On the possibility of constructing meaningful hash collisions for public keys

full version*, with an appendix** on colliding X.509 certificates

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

[1] Lucent Technologies, Bell Laboratories, Room 2T-504
600 Mountain Avenue, P.O.Box 636, Murray Hill, NJ 07974-0636, USA
[2] Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. It is sometimes argued (as in [6]) that finding meaningful hash collisions might prove difficult. We show that at least one of the arguments involved is wrong, by showing that for several common public key systems it is easy to construct pairs of meaningful and secure public key data that either collide or share other characteristics with the hash collisions as quickly constructed in [22]. We present some simple results, investigate what we can and cannot (yet) achieve, and formulate some open problems of independent interest. At this point we are not yet aware of truly interesting practical implications. Nevertheless, our results may be relevant for the practical assessment of the recent hash collision results in [22]. For instance, we show how to use hash collisions to construct two X.509 certificates that contain identical signatures and that differ only in the public keys. Thus hash collisions indeed undermine one of the principles underlying Public Key Infrastructures.

Keywords: hash collisions, public keys

1 Introduction

Based on the birthday paradox a random collision for any n-bit hash function can be constructed after an effort proportional to $2^{n/2}$ hash applications, no matter how good the hash function is. From the results presented at the Crypto 2004 rump session (cf. [22]), and since then described in more detail in [23], [24], [25], and [26], it follows that for many well known hash functions the effort required to find random collisions is considerably lower. Indeed, in some cases the ease with which collisions can be found is disconcerting.

However, most of the hash functions affected by the results announced in [22] were already known to be weak. Prudent applications that relied on their random collision resistance should have been phased out years ago. Their application in digital certificates, however, is still rather common. In particular MD5, one of the affected hash functions, is still being used by Certification Authorities to generate new certificates. The affected hash functions are also widely used for integrity protection of binary data. For example, executables distributed over the Internet often come with a published hash value so that users can check that the proper code was downloaded. And the occurrence of changes in the contents of a file system can be detected by hash checking programs such as Tripwire.

We sketch the commonly used arguments why such applications are not affected by the lack of random collision resistance. In this note we concentrate on applications in the area of

* This paper will be presented at ACISP 2005, Brisbane, Australia, July 4-6, 2005. See [14] for the "proceedings version" of this paper, which contains somewhat less detail than this "full version".

** The appendix is written by Arjen Lenstra, Xiaoyun Wang and Benne de Weger, and is an adapted version of [13].

public key cryptography, see [8] and [16] for interesting ideas about the application of hash collisions in other areas.

A successful attack on an existing certificate (or some other data structure such as an executable) requires *second preimage resistance* of one message: given a pre-specified value and its hash, it must be practically infeasible to find another value with the same hash. As far as we are aware, the results announced in [22] do not imply that second preimages are essentially easier to find than they should, namely with an effort proportional to 2^n for an n-bit hash function. According to a result first published in [4] and later (and independently) generalized in [10], second preimages for many common hash functions can be found in overall runtime proportional to 2^{n-k} for reasonably sized $k > 0$, but for 2^k input blocks, so also at memory cost 2^k. Thus, using the *full cost* time×memory of an attack effort, as suggested in [1] and [27], finding a second preimage can still be argued to cost the full 2^n. In any case, certificates that existed well before the results from [22] were obtained should be fine.

For newly to be constructed data structures such as certificates the argument goes that random collisions do not suffice because the values to be hashed are *meaningful* (cf. [6] and [19]). Dobbertin's cryptanalytic work on MD4 was so strong that meaningful collisions could be found easily, cf. [5]. The recent results of [22] seem not (yet) to have similar strength, so revisiting the concept of meaningfulness is of interest.

A certificate, such as an X.509 or PGP certificate, is a highly structured document, and also executable code will have a lot of structure to be able to execute properly. Nevertheless, both these data structures may contain pieces of data that look random, and may have been constructed to fit a hash collision. A hash collision may be inserted on purpose inside an executable; see [8] and [16] for interesting exploit ideas in this area. In certificates there will be random looking binary data related to public keys. Also the Diffie-Hellman group size may be related to a random-looking large prime, which is a system parameter that could be hard-coded into a binary executable. As was shown in [9], given any hash collision as for instance presented in [22], it is trivial to construct a 'real' Diffie-Hellman prime and a 'fake' one that hash to the same value. One may ask whether the mathematical requirements that lie behind public key constructions enforce so much meaningful structure that it may be expected to be incompatible with the collision requirement. We show that this is not the case.

The collisions found by [22] all have a special structure: two inputs are found that hash to the same value, and that differ in a few spread-out and precisely specified bit positions only. This leads us to the following question. Suppose the value to be hashed contains an RSA modulus, i.e., a hard to factor composite integer, or an element g^v for a (sub)group generator g and secret exponent v. How would one come up with two different RSA moduli or two different powers of g that have the subtle differences that seem to be required for the collisions as constructed in [22]?

Having the right type of difference structure does not, as far as we know, imply a hash collision. Indeed, it is as yet unclear to us what conditions have to be imposed on the matching bits in order to realize the collisions announced in [22], but it is clear from [23] that they will be severe. Presently, specially crafted data blocks seem to be required for collisions. But colliding data blocks can be used to generate more collisions as follows. All affected hash functions are based on the *Merkle-Damgård construction*, where a *compression function* is iteratively applied to a changing *chaining variable* and the successive data blocks followed by a length dependent final block. New collisions can therefore be constructed by appending arbitrary, but identical, data to any existing pair of colliding data consisting of the same number of blocks. Thus, to produce two different but colliding public keys one could try to use the specially crafted data blocks as their most significant parts, and then append equal data blocks, carefully chosen such that well-formed and secure public keys result with identical hash values.

Apparently, colliding data blocks can be found for the compression function with an arbitrary value of the chaining variable. This implies that identical data can also be prepended to colliding pairs if the resulting data have the same length and the colliding pairs have been specifically crafted to work with the chaining variable value that results from the prepended data.

In this paper we investigate the various problems and possibilities. We show how we can generate public keys with prescribed differences but with a priori unknown most significant parts. Even though the resulting public keys will, in general, not collide, it cannot be excluded, and it can indeed be expected, that in the future new collision methods will be found that have different, less severe restrictions. Therefore it is relevant to know if the two requirements—being meaningful and having the proper difference structure—are mutually exclusive or not, and if not if examples can be constructed in a reasonable amount of time. We address this question both for RSA and for discrete logarithm systems. We explicitly restrict ourselves to *known* and *secure* private keys as the construction of unknown or non-secure private keys is hardly challenging (cf. [20]): for instance, a number that differs slightly from a proper RSA modulus may be expected to behave as a random number (with respect to factorization properties), is thus often enough easy to factor, and thereby insecure. And if it turns out to be too hard to factor, it is useless because the secure private key cannot be found.

Furthermore, using the appending trick, we show how we can generate actually colliding pairs consisting of proper public RSA keys, albeit with moduli comprised of unbalanced prime factors. Combining this construction with the prepending idea, we show how very closely related X.509 certificates can be constructed that have identical signatures on different hard to factor moduli. It is conceivable that such certificate 'pairs' may be used for ulterior purposes.

We are not aware yet of real life practical implications of our results. Our sole goal is to point out that one may have to be more careful than expected when relying on the 'meaningful message' argument against hash collisions in certification applications, as this argument may be weaker than it appears at first sight.

A summary of our results is as follows. It is straightforward to generate secure pairs of RSA moduli with any small difference structure. For the sake of completeness our simple method is presented in Section 2 along with some runtimes of a proof-of-concept implementation. Furthermore, in Section 2 it is shown how any actual Merkle-Damgård based hash collision can be used to construct colliding pairs consisting of two hard to factor moduli, and how such moduli can be embedded in X.509 certificates with identical signatures. Full details of the construction method for colliding certificates are presented in an appendix, together with an illustrative example. For discrete logarithm systems there is a much greater variety of results, and even some interesting open questions. Briefly, one can do almost anything one desires if one may pick any generator of the full multiplicative group, but if a prescribed generator, or a subgroup generator, has to be used, then we cannot say much yet. Our observations are presented in Section 3. In Section 4 we investigate the practicality of generating colliding DL system parameters, à la Kelsey and Laurie [9]. Some attack scenarios and applications that use our constructions are sketched in Section 5.

2 Generating pairs of hard to factor moduli

The first problem we address in this section is constructing pairs of RSA public key values that differ in a prescribed small number of bit positions. The second problem is constructing pairs of colliding hard to factor moduli, with an application to the construction of pairs of X.509 certificates with identical signatures.

An RSA public key value ordinarily consists of an RSA modulus and a public exponent. A single RSA modulus with two different public exponents that differ in the prescribed way is in principle a solution to the first problem. But in practice one often fixes the public exponent (popular values are 3, 17, and 65537), and even if one does not, selecting two proper public exponents that differ in the right way is trivial and does not lead to an entertaining mathematical question.

The first problem: RSA moduli with prescribed difference. We address the more interesting problem where the public exponent is fixed and where the two RSA moduli differ in the prescribed bit positions. The latter is the case if the XOR of the regular binary representations of the moduli consists of the prescribed bits. Unfortunately, the XOR of two integers is not a convenient representation-independent mathematical operation. This slightly complicates matters. If the hamming weight of the prescribed XOR is small, however, the XOR corresponds often enough to the regular, representation-independent integer difference. Therefore a probabilistic method to generate moduli with a prescribed difference may be expected to eventually produce a pair with the right XOR.

Algorithm to generate moduli with prescribed difference. Let $N \in \mathbf{Z}_{>0}$ be an integer indicating the bitlength of the RSA moduli we wish to construct, and let δ be a positive even integer of at most N bits containing the desired difference. We describe a fast probabilistic method to construct two secure N-bit RSA moduli m and n such that $m-n=\delta$. Informally, pick primes p and q at random, use the Chinese Remainder Theorem to find m with $m \equiv 0 \bmod p$ and $m \equiv \delta \bmod q$, and add pq to m until both cofactors m/p and $(m-\delta)/q$ are prime. More formally:

- Let ℓ be a small positive integer that is about $2\log_2(N)$.
- Pick distinct primes p and q of bitlength $N/2-\ell$, calculate integers $r=\delta/p \bmod q$ and $s=(rp-\delta)/q$, then for any k

$$p(r+kq)-q(s+kp)=\delta.$$

- Search for the smallest integer k such that $r+kq$ and $s+kp$ are both prime and such that $p(r+kq)$ and $q(s+kp)$ both have bitlength N.
- For the resulting k let $m=p(r+kq)$ and $n=q(s+kp)$.
- If k cannot be found, pick another random p or q (or both), recalculate r and s, and repeat the search for k.

Runtime analysis. Because the more or less independent $(N/2+\ell)$-bit numbers $r+kq$ and $s+kp$ have to be simultaneously prime, one may expect that the number of k's to be searched is close to $(N/2)^2$. Thus, a single choice of p and q should suffice if 2^ℓ is somewhat bigger than $(N/2)^2$, which is the case if $\ell \approx 2\log_2(N)$. The algorithm can be expected to require $O(N^2)$ tests for primality. Depending on the underlying arithmetic and how the primality tests are implemented—usually by means of trial division combined with a probabilistic compositeness test—the overall runtime should be between $O(N^4)$ and $O(N^5)$.

A larger ℓ leads to fewer choices for p and q and thus a faster algorithm, but it also leads to larger size differences in the factors of the resulting RSA moduli m and n. The algorithm can be forced to produce balanced primes (i.e., having the same bitlength) by taking $\ell = 0$, and for instance allowing only $k = 0$, but then it can also be expected to run $O(N)$ times slower.

From prescribed difference to prescribed XOR. If required, and as discussed above, the method presented above may be repeated until the resulting m and n satisfy m XOR $n = \delta$ (where, strictly speaking, m and n in the last equation should be replaced by one's favorite binary representation of m and n). The number of executions may be expected to increase exponentially with the hamming weight $H(\delta)$ of δ. If $H(\delta)$ is small, as apparently required for the type of collisions constructed in [22], this works satisfactorily.

It is much faster, however, to include the test for the XOR condition directly in the algorithm before $r + kq$ and $s + kp$ are subjected to a primality test. In that case ℓ may be chosen about $H(\delta)$ larger to minimize the number of p and q choices, but that also leads to an even larger size difference between the factors. As shown in the runtimes below, the overhead caused by the XOR condition compared to the difference is quite small.

Security considerations. Given two regular RSA moduli m and n, their difference $\delta = |m - n|$ can obviously be calculated. But knowledge of δ and the factorization of one of the moduli, does, with the present state of the art in integer factorization, not make it easier to factor the other modulus, irrespective of any special properties that δ may have. Indeed, if the other modulus could be factored, the RSA cryptosystem would not be worth much. If m is the product of randomly selected primes p and r of the same size, as is the case in regular RSA, then $r = \delta/p \bmod q$ for any other RSA modulus n with prime factor q and $\delta = m - n$. Thus, the randomly selected prime factor r satisfies the same identity that was used to determine r in our algorithm above (given p, q, and δ), but as argued that does not make r easier to calculate given just q and δ (but not p). This shows that the '$\ell = 0$ and allow only $k = 0$' case of our algorithm produces RSA moduli pairs that are as hard to factor as regular RSA moduli, and that knowledge of the factorization of one of them does not reveal any information about the factors of the other.

The same argument and conclusion applies in the case of regular RSA moduli with unbalanced factors: with the present state of the art such factors are not easier to find than others (avoiding factors that are so small that the elliptic curve factoring method would become applicable), also not if the difference with another similarly unbalanced RSA modulus is known. If an N-bit RSA modulus m has an $(N/2 - \ell)$-bit factor p with $(N/2 + \ell)$-bit cofactor $\tilde{r}$, both randomly selected, then $\tilde{r} \bmod q = \delta/p \bmod q$ for any other RSA modulus n with $(N/2 - \ell)$-bit prime factor q and $\delta = m - n$. The randomly selected prime factor $\tilde{r}$ when taken modulo q satisfies the same identity that was used to determine r in our algorithm and the cofactor $\tilde{s}$ of q in n, when taken modulo p, satisfies the same identity, with r replaced by $\tilde{r} \bmod q$, that was used to determine s in our algorithm. Because $m - n = \delta$ the integers $\tilde{r}$, r, $\tilde{s}$, and s satisfy $\tilde{r} - r = kq$ and $\tilde{s} - s = kp$ for the same integer valued k. This means that the allegedly hard to find $\tilde{r}$ equals the prime factor $r + kq$ as determined by our algorithm.

Runtimes. Lots of obvious tricks can be used when implementing the above algorithm. We do not elaborate but just note that over a wide range of bitlengths, namely N ranging from 1024 to 4096, the average runtime to generate a pair of moduli m, n with m XOR $n = \delta$ grows slightly faster than N^4. For $\delta = 2^{927} + 2^{687} + 2^{607} + 2^{415} + 2^{175} + 2^{95}$ with $H(\delta) = 6$, a possible interpretation of a δ suggested by one of the examples in [22], we found the following runtimes on a 1GHz Pentium III, averaged over 100 modulus pairs per bitlength and using the fast unbalanced size approach: $N = 1024$ in 9.2 seconds, $N = 1536$ in 42 seconds, $N = 2048$ in 133 seconds, $N = 3072$ in 773 seconds, and $N = 4096$ in 2650 seconds. As expected, the '$\ell = 0$ and allow only k=0' variant works considerably slower, but we have not conducted enough experiments to be able to present meaningful runtime data. If the condition m XOR $n = \delta$ is replaced by $m - n = \delta$ the average runtimes are about 10% faster.

Remark on simultaneous versus consecutive construction. The method presented in this section simultaneously constructs two moduli with a prescribed difference. One may wonder if the moduli have to be constructed simultaneously and whether consecutive construction is possible: given a difference δ and an RSA modulus m (either with known or unknown factorization), efficiently find a secure RSA modulus n (and its factorization) such that m XOR $n = \delta$. But if this were possible, any modulus could be efficiently factored given its (easy to calculate) difference δ with m. Thus, it is highly unlikely that moduli with prescribed differences can be constructed both efficiently and consecutively.

The second problem: actually colliding hard to factor moduli. The object of our investigation so far has been to find out if the requirement to be meaningful (i.e., proper RSA moduli) excludes the apparent requirement of a prescribed difference structure. As shown above, that is not the case: proper RSA moduli with any prescribed difference can easily be constructed. A much stronger result would be to construct RSA moduli that actually *do* have the same hash value. We don't know yet how to do this if the two moduli must have factors of approximately equal size, a customary property of RSA moduli. We can, however, construct actually colliding composite moduli that are, with the proper parameter choices, as hard to factor as regular RSA moduli but for which, in a typical application, the largest prime factor is about three times longer than the smallest factor. Unbalanced moduli for instance occur in [21]. Our method combines the ideas mentioned in the introduction and earlier in this section with the construction from [11].

Algorithm to generate actually colliding hard to factor moduli. Let b_1 and b_2 be two bitstrings of equal bitlength B that collide under a Merkle-Damgård based hash function. Following [22], B could be 512 if b_1 and b_2 collide under MD4, or 1024 if they collide under MD5. It is a consequence of the Merkle-Damgård construction that for any bitstring b the concatenations $b_1||b$ and $b_2||b$ also collide. Denoting by $N > B$ the desired bitlength of the resulting moduli, we are thus looking for a bitstring b of length $N - B$ such that the integers m_1 and m_2 represented by $b_1||b$ and $b_2||b$, respectively, are hard to factor composites. Assuming that $N - B$ is sufficiently large, let p_1 and p_2 be two independently chosen random primes such that p_1p_2 has bitlength somewhat smaller than $N - B$. Two primes of bitlength $(N - B)/2 - \log_2(B)$ should do in practice. Using the Chinese Remainder Theorem, find an integer b_0, $0 \leq b_0 < p_1p_2$ such that p_i divides $b_i 2^{N-B} + b_0$ for $i = 1, 2$. Finally, look for the smallest integer $k \geq 0$ with $b_0 + kp_1p_2 < 2^{N-B}$ and such that the integers $q_i = (b_i 2^{N-B} + b_0 + kp_1p_2)/p_i$ are prime for $i = 1, 2$. If such an integer k does not exist, select new p_1 and p_2 and try again. The resulting moduli are $m_i = p_i q_i = b_i||b$ for $i = 1, 2$, where $b = b_0 + kp_1p_2$ is to be interpreted as $(N - B)$-bit integer. The security of each modulus constructed in this fashion, though unproven, is argued in [11]; since then no weaknesses in this construction have been published. Since p_1 and p_2 are independent, knowledge of the factorization of one of the moduli does not reveal information about the factorization of the other one. The argument follows the lines of the security argument presented earlier in this section. We do not elaborate.

The following example with $B = 1024$ and $N = 2048$ was found after a brief search:

$$
\begin{aligned}
b_1 = {} & \texttt{D131DD02 C5E6EEC4 693D9A06 98AFF95C 2FCAB587 12467EAB 4004583E B8FB7F89}\backslash \\
& \texttt{55AD3406 09F4B302 83E48883 2571415A 085125E8 F7CDC99F D91DBDF2 80373C5B}\backslash \\
& \texttt{960B1DD1 DC417B9C E4D897F4 5A6555D5 35739AC7 F0EBFD0C 3029F166 D109B18F}\backslash \\
& \texttt{75277F79 30D55CEB 22E8ADBA 79CC155C ED74CBDD 5FC5D36D B19B0AD8 35CCA7E3},
\end{aligned}
$$

$$
\begin{aligned}
b_2 = {} & \texttt{D131DD02 C5E6EEC4 693D9A06 98AFF95C 2FCAB507 12467EAB 4004583E B8FB7F89}\backslash \\
& \texttt{55AD3406 09F4B302 83E48883 25F1415A 085125E8 F7CDC99F D91DBD72 80373C5B}\backslash \\
& \texttt{960B1DD1 DC417B9C E4D897F4 5A6555D5 35739A47 F0EBFD0C 3029F166 D109B18F}\backslash \\
& \texttt{75277F79 30D55CEB 22E8ADBA 794C155C ED74CBDD 5FC5D36D B19B0A58 35CCA7E3},
\end{aligned}
$$

$$
\begin{aligned}
b = {} & \texttt{6DC99F24 E608F367 296D6536 91D7A2D7 4D216E84 8E7AF0AE 1C0E8B9D 59B3F3F1}\backslash \\
& \texttt{D3F6AB04 70832664 2C1AD4B8 E19C43E6 81B97B54 0960D2A2 3F92D141 D25FF166}\backslash \\
& \texttt{B71BEADC 1C34D830 2EFE0453 CFB4B06E F058C6A1 0D9DA967 382B53AD 549F4118}\backslash \\
& \texttt{7294E310 A093A4BD 849CD94D EAE6B25A 85E88C04 41973141 8CD5FFCF 17AF7703},
\end{aligned}
$$

$$
\begin{aligned}
p_1 = {} & \texttt{E8C208AE 3809DD82 969E9DC6 858D6C06 EB811E54 928D2BD9 71CD4847 776B0CB1}\backslash \\
& \texttt{EB7C1DC3 B3C8EE47 87D30965 812D8356 3A041081 019D72D1 205B3CB6 4F35A23F},
\end{aligned}
$$

$$
\begin{aligned}
p_2 = {} & \texttt{EFDA8662 E6AF382B 95011409 17CFC002 078B87C7 BBC6A6EC 7BBA4566 DAD95449}\backslash \\
& \texttt{07F74D4D 58D6002C D7C493A4 1836A8DE AD6C5771 02754860 4F698DF3 D6B7C107}.
\end{aligned}
$$

Here b_1 and b_2 are taken from [22], $b_1||b$ and $b_2||b$ are both 2048-bit integers with 512-bit prime factors p_1 and p_2, respectively, with prime cofactors, and $\text{MD5}(b_1||b) = \text{MD5}(b_2||b) =$ `116346B2 D5C5E569 F4B65C52 B8125B07`. As analyzed in [12], according to the current state of the art in factoring these moduli are as hard to factor as regular 2048-bit RSA moduli.

Special case. Note that, as far as the construction of the moduli is concerned, b_1 and b_2 are arbitrary and may have any part in common. More specifically, with colliding b_1 and b_2 and for any prefix bitstring c, the above method allows construction of moduli $c||b_1||b$ and $c||b_2||b$ where $b_1||b$ and $b_2||b$ collide.

Remark. Given the restrictions of the MD5-collisions as found by the methods from [22] and [23], our method does not allow us to target 1024-bit moduli that collide under MD5, only substantially larger ones. Asymptotically, with growing modulus size but fixed collision size, the prime factors in the moduli ultimately become balanced. The above method can easily be changed to produce a colliding pair of balanced N-bit RSA modulus and N-bit prime. A variation of our construction leads to moduli $b||b_1$ and $b||b_2$, which may be useful for collision purposes if moduli are represented from least to most significant bit.

Colliding X.509 certificates. Based on the ideas presented above we have constructed a pair of X.509 certificates that are different only in the hard to factor RSA moduli, but that have the same CA signature. A detailed description of our approach is given in the Appendix to this note. Briefly, it works as follows. Based on the initial part of the data to be certified, a value of the MD5 chaining variable is determined. Using this value as initialization vector, a pair of 1024-bit values that collide under MD5 is calculated using the methods from [23]. This collision is used as described above to produce two colliding hard to factor 2048-bit moduli, which then enables the construction of two X.509 certificates with identical signatures. Given the current limitations of the MD5-collision methods from [22] and [23], new MD5-based X.509 certificates for 2048-bit RSA moduli should be regarded with more suspicion than X.509 certificates for 1024-bit RSA moduli.

3 Generating DL public keys with prescribed difference

The problem. In the previous section RSA moduli were constructed with a prescribed XOR of small hamming weight by looking for sufficiently many pairs of moduli with a particular integer difference. Thus, the XOR-requirement was translated into a regular integer difference because the latter is something that makes arithmetic sense. In this section we want to generate discrete logarithm related public key values with a prescribed small XOR: for a generator g of some multiplicatively written group of known finite order, we want integers a_1 and a_2 (the secret keys) such that g^{a_1} and g^{a_2} (the public values) have a prescribed small XOR. Obviously, g^{a_1} XOR g^{a_2} depends on the way group elements are represented. For most common representations that we are aware of the XOR operation does not correspond to a mathematical operation that we can work with. Elements of binary fields are an exception: there XOR is the same as addition.

Representation of elements of multiplicative groups of finite fields. If $\langle g \rangle$ lives in a multiplicative group of a prime field of characteristic p, the group elements can be represented as non-zero integers modulo p, and the XOR can, probabilistically if $p > 2$ and deterministically if $p = 2$, be replaced by the regular integer difference modulo p, similar to what was done in Section 2. In this case the resulting requirement $g^{a_1} - g^{a_2} = \delta$ even has the advantage that it makes sense mathematically speaking, since the underlying field allows both multiplication and addition. Because of this convenience, multiplicative groups of prime fields is the case we concentrate on in this section. Multiplicative groups of extension fields have the same advantage, and most of what is presented below applies to that case as well.

Representation issues for elements of other types of groups. Other cryptographically popular groups are groups of elliptic curves over finite fields. In this case the group element g^{a_1} to be hashed[1] is represented as some number of finite field elements that represent the coordinates of certain 'points', either projectively or affinely represented, or in some cases even trickier as just a single coordinate, possibly with an additional sign bit. Given such a representation, it is not always immediately clear how the XOR operation should be translated into an integer subtraction that is meaningful in elliptic curve groups. It is conceivable that, for instance, the integer difference of the x-coordinates allows a meaningful interpretation, again with characteristic 2 fields as a possibly more convenient special case. We leave this topic, and the possibility of yet other groups, for future research.

Restriction to multiplicative groups of prime fields. Unless specified otherwise, in the remainder of this section we are working in the finite field $\mathbf{Z}/p\mathbf{Z}$ with, as usual, multiplication and addition the same as integer multiplication and addition modulo p. The problem we are mostly interested in is: given $\delta \in \mathbf{Z}/p\mathbf{Z}$ find non-trivial solutions to $g^{a_1} - g^{a_2} = \delta$ with $g \in (\mathbf{Z}/p\mathbf{Z})^*$ and integers a_1 and a_2. Several different cases and variants can be distinguished, depending on the assumptions one is willing to make.

Variant I: Prescribed generator g of $(\mathbf{Z}/p\mathbf{Z})^*$ and $\delta \neq 0$. Assume that g is a fixed prescribed generator of $(\mathbf{Z}/p\mathbf{Z})^*$ and that $\delta \neq 0$. Obviously, if the discrete logarithm problem in $\langle g \rangle = (\mathbf{Z}/p\mathbf{Z})^*$ can be solved, $g^{a_1} - g^{a_2} = \delta$ can be solved as well: a solution with any desired non-zero value $z = a_1 - a_2$ can be targeted by finding the discrete logarithm a_2 with

[1] Note that we keep using multiplicative notation for the group operation, and that our "g^{a_1}" would more commonly be denoted "$a_1 g$" in the elliptic curve cryptoworld.

respect to g of $\delta/(g^z-1)$, i.e., a_2 such that $g^{a_2}=\delta/(g^z-1)$. It follows that there are about p different solutions to $g^{a_1}-g^{a_2}=\delta$.

The other way around, however, is unclear: if $g^{a_1}-g^{a_2}=\delta$ can be solved for a_1 and a_2, can the discrete logarithm problem in $\langle g\rangle=(\mathbf{Z}/p\mathbf{Z})^*$ be solved? Annoyingly, we don't know. Intuitively, the sheer number of solutions to $g^{a_1}-g^{a_2}=\delta$ for fixed δ and g seems to obstruct all attempts to reduce the discrete logarithm problem to it. This is illustrated by the fact that if the $g^{a_1}-g^{a_2}=\delta$ oracle would produce solutions a_1, a_2 with fixed $z=a_1-a_2$, the reduction to the discrete logarithm problem becomes straightforward: to solve $g^y=x$ for y (i.e., given g and x), apply the $g^{a_1}-g^{a_2}=\delta$ oracle to $\delta=(g^z-1)x$ and set y equal to the resulting a_2.

Lacking a reduction for the general case (i.e., non-fixed a_1-a_2) from the discrete logarithm problem, neither do we know if, given δ and g, solving $g^{a_1}-g^{a_2}=\delta$ for a_1 and a_2 is easy. We conjecture that the problem is hard, and pose the reduction from the regular discrete logarithm problem to it as an interesting open question.

Summarizing, if $\delta\neq 0$ and g is a given generator of the full multiplicative group modulo p, the problem of finding a_1, a_2 with $g^{a_1}-g^{a_2}=\delta$ is equivalent to the discrete logarithm problem in $\langle g\rangle$ if a_1-a_2 is fixed, and the problem is open (but at most as hard as the discrete logarithm problem) if a_1-a_2 is not pre-specified.

Variant II: Prescribed generator g of a true subgroup of $(\mathbf{Z}/p\mathbf{Z})^*$ and $\delta\neq 0$. Let again $\delta\neq 0$, but now let g be a fixed prescribed generator of a true subgroup of $(\mathbf{Z}/p\mathbf{Z})^*$. For instance, g could have order q for a sufficiently large prime divisor q of $p-1$, in our opinion the most interesting case for the hash collision application that we have in mind. If $z=a_1-a_2$ is pre-specified, not much is different: a solution to $g^{a_1}-g^{a_2}=\delta$ exists if $\delta/(g^z-1)\in\langle g\rangle$ and if so, it can be found by solving a discrete logarithm problem in $\langle g\rangle$, and the discrete logarithm problem $g^y=x$ given an $x\in\langle g\rangle$ can be solved by finding a fixed $z=a_1-a_2$ solution to $g^{a_1}-g^{a_2}=(g^z-1)x$.

But the situation is unclear if a_1 and a_2 may vary independently: we do not even know how to establish whether or not a solution exists. We observe that for the cryptographically reasonable case where g has prime order q, with q a 160-bit prime dividing a 1024-bit $p-1$, the element $g^{a_1}-g^{a_2}$ of $\mathbf{Z}/p\mathbf{Z}$ can assume at most $q^2\approx 2^{320}$ different values. This means that the vast majority of unrestricted choices for δ is infeasible and that a δ for which a solution would exist would have to be constructed with care. However, the δ's that we are interested in have low hamming weight. This makes it exceedingly unlikely that a solution exists at all. For instance, for $H(\delta)=6$ there are fewer than 2^{51} different δ's. For each of these δ we may assume that it is of the form $g^{a_1}-g^{a_2}$ with probability at most $\approx 2^{320}/2^{1024}$. Thus, with overwhelming probability, none of the δ's will be of the form $g^{a_1}-g^{a_2}$. And, even if one of them has the proper form, we don't know how to find out.

Variant III: Free choice of generator of $(\mathbf{Z}/p\mathbf{Z})^*$ and $\delta\neq 0$. Now suppose that just $\delta\neq 0$ is given, but that one is free to determine a generator g of $(\mathbf{Z}/p\mathbf{Z})^*$, with p either given or to be determined to one's liking. Thus, the problem is solving $g^{a_1}-g^{a_2}=\delta$ for integers a_1 and a_2 and a generator g of the multiplicative group $(\mathbf{Z}/p\mathbf{Z})^*$ of a prime field $\mathbf{Z}/p\mathbf{Z}$. Not surprisingly, this makes finding solutions much easier. For instance, one could look for a prime p and small integers u and v such that the polynomial $X^u-X^v-\delta\in(\mathbf{Z}/p\mathbf{Z})[X]$ has a root $h\in(\mathbf{Z}/p\mathbf{Z})^*$ (for instance, by fixing $u=2$ and $v=1$ and varying p until a root exists). Next, one picks a random integer w coprime to $p-1$ and calculates $g=h^{1/w}$, $a_1=uw \bmod (p-1)$, and $a_2=vw \bmod (p-1)$. As a result $g^{a_1}-g^{a_2}=\delta$. With appropriately chosen p it can quickly be verified if g is indeed a generator; if not, one tries again with a different w or p, whatever is appropriate.

Obviously, this works extremely quickly, and solutions to $g^{a_1} - g^{a_2} = \delta$ can be generated on the fly. The disadvantage of the solution is, however, that any party that knows a_1 (or a_2) can easily derive a_2 (or a_1) because $va_1 = ua_2 \bmod (p-1)$ for small u and v. In our 'application' this is not a problem if one wants to spoof one's own certificate. Also, suspicious parties that do not know either a_1 or a_2 may nevertheless find out that g^{a_1} and g^{a_2} have matching small powers. It would be much nicer if the secrets (a_1 and a_2) are truly independent, as is the case for our RSA solution. We don't know how to do this. Similarly, we do not know how to efficiently force g into a sufficiently large but relatively small (compared to p) subgroup.

Variant IV: Two different generators, any δ. In our final variant we take g again as a generator of $(\mathbf{Z}/p\mathbf{Z})^*$, take any $\delta \in \mathbf{Z}/p\mathbf{Z}$ including $\delta = 0$, and ask for a solution h, a_1, a_2 to $g^{a_1} - h^{a_2} = \delta$. Obviously, this is trivial, even if a_1 is fixed or kept secret by hiding it in g^{a_1}: for an appropriate a_2 of one's choice compute h as the a_2th root of $g^{a_1} - \delta$. For subgroups the case $\delta \neq 0$ cannot be expected to work, as argued above.

The most interesting application of this simple method is the case $\delta = 0$. Not only does $\delta = 0$ guarantee a hash collision, it can be made to work in any group or subgroup, not just the simple case $(\mathbf{Z}/p\mathbf{Z})^*$ we are mostly considering here, and g and h may generate entirely different (sub)groups, as long as the representations of the group elements is sufficiently 'similar': for instance, an element of $(\mathbf{Z}/p\mathbf{Z})^*$ can be interpreted as an element of $(\mathbf{Z}/p'\mathbf{Z})^*$ for any $p' > p$, and most of the time vice versa as long as $p' - p$ is relatively small. Because, furthermore, just g^{a_1} but not a_1 itself is required, coming up with one's own secret exponent and generator (possibly of another group) seems to be the perfect way to spoof someone else's certificate on g^{a_1}. It follows that in practical cases of discrete logarithm related public keys, information about the generator and (sub)group (the *system parameters*) must be included in the certificate or that the system parameters must be properly authenticated in some other way.

This illustrates once more that one should never trust a generator whose construction method is not specified, since it may have been concocted to collide, for some exponents, with a 'standard' or otherwise prescribed generator. This has been known for a long time, cf. [17] and [2], and, according to [28], this issue came up in the P1363 standards group from time to time. Nevertheless it still seems to escape the attention of many implementors and practitioners.

Remark on actually colliding powers of a fixed g. As shown above, $\delta = 0$ and the freedom to select a generator makes it trivial to generate actually colliding powers. One may wonder if less straightforward examples with a fixed generator g can be constructed in a way similar to the construction shown at the end of Section 2. Let N be such that the elements of $\langle g \rangle$ can be represented as bitstrings of length N, and let (b_1, b_2) be a pair of B-bit values that collide under a Merkle-Damgård hash. The question is if an $(N-B)$-bit value b and integers a_1 and a_2 can be found such that the colliding values $b_1||b$ and $b_2||b$ satisfy $b_1||b = g^{a_1}$ and $b_2||b = g^{a_2}$. We don't know how to do this—except that it can be done in any number of ways if discrete logarithms with respect to g can be computed. The ability to solve Variant I, however, makes it possible to solve the related problem of finding b such that $b_1 2^{N-B} + b = g^{a_1}$ and $b_2 2^{N-B} + b = g^{a_2}$: simply take $\delta = (b_1 - b_2)2^{N-B}$, apply Variant I to find a_1 and a_2 with $g^{a_1} - g^{a_2} = \delta$ and define $b = g^{a_1} - b_1 2^{N-B}$, which equals $g^{a_2} - b_2 2^{N-B}$. Unfortunately, the resulting b will in general not be an $(N-B)$-bit value, so that the '+' cannot be interpreted as '||', and the resulting pair (g^{a_1}, g^{a_2}) will most likely no longer collide.

4 Generating colliding DL system parameters

John Kelsey suggested on a mailing list to generate Diffie-Hellman system parameters (specifically a large prime) for which a collision with cryptographically weak system parameters exists, to facilitate compromising private keys. Immediately Ben Laurie produced a large prime and a composite replacement with the same MD5-value (cf. [9]). Laurie's composite number, however, seems to be far from smooth and is hardly useful for the intended purpose. Therefore, the question is raised whether we can produce large primes p for which the discrete logarithm problem in $(\mathbf{Z}/p\mathbf{Z})^*$ is hard and that collide (e.g. for MD4 or MD5) with moduli for which the discrete logarithm problem is easy.

Denote by p_1 and p_2 the colliding moduli. We assume that p_1 is prime and that the discrete logarithm problem in the multiplicative group $(\mathbf{Z}/p_1\mathbf{Z})^*$ is hard. This means that p_1 should be large enough (i.e., say, 1024 bits) and that $p_1 - 1$ should contain a prime factor of, say, at least 160 bits. The last requirement complicates the description somewhat and may, if the large prime order subgroup is not explicitly needed, be omitted based on the argument that in most cases such a prime factor will exist. The number p_2 must be chosen in such a way that discrete logarithms modulo p_2 are easy. This can be achieved as follows.

1. Construct p_2 such that it is the product of relatively small primes. Discrete logarithms modulo p_2 can be calculated by computing them in the finite fields defined by the prime factors of p_2. This can effectively be done using subexponential-time index calculus based methods if the prime factors are at most, say, 400 bits.
2. Construct p_2 such that it is prime but such that the prime factors of $p_2 - 1$ are small enough so that discrete logarithms in $(\mathbf{Z}/p_2\mathbf{Z})^*$ can be computed using the Pohlig-Hellman method. This means that the prime factors of $p_2 - 1$ should be at most about 100 bits.
3. Combining the two methods above: a composite p_2 such that the finite fields defined by the prime factors of p_2 have multiplicative groups with orders divisible by primes of at most about 100 bits.

Construction of pairs of colliding moduli (p_1, p_2) based on an existing hash collision is straightforward, and in practice a bit cumbersome. Below we sketch how pairs may be constructed that satisfy one of the first two possibilities for p_2.

Let b_1, b_2 be a known pair of colliding B-bit values. If a large enough prime factor is explicitly desired in $p_1 - 1$, then generate a 160-bit prime q. Generate a number of small primes of, say, 32 bits, such that their product M is approximately B bits long (or $\approx B - 160$, if q has been generated). Values b can now be constructed, efficiently and in large quantities, such that $p_1 = b_1||b$ and $p_2 = b_2||b$ are $2B$-bit numbers, the large smooth factor M either divides p_2 or $p_2 - 1$ (depending on whether the first or the second possibility for p_2 is chosen) and, if applicable, q divides $p_1 - 1$. Among those b's, look for values such that p_1 is prime, and such that p_2/M or $(p_2 - 1)/M$ has all prime factors of the required size. This requires factoring an approximately B-bit (or $B + 160$-bit, if q is used) number, which sometimes may be doable, but often will be difficult.

To give an indication how many b's are needed, we consider the easiest case where $B = 512$ (as for the MD4 collisions from [22]), q is not used, and where we attempt to realize the first possibility for p_2. Let $\psi(x, y)$ be the number of y-smooth integers below x. Based on De Bruijn's estimate in [3]

$$\log \psi(x, y) \approx \frac{\log x}{\log y} \log\left(1 + \frac{y}{\log x}\right) + \frac{y}{\log y} \log\left(1 + \frac{\log x}{y}\right)$$

(neglecting error terms) we estimate that we have to generate 1.3 million b values before a good one turns up. This is feasible, despite the fact that each b requires a primality test (for p_1), possibly followed by a smoothness test on a number of approximately 512 bits (p_2/M). For $B = 1024$ (as for the MD5 collisions from [22]), however, one may expect that the number of b's to be inspected grows by a factor of at least 10^4, and the numbers involved get considerably larger. For instance, the smoothness tests would have to be applied to approximately 1024-bit numbers. Thus, constructing p_1 and p_2 for $B = 1024$ becomes a rather time-consuming task.

We mention just one example that we generated using a known MD4-collision (cf. [22]):

$b_1 =$ 839C7A4D 7A92CB56 78A5D5B9 EEA5A757 3C8A74DE B366C3DC 20A083B6 9F5D2A3B\
B3719DC6 9891E9F9 5E809FD7 E8B23BA6 318EDD45 E51FE397 08BF9427 E9C3E8B9,

$b_2 =$ 839C7A4D 7A92CBD6 78A5D529 EEA5A757 3C8A74DE B366C3DC 20A083B6 9F5D2A3B\
B3719DC6 9891E9F9 5E809FD7 E8B23BA6 318EDC45 E51FE397 08BF9427 E9C3E8B9,

$b =$ 13F449AF C2986A9E 529F545E 70E08FD0 54E5A316 EF7909EE 5157F452 236A8B1A\
C6945C7F 0EC7C00D 09E36FB8 03D954F3 B31E82C3 89A7DFD2 3A84A6FA CF35AA79,

where p_1 is a prime without special properties, and $p_2 = b_2||b$ has the prime factorization

3 × B6F × 2B97 × 8105 × 817D × 8225 × 8447 × 85A3 × 85EB × 87DD × 8AB5 × 9043 × 92A1 × 944B × 95E3 × 96FB × 997D × 9B9F × 9D15 × 9DE7 × A141 × A175 × A243 × A26B × A4F3 × A56D × A5D9 × A673 × AB5B × B01B × B17F × B1A9 × B567 × B951 × B993 × 2D061 × 4C24E1 × D3357A5 × 16164973 × 7FD131763 × 98BB302F87 × 20A7312C4827D × 6AFB9B7C2BE3A759 × 22EDF99B7227D62C8846F × 1780C6C1BB502D4E9F6627C7B47519E02D95B.

Here the largest prime factor has 145 bits, so p_2 can be considered sufficiently smooth. Generating this example took several hours, the bottleneck being the factorization attempts of the candidate p_2's.

5 Attack scenarios and applications

We describe some possible (ab)uses of colliding public keys. None of our examples is truly convincing, and we welcome more realistic scenarios.

One possible scenario is that Alice generates colliding public keys for her own use. We assume that it is possible to manufacture certificates for these public keys in such a way that the parts of the certificates that are signed by a Certification Authority (CA) also collide, so that the signatures are in fact identical. For RSA we have shown how this goal can actually be achieved for X.509 certificates. Then Alice can ask the CA for certification of one of her public keys, and obtain a valid certificate. By replacing the public key with the other one, she can craft a second certificate that is equally valid as the first one. If so desired this can be done without any involvement of the CA, in which case she obtains two valid certificates for the price of only one.

The resulting certificates differ in only a few bit positions in random looking data, and are therefore hard to distinguish by a cursory glance of the human eye. For standard certificate validating software both certificates will be acceptable, as the signature can be verified with the CA's public key.

A ‘positive’ application of the pairs of X.509 certificates would be that it enables Alice to distribute two RSA certificates, one for encryption and the other for signature purposes, for the transmission cost of just one certificate plus the few positions where the RSA moduli differ (similar ideas will be worked out in [15]). Indeed, the CA may knowingly participate in this application and verify that Alice knows both factorizations. However, if that is not done and the CA is tricked into signing one of the keys without being aware of the other one, the principle underlying Public Key Infrastructure that a CA guarantees the binding between an identity and a public key, has been violated. A CA usually requires its customers to provide proof of possession of the corresponding private key, to prevent key substitution attacks in which somebody tries to certify another person’s public key in his own name. Although the way our certificates have been constructed makes it highly improbable that somebody could come up with either of them independent of Alice, it should be clear that the proof of possession principle has been violated. It would be more interesting to be able to produce two colliding certificates that have differences in the subject name, but at present this seems infeasible because it requires finding a second preimage.

Alice can also, maliciously, spread her two certificates in different user groups (different in space or time). When Bob sends Alice an encrypted message that has been encrypted by means of the wrong certificate, Alice may deny to be able to read it. When however the dispute is seriously investigated, it will be revealed that Alice has two colliding certificates. Alice may claim that she does not know how this is possible, but as finding second preimages still is prohibitively expensive, it is clear that either Alice is lying, or she has been misled by the key pair generating software.

Alice can produce digital signatures with one key pair, that are considered perfectly valid in one user group, and invalid in the other. This may be convenient for Alice, when she wants to convince one person of something, and to deny it to another person. Again, on serious investigation the colliding certificates will be revealed.

Another possible scenario is that Alice does not generate key pairs herself, but obtains her key pair(s) from a Key Generation Centre (KGC). This KGC may maliciously produce colliding public keys, of which one is sold to Alice, and the other one kept for the KGC’s own use, without Alice’s consent. The KGC can distribute Alice’s false certificate to Bob, and then Bob, when he thinks he is sending a message that only Alice can decrypt, ends up sending a message that only the KGC or a party collaborating with it can decrypt. Furthermore, when Alice sends a signed message to Bob, Bob will not accept her signature. So this constitutes a small denial of service attack. Note that a KGC in principle *always* has the possibility to eavesdrop on encrypted messages to Alice, and to spoof her signature. Our ability to construct colliding certificate does not add much value to this malicious application.

In all the above cases, when the colliding public keys are both secure keys, it cannot be detected from one key (or one certificate) that it has a twin sister. When e.g. one of the colliding public keys is intentionally weak, e.g. a prime as opposed to a composite modulus, this can be in principle detected by compositeness testing. Unless there is a concrete suspicion such tests are not carried out in practice, since they would make the public operation substantially more costly.

For the case of colliding DL-parameters a realistic scenario has already been described by John Kelsey [9]. Note that in this case the ‘fake’ DL-prime can be detected by compositeness testing or factoring attempts.

In conclusion it seems that possibilities for abuse seem not abundant, as the two public keys are very much related, and generated at the same time by the same person. Nevertheless, the principle of Public Key Infrastructure, being a certified binding between an identity and a public key, is violated by some of the scenarios we have described, based on random collisions for (a.o.) the hash function MD5, which is still popular and in use by certificate generating institutions. Particularly worrying is that any person, including the certificate owner, the Certification Authority, and any other party trusting a certificate, cannot tell from the information in one certificate whether or not there exists a second public key or certificate with the same hash or digital signature on it. In particular, the relying party (the one that does the public key operation with somebody else's public key) cannot be sure anymore of the Certification Authority's guarantee that the certificate owner indeed is in possession of the corresponding private key.

6 Conclusion

We demonstrated that on the basis of the existence of random hash collisions, in particular those for MD5 as shown by Wang et al. in [22], one can craft public keys and even valid certificates that violate one of the principles underlying Public Key Infrastructures. We feel that this is an important reason why hash functions that have been subject to collision attacks should no longer be allowed in certificate generation.

Acknowledgments. Acknowledgments are due to Hendrik W. Lenstra, Berry Schoenmakers, Xiaoyun Wang, Mike Wiener and an anonymous referee for helpful remarks and fruitful discussions.

Appendix† - Colliding X.509 certificates

by Arjen Lenstra, Xiaoyun Wang[3] and Benne de Weger

Introduction. We describe in detail our method for the construction of pairs of valid X.509 certificates in which the "to be signed" parts form a collision for the MD5 hash function. As a result the issuer signatures in the certificates will be the same when the issuer uses MD5 as its hash function. Furthermore we provide an explicit example.

Construction outline. Our method constructs X.509 certificates in which all fields except the public key can be taken arbitrary. We use specially crafted but secure public RSA keys.

The heart of our construction is that, starting from a specially crafted MD5-collision produced by the method of Wang et al. [23], we can construct a pair of different RSA moduli that yield a collision for the MD5 compression function. Due to the ability of this method to produce MD5 compression function collisions for any IV, and due to the iterative structure of MD5, we can append a collision to any block of data of our choice, while maintaining the collision property. Similarly we can then append data of our choice to the constructed collisions. In this way we can build colliding certificates.

The RSA moduli are secure in the sense that they are built up from two large primes. Due to our construction these primes have rather different sizes, but since the smallest still are around 512 bits in size while the moduli can be made to have 2048 bits, this does not constitute a realistic vulnerability, as far as we know.

Construction details. We provide a detailed description of our construction.

1. We first construct a template for the certificate, in which all fields are completely filled in, with the exception of the RSA public key modulus and the signature (apart from a first zero byte which is there to prevent the bitstring from representing a negative integer). We can easily meet the following three requirements:
   - the data structure should be compliant to the X.509 standard [7] and the ASN.1 DER encoding rules;
   - the byte lengths of the modulus and the public exponent have to be fixed in advance;
   - the position where the public key modulus starts should be an exact multiple of 64 bytes after the beginning of the "to be signed" part.

   The third condition can e.g. be dealt with by adding some dummy information to the subject Distinguished Name. Using the special case mentioned in Section 2, it is possible to avoid the third condition by adding an additional random prefix bitstring (c in Section 2) to the moduli, but this means that the moduli will become longer. Note that the public key exponent bitlength has to be fixed in advance, but that 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.
2. We run the MD5 algorithm on the first portion of the "to be signed" part, making sure that the bitlength of the input to MD5 is an exact multiple of 512. The latter can be done either by adding dummy information to the subject Distinguished Name, or by randomly selecting a bitstring c of appropriate length that will act as the prefix for the RSA moduli. We suppress the padding normally used in MD5, and then get as output an IV that we use as input for the next step.

† This appendix is an adapted version of [13].

[3] School of Mathematics and System Sciences, Shandong University, Jinan 250100, China

3. Using the techniques developed in [23] we construct two different bitstrings b_1 and b_2, of 1024 bits each, for which the MD5 compression function with the IV from the previous step produces a collision.
4. The next step is to construct two RSA moduli from these bitstrings b_1 and b_2 respectively, by appending to each the same bitstring b, also of 1024 bits. This we do as follows, following the method outlined in Section 2:
   – generate random primes p_1 and p_2 of approximately 512 bits, 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 $p_1 | b_1 2^{1024} + b_0$ and $p_2 | b_2 2^{1024} + b_0$ (by the Chinese Remainder Theorem);
   – let k run through $0, 1, 2, \ldots$, and for each k compute $b = b_0 + k p_1 p_2$; check whether both $q_1 = (b_1 2^{1024} + b)/p_1$ and $q_2 = (b_2 2^{1024} + b)/p_2$ are primes, and whether e is coprime to both $q_1 - 1$ and $q_2 - 1$;
   – when k has become so large that $b \geq 2^{1024}$, restart with new random primes p_1, p_2;
   – when primes q_1 and q_2 have been found, stop, and output $n_1 = b_1 2^{1024} + b$ and $n_2 = b_2 2^{1024} + b$ (as well as p_1, p_2, q_1, q_2).

   As mentioned above, if the prefix c is used, then b_1 and b_2 in the above description can be replaced by $c||b_1$ and $c||b_2$, respectively, increasing the lengths of the resulting moduli by the length of c. It is reasonable to expect, based on the Prime Number Theorem, that this algorithm will produce in a feasible amount of computation time, two RSA moduli $n_1 = p_1 q_1$ and $n_2 = p_2 q_2$, that will form an MD5-collision with the specified IV. When the smaller primes p_1 and p_2 are around 500 bits in size, this algorithm usually returns a result in a few minutes of computation time. When this bitsize increases towards 512 the computation time grows considerably, because the search range for k then becomes almost empty. Nevertheless we have been able to find results with exactly 512-bit p_1, p_2 and 1536-bit q_1, q_2 in a few days of computation time.
5. We insert the modulus n_1 into the certificate. Now the "to be signed" part is complete, and we compute the MD5 hash of the entire "to be signed" part (including MD5-padding, and using the standard MD5-IV).
6. We apply standard PKCS#1v1.5-padding [18], 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 now is complete.
7. To obtain the second valid certificate, all we have to do is to replace n_1 for n_2 as the public key modulus. The signature remains valid.

Note that the prime factors of each modulus have rather different sizes. Although this is unusual, for the parameter choices we make (smallest primes of around 500 bits for a modulus of 2048 bits) we see no reason to believe that these moduli are insecure, 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 MD5 collisions seems to be the computationally hardest part of our method, unless one insists on a bitsize for the smallest primes of at least 512.

Example. Below is an example pair of colliding certificates in full detail.

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/~bdeweger/CollidingCertificates/`.

Both certificates are valid in the sense that they comply with the relevant standards (RFC 3280, ASN.1 DER encoding), and also in the sense that their digital signature can be verified against the issuing Certification Authority's certificate. The reader may verify this using widely available tools such as `OpenSSL`, Peter Gutmann's `dumpasn1`, and certificate viewing programs such as the one in Microsoft Windows.

In the left column the exact bytes are presented in a form that clarifies the ASN.1 structure.

```
tag length data                              | comment
=== ====== ================================= | ====================================================
 30 820335                                   | ASN.1 header
-------------------------------------------- | ----------------------------------------------------
 30 82021D                                   | ``to be signed'' part begins here
 A0 03                                       |
 02 01     02                                | X.509 version 3
 02 04     03507449                          | serial number (0x03507449)
 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     3035303230313030303030315A        | not valid before (Feb. 1, 2005, 0h0m1s)
 17 0D     3037303230313030303030315A        | not valid after  (Feb. 1, 2007, 0h0m1s)
-------------------------------------------- | ----------------------------------------------------
 30 60                                       | subject distinguished name starts here
 31 17                                       |
 30 15                                       |
 06 03     550403                            |
 13 0E     4861736820436F6C6C6973696F6E      | subject common name (``Hash Collision'')
 31 24                                       |
 30 22                                       |
 06 03     55040A                            |
 13 1B     77652075736564206120636F6C6C6973  | subject organization (``we used a collision for MD5'')
           696F6E20666F72204D4435            | (dummy text, used to fill up to multiple of 64 bytes)
 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 820122                                   |
-------------------------------------------- | ----------------------------------------------------
 30 0D                                       |
 06 09     2A864886F70D010101                | public key algorithm (rsaEncryption)
 05 00                                       |
```

```
 03 82010F 00                              | subject public key info
 30 82010A                                 |
 02 820101 00                              | public key modulus (2048 bits, 257 bytes)
                                           | ``to be signed'' part until here has a multiple of 64 bytes
                                           | different bytes are indicated by colors and underlining
                                           \----------------------------------\
           (certificate #1)                  (certificate #2)                 |
           CAB9E742C4B626871AB9A524846B05C1  CAB9E742C4B626871AB9A524846B05C1 |
           8895FB9365E9A69F480392FF2C3B3F79  8895FB1365E9A69F480392FF2C3B3F79 |
           41AD3406FFADB4034BDF847A4D37014F  41AD3406FFADB4034BDF847A4DB7014F |
           DB3283CB19D46FA8A765C6B3F016BF30  DB3283CB19D46FA8A765C633F016BF30 |
           6AFF7C2E5773689B3319B81564ABE7F5  6AFF7C2E5773689B3319B81564ABE7F5 |
           B9CF66C5E4FE790CEE047D36CC77B0AE  B9CF6645E4FE790CEE047D36CC77B0AE |
           5D087F30B560EB8872B34D4067786 62D  5D087F30B560EB8872B34D4067F8652D |
           D88464677DBD9B80989EF24FB82E0EA3  D88464677DBD9B80989EF2CFB82E0EA3 |
           2B5864AF33B8FE8659B094464699F477  2B5864AF33B8FE8659B094464699F477 |
           A6BFCA348C23CF681EC0A846A8B27A29  A6BFCA348C23CF681EC0A846A8B27A29 |
           071B563A1316B05F3827B82FB1F9DE1F  071B563A1316B05F3827B82FB1F9DE1F |
           238F3D12AD0DDAA97DDBCFCEEAD10939  238F3D12AD0DDAA97DDBCFCEEAD10939 |
           5E46E018AE237CE59355AC931872284C  5E46E018AE237CE59355AC931872284C |
           3A293FE9117941A1AD528364A0687AFF  3A293FE9117941A1AD528364A0687AFF |
           6083B14B009DD952C866CA43A0F41A7D  6083B14B009DD952C866CA43A0F41A7D |
           CE5876C16CB346E9A718091CEC3D57D9  CE5876C16CB346E9A718091CEC3D57D9 |
                                           /----------------------------------/
 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)
           1319E6FF66EF8621AEAE0CFBD2C067B9 |
           9C3834C00BE88E0A97E60205BC5ECD85 |
           646B6698BD2E91324826C8B10E2167EF |
           F264C5E45A234FDE5723A751EA2B7913 |
           06221B54B4C20E4CD16562D698ADE4D6 |
           33F053D653F8BE9C4D402EC9F92D3630 |
           98DD560596F7BF095AF3C9FED7EE2B49 |
           218018003F5C65F0511D454E6E522913 |
           2D0494B7B65EF9585AA9D433094FDB4F |
           9C994610AFE0F23FB26E5D246539AEFF |
           B6E0B0DF35B4D9AE3CF768C5AABC9355 |
           8DF87BF421288E79E9ADCBB8DA236452 |
           8E74F81348FFB9F5FAC43E974F3D79CC |
           A222FD675BFD3B808A3F66104232C806 |
           A25309A187D103D750893436D4A32909 |
           FE5C76B45495F52F29CF66A9E3DD473F |
=== ====== ================================ | =====================================================
```

References

1. D.J. Bernstein, *Circuits for integer factorization: a proposal*, manuscript, November 2001; available at `cr.yp.to/papers.html#nfscircuit`.
2. D. Bleichenbacher, *Generating ElGamal signatures without knowing the secret key*, Advances in Cryptology - EUROCRYPT '96, LNCS vol. 1070, Springer Verlag, pp. 10-18, 1996.
3. N.G. De Bruijn, *On the number of positive integers* $\leq x$ *and free of prime factors* $> y$*, II*, Indag. Math. **38** (1966) 239-247.
4. R.D. Dean *Formal aspects of mobile code security*, PhD thesis, Princeton University, January 1999, `http://www.cs.princeton.edu/sip/pub/ddean-dissertation.php3A`.
5. H. Dobbertin, *Alf swindles Ann*, Cryptobytes 1 (3) (1995), p. 5.
6. *Recent collision attacks on hash functions: ECRYPT position paper*, revision 1.1, February 2005, `http://www.ecrypt.eu.org/documents/STVL-ERICS-2-HASH_STMT-1.1.pdf`.
7. 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`.
8. D. Kaminsky, *MD5 to be considered harmful someday*, preprint, December 2004, `http://www.doxpara.com/md5_someday.pdf`.
9. J. Kelsey, B. Laurie, Contributions to the mailing list "cryptography@metzdowd.com", December 22, 2004, available at `http://diswww.mit.edu/bloom-picayune/crypto/16587`.
10. J. Kelsey, B. Schneier, *Second preimages on* n*-bit hash functions for much less than* 2^n *work*, Cryptology ePrint Archive, Report 2004/304, `http://eprint.iacr.org/2004/304`.
11. A.K. Lenstra, *Generating RSA moduli with a predetermined portion*, Asiacrypt'98, Springer-Verlag LNCS 1514 (1998), 1–10.
12. A.K. Lenstra, *Unbelievable security*, Asiacrypt 2001, LNCS 2248, Springer-Verlag 2001, 67–86.
13. 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`.
14. 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.
15. A.K. Lenstra and B.M.M. de Weger, *Twin RSA*, submitted for publication, April 2005.
16. O. Mikle, *Practical Attacks on Digital Signatures Using MD5 Message Digest*, Cryptology ePrint Archive, Report 2004/356, `http://eprint.iacr.org/2004/356`.
17. NIST, *Digital Signature Standard*, NIST FIPS PUB 186, US Dept. of Commerce, May 1994.
18. *PKCS #1 v2.1: RSA Cryptography Standard*, RSA Laboratories, June 2002, `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.
19. J. Randall, M. Szydlo, *Collisions for SHA0, MD5, HAVAL, MD4, and RIPEMD, but SHA1 still secure*, RSA Laboratories technical notes, `http://www.rsasecurity.com/rsalabs/node.asp?id=2738`.
20. E. Rescorla, *What's the Worst That Could Happen?* presentation at the DIMACS Workshop on Cryptography: Theory Meets Practice October 14–15, 2004, `http://dimacs.rutgers.edu/Workshops/Practice/slides/rescorla.pdf`.
21. A. Shamir, *RSA for paranoids*, Cryptobytes 1 (3) (1995) 1–4.
22. X. Wang, D. Feng, X. Lai, H. Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, Cryptology ePrint Archive, Report 2004/199, `http://eprint.iacr.org/2004/199`, presented at the Crypto 2004 rump session, August 17, 2004.
23. X. Wang and H. Yu , *How to Break MD5 and Other Hash Functions*, EuroCrypt 2005, Springer LNCS 3494 (2005), 19–35.
24. X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, *Cryptanalysis of the Hash Functions MD4 and RIPEMD*, EuroCrypt 2005, Springer LNCS 3494 (2005), 1–18.
25. X. Wang, H. Chen, X. Yu, *How to Find Another Kind of Collision for MD4 Efficiently*, Preprint, 2004.
26. X. Wang, D. Feng, X. Yu, *An Attack on Hash Function HAVAL-128*, Science in China Ser. F (Information Sciences) 35(4) (2005), 405–416 (in Chinese).
27. M.J. Wiener, *The full cost of cryptanalytic attacks*, Journal of Cryptology 17 (2004) 105–124.
28. M.J. Wiener, personal communication, November 17, 2004.