Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities

Marc Stevens[1], Arjen Lenstra[2], and Benne de Weger[1]

[1] TU Eindhoven, Faculty of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[2] EPFL IC LACAL, Station 14, and Bell Laboratories
CH-1015 Lausanne, Switzerland

Abstract. We present a novel, automated way to find differential paths for MD5. As an application we have shown how, at an approximate expected cost of 2^{50} calls to the MD5 compression function, for any two chosen message prefixes P and P', suffixes S and S' can be constructed such that the concatenated values $P\|S$ and $P'\|S'$ collide under MD5. Although the practical attack potential of this construction of *chosen-prefix collisions* is limited, it is of greater concern than random collisions for MD5. To illustrate the practicality of our method, we constructed two MD5 based X.509 certificates with identical signatures but different public keys *and* different Distinguished Name fields, whereas our previous construction of colliding X.509 certificates required identical name fields. We speculate on other possibilities for abusing chosen-prefix collisions. More details than can be included here can be found on `www.win.tue.nl/hashclash/ChosenPrefixCollisions/`.

1 Introduction

In March 2005 we showed how Xiaoyun Wang's ability [17] to quickly construct random collisions for the MD5 hash function could be used to construct two different valid and unsuspicious X.509 certificates with identical digital signatures (see [10] and [11]). These two *colliding certificates* differed in their public key values only. In particular, their Distinguished Name fields containing the identities of the certificate owners were equal. This was the best we could achieve because

- Wang's hash collision construction requires identical Intermediate Hash Values (IHVs);
- the resulting colliding values look like random strings: in an X.509 certificate the public key field is the only suitable place where such a value can unsuspiciously be hidden.

A natural and often posed question (cf. [7], [3], [1]) is if it would be possible to allow more freedom in the other fields of the certificates, at a cost lower than 2^{64} calls to the MD5 compression function. Specifically, it has often been suggested that it would be interesting to be able to select Distinguished Name fields that are different and, preferably, chosen at will, non-random and human readable as one would expect from these fields. This can be realized if two arbitrarily chosen messages, resulting in two different IHVs, can be extended in such a way that the extended messages collide. Such collisions will be called *chosen-prefix collisions*.

We describe how chosen-prefix collisions for MD5 can be constructed, and show that our method is practical by constructing two MD5 based X.509 certificates with different Distinguished Name fields and identical digital signatures. The full details of the chosen-prefix collision construction and the certificates can be found in [16] and [14], respectively.

Section 2 contains a bird's eye view of the chosen-prefix collision construction method and its complexity. Its potential applications are discussed in Section 3 with Section 4 containing implications and details of the application to X.509 certificates. Details of the automated differential path construction for MD5 are provided in Section 5.

2 Chosen-prefix collisions for MD5

The main contribution of this paper is a method to construct MD5 collisions starting from two arbitrary IHVs. Given this method one can take any two chosen message prefixes and construct bitstrings that, when appended to the prefixes, turn them into two messages that collide under MD5. We refer to such a collision as a *chosen-prefix collision*. Their possibility was mentioned already in [3, Section 4.2 case 1] and, in the context of SHA-1, in [1] and on `www.iaik.tugraz.at/research/krypto/collision/`.

We start with a pair of arbitrarily chosen messages, not necessarily of the same length. Padding with random bits may be applied so that the padded messages have the same bitlength which equals 416 modulo 512 (incomplete last block). Equal length is unavoidable, because Merkle-Damgård strengthening, involving the message length, is applied after the last message block has been compressed by MD5. The incomplete last block condition is a technical requirement. In our example of colliding certificates the certificate contents were constructed in such a way that padding was not necessary, to allow for shorter RSA moduli.

Given the padded message pair, we followed a suggestion by Xiaoyun Wang[1] to find a pair of 96-bit values that, when used to complete the last blocks by appending them to the messages and applying the MD5 compression function, resulted in a specific form of difference vector between the IHVs. Finding these 96-bit values was done using a birthdaying procedure.

The remaining differences between the IHVs were then removed by appending *near-collision blocks*. Per pair of blocks this was done by constructing new differential paths using an automated, improved version of Wang's original approach. This innovative differential path construction is described in detail in Section 5 below. Due to the specific form of the near-collisions and the first difference vector, essentially one triple of bit differences could be removed per near-collision block, thus shortening the overall length of the colliding values. For our example 8 near-collision blocks were needed to remove all differences. Thus, a total of $96 + 8 \times 512 = 4192$ bits were appended to each of the chosen message prefixes to let them collide.

The birthdaying step can be entirely avoided, thereby making it harder to find the proper differential paths and considerably increasing the number of near-collision blocks. Or the birthdaying step could be simplified, increasing the number of near-collision blocks from 8 to about 14. Our approach was inspired by our desire to minimize the number of near-collision blocks. Using a more intricate differential path construction it should be possible to remove more than a single triple of bit differences per block, which would reduce the number of near-collision blocks. Potential enhancements and variations, and the full details of the construction as used, will be discussed in [16].

The expected complexity of the birthdaying step is estimated at 2^{49} MD5 compression function calls. Estimating the complexity of the near-collision block construction is hard, but it turned out to be a small fraction of the birthdaying complexity. Based on our observations we find it reasonable to estimate the overall expected complexity of finding a chosen-prefix collision for MD5 at about 2^{50} MD5 compression function calls. For the example we constructed, however, we had some additional requirements and also were rather unlucky in the birthdaying step, leading to about 2^{52} MD5 compression function calls. Note that, either way, this is substantially faster than the trivial birthday attack which has complexity 2^{64}.

[1] Private communication.

The construction of just a single example required, apart from the development of the automated differential path construction method, substantial computational efforts. Fortunately, the work is almost fully parallelizable and suitable for grid computing. It was done in the "HashClash" project (see `www.win.tue.nl/hashclash/`) and lasted about 6 months: using BOINC software (see `boinc.berkeley.edu/`) up to 1200 machines contributed, involving a cluster of computers at TU/e and a grid of home PCs. We expect that another chosen-prefix collision can be found much faster, but that it would again require substantial effort, both human and computationally: say 2 months real time assuming comparable computational resources.

3 Applications of chosen-prefix collisions

We mention some potential applications of chosen-prefix collisions.

- The example presented in the next section, namely colliding X.509 certificates with different fields before the appended bitstrings that cause the collision. Those bitstrings are 'perfectly' hidden inside the RSA moduli, where 'perfect' means that inspection of either one of the RSA moduli does not give away anything about the way it is constructed (namely, crafted such that it collides with the other one). In particular it could be of interest to be able to freely choose the Distinguished Name fields, which contain the identities of the alleged certificate owners.
- It was suggested to combine different Distinguished Names with equal public keys, to lure someone to encrypt data for one person, which can then be decrypted by another. It is unclear to us how realistic this is—or why one would need identical digital signatures. Nevertheless, if the appendages are not hidden in the public key field, some other field must be found for them, located before or after the public key field. Such a field may be specially defined for this purpose, and there is a good chance that the certificate processing software will not recognize this field and ignore it. However, as the appendages have non-negligible length, it will be hard to define a field that will not look suspicious to someone who looks at the certificate at bit level.
- A way to realize the above variant is to hide the collision-causing appendages in the public exponent. Though the public exponent is often taken from a small set (3, 17, and 65537 are common choices), a large, random looking one is in principle possible. It may even be larger than the modulus, but that may raise suspicion. In any case, the two certificates can now have identical RSA moduli, making it easy for the owner of one private key to compute the other one.
- Entirely different abuse scenarios are conceivable. In [2] (see also [4]) it was shown how to construct a pair of Postscript files that collide under MD5, and that send different messages to output media such as screen or printer. However, in those constructions both messages had to be hidden in each of the colliding files, which obviously raises suspicions upon inspection at bit level. With chosen-prefix collisions, this can be avoided. For example, two different messages can be entered into a document format that allows insertion of color images (such as Microsoft Word), with one message per document. At the last page of each document a color image will be shown—a short one pixel wide line will do, for instance hidden inside a layout element, a company logo, or in the form of a nicely colored barcode claiming to be some additional security feature, obviously offering far greater security than those old-fashioned black and white barcodes—carefully constructed such that the hashes of the documents collide when their color codes are inserted. In Figure 1 the actual 4192-bit collision-causing appendages computed for the certificates are built into bitmaps to get two different barcode examples. Each string of 4192

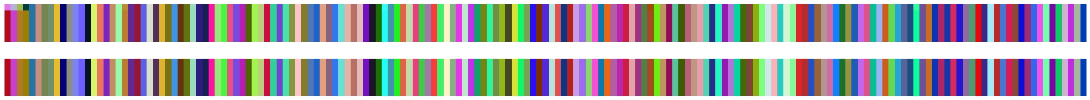

Fig. 1. A collision built into bitmap images.

bits leads to one line of 175 pixels, say A and B, and the barcodes consist of the lines ABBBBB and BBBBBB respectively. Apart from the 96 most significant bits corresponding to the 4 pixels in the upper left corner, the barcodes differ in only a few bits, which makes the resulting color differences hard to spot for the human eye. As noted above the 'obviously differing' 4 initial pixels can be avoided at the cost of more near-collision blocks (thus longer barcodes), and the barcodes can be shortened again at the cost of more elaborate differential path constructions.

- In [12] and [8] it was shown how to abuse existing MD5 collisions to mislead integrity checking software based on MD5. Similar to the colliding Postscript applications, they also used the differences in the colliding inputs to construct deviating execution flows of some programs. Here too chosen-prefix collisions allow a more elegant approach, especially since common operating systems ignore bitstrings that are appended to executables: the programs will run unaltered. Thus one can imagine two executables: a 'good' one (say Word.exe) and a bad one (the attacker's Worse.exe). A chosen-prefix collision for those executables is computed, and the collision-causing bitstrings are appended to them. The resulting altered file Word.exe, functionally equivalent to the original Word.exe, can then be offered to Microsoft's Authenticode signing program and receive an 'official' MD5 based digital signature. This signature will be equally valid for the attacker's Worse.exe, and the attacker might be able to replace Word.exe by his Worse.exe (renamed to Word.exe) on the appropriate download site. This construction affects a common functionality of MD5 hashing and may pose a practical threat, also because there is no a priori reason why the collision-causing bitstrings could not be hidden *inside* the executables.
- More ideas can be found on `www.iaik.tugraz.at/research/krypto/collision/`.

Further study is required to assess the impact of chosen-prefix collisions on applications of hash functions. Commonly used protocols and message formats such as SSL, S/MIME (CMS) and XML Signatures should be studied, with special attention to whether random looking data can be hidden in these protocols and data formats, in such a way that some or all implementations will not detect them. For instance, it was suggested by Pascal Junod to let a 'proper' certificate collide with one that contains executable code in the Distinguished Name field, thereby potentially triggering a buffer overflow, but we have not seen an actually working example of this idea yet. It also requires more study to see if there are formats that even allow the much easier random collision attacks.

4 Colliding X.509 certificates for different identities

In this section we concentrate on the first application mentioned above, that of two X.509 certificates with identical digital signatures but different Distinguished Name fields, where the collisions are perfectly hidden inside the public key moduli.

4.1 Attack scenarios

Though our current X.509 certificates construction, involving different Distinguished Names, should have more attack potential than the one with identical names fields in [11], we have not been able to find truly convincing attack scenarios yet. Ideally, a realistic attack targets the core of PKI: provide a relying party with trust, beyond reasonable cryptographic doubt, that the person indicated by the Distinguished Name field has exclusive control over the private key corresponding to the public key in the certificate. The attack should also enable the attacker to cover his trails.

Getting two certificates for the price of one could be economically advantageous in some situations, e.g. with two different owner names, or for two different validity periods. Such certificates undermine the proof of knowledge of the secret key corresponding to a certified public key. These possibilities have been noted before (cf. [10]) and do, in our opinion, not constitute attacks.

Our construction requires that the two colliding certificates are generated simultaneously. Although each resulting certificate by itself is completely unsuspicious, the fraud becomes apparent when the two certificates are put alongside, as may happen during a fraud analysis. An attacker can generate one of the certificates for a targeted person, the other one for himself, and attempt to use his own credentials to convince an external and generally trusted CA to sign the second one. If successful, the attacker can then distribute the first certificate, which will be trusted by relying parties, e.g. to encrypt messages for the targeted person. The attacker however is in control of the corresponding private key, and can thus decrypt confidential information embedded in intercepted messages meant for the targeted person. Or the attacker can masquerade as the targeted person while signing messages, which will be trusted by anyone trusting the CA. In this scenario it does not matter whether the two certificates have different public keys (as in our example) or identical ones (in which case the colliding blocks would have to be hidden somewhere else in the certificate).

A problem is, however, that the CA will register the attacker's identity. As soon as a dispute arises, the two certificates will be produced and revealed as colliding, and the attacker will be identified. Another problem is that the attacker must have sufficient control over the CA to predict all fields appearing before the public key, such as the serial number and the validity periods. It has frequently been suggested that this is an effective countermeasure against colliding certificate constructions in practice, but there is no consensus how hard it is to make accurate predictions. When this condition of sufficient control over the CA by the attacker is satisfied, colliding certificates based on chosen-prefix collisions are a bigger threat than those based on random collisions.

Obviously, the attack becomes effectively impossible if the CA adds a sufficient amount of fresh randomness to the certificate fields before the public key, such as in the serial number (as some already do, though probably for different reasons). This randomness is to be generated after the approval of the certification request. On the other hand, in general a relying party cannot verify this randomness. In our opinion, trustworthiness of certificates should not crucially depend on such secondary and circumstantial aspects. On the contrary, CAs should use a trustworthy hash function that meets the design criteria. Unfortunately, this is no longer the case for MD5, or SHA-1.

We stress that our construction (we prefer this wording to 'attack') is not a preimage attack. As far as we know, existing certificates cannot be forged by chosen-prefix collisions if they have not been especially crafted for that purpose. However, a relying party cannot distinguish any given trustworthy certificate from a certificate that has been crafted by our method to violate PKI principles. Therefore we repeat, with more urgency, our recommendation that MD5 is no longer used in new X.509 certificates. Similar work [1] is in development for the SHA-1 hash function, so we feel that a renewed assessment of the use of SHA-1 in certificate generation is also appropriate.

4.2 Certificate construction outline

Table 1 outlines the to-be-signed fields of the colliding certificates that were constructed.

Table 1. The to-be-signed parts of the colliding certificates.

field	comments	value first certificate	value second certificate
X.509 version number	identical, standard X.509	version 3	
serial number	different, chosen by CA	0x010C0001	0x020C0001
signature algorithm identifier	identical, standard X.509	md5withRSAEncryption	
issuer distinguished name	identical, chosen by CA	CN = "Hash Collision CA" L = "Eindhoven" C = "NL"	
not valid before	identical, chosen by CA	Jan. 1, 2006, 00h00m01s GMT	
not valid after	identical, chosen by CA	Dec. 31, 2007, 23h59m59s GMT	
subject distinguished name	different, chosen by us	CN = "Arjen K. Lenstra" O = "Collisionairs" L = "Eindhoven" C = "NL"	CN = "Marc Stevens" O = "Collision Factory" L = "Eindhoven" C = "NL"
public key algorithm	identical, standard X.509	rsaEncryption	
subject public key info	different, constructed by us	modulus $S_{\mathrm{b}} \| S_{\mathrm{c}} \| E$ as below	modulus $S'_{\mathrm{b}} \| S'_{\mathrm{c}} \| E$ as below
version 3 extensions	identical, standard X.509	(irrelevant for the present description)	

Here, S_{b} and S'_{b} are 96-bit values found using birthdaying, S_{c} and S'_{c} each consist of 8 near-collision blocks found using the automated method to find differential paths, and E is a 4000-bit value such that the 8192-bit values $S_{\mathrm{b}} \| S_{\mathrm{c}} \| E$ and $S'_{\mathrm{b}} \| S'_{\mathrm{c}} \| E$ are both RSA moduli. The details of the construction are set forth below.

Before the collision search (i.e., the searches for S_{b}, S'_{b} and for S_{c}, S'_{c}) is started the contents needs to be known of all to-be-signed fields of the certificate that appear before the modulus. Therefore, to be able to construct the certificates, sufficient control over the CA is necessary. This was achieved by implementing and operating this CA ourselves. In fact, we used the CA that had already been set up for [10]. It is used solely for the purposes of signing colliding certificates.

4.3 Certificate construction details

We provide a detailed description of our construction.

1. We construct two templates for the certificates in which all fields are filled in, with the exception of the RSA public key moduli and the signature, meeting the following three requirements:
   – The data structures must be compliant with the X.509 standard and the ASN.1 DER encoding rules (see [5], but see also the final section of [14]);
   – The byte lengths of the moduli and the public exponent (in fact, also the byte lengths of the entire to-be-signed parts of the certificates) must be fixed in advance, because these numbers have to be specified as parts of the ASN.1 structure, coming before the modulus;
   – The position where the RSA moduli start must be controlled. We chose to have this at an exact multiple of 64 bytes (512 bits) minus 96 bits, after the beginning of the to-be-signed fields. This gives convenient space for the results of the birthdaying step (described below).

   The third condition can be dealt with by adding dummy information to the subject Distinguished Name. This we did in the Organization-field (i.e., the value O in the outline above).
2. We apply MD5 to each of the first parts of the two to-be-signed fields, truncated at the last full block (thus excluding the incomplete blocks whose last 96 bits will consist of the most significant bits of the RSA moduli under construction), suppressing the padding normally used

in MD5. As output we get a pair of IHVs that we use as input for the next step. These IHVs will be completely different and have no special properties built in.

3. Using the IHVs and their corresponding incomplete blocks (the ones that still fail their last 96 bits) as input, we complete these blocks by appending 96-bit values S_b and S'_b. These values are computed by birthdaying, to satisfy 96 bit conditions on the output IHV difference. For this purpose each output IHV is interpreted as 4 little endian 32-bit integers, and the difference between the output IHVs is defined as the 4-tuple of differences modulo 2^{32} between the four corresponding 32-bit integers. If we represent this IHV difference as $\delta a \| \delta b \| \delta c \| \delta d$ for 32-bit $\delta a, \delta b, \delta c, \delta d$, then the conditions are $\delta a = 0$ and $\delta b = \delta c = \delta d$, as suggested to us by Xiaoyun Wang. The reason for this choice is that it facilitates the search for near-collision blocks, as explained in Section 5.3. The resulting δb can be expressed as only 8 signed bit differences (these are not bitwise XOR but additive differences).

4. Using the techniques developed in [16] and described in Section 5, we compute two different bitstrings S_c and S'_c, of 4096 bits (8 near-collision blocks) each. Each near-collision block is used to eliminate one (triple) of the bit differences in the IHVs from the previous step, so that at the end of the 8 near-collision blocks the IHVs are equal, and a complete MD5 collision has been constructed. We now have $S = S_\mathrm{b} \| S_\mathrm{c}$ and $S' = S'_\mathrm{b} \| S'_\mathrm{c}$ that form the leading 4192 bits of the RSA moduli, such that the two to-be-signed fields up to and including S and S', respectively, collide under MD5. Therefore, in order not to destroy the collision, everything that is to be appended from now on must be identical for the two certificates.

5. Next we used the method from [10] to craft two secure 8192-bit RSA moduli from the two bitstrings S and S' of 4192 bits each, by appending to each the same 4000-bit E. As explained in [11] this means that we could in principle construct moduli that are products of primes of sizes roughly 2000 and 6192 bits. In order to speed up the RSA modulus construction process, we aimed somewhat lower here and settled for products of 1976 and 6216-bit primes. This took about an hour on a regular laptop. The strongly unbalanced RSA moduli may be unusual, but for our parameter choices (smallest prime factor around 1976 bits for a modulus of 8192 bits) we see no reason to believe that these moduli are less secure than more balanced, regular RSA moduli of the same size, given the present state of factoring technology.

6. We insert the subject public key info into the template for the first certificate, thereby completing the to-be-signed part of the first certificate. We compute the MD5 hash of the entire to-be-signed part, and from it we compute the signature, which is added to the certificate. The first certificate is now complete. To obtain the second valid certificate, we put the proper subject public key info and the same signature at their locations in the template for the second certificate.

Finding the chosen-prefix MD5 collisions (i.e., Steps 3 and 4) is by far the computationally hardest part of the above construction, a remark that is similar to one made in [10]. However, in the meantime the methods for constructing MD5 collisions with *identical* initial IHVs have been improved considerably: such collisions can now be found within seconds, see [15] and [9]. So in the scenario of [10] the bottleneck may now have shifted from the collision search to the moduli construction.

An example pair of colliding certificates is available in full detail in [14] and on `www.win.tue.nl/hashclash/ChosenPrefixCollisions/`.

5 Chosen-prefix collision construction

5.1 Preliminaries

MD5 operates on 32-bit words, and uses *little endian* byte ordering.

A *binary signed digit representation* (BSDR) for a 32-bit word X is defined as $(k_i)_{i=0}^{31}$, where

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

Many different BSDRs may exist for any given X. The *weight* of a BSDR is the number of non-zero k_i's. A particularly useful BSDR is the Non-Adjacent Form (NAF), where no two non-zero k_i's are adjacent. The NAF is not unique since we work modulo 2^{32} (making $k_{31} = +1$ equivalent to $k_{31} = -1$), but uniqueness of the NAF can be enforced by choosing $k_{31} \in \{0, +1\}$. Among the BSDRs of an integer, the NAF has minimal weight. We use the following notation:

- Integers are denoted in hexadecimal as $\texttt{12EF}_{16}$ and in binary as $\texttt{0001001011101111}_2$;
- $X \wedge Y$ is the bitwise AND of X and Y;
- $X \vee Y$ is the bitwise OR of X and Y;
- $X \oplus Y$ is the bitwise XOR of X and Y;
- $\bar{X}$ is the bitwise complement of X;

for 32-bit integers X and Y:

- $X[i]$ is the i-th bit of the regular binary representation of X;
- $X + Y$ resp. $X - Y$ is the addition resp. subtraction modulo 2^{32};
- $RL(X, n)$ (resp. $RR(X, n)$) is the cyclic left (resp. right) rotation of X by n bit positions:

$$RL(\texttt{10100100}\ldots\texttt{00000001}_2,\ 5) = \texttt{10000000}\ldots\texttt{00110100}_2;$$

and for a 32-digit BSDR X:

- $X[\![i]\!]$ is the i-th signed bit of X;
- $RL(X, n)$ (resp. $RR(X, n)$) is the cyclic left (resp. right) rotation of X by n positions.

For chosen message prefixes P and P' we seek suffixes S and S' such that $P\|S$ and $P'\|S'$ collide under MD5. In this section a variable occurring during the construction of S and intermediate P-related MD5 calculations, may have a corresponding variable during the construction of S' and intermediate P'-related MD5 calculations. If the former variable is X, then the latter is denoted X'. Furthermore, $\delta X = X' - X$ for such a 'matched' 32-bit integer variable X, and $\Delta X = (X'[i] - X[i])_{i=0}^{31}$, which is a BSDR of δX. For a 'matched' variable Z that consist of tuples of 32-bit integers, say $Z = (z_1, z_2, \ldots)$, we define δZ as $(\delta z_1, \delta z_2, \ldots)$.

5.2 Description of MD5

5.2.1 MD5 message processing

MD5 can be split up into these parts:

1. *Padding.* Pad the message with: first the '1'-bit, next as many '0' bits until the resulting length equals 448 mod 512, and finally the bitlength of the original message as a 64-bit little-endian integer. The total bitlength of the padded message is $512N$ for a positive integer N.

2. *Partitioning.* Partition the padded message into N consecutive 512-bit blocks $M_1, M_2, \ldots, M_N$.
3. *Processing.* MD5 goes through $N+1$ states IHV_i, for $0 \leq i \leq N$, called the *intermediate hash values*. Each intermediate hash value IHV_i consists of four 32-bit words a_i, b_i, c_i, d_i. For $i = 0$ these are initialized to fixed public values:

   $$(a_0, b_0, c_0, d_0) = (\texttt{67452301}_{16}, \texttt{EFCDAB89}_{16}, \texttt{98BADCFE}_{16}, \texttt{10325476}_{16}),$$

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

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

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

5.2.2 MD5 compression function

The input for the compression function MD5Compress(IHV, B) is an intermediate hash value $IHV = (a, b, c, d)$ and a 512-bit message block B. There are 64 *steps* (numbered 0 up to 63), split into four consecutive *rounds* of 16 steps each. Each step uses a modular addition, a left rotation, and a non-linear function. Depending on the step t, *Addition Constants* AC_t and *Rotation Constants* RC_t are defined as follows:

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

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

The non-linear function f_t depends on the round:

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

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

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

We follow the description of the MD5 compression function from [6] because its 'unrolling' of the cyclic state facilitates the analysis. For $t = 0, 1, \ldots, 63$, the compression function algorithm maintains a working register with 4 state words Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3}. These are initialized as $(Q_0, Q_{-1}, Q_{-2}, Q_{-3}) = (b, c, d, a)$ and, for $t = 0, 1, \ldots, 63$ in succession, updated as follows:

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

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

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

5.3 Outline of the collision construction

A chosen-prefix collision 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 $MD5(M) = MD5(M')$. The suffixes consist of three parts: random padding bitstrings $S_\mathrm{r}, S'_\mathrm{r}$, followed by 'birthday' bitstrings $S_\mathrm{b}, S'_\mathrm{b}$, followed by 'near collision' blocks $S_\mathrm{c}, S'_\mathrm{c}$. The random padding bitstrings are chosen to guarantee that the bitlengths of $P\|S_\mathrm{r}\|S_\mathrm{b}$ and $P'\|S'_\mathrm{r}\|S'_\mathrm{b}$ are both equal to $512n$ for a positive integer n. (In our example of the colliding certificates we engineered the prefixes such that S_r and S'_r were both empty.) The MD5 compression function applied to $P\|S_\mathrm{r}\|S_\mathrm{b}$ resp. $P'\|S'_\mathrm{r}\|S'_\mathrm{b}$ will result in IHV_n resp. IHV'_n, in the notation from Section 5.2.1. The birthday bitstrings $S_\mathrm{b}, S'_\mathrm{b}$ are taken in such a way that the resulting δIHV_n has certain desirable properties, to be described below.

The idea is to eliminate the difference δIHV_n using a series of pairs of near-collision blocks that together constitute $S_\mathrm{c}, S'_\mathrm{c}$. For each near-collision we need to construct a differential path such that the NAF weight of the new δIHV_{n+j} is lower than the NAF weight of δIHV_{n+j-1}, until after r pairs of near-collision blocks we have reached $\delta IHV_{n+r} = 0$.

Table 2. Partial differential path with $\delta m_{11} = \pm 2^d$.

t	δQ_t	δF_t	δW_t	δT_t	δR_t	RC_t
30	$\mp 2^d$					
31	0					
32	0					
33	0	0	$\pm 2^d$	0	0	16
$34-60$	0	0	0	0	0	$\cdot$
61	0	0	$\pm 2^d$	$\pm 2^d$	$\pm 2^{d+10 \bmod 32}$	10
62	$\pm 2^{d+10 \bmod 32}$	0	0	0	0	15
63	$\pm 2^{d+10 \bmod 32}$	0	0	0	0	21
64	$\pm 2^{d+10 \bmod 32}$					

For the j-th pair of near-collision blocks, i.e., M_{n+j} and M'_{n+j}, we fix all but one of the 32-bit words δm_i of δM_{n+j} as 0, and allow only δm_{11} to be $\pm 2^d$ with varying d, $0 \leq d < 32$. 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. This is illustrated in Table 2, where the corresponding partial differential path is shown for the final 31 steps. For these types of message differences we try to find in an automated way a differential path with the right properties, and then try to find a pair of near-collision blocks M_{n+j}, M'_{n+j} that satisfies the differential path.

The differential paths under consideration can only add (or substract) a tuple $(0, 2^i, 2^i, 2^i)$ to δIHV_{n+j} and therefore cannot eliminate arbitrary δIHV_n. To solve this we first use a birthday attack to find 'birthday' bitstrings S_{b} and S'_{b} such that $\delta IHV_n = (0, \delta b, \delta b, \delta b)$ for some δb. The birthday attack actually searches for a collision $(a, b - c, b - d) = (a', b' - c', b' - d')$ between $IHV_n = (a, b, c, d)$ and $IHV'_n = (a', b', c', d')$, implying indeed $\delta a = 0$ and $\delta b = \delta c = \delta d$. The search space consists of 96 bits and therefore the birthday step can be expected to require on the order of $2\sqrt{2^{96}} = 2^{49}$ calls to the MD5 compression function.

One may extend the birthdaying by searching for a δb of low NAF weight, as this weight is the number of near-collision block pairs to be found. On average one may expect to find a δb of NAF weight 11. In the case of our colliding certificates example we found a δb of NAF weight only 8, after having extended the search somewhat longer than absolutely necessary.

Let (k_i) be the NAF of δb. Then we can reduce $\delta IHV_n = (0, \delta b, \delta b, \delta b)$ to $(0, 0, 0, 0)$ by using, for each non-zero k_i, a differential path based on the partial differential path in Table 2 with $\delta m_{11} = -k_i 2^{i-10 \bmod 32}$. In other words, the signed bit difference at position i in δb can be eliminated by choosing a message difference only in δm_{11}, with just one opposite-signed bit set at position $i - 10 \bmod 32$. Let i_j for $j = 1, 2, \ldots, r$ be the indices of the non-zero k_i. Starting with n-block $M = P \| S_{\mathrm{r}} \| S_{\mathrm{b}}$ and $M' = P' \| S'_{\mathrm{r}} \| S'_{\mathrm{b}}$ and the corresponding resulting IHV_n and IHV'_n we do the following for $j = 1, 2, \ldots, r$ in succession:

1. Let $\delta m_{11} = -k_{i_j} 2^{i_j - 10 \bmod 32}$ and $\delta m_\ell = 0$ for $\ell \neq 11$ (note the slight abuse of notation, since we define just the message block differences, without specifying the message blocks themselves).
2. Starting from IHV_{n+j-1} and IHV'_{n+j-1}, find a differential path.
3. Find message blocks $S_{\mathrm{c},j}$ and $S'_{\mathrm{c},j} = S_{\mathrm{c},j} + \delta M_{n+j}$, that satisfy the differential path. This can be done by using collision finding techniques such as Klima's tunnels, cf. [9] and [15].
4. Let $IHV_{n+j} = \mathrm{MD5Compress}(IHV_{n+j-1}, S_{\mathrm{c},j})$, $IHV'_{n+j} = \mathrm{MD5Compress}(IHV'_{n+j-1}, S'_{\mathrm{c},j})$, and append $S_{\mathrm{c},j}$ to M and $S'_{\mathrm{c},j}$ to M'.

It remains to explain step 2 in this algorithm.

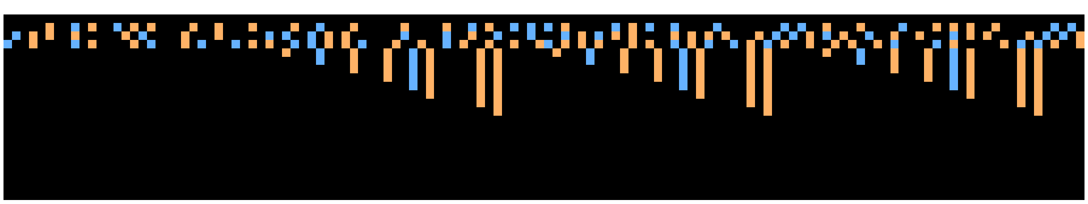

Fig. 2. δIHVs for the colliding certificates.

Figure 2 visualizes the entire process. The horizontal lines represent the NAFs of δIHV_i for $i = 0, 1, \ldots, 21$. The section $P \| S_{\mathrm{r}} \| S_{\mathrm{b}}$ consists of 4 blocks (i.e., $n = 4$), so at $i = 4$ only $r = 8$ triples of bit differences are left. They are annihilated one by one by the 8 near-collision block pairs (i.e., $S_{\mathrm{c},j}$ and $S'_{\mathrm{c},j}$ for $j = 1, 2, \ldots, 8$), so that at $i = 12$ a full collision is reached. The blocks after that (which include E from Section 4.3) are identical for the two messages, so that the collision is retained.

5.4 Differential paths and bitconditions

Assume MD5Compress is applied to pairs of inputs for both intermediate hash value and message block, i.e., to (IHV, B) and (IHV', B'). A differential path for MD5Compress is a precise description of the propagation of differences through the 64 steps caused by δIHV and δB:

$$\begin{aligned}
\delta F_t &= f_t(Q'_t, Q'_{t-1}, Q'_{t-2}) - f_t(Q_t, Q_{t-1}, Q_{t-2});\\
\delta T_t &= \delta F_t + \delta Q_{t-3} + \delta W_t;\\
\delta R_t &= RL(T'_t, RC_t) - RL(T_t, RC_t);\\
\delta Q_{t+1} &= \delta Q_t + \delta R_t.
\end{aligned}$$

Note that δF_t is not uniquely determined by $\delta Q_t, \delta Q_{t-1}$ and δQ_{t-2}, so it is necessary to describe the value of δF_t and how it can result from the Q_i, Q'_i in such a way that it does not conflict with other steps. Similarly δR_t is not uniquely determined by δT_t and RC_t, so also the value of δR_t has to be described.

5.4.1 Bitconditions

We use *bitconditions* on (Q_t, Q'_t) to describe differential paths, where a single bitcondition specifies directly or indirectly the values of the bits $Q_t[i]$ and $Q'_t[i]$. Thus a differential path consists of a matrix of bitconditions with 68 rows (for the possible indices $t = -3, -2, \ldots, 64$ in Q_t, Q'_t) and 32 columns (one for each bit). A *direct* bitcondition on $(Q_t[i], Q'_t[i])$ does not involve other bits $Q_j[k]$ or $Q'_j[k]$, while an *indirect* bitcondition does, and specifically one of $Q_{t-2}[i], Q_{t-1}[i], Q_{t+1}[i]$ or $Q_{t+2}[i]$. Using only bitconditions on (Q_t, Q'_t) we can specify all the values of δQ_t, δF_t and thus δT_t and $\delta R_t = \delta Q_{t+1} - \delta Q_t$ by the relations above. A bitcondition on $(Q_t[i], Q'_t[i])$ is denoted by $\mathfrak{q}_t[i]$, and symbols like $\mathtt{0}, \mathtt{1}, \mathtt{+}, \mathtt{-}, \mathtt{\hat{}}, \ldots$ are used for $\mathfrak{q}_t[i]$, as defined below. The 32 bitconditions $(\mathfrak{q}_t[i])_{i=0}^{31}$ are denoted by $\mathfrak{q}_t$. We discern between *differential* bitconditions and *boolean function* bitconditions. The former, shown in Table 3, are direct, and specify the value $k_i = Q'_t[i] - Q_t[i]$ which together

Table 3. 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: $\delta Q_t = \sum_{i=0}^{31} 2^i k_i$ and $\Delta Q_t = (k_i)$.

specify $\delta Q_t = \sum 2^i k_i$ by how each bit changes. Note that (k_i) is also a BSDR. The boolean function bitconditions, shown in Table 4, are used to resolve any ambiguity in

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

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

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

Table 4. 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

To resolve this ambiguity the bitcondition (`.`,`+`,`-`) can be replaced by (`0`,`+`,`-`) or (`1`,`+`,`-`).

All boolean function bitconditions include the constant bitcondition $Q_t[i] = Q'_t[i]$, so they do not affect δQ_t. Furthermore, indirect boolean function bitconditions never involve a bit with condition `+` or `-`, since then it could be replaced by one of the direct bitconditions `.`, `0` or `1`. We distinguish in the direction of indirect bitconditions, since that makes it easier to resolve an ambiguity later on. It is quite easy to change all backward bitconditions into forward ones in a valid (partial) differential pathm, and vice versa.

When all δQ_t and δF_t are determined by bitconditions then also δT_t and δR_t can be determined, which together describe the bitwise rotation of δT_t in each step. Note that this does not describe if it is a valid rotation or with what probability the rotation from δT_t to δR_t occurs. The differential paths we constructed for our example can be found at `www.win.tue.nl/hashclash/ChosenPrefixCollisions/`.

5.4.2 Differential path construction overview

The basic idea in constructing a differential path is to construct a partial lower differential path over steps $t = 0, 1, \ldots, 11$ and a partial upper differential path over steps $t = 16, 17, \ldots, 63$, so that the Q_i involved in the partial paths meet but do not overlap. Then try to connect those partial paths over the remaining 4 steps into one full differential path. Constructing the partial lower path can be done by starting with bitconditions $\mathtt{q}_{-3}$, $\mathtt{q}_{-2}$, $\mathtt{q}_{-1}$, $\mathtt{q}_0$ that are equivalent to the values of IHV, IHV' and then extend this step by step. Similarly the partial upper path can be constructed by extending the partial path in Table 2 step by step. To summarize, step 2 in the algorithm of section 5.3 consist of the following substeps:

2.1 Using IHV and IHV' determine bitconditions $(\mathtt{q}_i)_{i=-3}^{0}$.
2.2 Generate a partial lower differential path by extending $(\mathtt{q}_i)_{i=-3}^{0}$ forward up to step $t = 11$.
2.3 Generate a partial upper differential path by extending the path in Table 2 down to $t = 16$.
2.4 Try to connect these lower and upper differential paths over $t = 12, 13, 14, 15$.
If this fails generate other partial lower and upper differential paths and try again.

5.5 Extending differential paths

When constructing a differential path one must fix the message block differences $\delta m_0, \ldots, \delta m_{15}$. Suppose we have a partial differential path consisting of at least bitconditions $\mathtt{q}_{t-1}$ and $\mathtt{q}_{t-2}$ and that the values δQ_t and δQ_{t-3} are known. We want to extend this partial differential path forward with step t resulting in the value δQ_{t+1} and (additional) bitconditions $\mathfrak{q}_t, \mathfrak{q}_{t-1}, \mathfrak{q}_{t-2}$. We assume that all indirect bitconditions are forward and do not involve bits of Q_t. If we also have $\mathfrak{q}_t$ instead of only the value δQ_t (e.g. $\mathfrak{q}_0$ resulting from given values IHV, IHV'), then we can skip the carry propagation and continue at Section 5.5.2.

5.5.1 Carry propagation

First we want to use the value δQ_t to select bitconditions $\mathfrak{q}_t$. This can be done by choosing any BSDR of δQ_t, which directly translates into a possible choice for $\mathfrak{q}_t$ as given in Table 3. Since we want to construct differential paths with as few bitconditions as possible, but also want to be able to randomize the process, we may choose any low weight BSDR (such as the NAF).

5.5.2 Boolean function

For some i, let $(a, b, c) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ be any triple of bitconditions such that all indirect bitconditions involve only $Q_t[i]$, $Q_{t-1}[i]$ or $Q_{t-2}[i]$. The triple (a, b, c) is associated with the set U_{abc} of tuples of values $(x, x', y, y', z, z') = (Q_t[i], Q'_t[i], Q_{t-1}[i], Q'_{t-1}[i], Q_{t-2}[i], Q'_{t-2}[i])$:

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

If $U_{abc} = \emptyset$ then (a, b, c) is said to be contradicting and cannot be part of any valid differential path. We define $\mathcal{F}_t$ as the set of all triples (a, b, c) such that all indirect bitconditions involve only $Q_t[i]$, $Q_{t-1}[i]$ or $Q_{t-2}[i]$ and $U_{abc} \neq \emptyset$.

We define V_{abc} as the set of all possible boolean function differences $f_t(x', y', z') - f_t(x, y, z)$ for given bitconditions $(a, b, c) \in \mathcal{F}_t$:

$$V_{abc} = \left\{f_t(x', y', z') - f_t(x, y, z) \mid (x, x', y, y', z, z') \in U_{abc}\right\} \subset \{-1, 0, +1\}.$$

If $|V_{abc}| = 1$ then (a, b, c) leaves no ambiguity and the triple (a, b, c) is said to be a *solution*. Let $\mathcal{S}_t$ be the set of all solutions. If $|V_{abc}| > 1$ then for each $g \in V_{abc}$ we define $W_{abc,g}$ as the set of solutions $(d, e, f) \in \mathcal{S}_t$ that are compatible with (a, b, c) and that have g as boolean function difference:

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

Note that for all $g \in V_{abc}$ there is always a triple $(d, e, f) \in W_{abc,g}$ that consists only of direct bitconditions `01+-`, hence $W_{abc,g} \neq \emptyset$. The direct and forward (resp. backward) boolean function bitconditions were chosen such that for all t, i and $(a, b, c) \in \mathcal{F}_t$ and for all $g \in V_{abc}$ there exists a triple $(d, e, f) \in W_{abc,g}$ consisting of direct and forward (resp. backward) bitconditions such that

$$U_{def} \text{ is equal to } \left\{(x, x', y, y', z, z') \in U_{abc} \mid f_t(x', y', z') - f_t(x, y, z) = g\right\}.$$

In other words, these boolean function bitconditions allows one to resolve an ambiguity in an optimal way. If the triple (d, e, f) is not unique, then we prefer direct over indirect bitconditions and short indirect bitconditions (`vy^!`) over long indirect bitconditions (`whqm#?`) for simplicity reasons. For given t, bitconditions (a, b, c), and $g \in V_{abc}$ we define $FC(t, abc, g) = (d, e, f)$ and $BC(t, abc, g) = (d, e, f)$ as the preferred triple (d, e, f) consisting of direct and forward, respectively backward bitconditions. These should be precomputed for all cases.

For all $i = 0, 1, \ldots, 31$ we have by assumption valid bitconditions $(a, b, c) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ where only c can be an indirect bitcondition. If so, it must involve $Q_{t-1}[i]$. Therefore $(a, b, c) \in \mathcal{F}_t$.

If $|V_{abc}| = 1$ there is no ambiguity and we let $\{g_i\} = V_{abc}$. Otherwise, if $|V_{abc}| > 1$, then we choose any $g_i \in V_{abc}$ and we resolve the ambiguity left by bitconditions (a, b, c) by replacing them by $(d, e, f) = FC(t, abc, g_i)$, which results in boolean function difference g_i. Given all g_i, the values $\delta F_t = \sum_{i=0}^{31} 2^i g_i$ and $\delta T_t = \delta F_t + \delta Q_{t-3} + \delta W_t$ can be determined.

5.5.3 Bitwise rotation

The integer δT_t does not uniquely determine the value of $\delta R_t = RL(T'_t, n) - RL(T_t, n)$, where $n = RC_t$. Nevertheless, we simply use $\delta R_t = RL(NAF(\delta T_t), n)$ and determine $\delta Q_{t+1} = \delta Q_t + \delta R_t$ to extend our partial differential path forward with step t.

Another approach to determine δR_t uses the fact that any BSDR (k_i) of δT_t determines δR_t:

$$\delta R_t = \sum_{i=0}^{31} 2^{i+n \bmod 32}(T'_t[i] - T_t[i]) = \sum_{i=0}^{31} 2^{i+n \bmod 32} k_i = 2^n \sum_{i=0}^{31-n} 2^i k_i + 2^{n-32} \sum_{i=32-n}^{31} 2^i k_i.$$

Different BSDRs (k_i) and (ℓ_i) of δT_t result in the same δR_t as long as

$$\sum_{i=0}^{31-n} 2^i k_i = \sum_{i=0}^{31-n} 2^i \ell_i \quad \text{and} \quad \sum_{i=32-n}^{31} 2^i k_i = \sum_{i=32-n}^{31} 2^i \ell_i.$$

In general, let $(\alpha, \beta) \in \mathbb{Z}^2$ be a partition of the integer δT_t with $\alpha + \beta = \delta T_t \mod 2^{32}$, $|\alpha| < 2^{32-n}$, $|\beta| < 2^{32}$ and $2^{32-n} | \beta$. For a BSDR (k_i) of δT_t we say that $(\alpha, \beta) \equiv (k_i)$ if $\alpha = \sum_{i=0}^{31-n} 2^i k_i$ and $\beta = \sum_{i=32-n}^{31} 2^i k_i$. The rotation of (α, β) is defined as $RL((\alpha, \beta), n) = 2^n \alpha + 2^{n-32} \beta \mod 2^{32}$.

Let $x = (\delta T_t \mod 2^{32-n})$ and $y = (\delta T_t - x \mod 2^{32})$, then $0 \le x < 2^{32-n}$ and $0 \le y < 2^{32}$. This gives rise to at most 4 partitions of δT_t:

1. $(\alpha, \beta) = (x, y)$;
2. $(\alpha, \beta) = (x, y - 2^{32})$, if $y \neq 0$;
3. $(\alpha, \beta) = (x - 2^{32-n}, y + 2^{32-n} \mod 2^{32})$, if $x \neq 0$;
4. $(\alpha, \beta) = (x - 2^{32-n}, (y + 2^{32-n} \mod 2^{32}) - 2^{32})$, if $x \neq 0$ and $y + 2^{32-n} \neq 0 \mod 2^{32}$.

The probability of each partition (α, β) equals

$$p_{(\alpha,\beta)} = \sum_{(k_i) \equiv (\alpha,\beta)} 2^{-\text{weight of } (k_i)}.$$

One then chooses any partition (α, β) for which $p_{(\alpha,\beta)} \ge \frac{1}{4}$ and determines δR_t as $RL((\alpha, \beta), n)$. In practice $NAF(\delta T)$ most often leads to the highest probability, which validates the simpler approach we used.

5.5.4 Extending backward

Similar to extending forward, 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 δQ_{t-3} and (additional) bitconditions $\mathfrak{q}_t, \mathfrak{q}_{t-1}, \mathfrak{q}_{t-2}$. We assume that all indirect bitconditions are backward and do not involve bits of Q_{t-2}.

We choose a BSDR of δQ_{t-2} with weight at most 1 or 2 above the lowest weight, such as the NAF. We translate the chosen BSDR into bitconditions $\mathfrak{q}_{t-2}$.

For all $i = 0, 1, \ldots, 31$ we have by assumption valid bitconditions $(a, b, c) = (\mathfrak{q}_t[i], \mathfrak{q}_{t-1}[i], \mathfrak{q}_{t-2}[i])$ where only b can be an indirect bitcondition. If so, it must involve $Q_{t-2}[i]$. Therefore $(a, b, c) \in \mathcal{F}_t$. If $|V_{abc}| = 1$ there is no ambiguity and we let $\{g_i\} = V_{abc}$. Otherwise, if $|V_{abc}| > 1$, then we choose any $g_i \in V_{abc}$ and we resolve the ambiguity left by bitconditions (a, b, c) by replacing them by $(d, e, f) = BC(t, abc, g_i)$, which results in boolean function difference g_i. Given all g_i, the value $\delta F_t = \sum_{i=0}^{31} 2^i g_i$ can be determined.

To rotate $\delta R_t = \delta Q_{t+1} - \delta Q_t$ over $n = 32 - RC_t$ bits, we simply use $\delta T_t = RL(NAF(\delta R_t), n)$. Or we may choose a partition (α, β) of δR_t with $p_{(\alpha,\beta)} \geq \frac{1}{4}$ and determine $\delta T_t = RL((\alpha, \beta), n)$. As in the 'forward' case, $NAF(\delta R_t)$ often leads to the highest probability. 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.

5.6 Constructing full differential paths

Construction of a full differential path can be done as follows. Choose δQ_{-3} and bitconditions $\mathfrak{q}_{-2}$, $\mathfrak{q}_{-1}$, $\mathfrak{q}_0$ and extend forward up to step 11. Also choose δQ_{64} and bitconditions $\mathfrak{q}_{63}, \mathfrak{q}_{62}, \mathfrak{q}_{61}$ and extend backward down to step 16. This leads to bitconditions $\mathfrak{q}_{-2}, \mathfrak{q}_{-1}, \ldots, \mathfrak{q}_{11}, \mathfrak{q}_{14}, \mathfrak{q}_{15}, \ldots, \mathfrak{q}_{63}$ and differences $\delta Q_{-3}, \delta Q_{12}, \delta Q_{13}, \delta Q_{64}$. It remains to finish steps $t = 12, 13, 14, 15$. As with extending backward we can, for $t = 12, 13, 14, 15$, determine δR_t, choose the resulting δT_t after right rotation of δR_t over RC_t bits, and determine $\delta F_t = \delta T_t - \delta W_t - \delta Q_{t-3}$.

We aim to find new bitconditions $\mathfrak{q}_{10}, \mathfrak{q}_{11}, \ldots, \mathfrak{q}_{15}$ that are compatible with the original bitconditions and that result in the required δQ_{12}, δQ_{13}, δF_{12}, δF_{13}, δF_{14}, δF_{15}, thereby completing the differential path. First we can test whether it is even possible to find such bitconditions.

For $i = 0, 1, \ldots, 32$, let $\mathcal{U}_i$ be a set of tuples $(q_1, q_2, f_1, f_2, f_3, f_4)$ of 32-bit integers with $q_j \equiv f_k \equiv 0 \mod 2^i$ for $j = 1, 2$ and $k = 1, 2, 3, 4$. We want to construct each $\mathcal{U}_i$ so that for each tuple $(q_1, q_2, f_1, f_2, f_3, f_4) \in \mathcal{U}_i$ there exist bitconditions $\mathfrak{q}_{10}[\ell], \mathfrak{q}_{11}[\ell], \ldots, \mathfrak{q}_{15}[\ell]$, determining the $\Delta Q_{11+j}[\![\ell]\!]$ and $\Delta F_{11+k}[\![\ell]\!]$ below, over the bits $\ell = 0, \ldots, i-1$, such that

$$\delta Q_{11+j} = q_j + \sum_{\ell=0}^{i-1} 2^\ell \Delta Q_{11+j}[\![\ell]\!], \quad j = 1, 2,$$

$$\delta F_{11+k} = f_k + \sum_{\ell=0}^{i-1} 2^\ell \Delta F_{11+k}[\![\ell]\!], \quad k = 1, 2, 3, 4.$$

This implies $\mathcal{U}_0 = \{(\delta Q_{12}, \delta Q_{13}, \delta F_{12}, \delta F_{13}, \delta F_{14}, \delta F_{15})\}$. The other $\mathcal{U}_i$ are constructed inductively by Algorithm 1. Furthermore, $|\mathcal{U}_i| \leq 2^6$, since for each q_j, f_k there are at most 2 possible values that can satisfy the above relations.

If we find $\mathcal{U}_{32} \neq \emptyset$ then there exists a path $u_0, u_1, \ldots, u_{32}$ with $u_i \in \mathcal{U}_i$ where each u_{i+1} is generated by u_i in Algorithm 1. Now the desired new bitconditions $(\mathfrak{q}_{15}[i], \mathfrak{q}_{14}[i], \ldots, \mathfrak{q}_{10}[i])$ are $(a', b'', c''', d''', e'', f')$, which can be found at step 13 of Algorithm 1, where one starts with u_i and ends with u_{i+1}.

5.7 Implementation details

Implementation of these techniques was done in C++ using the general purpose library Boost and the BOINC framework. BOINC is an open source distributed computing framework that allows volunteers on the Internet to join a project and donate cpu-time. Each project running a BOINC

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

Suppose $\mathcal{U}_i$ is constructed as desired in Section 5.6.
Let $\mathcal{U}_{i+1} = \emptyset$ and $(a, b, e, f) = (\mathfrak{q}_{15}[i], \mathfrak{q}_{14}[i], \mathfrak{q}_{11}[i], \mathfrak{q}_{10}[i])$.
For each tuple $(q_1, q_2, f_1, f_2, f_3, f_4) \in \mathcal{U}_i$ do the following:

For each bitcondition $d = \mathfrak{q}_{12}[i] \in \begin{cases} \{\texttt{.}\} & \text{if } q_1[i] = 0 \\ \{\texttt{-}, \texttt{+}\} & \text{if } q_1[i] = 1 \end{cases}$ do
  Let $q_1' = 0, -1, +1$ for resp. $d = \texttt{.}, \texttt{-}, \texttt{+}$
  For each different $f_1' \in \{-f_1[i], +f_1[i]\} \cap V_{def}$ do
    Let $(d', e', f') = FC(12, def, f_1')$
    For each bitcondition $c = \mathfrak{q}_{13}[i] \in \begin{cases} \{\texttt{.}\} & \text{if } q_2[i] = 0 \\ \{\texttt{-}, \texttt{+}\} & \text{if } q_2[i] = 1 \end{cases}$ do
      Let $q_2' = 0, -1, +1$ for resp. $c = \texttt{.}, \texttt{-}, \texttt{+}$
      For each different $f_2' \in \{-f_2[i], +f_2[i]\} \cap V_{cd'e'}$ do
        Let $(c', d'', e'') = FC(13, cd'e', f_2')$
        For each different $f_3' \in \{-f_3[i], +f_3[i]\} \cap V_{bc'd''}$ do
          Let $(b', c'', d''') = FC(14, bc'd'', f_3')$
          For each different $f_4' \in \{-f_4[i], +f_4[i]\} \cap V_{ab'c''}$ do
            Let $(a', b'', c''') = FC(15, ab'c'', f_4')$
            Insert $(q_1 - 2^i q_1', q_2 - 2^i q_2', f_1 - 2^i f_1', f_2 - 2^i f_2', f_3 - 2^i f_3', f_4 - 2^i f_4')$ into $\mathcal{U}_{i+1}$.

Keep only one of each tuple in $\mathcal{U}_{i+1}$ that occurs multiple times. By construction we find $\mathcal{U}_{i+1}$ as desired.

server automatically handles compute-client inputs and outputs specific to any number of applications, including output validation and re-assignment of jobs, if required. Volunteers, which can form teams, can monitor their own and others' progress, thus providing an inspiring competitive environment. Our BOINC project had a peak performance of approximately 400 GigaFLOPS.

To construct our chosen-prefix collision we used six applications:

1. One that generates birthday trails ending in a distinguished point [13];
2. One that collects birthday trails and computes collisions when found;
3. One that loads a set of partial lower differential paths and extends those forward with step t and saves only the paths with the fewest bitconditions;
4. One that loads a set of partial upper differential paths and extends those backward with step t and saves only the paths with the fewest bitconditions;
5. One that loads sets of lower and upper differential paths and tries to connect each combination;
6. One that searches for near-collision blocks that satisfy a given full differential path.

While extending a partial differential path we exhaustively try all BSDRs of δQ_t with weight at most 2 above the lowest weight, and all possible δF_t and all high-probability rotations. We keep only the N paths with the fewest bitconditions, for some preset value of N. Also we keep only those paths that have a preset minimum total tunnel strength over the Q_4, Q_5, Q_9, Q_{10}-tunnels [9]. With the exception of the 2nd, all applications can be fully parallelized. For the 1st and 5th application, which were by far the most cpu-time consuming, we used BOINC; the others were run on a cluster.

6 Concluding remark

We have presented an automated way to find differential paths for MD5, have shown how to use them to construct chosen-prefix collisions, and have constructed two X.509 certificates with different name fields but idential signatures. Our construction required substantial cpu-time, but chosen-prefix collisions can be constructed much faster by using a milder birthday condition (namely, just $\delta a = 0$ and $\delta c = \delta d$) and allowing more near-collision blocks (about 14). See [16] for details.

Acknowledgements

This work benefited greatly from suggestions by Xiaoyun Wang. We are grateful for comments and assistance received from the Eurocrypt 2007 reviewers, Bart Asjes, Stuart Haber, Paul Hoffman, Pascal Junod, Vlastimil Klima, Bart Preneel, NBV, Gido Schmitz, Eric Verheul, and Yiqun Lisa Yin. Finally, we thank hundreds of BOINC enthousiasts all over the world, most unknown to us, who donated an impressive amount of cycles to the HashClash project running with BOINC software.

References

1. Christophe de Cannière and Christian Rechberger, *Finding SHA-1 Characteristics: General results and applications*, AsiaCrypt 2006, Springer LNCS 4284 (2006), 1–20.
2. M. Daum and S. Lucks, *Attacking Hash Functions by Poisoned Messages, "The Story of Alice and her Boss"*, June 2005, `www.cits.rub.de/MD5Collisions/`.
3. P. Gauravaram, A. McCullagh and E. Dawson, *Collision Attacks on MD5 and SHA-1: Is this the "Sword of Damocles" for Electronic Commerce?*, AusSCERT 2006 R&D Stream, May 2006, `www.isi.qut.edu.au/people/subramap/AusCert-6.pdf`.
4. M. Gebhardt, G. Illies and W. Schindler, *A Note on Practical Value of Single Hash Collisions for Special File Formats*, NIST First Cryptographic Hash Workshop, October/November 2005, `csrc.nist.gov/pki/HashWorkshop/2005/Oct31%5FPresentations/Illies%5FNIST%5F05.pdf`.
5. R. Housley, W. Polk, W. Ford and D. Solo, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, IETF RFC 3280, April 2002, `www.ietf.org/rfc/rfc3280.txt`.
6. Philip Hawkes, Michael Paddon and Gregory G. Rose, *Musings on the Wang et al. MD5 Collision*, Cryptology ePrint Archive, Report 2004/264, `eprint.iacr.org/2004/264`.
7. P. Hoffman and B. Schneier, *Attacks on Cryptographic Hashes in Internet Protocols*, IETF RFC 4270, November 2005, `www.ietf.org/rfc/rfc4270.txt`.
8. D. Kaminsky, *MD5 to be considered harmful someday*, December 2004, `www.doxpara.com/md5%5Fsomeday.pdf`.
9. Vlastimil Klima, *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, Cryptology ePrint Archive, Report 2006/105, `eprint.iacr.org/2006/105`.
10. A.K. Lenstra, X. Wang and B.M.M. de Weger, *Colliding X.509 certificates*, Cryptology ePrint Archive, Report 2005/067, `eprint.iacr.org/2005/067`. An updated version has been published as an appendix to [11].
11. A.K. Lenstra and B.M.M. de Weger, *On the possibility of constructing meaningful hash collisions for public keys*, ACISP 2005, Springer LNCS 3574 (2005), 267–279.
12. O. Mikle, *Practical Attacks on Digital Signatures Using MD5 Message Digest*, Cryptology ePrint Archive, Report 2004/356, `eprint.iacr.org/2004/356`.
13. Paul C. van Oorschot and Michael J. Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12**(1), 1–28, 1999.
14. Marc Stevens, Arjen Lenstra and Benne de Weger, *Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities*, Cryptology ePrint Archive, Report 2006/360, `eprint.iacr.org/2006/360`.
15. Marc Stevens, *Fast Collision Attack on MD5*, Cryptology ePrint Archive, Report 2006/104, `eprint.iacr.org/2006/104`.
16. Marc Stevens, TU Eindhoven MSc thesis, in preparation. See `www.win.tue.nl/hashclash/`.
17. X. Wang and H. Yu, *How to Break MD5 and Other Hash Functions*, EuroCrypt 2005, Springer LNCS 3494 (2005), 19–35.