X[i] \cdot 2^i$. However, there is a normal form, called the non-adjacent form (NAF). The non-adjacent form of δX is the *unique* BSDR ΔX such that $\sum_{i=0}^{31} \Delta X[i] \cdot 2^i = \delta X$, $\Delta X[31] \geq 0$ and ΔX has no adjacent non-zero entries. The NAF is a minimal-weight BSDR of δX. We define the *NAF-weight* $w(\delta X)$ as the weight of the NAF of δX.

3.4 Differential Paths

A *differential path* is an exact description of how differences propage through two related evaluations of `MD5Compress`. In particular, a differential path describes for every step t the differences δQ_{t-3}, ΔQ_{t-2}, ΔQ_{t-1}, ΔF_t, δW_t, δT_t, δR_t and δQ_{t+1} such that:

- $\delta T_t = \delta Q_{t-3} + \sigma(\Delta F_t) + \delta W_t$;
- $\delta Q_{t+1} = \sigma(\Delta Q_t) + \delta R_t$;
- $Pr[\Delta F_t | \Delta Q_{t-2}, \Delta Q_{t-1}, \Delta Q_t] > 0$;
- $Pr[\delta R_t | \delta T_t] > 0$.

We say that an input pair $(IHV, m_0 \| \ldots \| m_{15})$, $(IHV', m'_0 \| \ldots \| m'_{15})$ for `MD5Compress` *solves* the differential path up to step t if differences for the message block and the intermediate variables are as specified in the differential path up to step t.

Although the first differential paths for MD5 were constructed entirely by hand [29], two quite different ways to construct differential paths have since been introduced: Stevens' meet-in-the-middle approach [26] and De Cannière and Rechberger's coding-theory based technique [4,14].

Suppose a pair of inputs solves a differential path up to some step. This pair of inputs might fail to solve the next step because of the Boolean function or because of the bit rotation. To handle the Boolean functions, bit conditions are used that allow efficient checks whether our inputs have the correct values for ΔF_t. The rotations are taken care of probabilistically.

3.5 Tunnels

Message modification, specifically Tunneling [10], is an important technique that can drastically speed up collision attacks. Under some preconditions, a tunnel allows us to change a certain working state bit $Q_t[i]$ and corresponding message bits without affecting $Q_{t+1}, \ldots, Q_{t'}$ for some $t' > t$. As an example, consider the most important known tunnel $\mathcal{T}_8$ with the following requirements:

- $Q_9[i]$ is free, i.e., no difference and no boolean function bitcondition
- $Q'_{10}[i] = Q_{10}[i] = 0$, and $Q'_{11}[i] = Q_{11}[i] = 1$

Under these conditions, we can flip bits $Q_9[i] = Q'_9[i]$ and adjust m_8, m_9 and m_{12} without affecting $Q_{10}, \ldots, Q_{24}$ and $Q'_{10}, \ldots, Q'_{24}$.

To see why $\mathcal{T}_8$ is useful, suppose that we have a differential path and a partial solution thereof up to and including Q_{24}. We say that a bit-position $i \in \{0, \ldots, 31\}$ is *active* for $\mathcal{T}_8$ if it satisfies the requirements. We call the number k of active bit-positions the *strength* of $\mathcal{T}_8$. The tunnel allows us to generate 2^k different partial solution up to Q_{24} — one for each possible value of the active bit-positions. Since the probability that a partial solution can be extended to a full solution is rather small, cheaply generating *many* partial solutions reduces the cost of the collision attack significantly. In Table 1, we describe the three tunnels that are the most relevant for the Flame collision attack. In Sect. 4.1.3, we discuss how these tunnels might have been used in the collision attack.

Table 1. Most important tunnels for `MD5Compress`

Tunnel	Flip bit	Aux. bitconditions	Affected states	Affected message words
$\mathcal{T}_4$	$Q_9[b]$	$Q_{10}[b] = Q_{11}[b] = 1$	$Q_{22}, \ldots, Q_{64}$	m_8, m_9, m_{10}, m_{12}
$\mathcal{T}_5$	$Q_{10}[b]$	$Q_{11}[b] = 0$	$Q_{22}, \ldots, Q_{64}$	$m_9, m_{10}, m_{12}, m_{13}$
$\mathcal{T}_8$	$Q_9[b]$	$Q_{10}[b] = 0, Q_{11}[b] = 1$	$Q_{25}, \ldots, Q_{64}$	m_8, m_9, m_{12}

4 Reverse-Engineering Flame's Attack

4.1 Breakdown of Data

In Appendix B we list the chaining values and near-collision blocks from the available Flame certificate and the ones for the associated 'legitimate' certificate that can be recovered using counter-cryptanalysis. The differential path for each near-collision block pair can directly be observed by comparing the two compression function computations. In this section we first list several specific observations about these (reconstructed) Flame near-collision blocks and the observed differential paths that are relevant to our reconstruction.

4.1.1 Some Features of the Near-Collision Blocks

Observation 1 ([23])**.** *Due the constrained space where the near-collision blocks were to be hidden in the certificate, the collision attack could only use four near-collision blocks.*

Observation 2. *Blocks 1 and 3 of the Flame collision attack use the message block differences from the first differential path of Wang et al.'s identical-prefix attack, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = 2^{15}$ and $\delta m_i = 0$ for $i \neq 4, 11, 14$. Blocks 2 and 4 use the differences from the second differential path of the identical prefix attack, $\delta m_4 = \delta m_{14} = 2^{31}$, $\delta m_{11} = -2^{15}$ and $\delta m_i = 0$ for $i \neq 4, 11, 14$.*

Observation 3. *The working state differences ΔQ_6 are maximal in all four near-collision blocks, i.e., for every $i = 0, \ldots, 31$, we have $\Delta Q_6[i] \neq 0$. The ΔQ_6 of Blocks 1 and 3 are equal, likewise for Blocks 2 and 4.*

Observation 4. *The four blocks all have a common structure: Up to and including step 5, the differences δQ_t vary among all four blocks. Then, there is a maximal difference in step 6. After that, the values for ΔQ_t and ΔF_t are mostly identical in the first and third and in Blocks 2 and 4, leading up to long sequences of trivial steps. The final five steps again differ greatly among all four blocks.*

4.1.2 Notes on Differential Path Construction

From this last observation, we conclude that a differential path beginning based on the input IHVs and a differential path ending were generated separately and then combined. Such differential path construction can be done for MD5 using

Stevens' meet-in-the-middle approach [26] or De Cannière and Rechberger's coding-theory based technique [4,14]. The latter technique is less likely to have been used, since all observed differential paths don't show its characteristic very long carry chains over the non-predetermined part $Q_1, \dots, Q_5$. Stevens already showed that suitable differential paths can be constructed in about 15 s on an Intel i7-2600 CPU, so in time equivalent to approximately 2^{29} MD5 compressions [25]. As this shows that differential path construction can be done very fast and does not have to cost a significant fraction of the overall attack complexity and lacking more example collisions for analysis, our paper will focus on the complexity-wise more costly parts of the attack.

4.1.3 Tunnel Strengths in the Near-Collision Blocks

In order to estimate the complexity of the Flame collision attack, it is important to know whether and to what extent the attackers used *tunnels*. The tunnels $\mathcal{T}_4$, $\mathcal{T}_5$ and $\mathcal{T}_8$ are the most important in speeding up the attack. See Table 1 for a description of the three relevant tunnels.

Observation 5 ([25, Sect. 3.3]). *The table below lists per near-collision block the observed strength of tunnel $\mathcal{T}_8$, the maximal strength possible given the respective differential path and the average strength that would have been observed if the tunnel was* not *used.*

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

It is clear that the tunnel $\mathcal{T}_8$ has been used, since the observed tunnel strengths are much larger than one expects to see if $\mathcal{T}_8$ was *not* used. Although not presented here, we'd like to note that the tunnel strengths for $\mathcal{T}_4$ and $\mathcal{T}_5$ are smaller than average, but one cannot conclude that $\mathcal{T}_4$ and $\mathcal{T}_5$ were not used since for each bit only one of $\mathcal{T}_4, \mathcal{T}_5$ and $\mathcal{T}_8$ can be active due to conflicting preconditions.

For our complexity estimates, we will assume the strengths of these three tunnels to be the average over all four blocks. That is, we assume that tunnel $\mathcal{T}_4$ has strength 3, $\mathcal{T}_5$ has strength 7.5 and $\mathcal{T}_8$ has strength 9.75. Reconstructing the exact tunnel-exploitation method would be interesting and could lead to more precise complexity estimates. We discuss some possible methods in Sect. 4.2.3.

4.2 Our Reconstruction of the Chosen-Prefix Collision Attack

In this section, we describe our reconstruction of the collision attack, in particular the differential path construction, the *families* of differential paths endings that were used, the cost of the Birthday Search and of the message block construction.

Central to our reconstruction attempt is the idea that the first two blocks eliminate δc from $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$ up to a constant term while allowing random changes in parts of δb. The second two blocks then eliminate δb and the constant term in δc. This allows for the first two blocks to be constructed faster than estimated in the preliminary analysis in [25].

It seems that the four near-collision attacks can be grouped into two *pairs*: Blocks 1 and 2 form a pair, and, likewise, Blocks 3 and 4. In each of the pairs, the first block uses the message block differences of the first near-collision block in the identical-prefix attack by Wang et al. and the second block uses the difference of the second near-collision block in that attack. That is, in Blocks 1 and 3, the only differences in the message block are $\delta m_4 = \delta m_{14} = 2^{31}$ and $\delta m_{11} = 2^{15}$. In Blocks 2 and 4, the differences are negated, i.e., $\delta m_{11} = -2^{15}$.

To determine the complexity of the Birthday Search and of the message block construction algorithm, we describe a family of end-segments of the differential path for each of the four near-collision blocks. We compute the complexity of the Birthday Search and the complexity of the algorithm for generating near-collision blocks on the basis of our reconstruction of the end-segments.

Table 2. Chaining value difference corrections $(\delta IHV_{\text{out}} - \delta IHV_{\text{in}})$ for each block

	Block 1	Block 2
δa	$[31]$	$[31, 5]$
δd	$[31, 25]$	$[31, -25, -9, 5]$
δc	$[31, 25, -14, -12, 9]$	$[31, 26, 24, 20, -9, 5]$
δb	$[31, 25, -18, -15, -12, 9, 1]$	$[-26, 24, 21, -14, -9, 5, 0]$
	Block 3	**Block 4**
δa	$[31]$	$[31]$
δd	$[31, 25, 9]$	$[31, -25, -9]$
δc	$[31, 26, -24, -14, 9]$	$[31, -25, 14, -9]$
δb	$[30, 26, -24, 20, -17, 15, 9, -3]$	$[-25, 14, -9, -5, 3, 0]$

4.2.1 Differential Path Family

Four near-collision blocks are used to eliminate the chaining value differences after the birthday search of the chosen-prefix collision attack. In this section we reconstruct the family of differential paths used for each of the four near-collision blocks based on the observed chaining value differences, the observed differential paths and the possible variations thereof that are compatible with the overall attack.

In particular, each of the four near-collision attacks uses a carry expansion of a particular bit difference in the last few steps of Wang's original differential paths for MD5 to allow for some controlled additional differences to affect the chaining value differences. This can be seen in the recovered paths shown in Appendix A: for each block there is a primary carry chain either in δQ_{62} or δQ_{63} starting at bit position either 5 or 25 used for controlled differences, other small carry chains are random artifacts and not actively used. Our reconstruction is based on these primary carry chains and we will parametrize the amount of allowed carries. Using other carry chains significantly complicates the overall attack strategy, does not lead to significant benefits and does not fit the observed paths, hence we apply Ockham's razor principle and keep to the most straightforward explanation.

The differences that are added to $\delta IHV = (\delta a, \delta b, \delta c, \delta d)$ using each near-collision block are summarized in Table 2. We begin with an outline of what we assume to be the elimination strategy. The differences in δc are eliminated by the first two blocks using carry chains in δQ_{62} starting at bit positions 25 and 5 respectively, but a difference of -2^{24} is introduced which is then eliminated in the final two blocks. For δb, matters are more complicated. Given the following observations:

- deliberate changes to δb possible in blocks 1,2 can be deferred to blocks 3,4;
- random changes to δb possible in blocks 1,2 can be handled in blocks 3,4;
- blocks 1 and 2 actually increase the NAF-weight of δb;

we found that the best explanation is that the changes to δb in the first two blocks are mostly random and that the elimination of differences in δb is done in Block 3 and Block 4 using carry chains in δQ_{63} starting at bit positions 25 and 5 respectively. This explanation in fact reduces the complexity for Blocks 1 and 2 as they only need to control δQ_{64} (that affects δb) to a very small extent.

Table 3. End segment of block 1.

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

$\delta Q_{63} = 2^{31} + 2^{25} + 2^{9} + C_{14}2^{14} + \sum_{i=8}^{8+w_1} C_i 2^i, 1 \leq w_1 \leq 5$

$\delta Q_{64} = \delta Q_{63} + \sum_{i=29}^{31} X_i 2^i + \sum_{i=14}^{20} X_i 2^i + \sum_{i=0}^{v_1} X_i 2^i, -1 \leq v_1 \leq 3$

We have generalized the observed differential path endings to a reasonable extent, i.e., making our reconstructed path families more general would make matters significantly more complex, while similar benefits might also be obtained by simply choosing larger parameters for our families below. The four differential path families are described as follows. For Block i we use a parameter w_i that specifies the length of the carry chain and thus over how many bits one can fully control the differences. Block 4 uses an additional carry chain whose length is determined by u_4. For Blocks 1 and 2 we use an additional parameter v_1 and v_2 that control an amount of bit positions in which random differences are allowed as they can be handled in Blocks 3 and 4, these parameters v_1 and v_2 depend on the value of u_4.

Table 4. End segment of block 2.

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

$\delta Q_{63} = 2^{31} - 2^{26} + 2^{24} - 2^{9} + 2^{5} + \sum_{i=20}^{24+w_2} C_i 2^i$,
$\delta Q_{64} = \delta Q_{63} + \sum_{i=27}^{29} B_i 2^i + B_{20} 2^{20} + \sum_{i=0}^{v_2} X_i 2^i + \sum_{i=13}^{19} X_i 2^i + \sum_{i=30}^{31} X_i 2^i$
$0 \leq w_2 \leq 6, \quad -1 \leq v_2 \leq 4$

1. Block 1 uses a carry chain starting in δQ_{62} at bit position 25 up to $25 + w_1$ to control differences in δQ_{63} over bit positions 8 up to $8 + w_1$. Given the differences that can be covered in Blocks 3 and 4, we can allow arbitrary differences in δQ_{64} at bit position ranges $[0, v_1]$, $[14, 20]$, and $[29, 31]$.
2. Block 2 uses a carry chain starting in δQ_{62} at bit position 5 up to bit position $9+w_2$ to control differences in δQ_{63} over bit positions 20 up to $24+w_2$. Given the differences that can be covered in Blocks 3 and 4, we can allow arbitrary differences in δQ_{64} at bit position ranges $[0, v_2]$, $[13, 19]$, and $[30, 31]$.
3. Block 3 uses a carry chain starting in δQ_{63} at bit position 24 up to $26 + w_3$ to control differences in δQ_{64} over bit positions 13 up to $15 + w_3$.
4. Block 4 uses a carry chain starting in δQ_{63} at bit position 14 up to bit position $14 + w_4$ to control differences in δQ_{64} over bit positions 13 up to $15 + w_3$, and a second carry chain at bit positions 9 up to $9 + u_4$ to control differences over bit positions 30 up to $(30 + u_4 \bmod 32)$ that wrap around to the lower bit positions.

Note that in Block 4, the parameter u_4 must be large enough to eliminate the random changes to δb that are made in Blocks 1 and 2. That is, if $\max(v_1, v_2) \leq 2$, we need $u_4 \geq \max(v_1, v_2) + 2$ and otherwise, we need $u_4 = 4$. Also, in Sect. 4.3.2 we will estimate the complexity of solving these differential paths.

Table 5. End segment of block 3.

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

$\delta Q_{64} = 2^{30} + 2^{26} - 2^{24} + 2^{9} - 2^{3} + \sum_{i=13}^{15+w_3} B_i 2^i, 0 \leq w_3 \leq 4$

We now describe each differential path family more fully in Tables 3, 4, 5 and 6 by giving a template and specifying equations that the values of $\delta Q_{61}, \ldots, \delta Q_{64}$

Table 6. End segment of block 4.

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

$\delta Q_{64} = -2^{25} + 2^{14} - 2^{9} - 2^{5} + 2^{3} + \sum_{i=30}^{\min(u_4,1)} B_i 2^i + \sum_{i=0}^{u_4-2} B_i 2^i + \sum_{i=3}^{3+w_4} B_i 2^i$

$1 \leq w_4 \leq 6, \quad 0 \leq u_4 \leq 4$

must satisfy. In the templates, a symbol $\mathtt{q}_t[i]$ at step (row) i and bit position (column) i can be any of the following:

- '`.`': represents $Q_t[i] = Q'_t[i]$;
- '`+`': represents $Q_t[i] = 0$, $Q'_t[i] = 1$;
- '`-`': represents $Q_t[i] = 1$, $Q'_t[i] = 0$;
- '`0`': represents $Q_t[i] = Q'_t[i] = 0$;
- '`1`': represents $Q_t[i] = Q'_t[i] = 1$;
- '`^`': represents $Q_t[i] = Q'_t[i] = Q_{t-1}[i]$;
- '`?`': represents $(Q_t[i] = Q'_t[i]) \wedge (Q_t[i] = 1 \vee Q_{t-2}[i] = 0)$;
- '`D`': a variable differential bitcondition, i.e., $\mathtt{q}_t[i] \in \{\mathtt{.}, \mathtt{+}, \mathtt{-}\}$;
- '`B`': a variable Boolean function bitcondition, i.e., $\mathtt{q}_t[i] \in \{\mathtt{.}, \mathtt{0}, \mathtt{1}, \mathtt{?}\}$;
- '`X`': a non-constant bitcondition, i.e., $\mathtt{q}_t[i] \in \{\mathtt{+}, \mathtt{-}\}$;
- '`*`': a (for now) irrelevant differential bitcondition;
- '`A`': the same differential as above, $\mathtt{q}_t[i] = \mathtt{q}_{t-1}[i]$.

The equations may contain the following terms:

- w_i, v_i, u_i: Parameters of the differential path family. Higher values for the w_i mean that the differential path family can cancel more differences but is, on average, harder to solve.
- C_i, B_i: These terms can take on values in $\{-1, 0, 1\}$ and correspond to the variable differential bitconditions ('`D`'s) in step 63 or 64, respectively. A member of the differential path family is determined by the C_i and B_i.
- X_i: These terms take on values in $\{-1, 0, 1\}$ and correspond to the irrelevant bitconditions ('`*`'s). While the B_i and C_i fix a differential path in the family, the X_i are determined only after a successful near-collision block search.

We say that a pair of inputs (IHV, B), (IHV', B') to `MD5Compress` solves the last four steps of the differential path if there is some setting for the X_i such that $\delta Q_{61}, \ldots, \delta Q_{64}$ satisfy the given equations. This is a more lax definition than what we use elsewhere, i.e., we do *not* require a solution to solve the exact bitconditions but use bitconditions as a tool to show which δQ_i are possible.

4.2.2 Birthday Search

We calculate the Birthday Search complexity for the maximal parameter values. It is easy to compute the Birthday Search complexity for lower values, namely for each carry that is dropped, the complexity increases by a factor of $2^{0.5}$.

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

$$f(x) = (a, \tilde{b}_{10}, \dots, \tilde{b}_{13}, \tilde{b}_{21}, \dots, \tilde{b}_{26}, c_0, \dots, c_7, c_{15}, \dots, c_{19}, c_{31}, d)$$

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

$$\text{and } \tilde{b} = b - c$$

with IHV and IHV' the intermediate hash values after processing the two chosen prefixes. Not every collision of f is useful. The probability p that a collision is useful is at most 1/2 since we require that the two parts use different prefixes. Therefore the expected number of compression function calls required to find a useful collision is $\sqrt{\pi \cdot 2^{88}/(2 \cdot p)} \approx \sqrt{\pi} \cdot 2^{44} \approx 2^{44.8}$ [18].

As we already mentioned, we use parameters to make trade-offs between message block construction and Birthday Search cost. For every carry we do not rely on, we introduce another bit position where b and b' or c and c' may not differ, increasing the Birthday Search complexity by a factor of $2^{0.5}$. This allows us to trade off Birthday Search complexity against complexity in the message block construction.

4.2.3 Tunnel Exploitation Analysis

As explained in Sect. 4.1.3, the tunnel strength in the Flame differential paths was neither average nor maximized. We now derive a formula for the expected tunnel strength when each tunnel bit is active with probability α. Let m be the number of bits that could be active for $\mathcal{T}_8$. For a random solution up to step 24, let $\mathbf{S}$ be the random variable measuring the strength of $\mathcal{T}_8$ and **solve** the event that the partial solution can be extended to a full solution using $\mathcal{T}_8$. Assuming $\Pr[\textbf{solve} \mid \mathbf{S} = k] \approx 2^k \cdot p$ for some p independent of k, we can calculate

$$\mathbb{E}[\mathbf{S} \mid \textbf{solve}] \approx \sum_{k=0}^{m} \frac{k \cdot \binom{m}{k}}{\sum_{i=0}^{m} \binom{m}{i} \cdot (2\alpha)^{i-k} \cdot (1-\alpha)^{k-i}}$$

An explanation for the observed tunnel strengths (Observation 5) proposed in [25] is that the Flame authors did not try to maximize the tunnel strength but used tunnels in their message block construction algorithm to the extent that they were available. This corresponds to setting $\alpha = 1/4$. On the other hand, we consider the alternative hypothesis that many bits in working state

Q_{10} were fixed to '0' to bring the probability closer to $\alpha = 1/2$. In Table 7, we list the expectation and variance of the tunnel strength for both values of α. These results show that the initial explanation by Stevens with $\alpha = 1/4$ is rather unlikely, while the explanation with $\alpha = 1/2$ is more probable.

Table 7. Summary of the observed (s), maximal (m) and expected (μ) tunnel strength, and the standard deviation (σ).

			$\alpha = 1/4$		$\alpha = 1/2$	
Block	s	m	μ	σ	μ	σ
1	7	17	6.80	2.02	8.67	1.70
2	13	18	7.20	2.08	10.00	1.83
3	10	17	6.80	2.02	9.33	1.76
4	9	18	7.20	2.08	10.67	1.89

4.3 Cost Estimation

4.3.1 A Formula for the Expected Cost

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

1. Generate a full differential path/generate a set of initial working states that connects to the lower differential path.
2. Select $Q_1, \ldots, Q_{16}$ according to the path and tunnel requirements.
3. Try to obtain a solution[6] up to step 24 with the help of tunnels $\mathcal{T}_4$ and $\mathcal{T}_5$. Go back to step 2 and choose different Q_i if it is not possible to obtain a solution and use early abort to reduce the cost of this step.
4. Attempt to generate a solution for the whole path from our solution up to step 24 using tunnel $\mathcal{T}_8$. We use early abort to some extent.
5. Check if the values for $\delta Q_{61}, \ldots, \delta Q_{64}$ are correct. If yes, we have a solution.

The expected cost of this algorithm is as follows: $\mathcal{C}_{\text{path}}$ is the differential path construction cost; let the random variable $\mathcal{Z}$ be the number of input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ that we need to evaluate until we find an input pair where $\delta Q_{61}, \ldots, \delta Q_{64}$ are as specified by the differential path. The expected cost of finding a solution with the correct values for $\delta Q_{61}, \ldots, \delta Q_{64}$ is then

$$\mathcal{C}_{\text{block}} = \mathcal{C}_{\text{path}} + \mathbb{E}[\mathcal{Z}] \cdot 2^{13.6}.$$

The factor $2^{13.6}$ represents the measured average complexity of finding input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ for Flame's differential paths. This complexity is very stable for all near-collision blocks as the differential paths are

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

only varied in the first 16 steps which don't affect complexity and the last few steps which are instead covered by $\mathcal{Z}$. Hence, the expected complexity of finding a near-collision block is $\mathbb{E}[\mathcal{Z}] \cdot 2^{13.6}$.

We give estimates for $\mathbb{E}[\mathcal{Z}]$ in the next section. As discussed in Sect. 4.1.2, $\mathcal{C}_{\text{path}}$ can be as low as 2^{29} MD5 compressions, which will be negligible compared to the other parts of the attack.

4.3.2 Estimating the Expected Number of Attempts

In this section, we want to estimate the expected number of input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ we have to generate until a solution for the differential path is found. We call input pairs with $\delta Q_{57} = \cdots = \delta Q_{60} = 2^{31}$ *admissible* input pairs and we call the values for $\delta Q_{61}, \ldots, \delta Q_{64}$ that we want the *target.*

Let $\mathcal{T}_{i,u_i,v_i,w_i}$ be the random variable that gives the target for Block i with parameters u_i, v_i and w_i. Selecting a target is done by selecting the values for $B_k, C_k \in \{-1, 0, 1\}$ as in Sect. 4.2.1. Let $\mathcal{Z}_\tau$ be the random variable that counts the admissible inputs we have to try until τ is solved and let $\mathcal{Z}_{i,u_i,v_i,w_i}$ be the random variable obtained by first sampling $\tau \leftarrow \mathcal{T}_{i,u_i,v_i,w_i}$ and then sampling $\mathcal{Z}_\tau$. To compute the total expected cost, we need $\mathbb{E}\left[\mathcal{Z}_{i,u_i,v_i,w_i}\right]$. To obtain an empirical estimate λ_{i,u_i,v_i,w_i}, we repeat the following process until a fixed number of targets is solved: We first sample $\tau \leftarrow \mathcal{T}_{i,u_i,v_i,w_i}$ and then select random admissible inputs and message blocks until we find one that solves the target.[7] When the chosen number of targets is solved, we let the average number of attempts to solve a target be our estimate λ_{i,u_i,v_i,w_i}. We then obtain $\mathcal{C}_{i,u_i,v_i,w_i} = \lambda_{i,u_i,v_i,w_i} \cdot 2^{13.6}$ as an estimate for the cost of solving the differential path for Block i with parameters u_i, v_i and w_i. The simulation outcomes for the four blocks are given in Tables 8, 9, 10 and 11.

To save time, we do *not* generate admissible inputs as in Sect. 4.3.1. Instead, we select working states $Q_{57}, \ldots, Q_{60}$ and message words $m_0, \ldots, m_{15}$ at random and compute $Q'_{57}, \ldots, Q'_{60}$ and $m'_0, \ldots, m'_{15}$ by applying the appropriate arithmetic differentials. This procedure requires the assumption that the probability for hitting the target does not change when we select $Q_{57}, \ldots, Q_{60}$ and message words at random, which is justified by the pseudo-randomness of MD5.

Our estimate of the Birthday Search cost in Sect. 4.2.2 assumes that the parameters w_i and u_4 are maximal. For smaller parameter values, the cost must be multiplied by a "Birthday Factor" μ_i which we give in Table 12.

4.3.3 Total Cost.

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

[7] Recall that we say that an input solves a differential path if there exists a setting for the X_k such that $\delta Q_{61}, \ldots, \delta Q_{64}$ are as described by the path.

Table 8. Estimated complexities for the first near-collision block.

$\log_2 \mathcal{C}_{1,v_1,w_1}$	$v_1 = -1$	$v_1 = 0$	$v_1 = 1$	$v_1 = 2$	$v_1 = 3$
$w_1 = 1$	24.1	23.7	23.6	23.4	22.9
$w_1 = 2$	25.8	25.2	25.0	24.8	24.3
$w_1 = 3$	27.9	27.2	26.8	26.5	26.1
$w_1 = 4$	29.8	29.1	28.6	28.2	27.8
$w_1 = 5$	30.4	29.7	29.2	28.7	28.3

Table 9. Estimated complexities for the second near-collision block.

$\log_2 \mathcal{C}_{2,v_2,w_2}$	$v_2 = -1$	$v_2 = 0$	$v_2 = 1$	$v_2 = 2$	$v_2 = 3$	$v_2 = 4$
$w_2 = 0$	35.4	34.8	34.7	34.6	34.6	34.6
$w_2 = 1$	37.0	36.2	36.1	36.0	36.0	36.0
$w_2 = 2$	39.2	38.2	37.9	37.8	37.8	37.8
$w_2 = 3$	41.6	40.5	40.0	39.8	39.7	39.7
$w_2 = 4$	44.0	42.9	42.4	42.0	41.8	41.8
$w_2 = 5$	46.7	45.5	45.0	44.6	44.2	44.0
$w_2 = 6$	49.3	48.1	47.6	47.2	46.8	46.4

Table 10. Estimated complexities for the third near-collision block.

	$\log_2 \mathcal{C}_{3,w_3}$
$w_3 = 0$	32.3
$w_3 = 1$	34.3
$w_3 = 2$	36.4
$w_3 = 3$	38.1
$w_3 = 4$	38.3

- $\mathcal{C}_{\text{msg}}$: expected cost when minimizing the message block construction cost.
- $\mathcal{C}_{\text{flame}}$: expected cost when minimizing the message block construction cost while keeping the parameters consistent with the observed paths.
- $\mathcal{C}_{\text{search}}$: expected cost when minimizing the Birthday Search cost.
- $\mathcal{C}_{\text{min}}$: minimal expected cost.

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

$$\mathcal{C}_{\text{msg}} = 4 \cdot \mathcal{C}_{\text{path}} + 2^{11.0} \cdot 2^{44.3} \cdot p^{-1/2} + 2^{24.1} + 2^{35.4} + 2^{32.3} + 2^{34.1} \approx 2^{55.8}$$

Table 11. Estimated complexities for the fourth near-collision block.

$\log_2 \mathcal{C}_{4,u_4,w_4}$	$u_4 = 0$	$u_4 = 1$	$u_4 = 2$	$u_4 = 3$	$u_4 = 4$
$w_4 = 1$	34.1	33.8	35.6	38.2	38.7
$w_4 = 2$	35.2	35.0	36.7	39.4	39.8
$w_4 = 3$	37.0	36.5	38.4	41.0	41.4
$w_4 = 4$	38.8	38.4	40.2	42.7	43.8
$w_4 = 5$	40.8	40.6	42.3	44.6	44.8
$w_4 = 6$	43.0	42.4	43.6	46.9	47.8

Table 12. "Birthday Factors" for the four near-collision blocks.

Block i	1	2	3	4
$\log_2 \mu_i$	$(5 - w_1)/2$	$(6 - w_2)/2$	$(4 - w_3)/2$	$(10 - w_4 - u_4)/2$

where $\mathcal{C}_{\text{path}}$ is the cost of constructing a full differential path and p is the probability that a collision is useful. We assume that $p \approx 1/2$ and use the fact that $4 \cdot \mathcal{C}_{\text{path}}$ can be negligible compared to the other costs (see Sect. 4.1.2).

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

$$\mathcal{C}_{\text{flame}} = 4 \cdot \mathcal{C}_{\text{path}} + 2^{48.8} \cdot p^{-1/2} + 2^{27.8} + 2^{39.7} + 2^{38.3} + 2^{38.7} \approx 2^{49.3}.$$

For $\mathcal{C}_{\text{search}}$, we have a Birthday Factor of 1 and

$$\mathcal{C}_{\text{search}} = 4 \cdot \mathcal{C}_{\text{path}} + 2^{44.3} \cdot p^{-1/2} + 2^{28.3} + 2^{46.4} + 2^{38.3} + 2^{47.8} \approx 2^{48.4}$$

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

$$\mathcal{C}_{\text{min}} = 4 \cdot \mathcal{C}_{\text{path}} + 2^{45.8} \cdot p^{-1/2} + 2^{28.3} + 2^{44.0} + 2^{38.3} + 2^{44.8} \approx 2^{46.6}$$

We now show that this cost is indeed minimal:

Theorem 6 . *Given the values for $\mathbb{E}[\mathcal{Z}]$ from Sect. 4.3.2 and assuming that the probability p for a useful collision in the Birthday Search is 1/2, the expected cost of the collision attack is equivalent to at least $\mathcal{C}_{min} = 2^{46.6}$ executions of* `MD5Compress`. *For suitably chosen parameters, this cost can be achieved.*

Proof. We have already given parameters which show that the second part of the theorem holds. To see that this parameter choice indeed gives us the minimal cost, let us try to improve upon it: It is easy to see that the Birthday Factor μ must satisfy $1 < \mu \leq 2^{1.5}$ for if $\mu = 1$, the attack complexity is $\mathcal{C}_{\text{search}} > \mathcal{C}_{\text{min}}$ and if $\mu = 2^{2.0}$, the Birthday Search cost is already larger than $\mathcal{C}_{\text{min}}$. If $\mu = 2^{0.5 \cdot k}$, we can reduce the w_i or u_i parameters by k. Since Blocks 2 and 4 have the highest complexity, this is where these reductions should be spent.

For $\mu = 2^{0.5}$, in order to improve upon $\mathcal{C}_{\text{min}}$, we need to construct the near-collision blocks with a cost $\leq 2^{45.8}$, for $\mu = 2^{1.0}$, the cost needs to be $\leq 2^{45.4}$ and for $\mu = 2^{1.5}$ it needs to be $\leq 2^{44.2}$. It turns out that the only way these constraints can be satisfied is by setting $\mu = 2^{1.0}$ and reducing the parameters w_2 and w_4. But these are precisely the parameters that give us $\mathcal{C}_{\text{min}}$. □

The parameters for $\mathcal{C}_{\text{min}}$ are consistent with the observed differential paths. Assuming that our reconstruction is correct, we can conclude that the expected cost of the collision attack used by the Flame authors is lower-bounded by $2^{46.6}$ calls to `MD5Compress`. However, it seems likely that the cost of the actual attack was higher than $\mathcal{C}_{\text{min}}$ since the observed number of carries is always lower than the $\mathcal{C}_{\text{min}}$-parameters. Nevertheless, the actual collision attack might have been faster in practice: Since Birthday Search can be executed very cost-effectively on massively parallel architecture (e.g., GPUs), it might be advantageous to shift a larger part of the workload to the Birthday Search step.

The expected cost of the [27]-attack with four near-collision blocks is roughly 1/4 of the lower bound of the Flame attack; its expected cost is equivalent to $2^{44.55}$ calls to `MD5Compress` (see [25, Sect. 3.7]). The cost of the Birthday Search dominates the total cost.

5 Conclusion

In this paper we have demonstrated for the first time that a cryptanalytic attack can be reconstructed from a single output example, specifically, a single example half of a collision pair. We have provided a complexity analysis proving a lower-bound for its cost. Furthermore, we showed that in terms of theoretical cost, the Flame attack is less efficient than the [27]-attack, although it might achieve a better real-world performance when the Birthday Search is performed on a massively parallel architecture.

Our reverse-engineering of a yet unknown cryptanalytic attack seems to be without precedent. As allegedly Flame was developed by some nation-state(s), the example collision and its analysis in this work provide some insights to their cryptanalytic knowledge and capabilities. With respect to the complexity, the closest fit of attack parameters is equivalent to $2^{49.3}$ MD5 compressions which takes roughly 40,000 CPUcore hours. That means for say 3-day attempts to succeed in reasonable time given the large number of required attempts, one needs about 560 CPUcores, which is large but not unreasonable even for academic research groups. With the respect to cryptanalytic knowledge there are no indications at all of superior techniques, rather various parts seem to be sub-optimal compared to the state-of-the-art in the literature. In particular it is clear that one could do better using the state-of-the-art in the literature, i.e., lower theoretical complexity to craft a 4-block chosen-prefix collision (see Theorem 6), and generate differential paths with significantly lower density of bitconditions in negligible time (as previously observed [25]). Nevertheless, the apparent significant resources more than make up for that and it seems a working attack that succeeds in reasonable time was more important than optimizing the overall attack using all state of the art techniques.

A Flame Differential Paths

Here, we show the differential paths for all four Flame near-collision blocks, see also Sect. 4.2.1. The column 'Probability' lists the theoretical unconditional rotation probabilities from δT_t to δR_t. If this rotation probability for this δT_t is not maximal, we list the maximal possible rotation probability for this δT_t albeit for a different δR_t between braces. In the next column 'Cond. Est.', we give *empirical estimates* for the probabilities of the rotations *conditioned* on that the Q_i satisfy their bitconditions (Tables 13, 14, 15 and 16).

Table 13. Differential path sections of the 1st near-collision block of Flame's attack

t	Bitconditions $\mathfrak{q}_t[31] \dots \mathfrak{q}_t[0]$	Probability	Cond. est.
2	+0-0-.00 .-++00+- 0-1-+.1+ 1+-0++^.	0.247 (0.628)	0.166
3	+010-000 .-+++0+1 +--.+^1+ -+-+++-.	0.911	1
4	-00-10+. .11-+-0+ +++11--0 -101-+0.	0.381 (0.561)	1
5	0-+-++-^ ^0110+1- -110+0-0 -0001+1^	0.229 (0.435)	1
6	++----+- ---+---- -----+++ ++++++++	0.425 (0.514)	1
7	111.-111 1101011. 110-1001 +0100.00	0.838	1
8	00+0.111 10111101 -1101100 .1110011	0.063 (0.444)	0.171
9	..0.1... .....-.. 0.10+... 0-....0.	0.516	0.563
59	+.......	1	1
60	+.11110.	1	1
61	+.11000. .... .001.00.	0.992	1
62	-.+----. ....0....	0.391 (0.609)	0.427
63	+.?0??+. ....--+.+-.	0.867	0.855
64	+......+ ++++++.. -..-.+-. .....+-.		

Table 14. Differential path sections of the 2nd near-collision block of Flame's attack

t	Bitconditions $\mathfrak{q}_t[31]\ldots\mathfrak{q}_t[0]$	Probability	Cond. est.
2	`.01.-011 00+-++0+ 0--+.--0 ++10+0+0`	0.849	0.492
3	`..1.-+11 +001++^+ 01-+0110 0+1++0++`	0.623	0.833
4	`..-.1-11 ++1-++-+ -1111--+ ++0+-+-1`	0.100 (0.547)	1
5	`^^1^+1-- 10-01011 0+10-1-+ 0-+++000`	0.399 (0.431)	0.499
6	`+-++++++ ++++---- ------+- --+-----`	0.458 (0.518)	1
7	`0010-000 01111011 1011-111 10.10010`	0.961	1
8	`00000100 1111111+ -1001111 1-010111`	0.468	0.673
9	`...-1... .-.....1 0..1+... .1....^.`	0.468 (0.469)	0.495
58	`+....... ........ ........ ........`	1	1
59	`+....... ........ ........ ..0.....`	1	1
60	`+.....0. ........ ...1001. 110.....`	0.5	0.507
61	`-....100 ...0.... ...1..1. 00+.....`	0.496	0.749
62	`+....1-. ........ ...-+++. +--.....`	0.972	0.948
63	`+....++- ...+.... ...???-. ?+-.....`	0.238 (0.270)	0.262
64	`......-- ..+..... .-....-. .+-....+`		

Table 15. Differential path sections of the 3rd near-collision block of Flame's attack

t	Bitconditions $\mathfrak{q}_t[31]\ldots\mathfrak{q}_t[0]$	Probability	Cond. est.
2	`10-01110 +++1---+ +10+.... 0-0++++1`	0.404(0.408)	0.374
3	`-0-01^1+ +0+1--10 0-++^^.0 01+0+00.`	0.941	1
4	`--0++-00 0-0+11++ ++-1-+10 -+00+-1.`	0.085(0.593)	1
5	`-1++-0-1 +1-00+1- +0++110- -1--1+^^`	0.776	1
6	`++----+- ---+---- -----+++ ++++++++`	0.514	1
7	`1000-010 00.1010. 101-0101 +0001.00`	0.838	1
8	`11+1.101 01011100 -1000101 .1000011`	0.437	0.0566
9	`..0.1... .....-.. 0.10+... 0-....0.`	0.516	0.573
58	`+....... ........ ........ ........`	1	1
59	`+....... ........ ........ ........`	1	1
60	`-.....0. ........ ......1. ........`	1	1
61	`-.0110.0 ........ .1....0. ........`	0.496	0.515
62	`+..01.+. ........ .0....+. ........`	0.498	0.492
63	`+.+---?- ........ .-....+. ........`	0.404	0.396
64	`.+...+.- ....++++ -.....+. ....-...`		

Table 16. Differential path sections of the 4th near-collision block of Flame's attack

t	Bitconditions $\mathfrak{q}_t[31] \ldots \mathfrak{q}_t[0]$	Probability	Cond. est.
2	+--.-0-. -+1+0--0 1+1-1-++ -1-00+--	0.691	0.757
3	+--1-^1. .+100--+ 10---1+0 ---0++-1	0.309	1
4	-010+-1. 10-1-01+ 0-000-1- 0+-10-1-	0.574	1
5	+00-+00^ 0++-11-0 +++0-111 01-+-100	0.749	1
6	+-++++++ ++++---- ------+- --+-----	0.518	0.507
7	.111-110 01.010.0 0101-110 1101.011	0.961	0.735
8	11110110 0101000+ -0101111 0-100111	0.032(0.476)	0.0508
9	...-1... .-.....1 0..1+... .1....^.	0.468(0.469)	0.522
58	-.......	1	1
59	-.......	1	1
60	+.....0. ....00.	1	1
61	+.....1. .... 11....1.	0.496	0.525
62	-.....-. .... 10...-+.	0.5	0.493
63	+.....-. .... +-...?-.	0.500	0.503
64	....-++. .... +-....-. ..-.+..+		

B Message Blocks and *IHV*s

	Flame certificate	Legitimate certificate
IHV_7	a262d0136907c960bb84d9d73b74732e	8262d01365179fa09bd4c9cf1b76732e
B_8	7f7b4b7bc6beeb3f9f983da38487547e	7f7b4b7bc6beeb3f9f983da38487547e
	728771254b6835ae65bd6c8fdc8dacc4	728771a54b6835ae65bd6c8fdc8dacc4
	e89892dedc5362f5726a2527a31246eb	e89892dedc5362f5726a2527a39246eb
	7f6d58cd3083d77a85b848e60e011168	7f6d58cd3083d77a85b848660e011168
IHV_8	63fc3d453bdacbc8826faa39cc7df2cc	43fc3dc5395c9d8a62719ab3ac7ff24e
B_9	657d53380b40f43b684359c13c05c340	657d53380b40f43b684359c13c05c340
	269d5197e2eb2eb8c2196e4e94463bd8	269d5117e2eb2eb8c2196e4e94463bd8
	d4fd0d00d168fadff3fa188a7c659bda	d4fd0d00d168fadff3fa188a7ce59ada
	23119f16a68b23248887226919c211ea	23119f16a68b2324888722e919c211ea
IHV_9	7aeea241ddd49e30b9ce4dab4b8e0ff4	7aeea241fc1490efb9ce4daa4b8e0ff4
B_{10}	9d3681adfbe88bd2d0eb06f21a868dc6	9d3681adfbe88bd2d0eb06f21a868dc6
	84f388c5e0d964c64895d4bed3544891	84f38845e0d964c64895d4bed3544891
	e66ce91e33971542eeb46d1f150b27dd	e66ce91e33971542eeb46d1f158b27dd
	08bb81deb6961639d926446a5fd16b3f	08bb81deb6961639d92644ea5fd16b3f
IHV_{10}	ac3aa31bd79e7f3a9b34ec0a850e3940	ac3aa39bee607f3c9bf6eb8c851039c2
B_{11}	1271dcf09962d2431458f86ef82235d2	1271dcf09962d2431458f86ef82235d2
	90f7fd936ac449b8cb0ce965a8f722b5	90f7fd136ac449b8cb0ce965a8f722b5
	f2051920ef2563c7b3974a823eb2e3ee	f2051920ef2563c7b3974a823e32e3ee
	b45ecb1db3598f8df47901b1b6688914	b45ecb1db3598f8df4790131b6688914

References

1. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag, London (1993)
2. den Boer, B., Bosselaers, A.: Collisions for the compression function of MD-5. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 293–304. Springer, Heidelberg (1994)
3. Brassard, G. (ed.): CRYPTO 1989. LNCS, vol. 435. Springer, Heidelberg (1990)
4. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: general results and applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
5. Damgård, I.: A design principle for hash functions. In: Brassard [3], pp. 416–427
6. Dobbertin, H.: The Status of MD5 After a Recent Attack. RSA CryptoBytes, 2(2) (1996)
7. Hawkes, P., Paddon, M., Rose, G.G.: Musings on the Wang et al. MD5 Collision. Cryptology ePrint Archive, Report 2004/264 (2004)
8. Hashclash project webpage. http://code.google.com/p/hashclash
9. Klima, V.: Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications. Cryptology ePrint Archive, Report 2005/102 (2005)
10. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105 (2006)
11. CrySyS Lab: sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks. Laboratory of Cryptography and System Security, Budapest University of Technology and Economics, 31 May 2012
12. Kaspersky Lab: The Flame: Questions and Answers. Securelist blog, 28 May 2012
13. Liang, J., Lai, X.: Improved Collision Attack on Hash Function MD5. Cryptology ePrint Archive, Report 2005/425 (2005)
14. Mendel, F., Rechberger, C., Schläffer, M.: MD5 is weaker than weak: attacks on concatenated combiners. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 144–161. Springer, Heidelberg (2009)
15. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard [3], pp. 428–446
16. Microsoft: Flame malware collision attack explained. Security Research and Defense, Microsoft TechNet Blog, 6 June 2012
17. Microsoft: Microsoft certification authority signing certificates added to the Untrusted Certificate Store. Security Research and Defense, Microsoft TechNet Blog, 3 June 2012
18. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. J. Cryptol. **12**(1), 1–28 (1999)
19. Nakashima, E., Miller, G., Tate, J.: U.S., Israel developed Flame computer virus to slow Iranian nuclear efforts, officials say. The Washington Post, June 2012
20. Ray, M.: Flame's Windows Update Certificate Chain. Randombit Cryptography Mailing List, June 2012. http://lists.randombit.net/pipermail/cryptography/2012-June/002969.html
21. Rivest, R.L.: The MD5 Message-Digest Algorithm. Internet Request for Comments, RFC 1321, April 1992
22. Sasaki, Y., Naito, Y., Kunihiro, N., Ohta, K.: Improved Collision Attack on MD5. Cryptology ePrint Archive, Report 2005/400 (2005)
23. Sotirov, A.: Analyzing the MD5 collision in Flame, June 2012
24. Stevens, M.: Fast Collision Attack on MD5. Cryptology ePrint Archive, Report 2006/104 (2006)

25. Stevens, M.: Counter-cryptanalysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 129–146. Springer, Heidelberg (2013)
26. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
27. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009)
28. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199 (2004)
29. Wang, X., Yu, H.: How to break MD5 and other Hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
30. Xie, T., Feng, D.: How To Find Weak Input Differences for MD5 Collision Attacks. Cryptology ePrint Archive, Report 2009/223 (2009)
31. Yajima, J., Shimoyama, T.: Wang's sufficient conditions of MD5 are not sufficient. Cryptology ePrint Archive, Report 2005/263 (2005)