On the Possibility of Constructing Meaningful Hash Collisions for Public Keys

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 that finding meaningful hash collisions might prove difficult. We show 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 by Wang et al. We present some simple results, investigate what we can and cannot (yet) achieve, and formulate some open problems of independent interest. 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. For instance, we show how to construct two different X.509 certificates that contain identical signatures.

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. [14]), and since then described in more detail in [15], [16], [17], and [18], 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. Their application for integrity protection of binary data and 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.

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 public key cryptography, see [4] and [9] for interesting ideas about the application of hash collisions in other areas.

A successful attack on an existing certificate 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 [14] do not imply that second preimages are essentially easier to find than they should, namely with an effort proportional

C. Boyd and J.M. González Nieto (Eds.): ACISP 2005, LNCS 3574, pp. 267–279, 2005.

to 2^n for an n-bit hash function. Therefore certificates that existed well before the results from [14] were obtained should be fine.

For newly to be constructed certificates the argument goes that random collisions do not suffice because the values to be hashed are *meaningful* (cf. [3] and [11]). Dobbertin's cryptanalytic work on MD4 was so strong that meaningful collisions could be found easily, cf. [2]. The recent results of [14] 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. Nevertheless, it contains pieces of data that look random, and may have been constructed to fit a hash collision. In particular 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 [5], given any hash collision 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 [14] 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 in [14]?

Having the right type of difference structure does not, as far as we know, imply a hash collision. 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*. 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. We show how this may be used to construct well-formed and secure public keys 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. [12]).

Furthermore, using the appending trick, we show how we can generate actually colliding pairs consisting of proper public RSA keys. Combining this construction with the prepending idea, we show how different X.509 certificates can be constructed that have identical signatures. 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 the 'meaningful message' argument against hash collisions in certification applications 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. 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. 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. Some attack scenarios and applications that use our constructions are sketched in Section 4. The full version [7] of this paper contains some more detail, an additional section on the practicality of generating colliding DL system parameters, à la Kelsey and Laurie [5], and an appendix giving full details of our construction of colliding X.509 certificates.

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.

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, and selecting two proper public exponents that differ in the right way is trivial.

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 [14], 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. It turns out that 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.

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 [13]. Our method combines the ideas mentioned in the introduction and earlier in this section with the construction from [6].

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 [14], 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, $0 \le b_0 < p_1p_2$ such that p_i divides $b_i 2^{N-B} + b_0$ for $i = 1, 2$. Finally, look for the smallest integer $k \ge 0$ 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 $m_i = p_iq_i = b_i||b$ 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 [6]; 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.

Remark. Given the restrictions of the MD5-collisions as found by the methods from [14] and [15], 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 full version of this note, cf. [7]. 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 [15]. 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 [14] and [15], 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[3] is represented as some number

[3] 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.

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 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. [10] and [1], and, according to [19], 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 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 [8]). 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.

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.

5 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 [14], 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.

References

1. D. Bleichenbacher, *Generating ElGamal signatures without knowing the secret key*, EuroCrypt '96, LNCS vol. 1070, Springer Verlag, pp. 10-18, 1996.
2. Hans Dobbertin, *Alf swindles Ann*, Cryptobytes 1(3) (1995), p. 5.
3. *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`.
4. D. Kaminsky, *MD5 to be considered harmful someday*, preprint, december 2004, `http://www.doxpara.com/md5_someday.pdf`.
5. 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`.
6. A.K. Lenstra, *Generating RSA moduli with a predetermined portion*, Asiacrypt'98, Springer-Verlag LNCS 1514 (1998), 1–10.
7. A.K. Lenstra and B. de Weger, *On the possibility of constructing meaningful hash collisions for public keys, full version, with an appendix on colliding X.509 certificates*, April 2005, `http://www.win.tue.nl/~bdeweger/CollidingCertificates/ddl-full.pdf`.
8. A.K. Lenstra and B. de Weger, *Twin RSA*, submitted for publication, April 2005.
9. O. Mikle, *Practical Attacks on Digital Signatures Using MD5 Message Digest*, eprint archive 2004/356, `http://eprint.iacr.org/2004/356`.
10. NIST, *Digital Signature Standard*, NIST FIPS PUB 186, 1994.
11. 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`.
12. 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`.
13. A. Shamir, *RSA for paranoids*, RSA Laboratories' Cryptobytes, 1(3) (1995), 1–4.
14. X. Wang, D. Feng, X. Lai, H. Yu, *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*, eprint archive 2004/199, `http://eprint.iacr.org/2004/199`, presented at the Crypto 2004 rump session, August 17, 2004.
15. X. Wang and H. Yu , *How to Break MD5 and Other Hash Functions*, EuroCrypt 2005, Springer LNCS 3494 (2005), 19-35.
16. 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.
17. X. Wang, H. Chen, X. Yu, *How to Find Another Kind of Collision for MD4 Efficiently*, Preprint, 2004.
18. X. Wang, D. Feng, X. Yu, *An Attack on Hash Function HAVAL-128* (Chinese), Science in China Ser. F (Information Sciences) 35(4) (2005), 405-416.
19. M. Wiener, personal communication, November 17, 2004.