4 Chosen-prefix collision abuse scenarios

Contents

4.1 Survey

When exploiting collisions in real world applications two major obstacles must be overcome.

- The problem of constructing *meaningful collisions*. Given current methods, collisions require appendages consisting of unpredictable and mostly uncontrollable bit strings. These must be hidden in the usually heavily formatted application data structure without raising suspicion.
- The problem of constructing *realistic attack scenarios*. As we do not have effective attacks against MD5's (second) pre-image resistance but only collision attacks, we cannot target existing MD5 hash values. In particular, the colliding data structures must be generated simultaneously, along with their shared hash, by the adversary.

In this section several chosen-prefix collision applications are surveyed where these problems are addressed with varying degrees of success. Sections 4.2, 4.3, and 4.4 describe the three most prominent applications in more detail. These applications are the result of work done jointly with Arjen Lenstra and Benne de Weger, Section 4.2 is the result of joint work with Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik and Benne de Weger [SLdW07c, SSA$^+$09b, SLdW12]. The theory, algorithms, implementation and practical execution of the underlying collision attacks of these applications as described in Chapter 6 and Chapter 7 are the work of the author of this thesis.

Digital certificates. Given how heavily they rely on cryptographic hash functions, digital certificates are the first place to look for applications of chosen-prefix collisions. Two X.509 certificates are said to collide if their to-be-signed parts have the same hash and consequently their digital signatures, as provided by the CA (Certification Authority), are identical. In [LdW05] it was shown how identical-prefix collisions can be used to construct colliding X.509 certificates with different RSA moduli but identical Distinguished Names. Here the RSA moduli absorbed the random-looking near-collision blocks, thus inconspicuously and elegantly solving the meaningfulness problem. Allowing different Distinguished

Names required chosen-prefix collisions, as we have shown in [SLdW07c] in collaboration with Arjen Lenstra and Benne de Weger. The certificates resulting from both constructions do not contain spurious bits, so superficial inspection at bit level of either of the certificates does not reveal the existence of a sibling certificate that collides with it signature-wise. Nevertheless, for these constructions to work the entire to-be-signed parts, and thus the signing CA, must be fully under the attacker's control, thereby limiting the practical attack potential.

A related but in detail rather different construction was carried out in collaboration with Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik and Benne de Weger, as reported in [SSA+09a, SSA+09b] and in Section 4.2. Although in practice a certificate's to-be-signed part cannot be for 100% under control of the party that submits the certification request, for some commercial CAs (that still used MD5 for their digital signature generation) the entire to-be-signed part could be predicted reliably enough to make the following guess-and-check approach practically feasible: prepare the prefix of the to-be-signed part of a legitimate certification request including a guess for the part that will be included by the CA upon certification, prepare a rogue to-be-signed prefix, determine different collision-causing and identical collision-maintaining appendages to complete two colliding to-be-signed parts, and submit the legitimate one for certification. If upon receipt of the legitimate certificate the guess turns out to have been correct, then the rogue certificate can be completed by pasting the CA's signature of the legitimate data onto the rogue data: because the data collide, the signature is equally valid for both. Otherwise, if the guess is incorrect, another attempt is made. Using this approach we managed (upon the fourth attempt) to trick a commercial CA into providing a signature valid for a rogue CA certificate. For the intricate details of the construction we refer to Section 4.2.

A few additional remarks about this construction are in order here. We created not just a rogue certificate, but a rogue CA certificate, containing identifying information and public key material for a rogue CA. The private key of this rogue CA is under our control. Because the commercial CA's signature is valid for the rogue CA certificate, all certificates issued by the rogue CA are trusted by anybody trusting the commercial CA. As the commercial CA's root certificate is present in all major browsers, this gives us in principle the possibility to impersonate any certificate owner. This is certainly a realistic attack scenario. The price that we have to pay is that the meaningfulness problem is only adequately – and most certainly not elegantly – solved: as further explained in the next paragraph, one of the certificates contains a considerable number of suspicious-looking bits.

To indicate that a certificate is a CA certificate, a certain bit has to be set in the certificate's to-be-signed-part. According to the X.509v3 standard [CSF+08], this bit comes after the public key field. Because it is unlikely that a commercial CA accepts a certification request where the CA bit is set, the bit must not be set in the legitimate request. For our rogue CA certificate construction, the fact that the two to-be-signed parts must contain a different bit *after* the public key field causes an incompatibility with our 'usual' colliding certificate construction as in [SLdW07c]. In that construction the collision-causing appendages correspond to the high order bits of RSA moduli, and they are followed by identical collision-maintaining appendages that transform the two appendages into valid RSA moduli. Anything following after the moduli must remain identical lest the collision property goes lost. As a consequence, the appendages on the rogue side can no longer be hidden in the public key field and some other field must be found for them. Such a field may be specially defined for this purpose, or an existing (proprietary) extension may be used. The Netscape Comment extension is a good example of the latter, as we found that it is ignored by the major certificate processing software. The upshot is, however, that as the appendages have non-negligible length, it will be hard to define a field that will not look suspicious to someone who looks at the rogue certificate at bit level.

Colliding documents. In [DL05] (see also [GIS05]) it was shown how to construct a pair of PostScript files that collide under MD5, but that display different messages when viewed or printed. These constructions use identical-prefix collisions and thus the only difference between the colliding files is in the generated collision bit strings. It follows that they have to rely on the presence of both messages in each of the colliding files and on macro-functionalities of the document format used to show either one of the two messages. Obviously, this raises suspicion upon inspection at bit level. With chosen-prefix collisions, one message per colliding document suffices and macro-functionalities are no longer required. For example, using a document format that allows insertion of color images (such as Microsoft Word or Adobe PDF), inserting one message per document, two documents can be made to collide by appending carefully crafted color images after the messages. A short one pixel wide line will do – for instance hidden inside a layout element, a company logo, or a nicely colored barcode – and preferably scaled down to hardly visible size (or completely hidden from view, as possible in PDF). An extension of this construction is presented in the paragraphs below and set forth in detail in Section 4.3.

Hash based commitments. Kelsey and Kohno [KK06] presented a method to first commit to a hash value, and next to construct faster than by a trivial pre-image attack a document with the committed hash value, and with any message of one's choice as a prefix. The method applies to any Merkle-Damgård hash function, such as MD5, that given an IHV and a suffix produces some IHV. Omitting details involving message lengths and padding, the idea is to commit to a hash value based on an IHV at the root of a tree, either that IHV itself or calculated as the hash of that IHV and some suffix at the root. The tree is a complete binary tree and is calculated from its leaves up to the root, so the IHV at the root will be one of the last values calculated. This is done in such a way that each node of the tree is associated with an IHV along with a suffix that together hash to the *IHV* associated with the node's parent. Thus, two siblings have *IHV* values and suffixes that collide under the hash function. The *IHV* values at the leaves may be arbitrarily chosen but are, preferably, all different. Given a prefix of one's choice one performs a brute-force search for a suffix that, when appended to the prefix and along with the standard *IHV*, results in the *IHV* at one of the leaves (or nodes) of the tree. Appending the suffixes one encounters on one's way from that leave or node to the root, results in a final message with the desired prefix and committed hash value.

Originally based on a birthday search, the construction of the tree can be done more efficiently by using chosen-prefix collisions to construct sibling node suffixes based on their *IHV* values. For MD5, however, it remains far from feasible to carry out the entire construction in practice. In a variant that *is* feasible, one commits to a prediction by publishing its hash value. In due time one reveals the correct prediction, chosen from among a large enough preconstructed collection of documents that, due to tree-structured chosen-prefix collision appendages, all share the same published hash value. In section 4.3 we present an example involving 12 documents.

Software integrity checking. In [Kam04] and [Mik04] it was shown how any existing MD5 collision, such as the ones originally presented by Xiaoyun Wang at the Crypto 2004 rump session, can be abused to mislead integrity checking software that uses MD5. A similar application, using freshly made collisions, was given on [Sel06]. As shown on [Ste09] this can even be done within the framework of Microsoft's Authenticode code signing program. All these results use identical-prefix collisions and, similar to the colliding PostScript application mentioned earlier, differences in the colliding inputs are used to construct deviating execution flows.

Chosen-prefix collisions allow a more elegant approach, since common operating systems ignore bit strings that are appended to executables: the programs will run unaltered. Thus, using tree-structured chosen-prefix collision appendages as above, any number of executables can be made to have the same MD5 hash value or MD5-based digital signature. See Section 4.4 for an example.

One can imagine two executables: a 'good' one (say Word.exe) and a 'bad' one (the attacker's Worse.exe). A chosen-prefix collision for those executables is computed, and the collision-causing bit strings are appended to both executables. The resulting altered file Word.exe, functionally equivalent to the original Word.exe, can be offered to a code signing program such as Microsoft's Authenticode and receive an 'official' MD5-based digital signature. This signature will then be equally valid for the attacker's Worse.exe, and the attacker might be able to replace Word.exe by his Worse.exe (renamed to Word.exe) on the appropriate download site. This construction affects a common functionality of MD5 hashing and may pose a practical threat. It also allows people to get many executables signed at once at the cost of getting a single such executable signed, bypassing verification of any kind (e.g., authenticity, quality, compatibility, non-spyware, non-malware) by the signing party of the remaining executables.

Computer forensics. In computer forensics so-called *hash sets* are used to quickly identify known files. For example, when a hard disk is seized by law enforcement officers, they may compute the hashes of all files on the disk, and compare those hashes to hashes in existing hash sets: a whitelist (for known harmless files such as operating system and other common software files) and a blacklist (for previously identified harmful files). Only files whose hashes do not occur in either hash set have to be inspected further. A useful feature of this method of recognizing files is that the file name itself is irrelevant, since only the content of the file is hashed.

MD5 is a popular hash function for this application. Examples are NIST's National Software Reference Library Reference Data Set[13] and the US Department of Justice's Hashkeeper application[14].

A conceivable, and rather obvious, attack on this application of hashes is to produce a harmless file (e.g., an innocent picture) and a harmful one (e.g., an illegal picture), and insert collision blocks that will not be noticed by common application software or human viewers. In a learning phase the harmless file might be submitted to the hash set and thus the common hash may end up on the whitelist. The harmful file will be overlooked from then on.

Peer to peer software. Hash sets are also used in peer to peer software. A site offering content may maintain a list of pairs (file name, hash). The file name is local only, and the peer to peer software uniquely identifies the file's content by means of its hash. Depending on how the hash is computed such systems may be vulnerable to a chosen-prefix attack. Software such as eDonkey and eMule use MD4 to hash the content in a two stage manner: the identifier of the content $c_1\|c_2\|\ldots\|c_n$ is $\mathrm{MD4}(\mathrm{MD4}(c_1)\|\ldots\|\mathrm{MD4}(c_n))$, where the chunks c_i are about 9 MB each. One-chunk files, i.e., files not larger than 9 MB, are most likely vulnerable; whether multi-chunk files are vulnerable is open for research. We have not worked out the details of a chosen-prefix collision attack against MD4, but this seems very well doable by adapting our methods and should result in an attack that is considerably faster than our present one against MD5.

Content addressed storage. In recent years *content addressed storage* is gaining popularity as a means of storing fixed content at a physical location of which the address is directly derived from the content itself. For example, a hash of the content may be used as the file name. See [PD05] for an example. Clearly, chosen-prefix collisions can be used by an attacker to fool such storage systems, e.g., by first preparing colliding pairs of files, by then storing the harmless-looking first one, and later overwriting it with the harmful second one.

13. `http://www.nsrl.nist.gov/`
14. `http://www.usdoj.gov/ndic/domex/hashkeeper.htm`

Further investigations are required to assess the impact of chosen-prefix collisions. We leave it to others to study to what extent commonly used protocols and message formats such as TLS, S/MIME (CMS), IPSec and XML Signatures (see [BR06b] and [HS05]) allow insertion of random looking data that may be overlooked by some or all implementations. The threat posed by identical-prefix collisions is not well understood either: their application may be more limited, but for MD5 they can be generated almost instantaneously and thus allow real-time attacks on the execution of cryptographic protocols, and, more importantly, for SHA-1 they may soon be feasible. We present a possible countermeasure against identical-prefix and chosen-prefix collision attacks for MD5 and SHA-1 in Chapter 8.

4.2 Creating a rogue Certification Authority certificate

In our conference paper [SLdW07c, Section 4.1] we daydreamed:

> "Ideally, a realistic attack targets the core of PKI: provide a relying party with trust, beyond reasonable cryptographic doubt, that the person indicated by the Distinguished Name field has exclusive control over the private key corresponding to the public key in the certificate. The attack should also enable the attacker to cover his trails."

Our dream scenario has been, mainly, realized with the construction of a rogue CA certificate. With the private key of a CA under our control, and the public key appearing in a certificate with a valid signature of a commercial CA that is trusted by all major browsers, we can create 'trusted' certificates at will. When scrutinized at bit level, however, our rogue CA certificate may look suspicious which may, ultimately, expose us. Bit level inspection is not something many users will engage in – if they know the difference between `https` and `http` to begin with – and, obviously, the software that is supposed to inspect a certificate's bits is expertly guided around the suspicious ones. So, it may be argued that our construction has a non-negligible attack potential. Below we discuss some possibilities in this direction. Upfront, however, we like to point out that our rogue CA is nothing more than a proof of concept that is incapable of doing much harm, because it expired, on purpose, in September of 2004, i.e., more than four years before it was created.

Any website secured using TLS can be impersonated using a rogue certificate issued by a rogue CA. This is irrespective of which CA issued the website's true certificate and of any property of that certificate (such as the hash function it is based upon – SHA-256 is not any better in this context than MD4). Combined with redirection attacks where `http` requests are redirected to rogue web servers, this leads to virtually undetectable phishing attacks.

But any application involving a Certification Authority that provides MD5-based certificates with sufficiently predictable serial number and validity period may be vulnerable. In contexts different from TLS this may include signing or encryption of e-mail or software, non-repudiation services, etc.

As pointed out earlier, bit-level inspection of our rogue CA certificate will reveal a relatively large number of bits that may look suspicious – and that *are* suspicious.

This could have been avoided if we had chosen to create a rogue certificate for a regular website, as opposed to a rogue CA certificate, because in that case we could have hidden all collision causing bits inside the public keys. Nevertheless, even if each resulting certificate by itself looks unsuspicious, as soon as a dispute arises, the rogue certificate's legitimate sibling can be located with the help of the CA, and the fraud becomes apparent by putting the certificates alongside, thus exposing the party responsible for the fraud.

Our attack relies on our ability to predict the content of the certificate fields inserted by the CA upon certification: if our prediction is correct with non-negligible probability, a rogue certificate can be generated with the same non-negligible probability. Irrespective of the weaknesses, known or unknown, of the cryptographic hash function used for digital signature generation, our type of attack becomes effectively impossible if the CA adds a sufficient amount of fresh randomness to the certificate fields before the public key fields. Relying parties, however, cannot verify this randomness. Also, the trustworthiness of certificates should not crucially depend on such secondary and circumstantial aspects. We would be in favor of a more fundamental solution – along with a strong cryptographic hash function – possibly along the lines as proposed in [HK06]. Generally speaking, it is advisable not to sign data that is completely determined by some other party. Put differently, a signer should always make a few trivial and unpredictable modifications before digitally signing a document provided by someone else.

The issue in the previous paragraph was recognized and the possibility of the attack presented in this paper anticipated in the catalogue [Bun08] of algorithms suitable for the German Signature Law ('Signaturgesetz'). This catalogue includes conditions and time frames for cryptographic hash algorithms to be used in legally binding digital signatures in Germany. One of the changes introduced in the 2008 version of the catalog is an explicit condition on the usage of SHA-1: only until 2010, and only for so-called "qualified certificates" that contain at least 20 bits of entropy in their serial numbers. We are grateful to Prof. Werner Schindler of the BSI for bringing this to our attention and for confirming that this change was introduced to thwart exactly the type of rogue certificates that we present here for MD5.

We stress that our attack on MD5 is not a pre-image or second pre-image attack. We cannot create a rogue certificate having a signature in common with a certificate that was not especially crafted using our chosen-prefix collision. In particular, we cannot target any existing, independently created certificate and forge a rogue certificate that shares its digital signature with the digital signature of the targeted certificate. Given any certificate with an MD5-based digital signature, so far a relying party cannot easily recognize if it is trustworthy or, on the contrary, crafted by our method. However, in Chapter 8 we present a method to distinguish near-collision attacks given only either certificate. This method could both be used to prevent legitimate-looking but malicious certificates to be signed by CAs and to block malicious certificates in the end-users applications. Nevertheless, we repeat our urgent recommendation not to use MD5 for new X.509 certificates. How existing MD5 certificates should be handled is a subject of further research. We also urgently recommend reconsidering usage of

MD5 in other applications. Proper alternatives are available; but compatibility with existing applications is obviously another matter.

The first colliding X.509 certificate construction was based on an identical-prefix collision, and resulted in two certificates with different public keys, but identical Distinguished Name fields [LdW05]. As a first application of chosen-prefix collisions we showed how the Distinguished Name fields could be chosen differently as well [SLdW07c]. In this section we describe the details of a colliding certificate construction that goes one step further by also allowing different "basic constraints" fields. This allows us to construct one of the certificates as an ordinary website certificate, but the other one as a CA certificate, the contents of both certificates can be found in Appendix E. As already pointed out in Section 4.1, this additional difference required a radical departure from the traditional construction methods from [LdW05] and [SLdW07c]. Also, unlike our previous colliding certificate constructions where the CA was under our control, a commercial CA provided the digital signature for the (legitimate) website certificate. This required us to sufficiently accurately predict its serial number and validity period well before the certification request was submitted to the signing CA.

We exploited the following weaknesses of the commercial CA that carried out the legitimate certification request:

- Its usage of the cryptographic hash function MD5 to generate digital signatures for new certificates.

- Its fully automated way to process online certification requests that fails to recognize anomalous behavior of requesting parties.

- Its usage of sequential serial numbers and its usage of validity periods that are determined entirely by the date and time in seconds at which the certification request is processed.

- Its failure to enforce, by means of the "basic constraints" field in its own certificate, a limit on the length of the chain of certificates that it can sign.

The first three points are further discussed below. The last point, if properly handled, could have crippled our rogue CA certificate but does not affect its construction. A certificate contains a "basic constraints" field where a bit is set to indicate if the certificate is a CA certificate. With the bit set, a "path length constraint" subfield may be present, specifying an integer that indicates how many CAs may occur in the chain between the CA certificate in question and end-user certificates. The commercial CA that we interacted with failed to use this option in its own certificate, implying that any number of intermediate CAs is permitted. If the "path length constraint" would have been present and set at 0 (zero), then our rogue CA certificate could still have been constructed. But whether or not the rogue CA certificate or certificates signed by it can then also be used depends on (browser-)software actually checking the "path length constraint" subfields in chains of certificates. Thus a secondary "defense in depth" mechanism was present that could have foiled our attack, but failed to do so simply because it was not used.

Before describing the construction of the colliding certificates, we briefly discuss the parameter choices used for the chosen-prefix collision search. First, the number of near-collision blocks is denoted by r and can be used to trade-off between birthday search time complexity and the cost of finding the r near-collision blocks. Second, k defines the birthday search space (its size is $64 + k$) and the birthday iteration function and can be used to trade-off between birthday search time complexity, birthday search memory complexity and average number of required near-collisions per birthday collision. Third, w defines the family of differential paths that can be used to construct the near-collision blocks and is the number of bit positions where arbitrary bit differences are allowed. It can be used to trade-off between the average number of required near-collision blocks per birthday collision and the cost of finding the r near-collision blocks. For more details on r, k and w we refer to Sections 6.5.2 and 6.5.3.

The 2048-bit upper bound on the length of RSA moduli, as enforced by some CAs, combined with other limitations of our certificate construction, implied we could allow for at most three near-collision blocks. Opting for the least difficult possibility (namely, three near-collision blocks), we had to decide on values for k and the aimed for value for w that determine the costs of the birthday search and the near-collision block constructions, respectively. Obviously, our choices were influenced by our computational resources, namely a cluster of 215 PlayStation 3 (PS3) game consoles. When running Linux on a PS3, applications have access to 6 Synergistic Processing Units (SPUs), a general purpose CPU, and about 150MB of RAM per PS3. For the birthday search, the 6×215 SPUs are computationally equivalent to approximately 8600 regular 32-bit cores, due to each SPU's 4×32-bit wide SIMD architecture. The other parts of the chosen-prefix collision construction are not suitable for the SPUs, but we were able to use the 215 PS3 CPUs for the construction of the actual near-collision blocks. With these resources, the choice $w = 5$ still turned out to be acceptable despite the 1000-fold increase in the cost of the actual near-collision block construction. This is the case even for the hard cases with many differences between IHV and IHV': as a consequence the differential paths contain many bitconditions, which leaves little space for the tunnels, thereby complicating the near-collision block construction.

For the targeted three near-collision blocks, the entries for $w = 5$ in the first table in Appendix D show the time-memory trade-off when the birthday search space is varied with k. With 150MB at our disposal per PS3, for a total of about 30GB, we decided to use $k = 8$ as this optimizes the overall birthday search complexity for the plausible case that the birthday search takes $\sqrt{2}$ times longer than expected. The resulting overall chosen-prefix collision construction takes on average less than a day on the PS3-cluster. In theory we could have used 1TB (or more) of hard drive space, in which case it would have been optimal to use $k = 0$ for a birthday search of about 20 PS3 days which is about 2.3 hours on the PS3-cluster.

We summarize the construction of the colliding certificates in the sequence of steps below, and then describe each step in more detail.

1. Construction of templates for the two to-be-signed parts, as outlined in Figure 7. Note that we distinguish between a 'legitimate' to-be-signed part on the left hand side, and a 'rogue' to-be-signed part on the other side.
2. Prediction of serial number and validity period for the legitimate part, thereby completing the chosen prefixes of both to-be-signed parts.
3. Computation of the two different collision-causing appendages.
4. Computation of a single collision-maintaining appendage that will be appended to both sides, thereby completing both to-be-signed parts.
5. Preparation of the certification request for the legitimate to-be-signed part.
6. Submission of the certification request and receipt of the new certificate.
7. If serial number and validity period of the newly received certificate are as predicted, then the rogue certificate can be completed. Otherwise return to Step 2.

The resulting rogue CA certificate and the end-user certificate, together with the differential paths used for the three near-collision blocks, can be found in Appendix E.

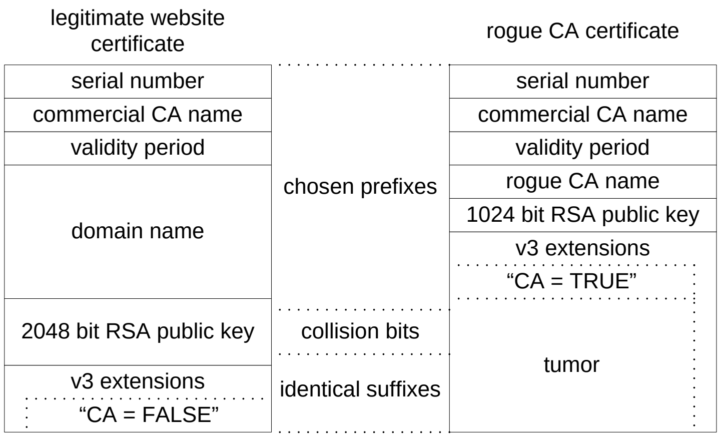

Figure 7: *The to-be-signed parts of the colliding certificates.*

Step 1. Templates for the to-be-signed parts. In this step all bits are set in the two to-be-signed parts, except for bits that are determined in later steps. For the latter bits space is reserved here. On the legitimate side the parts to be filled in later are the predictions for the serial number and validity period, and most bits of the public key. On the rogue side the largest part of the content of an extension field of the type “Netscape Comment” is for the moment left undetermined. The following roughly describes the sequence of steps.

- On the legitimate side, the chosen prefix contains space for serial number and validity period, along with the exact Distinguished Name of the commercial CA where the certification request will be submitted. This is followed by a subject Distinguished Name that contains a legitimate website domain name (owned by one of us) consisting of as many characters as allowed by the commercial CA (in our case 64), and concluded by the first 208 bits of an RSA modulus, the latter all chosen at random after the leading ‘1’-bit. These sizes were chosen in order to have as many corresponding bits as possible on the rogue side, while fixing as few bits as possible of the RSA modulus on the legitimate side (see Step 4 for the reason why).

- The corresponding bits on the rogue side contain an arbitrarily chosen serial number, the same commercial CA’s Distinguished Name, an arbitrarily chosen validity period (actually chosen as indicating “August 2004”, to avoid abuse of the rogue certificate), a short rogue CA name, a 1024-bit RSA public key generated using standard software, and the beginning of the X.509v3 extension fields. One of these fields is the “basic constraints” field, a bit that we set to indicate that the rogue certificate will be a CA certificate (in Figure 7 this bit is denoted by “CA=TRUE”).

- At this point the entire chosen prefix is known on the rogue side, but on the legitimate side predictions for the serial number and validity period still need to be inserted. That is done in Step 2.

- The various field sizes were selected so that on both sides the chosen prefixes are now 96 bits short of the same MD5 block boundary. On both sides these 96 bit positions are reserved for the birthday bits. Because only $64 + k = 72$ birthday bits per side are needed (and appended in Step 3) the first 24 bits at this point are set to 0. On the legitimate side these 96 bits are part of the RSA modulus, on the rogue side they are part of an extension field of the type “Netscape Comment”, denoted as ‘tumor’ in Figure 7.

- From here on forward, everything that goes to the rogue side is part of the “Netscape Comment” field, as it is not meaningful for the rogue CA certificate but only appended to cause and maintain a collision with bits added to the legitimate side. On the legitimate side we first make space for 3 near-collision blocks of 512 bits each (calculated in Step 3) and for 208 bits used to complete a 2048-bit RSA modulus (determined in Step 4), and then set the RSA public exponent (for which we took the common choice 65537) and the X.509v3 extensions including the bit indicating that the legitimate certificate will be an end-user certificate (in Figure 7 denoted by “CA=FALSE”).

Step 2. Prediction of serial number and validity period. Based on repeated certification requests submitted to the targeted commercial CA, it turned out that the validity period can very reliably be predicted as the period of precisely one year plus one day, starting exactly six seconds after a request is submitted. So, to control that field, all we need to do is select a validity period of the right length, and submit the legitimate certification request precisely six seconds before it starts. Though occasional accidents may happen in the form of one-second shifts, this was the easy part.

Predicting the serial number is harder but not impossible. In the first place, it was found that the targeted commercial CA uses sequential serial numbers. Being able to predict the next serial number, however, is not enough: the construction of the collision can be expected to take at least a day, before which the serial number and validity period have to be fixed, and only after which the to-be-signed part of the certificate will be entirely known. As a consequence, there will have been a substantial and uncertain increment in the serial number by the time the collision construction is finished. So, another essential ingredient of our construction was the fact that the CA's weekend workload is quite stable: it was observed during several weekends that the increment in serial number over a weekend does not vary a lot. This allowed us to pretty reliably predict Monday morning's serial numbers on the Friday afternoon before. Thus, on Friday afternoon we selected a number at the high end of the predicted range for the next Monday morning, and inserted it in the legitimate to-be-signed part along with a validity period starting that same Monday morning at the time corresponding to our serial number prediction. See Step 6 how we then managed, after the weekend, to target precisely the selected serial number and validity period.

Step 3. Computation of the collision. At this point both chosen prefixes have been fully determined so the chosen-prefix collision can be computed: first the 72 birthday bits per side, calculated in parallel on the 1290 SPUs of a cluster of 215 PS3s, followed by the calculation of 3 pairs of 512-bit near-collision blocks on a quad-core PC and the 215 PS3 CPUs. The entire calculation takes on average about a day.

Given that we had a weekend available, and that the calculation can be expected to take just a day, we sequentially processed a number of chosen-prefixes, each corresponding to different serial numbers and validity periods (targeting both Monday and Tuesday mornings). So, a near-collision block calculation on the CPUs would always run simultaneously with a birthday search on the SPUs for the 'next' attempt.

Step 4. Finishing the to-be-signed parts. At this point the legitimate and rogue sides collide under MD5, so that from here on only identical bits may be appended to both sides.

With $208 + 24 + 72 + 3 * 512 = 1840$ bits set, the remaining $2048 - 1840 = 208$ bits need to be set for the 2048-bit RSA modulus on the legitimate side. Because in the next step the RSA private exponent corresponding to the RSA public exponent is needed, the full factorization of the RSA modulus needs to be known, and the factors must be compatible with the choice of the RSA public exponent. Because common

CAs (including our targeted commercial CA) do not check for compositeness of RSA moduli in certification requests, we could simply have added 208 bits to make the RSA modulus a prime. We found that approach unsatisfactory, and opted for the rather crude but trivial to program method sketched below that results in a 224-bit prime factor with a prime 1824-bit cofactor. Given that at the time this work was done the largest factor found using the elliptic curve integer factorization method was 222 bits long, a 224-bit smallest prime factor keeps the resulting modulus out of reach of common factoring efforts. We could have used a relatively advanced lattice-based method to try and squeeze in a 312-bit prime factor along with a prime 1736-bit cofactor. Given only 208 bits of freedom to select a 2048-bit RSA modulus, it is unlikely that a more balanced solution can efficiently be found. Thus the reason why as few bits as possible should be fixed in Step 1, is that it allows us to construct a slightly less unbalanced RSA modulus.

Let N be the 2048-bit integer consisting of the 1840 already determined bits of the RSA modulus-to-be, followed by 208 one bits. We select a 224-bit integer p at random until $N = a \cdot p + b$ with $a \in \mathbb{N}$ and $b < 2^{208}$, and keep doing this until both p and $q = \lfloor N/p \rfloor$ are prime and the RSA public exponent is coprime to $(p-1)(q-1)$. Once such primes p and q have been found, the number pq is the legitimate side's RSA modulus, the leading 1840 bits of which are already present in the legitimate side's to-be-signed part, and the 208 least significant bits of which are inserted in both to-be-signed parts.

To analyze the required effort somewhat more in general, 2^{k-208} integers of k bits (with $k > 208$) need to be selected on average for pq to have the desired 1840 leading bits. Since an ℓ-bit integer is prime with probability approximately $1/\log(2^{\ell})$, a total of $k(2048-k)2^{k-208}(\log 2)^2$ attempts may be expected before a suitable RSA modulus is found. The coprimality requirement is a lower order effect that we disregard. Note that for $k(k-2048)(\log 2)^2$ of the attempts the k-bit number p has to be tested for primality, and that for $(2048-k)\log 2$ of those q needs to be tested as well (on average, obviously). For $k = 224$ this turned out to be doable in a few minutes on a standard PC.

This completes the to-be-signed parts on both sides. Now it remains to be hoped that the legitimate part that actually will be signed corresponds, bit for bit, with the legitimate to-be-signed part that we concocted.

Step 5. Preparing the certification request. Using the relevant information from the legitimate side's template, i.e., the subject Distinguished Name and the public key, a PKCS#10 Certificate Signing Request is prepared. The CA requires proof of possession of the private key corresponding to the public key in the request. This is done by signing the request using the private key – this is the sole reason that we need the RSA private exponent.

Step 6. Submission of the certification request. The targeted legitimate to-be-signed part contains a very specific validity period that leaves no choice for the moment at which the certification request needs to be submitted to the CA. Just hoping that at that time the serial number would have precisely the predicted value is unlikely to work, so a somewhat more elaborate approach is used. About half an hour before the targeted submission moment, the same request is submitted, and the serial number in the resulting certificate is inspected. If it is already too high, the entire attempt is abandoned. Otherwise, the request is repeatedly submitted, with a frequency depending on the gap that may still exist between the serial number received and the targeted one, and taking into account possible certification requests by others. In this way the serial number is slowly nudged toward the right value at the right time. Although there is nothing illegal about repeated certification requests, it should be possible for a CA to recognize the somewhat anomalous behavior sketched above and to take appropriate countermeasures (such as random delays or jumps in serial numbers) if it occurs.

Various types of accidents may happen, of course, and we experienced some of them, such as another CA customer ‘stealing’ our targeted serial number just a few moments before our attempt to get it, thereby wasting that weekend’s calculations. But, after the fourth weekend it worked as planned, and we managed to get an actually signed part that exactly matched our predicted legitimate to-be-signed part.

Step 7. Creation of the rogue certificate. Given the perfect match between the actually signed part and the hoped for one, and the MD5 collision between the latter and the rogue side’s to-be-signed part, the MD5-based digital signature present in the legitimate certificate as provided by the commercial CA is equally valid for the rogue side. To finish the rogue CA certificate it suffices to copy the digital signature to the right spot in the rogue CA certificate.

4.3 Nostradamus attack

In the original Nostradamus attack from [KK06] one first commits to a certain hash value, and afterwards for any message constructs a document that not only contains that message but that also has the committed hash value. In its full generality, this attack is at this point in time not feasible for MD5. It is easily doable, though, if a limited size message space has been defined upfront.

Suppose there are messages $m_1, m_2, \ldots, m_r$, then using $r-1$ chosen-prefix collisions we can construct r suffixes $s_1, s_2, \ldots, s_r$ such that the r documents $d_i = m_i \| s_i$ all have the same hash. After committing to the common hash, afterwards any of the r documents $d_1, d_2, \ldots, d_r$ can be shown, possibly to achieve some malicious goal. The other documents will remain hidden and their contents, i.e., the m_i-parts, cannot be derived – with overwhelming probability – from the single published document or from the common hash value.

To show the practicality of this variant, we have made an example consisting of 12 different PDF documents with a common MD5 hash value, where each document predicts a different outcome of the 2008 US presidential elections. The PDF format is convenient for this purpose because it allows insertion of extra image objects that are unreferenced in the resulting document and thus invisible to the viewer in any common PDF reader. The common MD5 hash value of our 12 colliding PDF documents containing our predictions is

$$\texttt{3d515dead7aa16560aba3e9df05cbc80}_{16}.$$

See [SLdW07a] for the actual PDF documents, one of which correctly predicted the outcome one year before the elections took place.

For each of the 11 collisions required for this example we used a 64-bit birthday search (on a single PS3) aiming for about 11 near-collision blocks (constructed on a quad-core PC). It took less than two days per chosen-prefix collision. Since we performed those computations our methods have improved as described in this thesis, so this attack would now run much faster.

Given the structure of PDF documents it is not entirely straightforward to insert different chosen-prefix collision blocks, while keeping the parts following those blocks identical in order to maintain the collision. The relevant details of both the PDF structure and our construction are covered here.

A PDF document is built up from the following four consecutive parts: a fixed header, a part consisting of an arbitrary number of numbered objects, an object lookup table and, finally, a trailer. The trailer specifies the number of objects, which of the objects is the unique root object (containing the document content) and which is the info object (containing the document's meta information such as authors and title etc.), and contains a filepointer to the start of the object lookup table.

Given a file containing a PDF document, additional objects can be inserted, as long as they are added to the object lookup table and the corresponding changes are made to the number of objects and the filepointer in the trailer. A template for an image object is given in Table 4-1. With the exception of binary images, the format is mostly text based. Any binary image is put between single line-feed characters (ASCII code 10) and the result is encapsulated by the keywords `stream` and `endstream`. The keyword `/Length` must specify the byte length of the image. Because in our case, the image is uncompressed and each pixel requires three bytes ('`RGB`'), the image byte length must be three times the product of the specified width and height. The object number (`42` in the example object header) must be set to the next available object number.

Table 4-1: *An example numbered image object in the PDF format.*

Part	Contents
object header	`42 0 obj`
image header	`<< /ColorSpace /DeviceRGB /Subtype /Image`
image size	`/Length 9216 /Width 64 /Height 48 /BitsPerComponent 8`
image contents	`>> stream...endstream`
object footer	`endobj`

When constructing colliding PDF files they must be equal after the collision-causing data. The object lookup tables and trailers for all files must therefore be the same. This was achieved as follows:

- Because all documents must have the same number of objects, dummy objects are inserted where necessary.

- Because all root objects must have the same object number, they can be copied if necessary to objects with the next available object number.
- The info objects are treated in the same way as the root objects.
- To make sure that all object lookup tables and filepointers are identical, the objects can be sorted by object number and if necessary padded with spaces after their `obj` keyword to make sure that all objects with the same object number have the same file position and byte length in all files.
- Finally, the object lookup tables and trailers need to be adapted to reflect the new situation – as a result they should be identical for all files.

Although this procedure works for basic PDF files (such as PDF version 1.4 as we produced using pdflatex), it should be noted that the PDF document format allows additional features that may cause obstructions.

Given r LaTeX files with the desired subtle differences (such as names of r different candidates), r different PDF files are produced using a version of LaTeX that is suitable for our purposes (cf. above). In all these files a binary image object with a fixed object number is then inserted, and the approach sketched above is followed to make the lookup tables and trailers for all files identical. Since this binary image object is present but not used in the PDF document, it remains hidden from view in a PDF reader. To ensure that the files are identical after the hidden image contents, their corresponding objects were made the last objects in the files. This then leads to r chosen prefixes consisting of the leading parts of the PDF files up to and including the keyword `stream` and the first line-feed character. After determining $r-1$ chosen-prefix collisions resulting in r collision-causing appendages, the appendages are put in the proper binary image parts, after which all files are completed with a line-feed character, the keywords `endstream` and `endobj`, and the identical lookup tables and trailers.

Note that the `Length` etc. fields have to be set before collision finding, and that the value of `Length` will grow logarithmically with r and linearly in the number of near-collision blocks one is aiming for.

4.4 Colliding executables

Using the same set-up as used for the Nostradamus attack reported in Section 4.3, i.e., 64-bit birthday search on a PS3 followed by the construction of about 12 near-collision blocks on a quad-core PC, it took us less than 2 days to create two different Windows executables with the same MD5 hash. Initially both 40960 bytes large, 13×64 bytes had to be appended to each executable, for a resulting size of just 41792 bytes each, to let the files collide under MD5 without changing their functionality.

See [SLdW07b] for details. As noted above, it has been shown on [Ste09] that this attack can be elevated to one on a code signing scheme.

As usual, the following remarks apply:

- An existing executable with a known and published hash value not resulting from this construction cannot be targeted by this attack: our attack is not a pre-image or second pre-image attack. In order to attack a software integrity protection or code signing scheme using this approach, the attacker must be able to manipulate the files before they are hashed (and, possibly, signed). Given the level of access required to realize the attack an attacker can probably do more harm in other simpler and more traditional ways.
- Any number r of executables can be made to collide, at the cost of $r-1$ chosen-prefix collisions and an $O(\log r)$-byte appendage to each of the r original executables.

In Chapter 8 we present a method that allows to distinguish near-collision attacks and thus distinguish potentially malicious MD5-based certificates or executables. It is better, however, not to rely on cryptographic primitives such as MD5 and SHA-1 that fail to meet their design criteria.