Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities

version 1.1, 4th November 2006

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 and Bell Laboratories
INJ 330 (Bâtiment INJ), Station 14
CH-1015 Lausanne, Switzerland

Abstract. We have shown how, at a cost of about 2^{52} calls to the MD5 compression function, for any two target messages m_1 and m_2, values b_1 and b_2 can be constructed such that the concatenated values $m_1 \| b_1$ and $m_2 \| b_2$ collide under MD5. Although the practical attack potential of this construction of *target collisions* is limited, it is of greater concern than random collisions for MD5. In this note we sketch our construction. To illustrate its practicality, we present 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 target collisions.

Announcement

In March 2005 we showed how Xiaoyun Wang's ability 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 the announcement [9], more technical information on the website `http://www.win.tue.nl/%7Ebdeweger/CollidingCertificates/`, and [10] for a broader theoretical description). 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

- at the time, Wang's hash collision construction required 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. [6], [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 at will Distinguished Name fields that are different, but non-random and human readable as one would expect from these fields. This can be realized if two arbitrary messages, resulting in two different IHVs, can be extended in such a way that the extended messages collide. Such collisions will be called *target collisions*. It is exactly the construction of such a target collision which has recently been completed by the first author. The full details of his work will be reported elsewhere, cf. [14].

In this note we sketch how target collisions for MD5 can be constructed, and we illustrate this by presenting a method to construct two MD5 based X.509 certificates with different Distinguished Name fields and identical digital signatures. To show that our methods are indeed practical, we have constructed an actual pair of such certificates with explicitly targeted Distinguished Name fields. The certificates are available for download from `http://www.win.tue.nl/hashclash/TargetCollidingCertificates/`. Below we describe their contents in full detail, as well as the way we constructed them.

Target Collisions

The main ingredient of our construction is a method, developed as part of the work on [14], to construct MD5 collisions starting from two arbitrary IHVs. Given this method one can take any two targeted messages and construct bitstrings that, when appended to the messages, turn them into MD5 collisions. We refer to such a collision as a *target collision*. Their possibility was mentioned already in [3, Section 4.2 case 1] and, in the context of SHA-1, in [1] and on `http://www.iaik.tugraz.at/research/krypto/collision/`. We are aware of the fact that terms similar to 'target collision' have been used before in different hash-related contexts – we are grateful to Bart Preneel for pointing this out too – but this will not lead to misunderstandings as far as this note is concerned.

In somewhat more detail, we started with a pair of arbitrarily chosen messages satisfying two conditions:

- they have equal bitlength,
- the bitlength equals 416 modulo 512 (incomplete last block).

The condition of equal bitlength seems unavoidable, because Merkle-Damgård strengthening, involving the message length, is applied after the last message block has been compressed by MD5. The second condition (incomplete last block) is not essential, as one can always add additional random bits to satisfy it, but we keep it for ease of exposition and to allow for shorter RSA moduli.

Given the message pair, we followed a suggestion by Xiaoyun Wang[1] to find a pair of 96-bit values that, when appended to the messages, resulted in a specific form of difference vector between the IHVs when the MD5 compression function was applied to the completed blocks. Finding this pair of 96-bit values was done using a birthdaying procedure. The 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 a semi-automated, improved version of Wang's original approach. Due to the specific form of the first difference vector, essentially one triple of bit differences was removed per near-collision block, thus shortening the overall length of the colliding values. For our example 8 additional 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 targeted messages to let them collide. The overall expected complexity of the target collision method for MD5 is estimated at about 2^{52} MD5 compression function calls. Note that this is substantially faster than the trivial birthday attack which has complexity 2^{64}.

In principle it is possible to omit the initial birthdaying step, but as a result finding the proper differential paths would become harder, and quite a few more additional blocks would be needed. A different and easier birthdaying procedure could have been used instead, and would have required about 14 additional blocks. Our approach reflected our desire to minimize the number of additional blocks using the new differential path construction method. Using a more intricate differential path construction we should be able to remove more than a single triple of bit differences per block, thereby further reducing the number of additional blocks. These potential enhancements and variations, and the full details of the construction as used, will be published shortly in [14]. Further announcements on this subject will appear on the website `http://www.win.tue.nl/hashclash/`, along with the thesis [14].

[1] Private communication.

The construction of just a single example required, apart from intensive study of the construction of differential paths, substantial computational efforts. This was done in the "HashClash" project (see `http://www.win.tue.nl/hashclash`), in which we needed about 6 months of real time, during which we employed a high performance cluster of computers at TU/e as well as a grid of home PCs, sometimes involving up to 1200 machines, using BOINC software (see `http://boinc.berkeley.edu/`). The computational work is almost fully parallelizable, and very well suited for grid computing. Constructing another target collision can probably be done much faster. Nevertheless, we expect that it will again require a substantial effort, both human and computational work, say 2 months real time assuming comparable computational resources.

Applications of target collisions

Given two target messages of equal length, we can effectively construct relatively short appendages in such a way that the extended messages collide under MD5. We mention the following potential applications of such a construction.

- The example presented here, namely colliding X.509 certificates with different fields before the appended bitstrings that cause the collision, where those bitstrings are perfectly hidden inside the RSA moduli. 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 us to keep the different Distinguished Names, but to insist on equal public keys: someone may be lured 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 possible way to realize the above variant is to hide the collision-causing appendages inside the RSA public exponent. Though the public exponent is commonly taken from a limited set (3, 17, and 65537 are popular 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. Daum and Lucks [2] (see also Gebhardt, Illies and Schindler [4]) have 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 target 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 colored layout element will be shown—for instance a company logo or 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 appended. The images in Figure 1 below are based on 4192-bit actual collision-causing appendages. In fact, we just took the collision computed for the certificates and built them into bitmaps to get two different barcode examples. Each string of 4192 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, they differ in only a few bits, so the resulting color differences will be hard to spot for the human eye. As

noted above the 'obvious' 4 initial pixels can be avoided at the cost of more blocks (thus longer barcodes), and the barcodes can be shortened again at the cost of more work on differential path constructions.

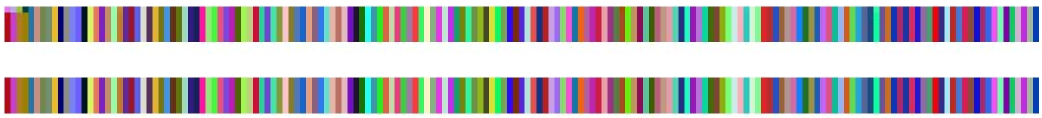

Figure 1. A collision built into a bitmap images.

– Mikle [11] and Kaminsky [7] have 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 target collisions allow a more elegant approach, especially since common operating systems ignore any bitstring that is appended to an executable: the program will run unaltered. Thus one can imagine two executables: a 'good' one (say Microsoft's Word.exe) and a bad one (the attacker's Worse.exe). A target collision for those two executable files 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 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 `http://www.iaik.tugraz.at/research/krypto/collision/`.

Further study is required to assess the impact of target collisions on these and other 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 to us 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.

In the remainder of this note 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.

Attack scenarios

Though our current X.509 certificates construction, involving different Distinghuished Names, should have more attack potential than our previous one in [10] (with identical name fields), 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. Also, such certificates undermine the proof of knowledge of the secret key corresponding to a certified public key. Both these possibilities have been noted before (cf. [9]) 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 target 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 target 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. As shown in [1], similar work 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.

Construction outline

The table below outlines the to-be-signed fields of the colliding certificates that were constructed.

field	comments	value first certificate	value second certificate
X.509 version number	identical, standard X.509	0x02, indicating 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, see below	as specified below	as specified below
version 3 extensions	identical, standard X.509	see below	

Before the collision search is started the exact 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 [9]. It is used solely for the purposes of signing colliding certificates.

Below we explain in more detail how each of the fields was determined. For this purpose it is helpful to know that the Subject Public Key Info was split in the following four parts:

Part 1, the 96 most significant bits of the RSA modulus. This part coincides with the last 96 bits of a 512-bit block of MD5 input during the certificate digital signature generation. This part is computed by birthdaying and will be 'entirely' (i.e., approximately half) different for the two certificates. The resulting IHVs have only 8 triples of bit differences (these are not bitwise xor differences but the additive differences of the IHVs, where each IHV is interpreted as a quadruple of 32-bit unsigned integers).

Part 2, the next $8 \times 512 = 4096$ bits of the RSA modulus, with each of the eight 512-bit near-collision blocks computed by a collision finding method: each near-collision block is used to eliminate one triple of the bit differences in the IHVs, so that at the end of the 8 near-collision blocks the IHVs are equal, and a complete collision has been constructed. This part of the moduli is different for the two certificates, but each of the 8 pairs of near-collision blocks has one bit difference only.

Part 3, the least significant 4000 bits of the RSA modulus, calculated in such a way that the concatenation of the three parts (for a total of $96 + 4096 + 4000 = 8192$ bits) is a hard to factor RSA modulus. This part is identical for the two certificates.

The public exponent, fixed at 65537 for both certificates.

Construction details

We provide a detailed description of our construction.

1. We first construct a pair of templates for the certificates, in which all fields are filled in, with the exception of the RSA public key moduli (apart from a first zero byte which is there to prevent the bitstring from representing a negative integer) and the signature. We can easily meet 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 this note);
- 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. Note that since the public key exponent bitlength has to be fixed in advance, it is just as easy to fix the entire public exponent. We take the usual "Fermat-4" number $e = 65537$. It is imperative to have the same e for both certificates, as it comes after the colliding blocks.

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 appropriately chosen bits to each. These bits are computed by birthdaying, to satisfy 96 bit conditions on the output IHV difference. For this purpose each IHV is interpreted as 4 little endian 32-bit integers, and the difference between the 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 $\Delta b = \Delta c = \Delta d$. This approach was suggested to us by Xiaoyun Wang[2], as it facilitates the search for the next near-collision blocks. Let b'_1 and b'_2 be the resulting bitstrings of length 96. This completes Part 1 of the Subject Public Key Info.
4. Using the techniques developed in Marc Stevens' MSc thesis [14] and as sketched in the Appendix, we compute two different bitstrings b''_1 and b''_2, of 4096 bits (8 near-collision blocks) each, for which the MD5 compression function with the IHVs from the previous step produces a collision. With $b_1 = b'_1 \| b''_1$ and $b_2 = b'_2 \| b''_2$ we now have b_1 and b_2 that form the leading 4192 bits of the RSA moduli. Note that the two to-be-signed fields up to and including b_1 and b_2, 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. This completes Part 2 of the Subject Public Key Info.
5. The next step is to construct two specially crafted but secure RSA moduli from the bitstrings b_1 and b_2, respectively, by appending to each the same bitstring b of 4000 bits. This we did in the same way as for our previous colliding certificates and as described in [9]. In the present case we have 4192-bit prefixes b_1 and b_2, and we target 8192-bit moduli. As explained in [10] 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. As a result, computing the moduli took about an hour on a regular laptop. Here is how it goes.
   - Generate random 1976-bit primes p_1 and p_2, such that e is coprime to $p_1 - 1$ and $p_2 - 1$.
   - Compute b_0 between 0 and $p_1 p_2$ such that $p_1 | b_1 2^{4000} + b_0$ and $p_2 | b_2 2^{4000} + b_0$ (by the Chinese Remainder Theorem).
   - Find a positive integer k for which $b = b_0 + k p_1 p_2$ satisfies the following conditions: both $q_1 = (b_1 2^{4000} + b)/p_1$ and $q_2 = (b_2 2^{4000} + b)/p_2$ are primes, and e is coprime to both $q_1 - 1$ and $q_2 - 1$:
     - use a sieve to eliminate candidates with a small prime divisor; we sieved with the primes below 2^{28} over an interval of 2^{24} odd numbers k, which resulted in 44601 survivors (out of $2^{24} \approx 1.678 \times 10^7$ candidates);

[2] Private communication.

    - for each of the survivors do a simple Miller-Rabin test (with only 2 as base) on q_1 and, if necessary, on q_2; the first candidate surviving the q_2 test was subjected to more thorough testing (for both q_1 and q_2) and turned out to be a satisfying example (we were lucky, as already the 1374th of the 44601 candidates was successful).
  - When primes q_1 and q_2 have been found, output $n_1 = b_1 2^{4000} + b$ and $n_2 = b_2 2^{4000} + b$ (as well as p_1, p_2, q_1, q_2), and stop.
  - When k becomes too large, i.e., the corresponding q_1 or q_2 may become too large, start all over with new random p_1 and p_2.

  This completes Part 3 of the Subject Public Key Info.

  It is reasonable to expect, based on the Prime Number Theorem, that this algorithm will produce in a feasible amount of computation time, two hard to factor RSA moduli $n_1 = p_1 q_1$ and $n_2 = p_2 q_2$. Furthermore, as argued above, when concatenated to their corresponding initial to-be-signed parts, they will collide under MD5. With p_1 and p_2 at around 1976 bits our RSA construction method is usually successful within a few hours of computing time. Theoretically, it still works for p_1 and p_2 up to 2000 bits, but the interval in which candidates are to be found gets shorter the closer one gets to 2000 bits, thereby leading to longer expected runtimes. So, we left it at 1976 bits.
6. We insert the modulus n_1 into the template for the first certificate, thereby completing the to-be-signed part of the first certificate, and we compute the MD5 hash of the entire to-be-signed part (including MD5 padding, and using the standard MD5-IHV).
7. We apply standard PKCS#1v1.5-padding (see [12, Section 9.2]), and perform a modular exponentiation using the issuing Certification Authority's private key. This gives the signature, which is added to the certificate. The first certificate is now complete.
8. To obtain the second valid certificate, all we have to do is to put the modulus n_2 and the signature as computed in the previous step at their locations in the template for the second certificate.

Note that the prime factors of each modulus have rather different sizes, i.e., the RSA moduli are strongly unbalanced. Although this is unusual for RSA moduli, for the parameter choices we make (smallest primes of 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. Further note that the corresponding private keys can easily be computed from the public exponent and the prime factors of the moduli.

Finding the target MD5 collisions is by far the computationally hardest part of the above construction, a remark that is similar to one made in [9]. However, in the meantime the methods for constructing MD5 collisions with identical initial IHVs have been improved considerably, see [13] and [8]. Such collisions can now be found within seconds, so the bottleneck in the colliding certificate scenario of [9] may now have shifted from the collision search to the moduli construction.

Example

Below is an example pair of colliding certificates in full detail (byte dump). The colliding certificates in binary form, as well as the CA certificate and some additional data, can be downloaded from `http://www.win.tue.nl/hashclash/TargetCollidingCertificates/`.

In the left column the exact bytes are presented in a form that clarifies the ASN.1 structure. Black characters indicate identical bits, underlined blue and red characters indicate different bits.

```
tag length data                                      | comment
=== ====== ================================ | ====================================================
 30 820629                                          | ASN.1 header
---------------------------------------------------- | ----------------------------------------------------
 30 820511                                          | to-be-signed part begins here
 A0 03                                              |
 02 01      02                                      | X.509 version 3
 02 04      010C0001 020C0001                       | serial number
 30 0D                                              |
 06 09      2A864886F70D010104                      | signature algorithm identifier (md5withRSAEncryption)
 05 00                                              |
---------------------------------------------------- | ----------------------------------------------------
 30 3D                                              | issuer distinguished name starts here
 31 1A                                              |
 30 18                                              |
 06 03      550403                                  |
 13 11      4861736820436F6C6C6973696F6E2043        | issuer common name (``Hash Collision CA'')
            41                                      |
 31 12                                              |
 30 10                                              |
 06 03      550407                                  |
 13 09      45696E64686F76656E                      | issuer locality (``Eindhoven'')
 31 0B                                              |
 30 09                                              |
 06 03      550406                                  |
 13 02      4E4C                                    | issuer country code (``NL'')
---------------------------------------------------- | ----------------------------------------------------
 30 1E                                              |
 17 0D      3036303130313030303030315A              | not valid before (Jan. 1, 2006, 0h0m1s GMT)
 17 0D      3037313233313233353935395A              | not valid after  (Dec. 31, 2007, 23h59m59s GMT)
---------------------------------------------------- | ----------------------------------------------------
 30 54                                              | subject distinguished name starts here
 31 19      15                                      |
 30 17      13                                      |
 06 03      550403                                  |
                                                    | subject common name:
 13 10      41726A656E204B2E204C656E73747261        | (``Arjen K. Lenstra'')
 13 0C      4D6172632053746576656E73                | (``Marc Stevens'')
 31 16      1A                                      |
 30 14      18                                      |
 06 03      55040A                                  |
                                                    | subject organization
 13 0D      436F6C6C6973696F6E61697273              | (``Collisionairs'')
 13 11      436F6C6C6973696F6E20466163746F72        | (``Collision Factory'')
            79                                      | (dummy text, used to fill up to convenient byte size)
 31 12                                              |
 30 10                                              |
 06 03      550407                                  |
 13 09      45696E64686F76656E                      | subject locality (``Eindhoven'')
 31 0B                                              |
 30 09                                              |
 06 03      550406                                  |
 13 02      4E4C                                    | subject country code (``NL'')
 30 820422                                          |
---------------------------------------------------- | ----------------------------------------------------
 30 0D                                              |
 06 09      2A864886F70D010101                      | public key algorithm (rsaEncryption)
 05 00                                              |
 03 82040F 00                                       | subject public key info
 30 82040A                                          |
 02 820401 00                                       | public key modulus (8192 bits, 1025 bytes)
                                                    | to-be-signed part until here has a multiple of 64 bytes minus 12 bytes
                                                    | different bytes are indicated by colors and underlining
                                                    \---------------------------------\
            EE73E7D6B3B34FBAA1393D02                 1A09B4CB40C7267AAF017F9B          | part 1: 96 birthday bits
                                                                                       |
            A47425818DC84F86736E907228BBE877         A47425818DC84F86736E907228BBE877 | part 2: 8 near-collision blocks
            0203858D8CF1837AFF5E6C2213036AF3         0203858D8CF1837AFF5E6C2213036AF3 |
            D95C77E9C2237D608CC4A9FB97308BBF         D95C77E9C2237D608CC4A9FB97307BBF | <-- bit difference on this line
            9828612F1599E2615BCCDEDA5930532F         9828612F1599E2615BCCDEDA5930532F |
            B3DD117278E494401433630E7461C1DC         B3DD117278E494401433630E7461C1DC |
            9B801B2E552015A513FF7AE7973EF44B         9B801B2E552015A513FF7AE7973EF44B |
            8352E4E04979B31EB600654D51F4A381         8352E4E04979B31EB600654D51F4A481 | <-- bit difference on this line
            CEBE3F0BD099D130D1456FABE04A3E98         CEBE3F0BD099D130D1456FABE04A3E98 |
            85C8C4FB297B86B57752CD6419809FE3         85C8C4FB297B86B57752CD6419809FE3 |
            7E6286F07732D1E069A5B4E56670B8BB         7E6286F07732D1E069A5B4E56670B8BB |
            BAE5C211742A131D05711CF1FE32AF93         BAE5C211742A131D05711CF1FE22AF93 | <-- bit difference on this line
            3F1EEF224762E3AADAC17C40E448CA41         3F1EEF224762E3AADAC17C40E448CA41 |
            A879A03D3CF665F239C7F3FE82B384E8         A879A03D3CF665F239C7F3FE82B384E8 |
            35E7C9E8BDEE30C268A2121284789DF4         35E7C9E8BDEE30C268A2121284789DF4 |
            2F44906F19B79026464436E1DA65FA0C         2F44906F19B79026464436E1DA64FA0C | <-- bit difference on this line
            53A377FA0D2B012B7DDC2855DAE5B551         53A377FA0D2B012B7DDC2855DAE5B551 |
            51E28034112120B5E79EC5F26A9F69DA         51E28034112120B5E79EC5F26A9F69DA |
```

```
           85D74EF6A97A0B1164EFA25FB1AE26BA   85D74EF6A97A0B1164EFA25FB1AE26BA |
           451CCDA7A2E784339C447D562549A60B   451CCDA7A2E784339C447D560549A60B | <-- bit difference on this line
           F0676294BF580C919EC457025D3C7860   F0676294BF580C919EC457025D3C7860 |
           B98296C0AB9FE5B1D353882E26C1F721   B98296C0AB9FE5B1D353882E26C1F721 |
           B41899D972B5A1D5050B684536448010   B41899D972B5A1D5050B684536448010 |
           AF8C7AFF7CE8EACCB9B1FBBDD129D4F5   AF8C7AFF7CE8EACCB9B1FBBDC929D4F5 | <-- bit difference on this line
           D499FB812924DF302CB3C45023386297   D499FB812924DF302CB3C45023386297 |
           9396B3A46CD0FF7F1426711C459297B6   9396B3A46CD0FF7F1426711C459297B6 |
           5D1CEF66C18751E094BF08F3B2981C5C   5D1CEF66C18751E094BF08F3B2981C5C |
           CE52D963D5A4259A64557E4D1B9EFE2D   CE52D963D5A4259A64557E4D1B9EFE0D | <-- bit difference on this line
           9A516D1E6EC8BB37066825AEA6361660   9A516D1E6EC8BB37066825AEA6361660 |
           2BD7D11625A06A90739B4D0A06EA872A   2BD7D11625A06A90739B4D0A06EA872A |
           3AF9EBA12629BED67940561BD9374A89   3AF9EBA12629BED67940561BD9374A89 |
           D60F0D722C9FEB6833EC53F0B0FD76AA   D60F0D722C9FEB6833EC53F0B0FD76A2 | <-- bit difference on this line
           047B66C90FCEB1D2E22CC099B9A4B93E   047B66C90FCEB1D2E22CC099B9A4B93E |
                                            /----------------------------------/ at this point an MD5 collision is reached
           0000000F54A895176E4C295A405FAF54 | part 3: identical parts of the modulus
           CEE82D043A45CE40B155BE34EBDE7847 |
           85A25B7F894D424FA127B157A8A120F9 |
           9FE53102C81FA90E0B9BDA1BA775DF75 |
           D9152A80257A1ED352DD49E57E068FF3 |
           F02CABD4AC97DBBC3FA0205A74302F65 |
           C7F49A419E08FD54BFAFC14D78ABAAB3 |
           0DDB3FC848E3DF02C5A40EDA248C9FF4 |
           7482850CFDFBDD9BC55547B7404F5803 |
           C1BB81632173127E1A93B24AFB6E7A80 |
           450865DB374676D576BA5296CCC6C130 |
           82D1AB36521F1A8AD945466B9EF06AF4 |
           3A02D70B7FB8B7DC6D268C3DBA6898F6 |
           552FA3FBB33DCBFADA7B33FA75D93AFE |
           262BD37AFF75995FD0E9774BA5A26A7C |
           443FF34E461502A2CB777E982D007375 |
           14B88ED28D61F428E88387DF2BF02230 |
           AD17A9D44FF364850A07DB42A7826AC2 |
           EE3899CAC3EC274721D476D96658F537 |
           16676587F8FF14DB8DE6741AFA2206DB |
           A3B11828BA87C6E1E88A022F1AA8DDD0 |
           37EAB049B5C7D3053D0A63D7861DEA07 |
           B3D8B720DE068CF47E657BB44450B85D |
           52F749D59572DF0C0E3433B47C9AA19A |
           856F1DC3CDADBAFB143035C85A53AF57 |
           22038F765C0D621B66B69FFFFD091D4A |
           661A453BF1DAED1A3A2341B37D7F623B |
           158F6EC02B49A25364430FCB5861483E |
           1E9543ED2EE7E54A4C108A6E64194098 |
           0EE60D14AEE559AF30037E75B2309CE0 |
           21FFE3109BF2053892AB0AE403516E2A |
           B58067F7                         |
 02 03     010001                           | public exponent (65537)
------------------------------------------- | -----------------------------------------------------
 A3 1A                                      | version 3 extensions start here
 30 18                                      |
 30 09                                      |
 06 03     551D13                           | basic constraints
 04 02     3000                             |
 30 0B                                      |
 06 03     551D0F                           | key usage
 04 04                                      |
 03 02     05E0                             | to-be-signed part ends here
------------------------------------------- | -----------------------------------------------------
 30 0D                                      |
 06 09     2A864886F70D010104               | signature algorithm identifier (md5withRSAEncryption)
 05 00                                      |
------------------------------------------- | -----------------------------------------------------
 03 820101 00                               | signature (2048 bits, 257 bytes)
           86C0876D20682DC897443F97690DDFB2 |
           9074CB25C358F09F81234CE265A44333 |
           CB6A78B23273291700DCD6BADF55088A |
           19A317A51D6092AC3F6FC6243601367A |
           6A2FC0969B4E8913BFC2315F5AF35D83 |
           FBD03C957839242217BEB9AD8873D442 |
           F3A36200CA198F6345BCB76CCB27FCF2 |
           DBEA239E50FDDD3CD69304C950E7094A |
           FF0A965902B72206D04E3759BAED05AE |
           05922D8BE93556C8CACDC3606C56EE37 |
           89C3775F767A8909AB444BC1D7EE4A41 |
           677302EFDF337B4CEE082D9218FE44AA |
           5D68D34EFB796AC43219DCF8DD4C2E6E |
           C458EFA482DA7E181C0864177124F0CF |
           214B0C5A28EFECA40EC532BB7673FFEA |
           9B9BD0A0B1EFE6DB97C518C4DB17B9A5 |
=== ====== ================================ | =====================================================
```

Here are the IHV values for the to-be-signed parts of the certificates (the differences are computed for 32-bit unsigned integer words):

block	certificate 1	certificate 2	difference	note
0	0123456789ABCDEFFEDCBA9876543210		0	1)
1	488FAE30B8259F77F81AA10709F1667D	8CD14B34EE2CE093EE1238A70A9449C1	many	
2	3E15562D935DC8950E86F877F650A439	7D99D701715647503BDA995E53F9EB07	many	
3	A2934A57268FC8FB99270DB2BD42867F	9756EBE66FC92AD60256345C8EC444A8	many	
4	2D857B4E0479B7259F7662D47771220B	2D857B4EA419FB613F17A61017126647	-2^5-2^7-2^13+2^15-2^18-2^22+2^26-2^30	2)
5	E745A14768C24DF4F16EF79A0EE57A77	E745A147086391F0910F3B97AE85BE73	-2^5-2^7-2^13+2^15-2^18-2^22+2^26	
6	6900F0DD6880AD3B8A559C5D95807BC7	6900F0DD0821F13B2AF6DF5D3521BFC7	-2^5-2^7-2^13+2^15-2^18-2^22	
7	6F48D9E5989D51D05CA3E94D800AF3F8	6F48D9E5383E55D0FC43ED4D20ABF6F8	-2^5-2^7-2^13+2^15-2^18	
8	80D9AE066685A793F953E15A6EDE318F	80D9AE060626A79399F4E05A0E7F318F	-2^5-2^7-2^13+2^15	
9	73A70AC0FAA8B2239EAB7BE423EC6388	73A70AC09AC9B2233ECC7BE4C30C6488	-2^5-2^7-2^13	
10	DE56FC8A9A091FEB1E6E537D16629AC4	DE56FC8A3A0A1FEBBE6E537DB6629AC4	-2^5-2^7	
11	DCA82596635B2D4F0EDB818BDEE0D521	DCA82596835B2D4F2EDB818BFEE0D521	-2^5	
12	505D9746FAB00B328018DBC34A87DF11		0	3)
13	DAC293C410FD4B465B174166617DA963		0	
14	524312A4FD34CF77AF144C437EAC0BBF		0	
15	AA6FAC2CFD95D7C22F35ACF82B55B146		0	
16	065C03F4E72681A54B874ABF80BC3C3D		0	
17	D4852EBAA84E005A8C82A34146D0AD3A		0	
18	FCABDB3144B842CCD7E3DFE8C94A6729		0	
19	80AC53D61C9869AEA32085761A042D0F		0	
20	0BA6111733324BB09A2227F50C4496E2		0	
final	C6B2FE88912770FC6F2DB71F58C7D251		0	4)

Notes:
1): Initial IHV, according to the MD5 standard.
2): This special difference is the result of birthdaying. Interpreting each IHV as 4 little endian 32-bit integers and defining the difference between the IHVs as the 4-tuple of differences modulo 2^{32}, as explained above, the difference between the IHVs can be written as $0\|\delta\|\delta\|\delta$ with $\delta = -2^5 - 2^7 - 2^{13} + 2^{15} - 2^{18} - 2^{22} + 2^{26} - 2^{30}$. At each consecutive near-collision block the highest 2-power of δ in this notation (i.e., using the Non-Adjacent Form) is chipped away, thus removing three 'bits' of the difference per step.
3): Here is the full collision.
4): The final IHV includes MD5 padding and Merkle-Damgård strengthening according to the MD5 standard. It is the MD5 output, that is subsequently used as input to the RSA signing operation using the CA private key.

The differences are also made visible in the pictures below.

Figure 2 shows the differences of the IHVs, one at each horizontal line. The colors refer to the signs of the bit differences.

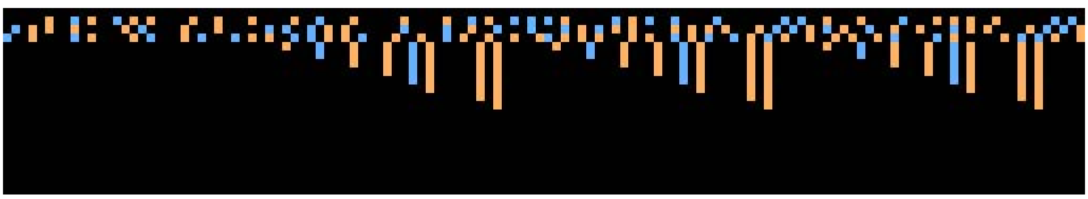

Figure 2. IHV differences for the colliding certificates.

Figure 3 shows also the differences of the internal states after each round inside the compression function, and shows the IHV differences between the yellow bars.

How to verify

The certificates are valid in the sense that they comply with the relevant standards (RFC 3280, ASN.1 DER encoding, but see the next section), and also in the sense that their digital signature can be verified against the issuing Certification Authority's certificate. For manual verification of our claims we have provided the above byte dumps, as well as further technical data (such as the prime factors of the moduli and the CA public key) at the mentioned website. Tools that provide more convenient ways to verify our claims are e.g. Peter Gutmann's `dumpasn1` (see `http://www.cs.auckland.ac.nz/%7Epgut001/`), `openssl` (see `http://www.openssl.org`), and Microsoft's standard Certificate Viewer as it comes with e.g. Windows XP. Unfortunately Microsoft's Certificate Viewer does not show the certificate's signature, but `dumpasn1` and `openssl` do, as the final byte string of length 257. Note that when the CA certificate is installed in the standard Windows (Internet Explorer) Certificate Store, the Certificate Viewer will automatically validate the certificate signatures against the CA certificate.

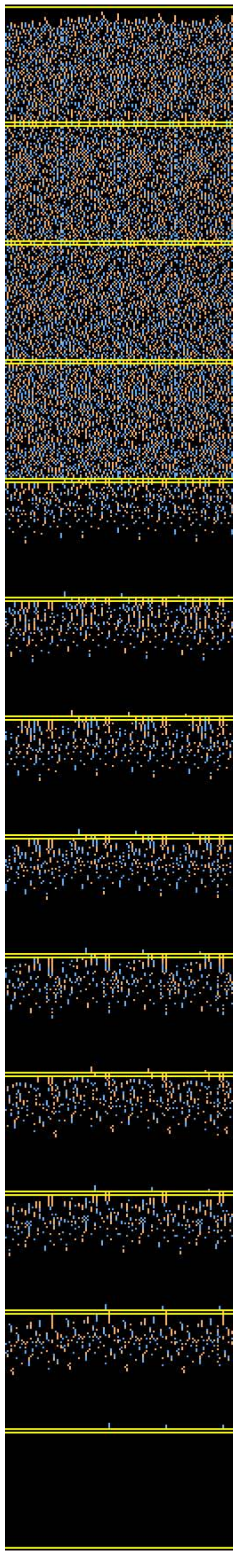

Figure 3. Internal state differences.

A small error

The reader who takes a close look at the bits of our certificates will notice that the second certificate does not have a 8192-bit modulus, but a 8189-bit one. This is due to the fact that in the result of the birthdaying computation it turned that one of the bitstrings of 96 bits had the three most significant bits not set. At the time we should have noticed this and we should have birthdayed a bit further at almost no additional effort to find a pair with for both the most significant bit set. Or we could simply have fixed one more byte. Unfortunately we overlooked this. When we did notice it, 6 months of hard work had already been based on these values, and we did not want to wait another few months to redo all the computations.

As a result we now have one 8192-bit modulus and one 8189-bit one. The main problem with this is that the DER encoded bitstring in which this 8189-bit modulus is located, is strictly speaking erroneous, i.e. not according to the DER encoding rules: the zero byte at the front, needed to make sure the integer is interpreted as a positive one, should be there only when the next byte has its most significant bit set. We could however not leave it out anymore: that would have changed the length values that occur earlier in the ASN.1 structure, which would have changed the IHVs, so that the entire collision computation would have to be done again. This would have meant a delay of several months, so we decided to leave the error there.

Peter Gutmann's `dumpasn1` program notices this error. The `openssl` software does not, and gives the correct modulus bit length of 8189. Microsoft's Certificate Viewer also does not notice the error, and moreover gives the erroneous value 8192 for the modulus bitlength. It could very well happen that other certificate parsing software will notice the error and reject the certificate because of it. This however does not undermine our method of construction of colliding certificates with different identities (let alone the method of constructing target collisions). It only happens to be the case that this specific example has a minor flaw in it, that could have easily been prevented had we been more alert, and that is not worth anyone's trouble to repair.

Acknowledgements

We are very grateful to:

– Xiaoyun Wang for her birthdaying suggestion and her further advice and support;
– Yiqun Lisa Yin and Vlastimil Klima for discussions and ideas on MD5 differential path construction;
– Paul Hoffman, Eric Verheul, Pascal Junod and Bart Preneel for discussions, ideas and comments;
– NBV and Gido Schmitz for providing a good environment for Marc to do his MSc Thesis project;
– many hundreds of BOINC enthousiasts all over the world, mostly completely unknown to us, who were willing to donate an impressive amount of cycles to the HashClash project running with BOINC software;
– Jan Hoogma at LogicaCMG for technical discussions and sharing his BOINC knowledge;
– Bas van der Linden at TU/e for making available the Elegast cluster;
– Wil Kortsmit and Vincent Huijgen at TU/e for technical support.

References

1. C. de Cannière and C. Rechberger, *Finding SHA-1 Characteristics*, AsiaCrypt 2006, to appear.
2. M. Daum and S. Lucks, *Attacking Hash Functions by Poisoned Messages, "The Story of Alice and her Boss"*, June 2005, `http://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.
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, `http://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, `http://www.ietf.org/rfc/rfc3280.txt`.
6. P. Hoffman and B. Schneier, *Attacks on Cryptographic Hashes in Internet Protocols*, IETF RFC 4270, November 2005, `http://www.ietf.org/rfc/rfc4270.txt`.
7. D. Kaminsky, *MD5 to be considered harmful someday*, December 2004, `http://www.doxpara.com/md5%5Fsomeday.pdf`.
8. Vlastimil Klima, *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, Cryptology ePrint Archive, Report 2006/105, `http://eprint.iacr.org/2006/105`.
9. A.K. Lenstra, X. Wang and B.M.M. de Weger, *Colliding X.509 certificates*, Cryptology ePrint Archive, Report 2005/067, `http://eprint.iacr.org/2005/067`. An updated version has been published as an appendix to [10].
10. 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.
11. O. Mikle, *Practical Attacks on Digital Signatures Using MD5 Message Digest*, Cryptology ePrint Archive, Report 2004/356, `http://eprint.iacr.org/2004/356`.
12. *PKCS#1 v2.1, RSA Cryptography Standard*, RSA Laboratories, June 2002, `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.
13. Marc Stevens, *Fast Collision Attack on MD5*, Cryptology ePrint Archive, Report 2006/104, `http://eprint.iacr.org/2006/104`.
14. Marc Stevens, TU Eindhoven MSc thesis, in preparation.
15. X. Wang and H. Yu , *How to Break MD5 and Other Hash Functions*, EuroCrypt 2005, Springer LNCS 3494 (2005), 19–35.

Appendix

We sketch the construction of the differential paths that are used to compute the near-collision blocks. First, to fix notation, we review the MD5 compression function.

MD5 compression function

The input for the MD5 Compression function is a 128-bit intermediate hash value $IHV^{(i-1)}$ (consisting of four 32-bit values IHV_0^{i-1}, IHV_1^{i-1}, IHV_2^{i-1}, IHV_3^{i-1}) and a 512-bit message block $M^{(i)}$. There are 64 steps (numbered 0 up to 63) grouped into four rounds. Each step is based on a non-linear function, modular addition and left rotation. In each step t an Addition Constant (AC_t) and a Rotation Constant (RC_t) is used. They are defined as follows:

$$AC_t = \lfloor \text{abs}(\sin(t+1)) \cdot 2^{32} \rfloor \,, \qquad 0 \le t \le 63,$$

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

The message block $M^{(i)}$ is expressed as sixteen 32-bit words $M_0^{(i)}, \ldots, M_{15}^{(i)}$ and expanded to 64 words W_t as follows:

$$W_t = \begin{cases} M_t^{(i)} & \text{for } 0 \le t \le 15; \\ M_{(1+5t) \bmod 16}^{(i)} & \text{for } 16 \le t \le 31; \\ M_{(5+3t) \bmod 16}^{(i)} & \text{for } 32 \le t \le 47; \\ M_{(7t) \bmod 16}^{(i)} & \text{for } 48 \le t \le 63. \end{cases}$$

The non-linear function f_t depends on the round

$$f_t(X,Y,Z) = \begin{cases} F(X,Y,Z) = (X \wedge Y) \oplus (\bar{X} \wedge Z) & \text{for } 0 \le t \le 15; \\ G(X,Y,Z) = (Z \wedge X) \oplus (\bar{Z} \wedge Y) & \text{for } 16 \le t \le 31; \\ H(X,Y,Z) = X \oplus Y \oplus Z & \text{for } 32 \le t \le 47; \\ I(X,Y,Z) = Y \oplus (X \vee \bar{Z}) & \text{for } 48 \le t \le 63. \end{cases}$$

The algorithm has a working register with 4 state words Q_t, Q_{t-1}, Q_{t-2} and Q_{t-3}, which are initialized for step $t = 0$ to

$$Q_0 = IHV_1^{(i-1)}, \;\; Q_{-1} = IHV_2^{(i-1)}, \;\; Q_{-2} = IHV_3^{(i-1)}, \;\; Q_{-3} = IHV_0^{(i-1)}.$$

After these initializations the 64 steps are computed as follows for $t = 0, \ldots, 63$:

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

where $RL(x,n)$ denotes bitwise cyclic left-rotation of x over n positions. After all steps are computed, the resulting state values are added to the intermediate hash value and then returned:

$$\begin{aligned}
&IHV_0^{(i)} = IHV_0^{(i-1)} + Q_{61}, IHV_1^{(i)} = IHV_1^{(i-1)} + Q_{64}, \\
&IHV_2^{(i)} = IHV_2^{(i-1)} + Q_{63}, IHV_3^{(i)} = IHV_3^{(i-1)} + Q_{62}.
\end{aligned}$$

MD5 differential paths for the computation of near-collision blocks

Construction of the differential paths that were used for the computation of the near-collision blocks was done in three steps. The first step consisted of constructing a set of lower partial differential paths, starting with two given IHVs, in a step by step manner for $t = 0, \ldots, 12$. The second step similarly consisted of constructing a set of upper partial differential paths, starting with no differences or bitconditions in the working state at $t = 34$, working backwards in a step by step manner for $t = 34, \ldots, 17$. The message differences $\delta m_{11} = \pm 2^b$ were chosen such that the differential paths would have no differences in the working state $(Q_{t-3}, Q_{t-2}, Q_{t-1}, Q_t)$ for $t = 35, \ldots, 61$ and such that $\delta Q_{62} = \delta Q_{63} = \delta Q_{64} = \pm 2^{b+10 \mod 32}$. In the final step combinations of lower and upper paths are taken and completed, if possible, to a full correct differential path.

Because of the boolean function and bitwise rotation, we need to describe precisely how the additive difference of each Q_t affects each bit. For this we use the binary signed-digit representation (SDR), where naturally a digit 0 indicates that a bit is unaffected (constant) and digits +1 and -1 indicate a change in a bit of 0 to 1 and 1 to 0, respectively. Since we aim for the lowest possible number of bitconditions, we always use SDRs that are close to the Non-Adjacent Form (NAF), which has the fewest affected bits. All bits that are not affected in the SDR are constant, nevertheless their value can affect the outcome of the boolean function. Therefore we use bitconditions that specify their value either directly as 0, 1 or free(0/1) or indirectly as the (inverted) value of some other bit. Given SDRs and bitconditions for a step t, one can easily find out which outcomes of the boolean function for each bit are possible: -1, 0, +1. After choosing a preferred outcome one can set extra bitconditions such that only that preferred outcome is possible. Bitwise rotation of a difference is handled by rotating the NAF of that difference, since the resulting difference would be one of the most likely differences after rotation, and is often the most likely one.

When extending a lower differential path over $t = 0, .., k-1$ with step $t = k$ we have to deal with SDRs, bitconditions over $Q_{-3}, \ldots, Q_{k-1}$ and an additive difference δQ_k. For each interesting SDR of δQ_k we do the following. We examine the possible outcomes of the boolean function over all bits, and for each combination of one possible outcome over all bits, we set extra bitconditions such that those outcomes are guaranteed. As a result we find an extended lower differential path with

$$\delta Q_{k+1} = \delta Q_k + RL(\delta F_k + \delta Q_{k-3} + \delta w_k, RC_k).$$

Similarly, when extending an upper differential path over $t = k+1, ..., 63$ with step $t = k$ we are dealing with SDRs, bitconditions over $Q_{k-1}, ..., Q_{35}$ and an additive difference δQ_{k-2}. For each interesting SDR of δQ_{k-2} we do the following. We examine the possible outcomes of the boolean function over all bits, and for each combination of one possible outcome over all bits, we set extra bitconditions such that those outcomes are guaranteed. As a result we find an extended upper differential path with

$$\delta Q_{k-3} = RR(\delta Q_{k+1} - \delta Q_k, RC_k) - \delta F_k - \delta w_k,$$

where $RR(x, n)$ denotes bitwise cyclic right-rotation of x over n positions. For a combination of a lower and an upper differential path we have SDRs, bitconditions over $Q_{-3}, ..., Q_{12}$ and over $Q_{15}, ..., Q_{35}$, and additive differences δQ_{13} and δQ_{14}. So all additive differences δQ_i are known, however steps $t = 13, 14, 15, 16$ are not handled yet in the differential path. For those steps we can determine the required

$$\delta \hat{F}_t = RR(\delta Q_{t+1} - \delta Q_t, RC_t) - \delta Q_{t-3} - \delta w_t.$$

We exhaustively try all possible SDRs of δQ_{13} and δQ_{14} and try to find extra bitconditions such that $\delta F_t = \delta \hat{F}_t$ for $t = 13, 14, 15, 16$. This will succeed with sufficiently large probability.