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 $\delta IHV_{n+r} = (0, 0, 0, 0)$.

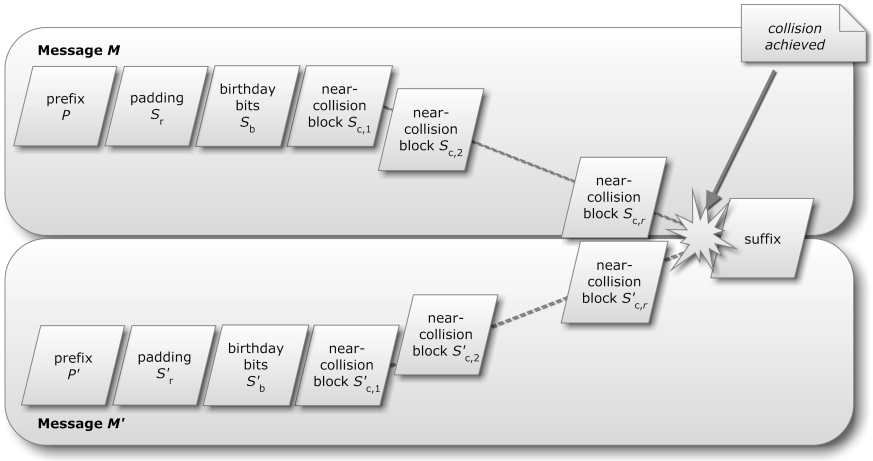

Figure 8: *Chosen-prefix collision sketch*

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.