MD5 considered harmful today

Creating a rogue CA certificate

December 30, 2008

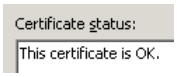
**Alexander Sotirov, Marc Stevens,
Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, Benne de Weger**

Latest news [Dec. 31] Responses from Verisign (RapidSSL) and Microsoft.

Summary We have identified a vulnerability in the Internet Public Key Infrastructure (PKI) used to issue digital certificates for secure websites. As a proof of concept we executed a practical attack scenario and successfully created a rogue Certification Authority (CA) certificate trusted by all common web browsers. This certificate allows us to impersonate any website on the Internet, including banking and e-commerce sites secured using the HTTPS protocol.

Our attack takes advantage of a weakness in the MD5 cryptographic hash function that allows the construction of different messages with the same MD5 hash. This is known as an MD5 "collision". Previous work on MD5 collisions between 2004 and 2007 showed that the use of this hash function in digital signatures can lead to theoretical attack scenarios. Our current work proves that at least one attack scenario can be exploited in practice, thus exposing the security infrastructure of the web to realistic threats.

As a result of this successfull attack, we are currently in possession of a rogue Certification Authority certificate. This certificate will be accepted as valid and trusted by all common browsers, because it appears to be signed by one of the root CAs that browsers trust by default. In turn, any website certificate signed by our rogue CA will be trusted as well. If an unsuspecting user is a victim of a man-in-the-middle attack using such a certificate, they will be assured that the connection is secure through all common security indicators: a "https://" url in the address bar, a closed padlock and messages such as "This certificate is OK" if they chose to inspect the certificate.



This successful proof of concept shows that the certificate validation performed by browsers can be subverted and malicious attackers might be able to monitor or tamper with data sent to secure websites. Banking and e-commerce sites are particularly at risk because of the high value of the information secured with HTTPS on those sites. With a rogue CA certificate, attackers would be able to execute practically undetectable phishing attacks against such sites.

The infrastructure of Certification Authorities is meant to prevent exactly this type of attack. Our work shows that known weaknesses in the MD5 hash function can be exploited in realistic attack, due to the fact that even after years of warnings about the lack of security of MD5, some root CAs are still using this broken hash function.

The vulnerability we expose is not in the SSL protocol or the web servers and browsers that implement it, but in the Public Key Infrastructure. This infrastructure has applications in other areas than the web, but we have not investigated all other possible attack scenarios. So other attack scenarios beyond the web are conceivable, such as in the areas of code signing, e-mail security, and in other areas that use certificates for enabling digital signatures or public key encryption.

The rest of this document will explain our work and its implications in a fair amount of detail. In the interest of protecting the Internet against malicious attacks using our technique, we have omitted the critical details of our sophisticated and highly optimized method for computing MD5 collisions. A scientific paper about our method is in preparation and will be released after a few months, so that the affected Certification Authorities have had some time to remedy this vulnerability.

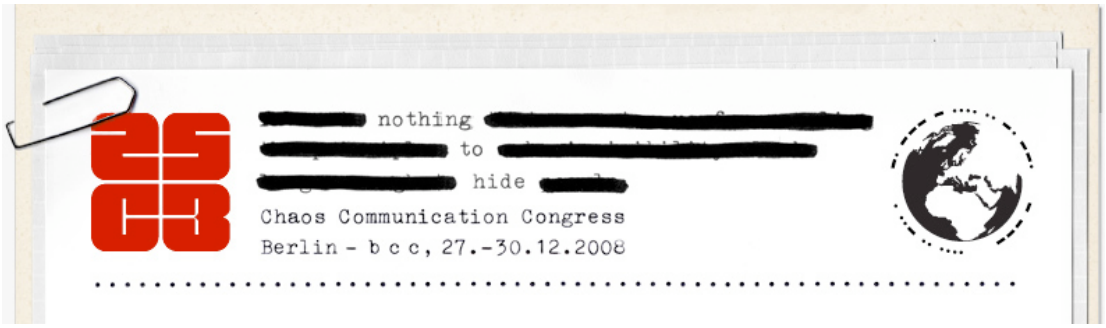
Organization

- Section 1 below gives an outline of our work.
- To fully appreciate our result some basic knowledge is required. Therefore in section 2 we provide some background on certificates, and in section 3 we describe the necessary details on hash functions. Readers familiar with those topics may want to skip (large parts of) these sections.
- Our present work can be seen as a continuation of recent work in the last few years, of which the history is briefly sketched in section 4.
- Section 5 describes our method in detail. Readers not interested in many technical topics may want to skip (large parts of) this section.
- The possible impact of the attack scenarios and possible countermeasures are dealt with in section 6 and

section 7 respectively.
- We conclude with "Frequently asked questions" in section 8.

Presentation

This work was presented on December 30, 2008 at the 25th Annual Chaos Communication Congress in Berlin.



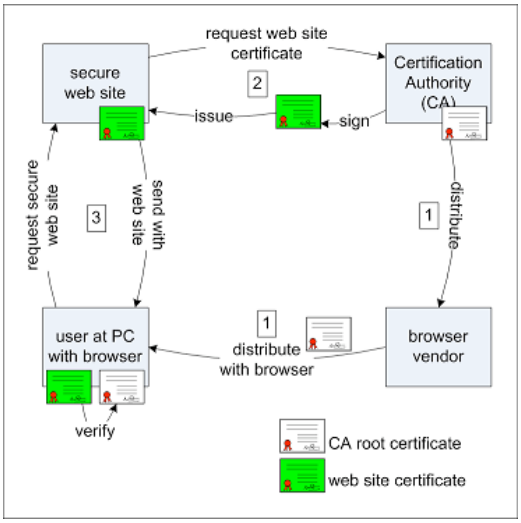
The presentation was originally announced as "Making the theoretical possible. Attacking a critical piece of Internet infrastructure", but finally was entitled "MD5 considered harmful today: creating a rogue CA certificate".

---

1. Outline of the attack

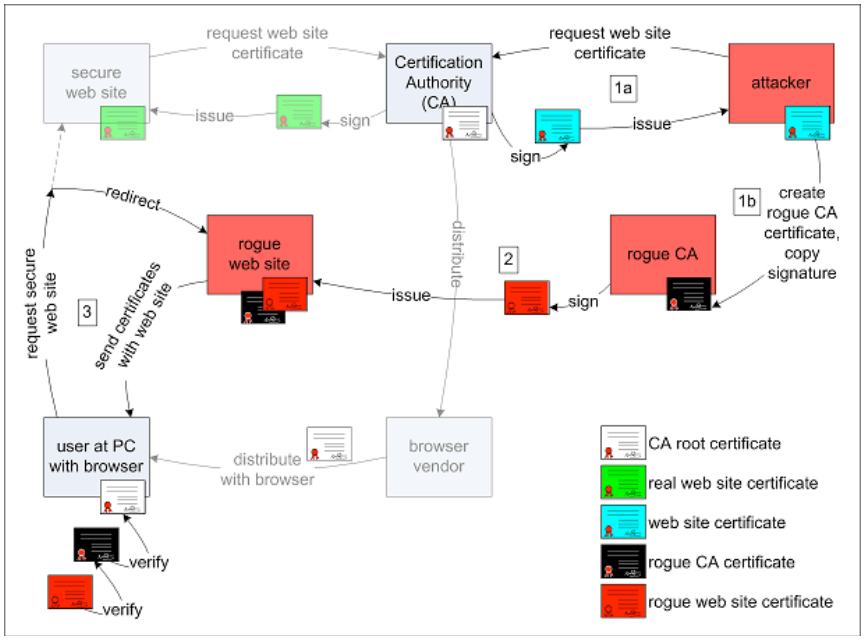
Our attack scenario basically is as follows. We request a legitimate website certificate from a commercial Certification Authority trusted by all common browsers. Since the request is legitimate, the CA signs our certificate and returns it to us. We have picked a CA that uses the MD5 hash function to generate the signature of the certificate, which is important because our certificate request has been crafted to result in an MD5 collision with a second certificate. This second certificate is not a website certificate, but an intermediary CA certificate that can be used to sign arbitrary other website certificates we want to issue. Since the MD5 hashes of both the legitimate and the rogue certificates are the same, the digital signature obtained from the commercial CA can simply be copied into our rogue CA certificate and it will remain valid.

Here is a schematic diagram and a description of how website certificates are meant to operate.



1. A Certification Authority distributes its CA root certificate (the white one in the diagram) via browser vendors to browsers. These root certificates reside in a "trust list" on the user's PC. This means that all certificates issued by this CA will be trusted by default by the users.
2. A company that wants its website to be secured, purchases a website certificate at the CA (the green one in the diagram). This certificate is signed by the CA and guarantees the identity of the website to the users.
3. When a user wants to visit the secure website, the web browser will first ask the web server for the certificate. If its signature can be verified with the certificate of a CA in the trust list, the website certificate will be accepted. Then the website will be loaded into the browser, and all traffic between the browser and the website will be secured by using encryption.

And here is a similar diagram and description of how our attack scenario may be used to impersonate an existing website.



1a A legitimate website certificate is obtained from a commercial CA (the blue one in the diagram).

1b A rogue CA certificate is constructed (the black one in the diagram). It bears exactly the same signature as the website certificate. Thus it appears as being issued by the CA, whereas in fact the CA has never even seen it.

2 Then a website certificate (the red one in the diagram) bearing the genuine website's identity but another public key is created and signed by the rogue CA. A copy of the genuine website is built, put on another web server, and equipped with the rogue website certificate.

3 When a user wants to visit the secure website, the web browser will look on the Internet for the genuine web server. There exist "redirection attacks", by which the communication from the browser can be redirected to the rogue website. This rogue website presents its certificate to the user, together with the rogue CA certificate. The signature in the rogue website certificate can be verified with the rogue CA certificate, and this rogue CA certificate in turn will be accepted by the browser, as its signature can be verified with the CA root certificate in the trust list. The user will not notice anything.

An effective and efficient countermeasure to remediate this vulnerability is to stop using MD5 for digital signatures. Acceptable alternatives to replace MD5, such as SHA-1, are widespread or, like SHA-2, at least becoming rapidly available. Fortunately most CAs have already taken such steps. It would be best to stop using MD5 for the creation of certificates altogether, as its continued use may lead to severe security problems. As a last resort, in the case that MD5 cannot be discontinued immediately, a CA can take simple alternative countermeasures such as randomizing the serial numbers of all newly issued certificates. We stress that such countermeasures are mere band-aids rather than fundamental solutions.

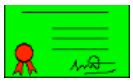
We also recommend to be similarly careful when using SHA-1 and to avoid its adoption in new products, as theoretically it has been shown that it suffers from similar problems as MD5. While the weaknesses in SHA-1 known so far do at this moment not allow the same type of attack scenarios we have shown to be possible for MD5, experts in the area believe that it is sufficiently likely that before long SHA-1 suffers the same fate as MD5.

2. Certificate background

2.1. The need for certificates

More and more the web is used for exchanging sensitive information. Think of internet banking websites, websites that require a login password, websites on which transactions with financial value can be performed, etc. Often the sensitivity of the information implies that it should not be exposed to unauthorized persons. The exchange of sensitive information requires that the user of a website wants assurance that the website he or she is visiting is not a rogue web site, but is the genuine website of the organisation he or she has to do business with. This is what certificates provide.

A certificate is a document that contains both an identity and a public key, binding them together



by a digital signature. This digital signature is created by an organisation called a Certification Authority (CA). This organisation guarantees that upon creating the digital signature it has checked the identity of the public key owner (e.g. by verifying a passport) and that is has checked that this public key owner is in possession of the corresponding private key. Anybody in possession of the CA's public key can verify the CA's signature on the certificate. In this way the CA guarantees that the public key in the certificate belongs to the individual whose identity is in the same certificate.

2.2. X.509 certificates

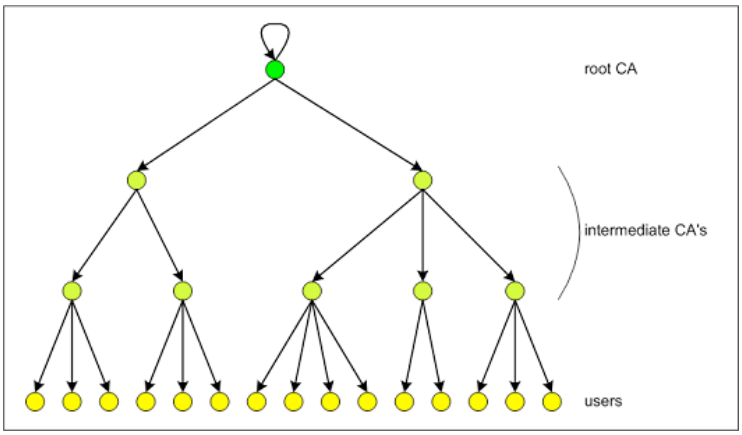
The worldwide accepted standard for digital certificates is called X.509. An X.509 certificate contains the following parts

- the "to-be-signed" part, consisting of:
  - serial number
  - validity period
  - issuer name - this is the identity of the issuing CA
  - subject name - this is the identity of the party being certified; this can be a user (person, organisation, website), another CA, or even the issuing CA itself
  - subject public key - the public key of the party being certified
  - "basic constraints" field, containing
    - a bit indicating whether this is a CA certificate or a user certificate (we will denote this by "CA = TRUE" resp. "CA = FALSE")
    - a path length field, indicating the maximum number of intermediate CAs that are allowed between the "root CA" and the end user
- the "signature" part, containing a digital signature over the "to-be-signed" part; this digital signature is produced using the issuing CA's private key.

By the digital signature in a certificate, the CA guarantees that it has checked (according to its policies) that the subject identity mentioned inside the certificate is indeed the owner of the public key that is inside the same certificate, and that this public key owner also is in possession of the corresponding private key. The signature in the certificate can be verified by anybody, by using the issuing CA's public key.

2.3. Certificate hierarchy

X.509 certificates fit into a hierarchy of CA and end user certificates. A signature on data can be verified by the signer's public key. This public key is linked to the owner's identity by a certificate. This link can be verified by verifying the certificate's signature, using the public key of the issuing CA. This CA public key can be found inside the CA certificate, one layer upwards in the hierarchy. This CA certificate will itself be signed by a CA another layer up. At the top of the hierarchy there is the "root certificate", that is "self signed", and that has to be trusted for its own sake.



Many of these self signed root certificates are installed in browsers or operating systems. Applications using them, such as web browsers, will trust them automatically, i.e. the applications will accept the certificates as authenticated and genuine without explicitly asking the user if it should do so. The user may be notified by small security signs such as a closed padlock symbol in the browser's frame. Every certificate that is in a hierarchy of which the root certificate is trusted, is also automatically trusted.

---

3. Hash function background

3.1. Hash functions

The cryptographic operation that uses a private key to sign data does not deal directly with the data itself, but with a purportedly unique representation of this data, that has a predetermined fixed length, is short and therefore convenient to work with. This can be compared to a fingerprint as a purportedly unique, short and convenient representation of a human being. The process of creating such short representations of data is called "hashing". Unfortunately, because of the fixed length of the hash, there must exist pairs of different inputs that yield the same hash value. Good hash functions, however, have the property that finding such pairs is

extremely difficult, even though they are guaranteed to exist. This is precisely where MD5 has a crucial weakness.

A "hash function" is a cryptographic function that takes as its input a bit string of any length, performs a deterministic algorithm with this input, and produces as output a bit string of fixed length. This output is called the "hash value" or the "hash" of the input. Other terms that can be found for "hash value" are "fingerprint" or "message digest"

If `H` is a hash function, `m` is an input bit string, and `h` is the output of `H` applied to the input `m`, then we write `h = H(m)`. Some common and useful terminology:

- if `h = H(m)` then
  - `h` is called the "hash" of `m`,
  - `m` is called a "preimage" of `h`,
- for a given input `m`, a "second preimage" of `m` is a different input `m'` such that `H(m) = H(m')`,
- if `m` and `m'` are different inputs such that `H(m) = H(m')` then the pair `{m,m'}` is called a "collision" for `H`.

The difference between the concepts of second preimage and collision is subtle, but crucial.

3.2. Hash function requirements

A hash function should satisfy the following requirements:

- (**practicality**) computing the hash `h(m)` of any input `m` can be done efficiently,
- (**preimage resistance**) given `h`, it is hard to compute a preimage of `h`, i.e. it is hard to compute an `m` such that `h = H(m)`,
- (**second preimage resistance**) given `m`, it is hard to compute a second preimage of `m`, i.e. it is hard to compute an `m'` such that $m \neq m'$ and yet `H(m) = H(m')`,
- (**collision resistance**) it is hard to compute a collision for `H`, i.e. it is hard to compute `m` and `m'` such that $m \neq m'$ and yet `H(m) = H(m')`.

Let us assume that we have a hash function which produces hashes with a length of `k` bits. Brute force methods simply try all possible inputs in a certain range, or try a large number of random inputs, until a (second) preimage or a collision are found. For (second) preimage resistance this requires an expected number of hash computations equal to 2^k. For collision resistance the situation is different, due to the "birthday paradox", reducing the number of hash computations to approximately $2^{k/2}$. Cryptographers say that a hash function with hash length `k` provides only `k/2` bits security.

A cryptographic hash function is said to be "cryptographically strong" if no better methods are known than the above mentioned brute force type methods with the mentioned complexities.

3.3. Hash function history

In 1990 Rivest designed MD4, a hash function with hash length 128 bits. Collisions for MD4 were found in 1995. In 2005 a method for finding preimages was published.

In 1991 Rivest announced MD4's successor, MD5, an improved design, also with hash length 128. In 1993 some weaknesses in its design were pointed out, but it took until 2004 before collisions were found. Producing collisions is nowadays a matter of seconds on a PC. The more advanced type of collisions used in our method can be produced in a matter of hours on special but commodity hardware.

In 1995 NIST proposed SHA-1, a hash function with hash length 160 bits. In 2005 weaknesses in its design were found, showing that finding collisions takes essentially fewer computations than the brute force method of complexity 2^{80}, but so far collisions have not been published. NIST advises to replace SHA-1 by 2010.

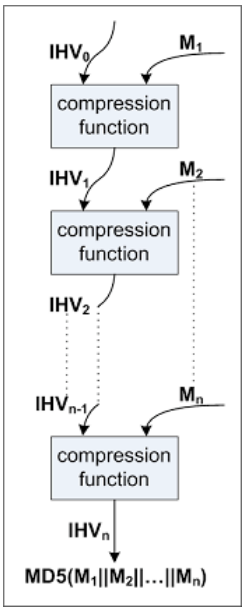
From their introduction until the present day, the hash functions MD5 and SHA-1 have been the work horses of many cryptographic systems. The replacement of MD5 in common applications is far from completed. The replacement of SHA-1 has not really taken off yet.

Even if SHA-1 would have lived up to its design objectives, its output length of 160 is too small to justify its prolonged use for more than the short term. NIST recognized this at an early stage, and came in 2001 with the new SHA-2 family of hash functions. So far these have withstood all cryptanalysis. Nevertheless NIST saw the need for mobilizing the cryptographic community to get a deeper understanding of hash function design and to come up with better hash functions for the next 10 years. Therefore it has started an open competition for selecting the successor of SHA-2, dubbed for the moment SHA-3. The winner of this competition is expected to be selected by 2012, and will most probably become the de facto hashing standard for the next decade.

3.4. The Design of MD5

MD5 uses the Merkle-Damgård iterative construction. The input bit string is padded to a multiple of 512 bits. The length of the unpadded bit string is incorporated in this padding. Then the padded input bit string is divided into blocks of 512 bits each, hereafter called "input blocks".

The core of MD5 is a compression function. The input blocks are fed to MD5 in successive calls to the compression function, that uses each input block in updating a state of 128 bits. This state is called IHV, which stands for "Intermediate Hash Value". Thus the compression function accepts as input a 128 bit state IHV and a 512 bit data block, and produces as output an updated state IHV. The initial state is a fixed value, and the final state is the hash value.



So if we denote the input blocks by M_1, M_2, ..., M_n (512 bits each), the initial IHV by IHV_0 (128 bits), and the compression function by CF, then the sequence of computations is

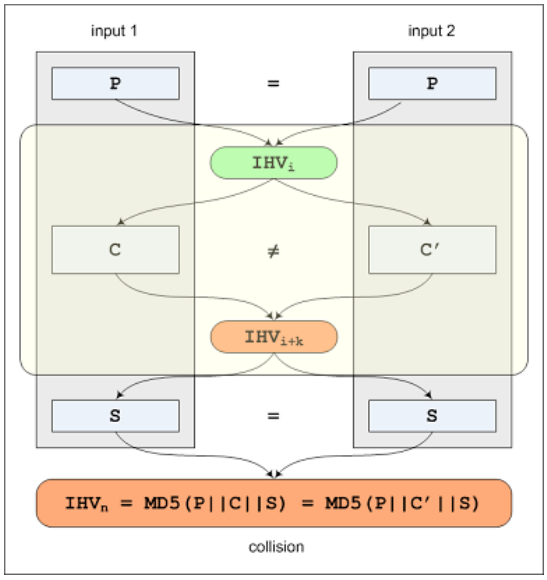
$$
\begin{aligned}
IHV_0 &= \text{fixed value} \\
IHV_1 &= CF(IHV_0, M_1) \\
IHV_2 &= CF(IHV_1, M_2) \\
IHV_3 &= CF(IHV_2, M_3) \\
&\vdots \\
IHV_{n-1} &= CF(IHV_{n-2}, M_{n-1}) \\
IHV_n &= CF(IHV_{n-1}, M_n) \\
\text{MD5 hash} &= IHV_n
\end{aligned}
$$

3.5. Collisions for MD5

In 2004 Xiaoyun Wang and Hongbo Yu presented a collision for MD5 consisting of 2 input blocks, neglecting padding. Details of the collision construction method were published at EuroCrypt 2005 in [WY]. Their method works for any value of the initial IHV_0.

In other words, their method produces on input of any 128 bit IHV_0 a pair $\{\{M_1,M_2\}, \{M_1',M_2'\}\}$, each consisting of two 512 bit input blocks, such that $\{M_1,M_2\} \neq \{M_1',M_2'\}$, and

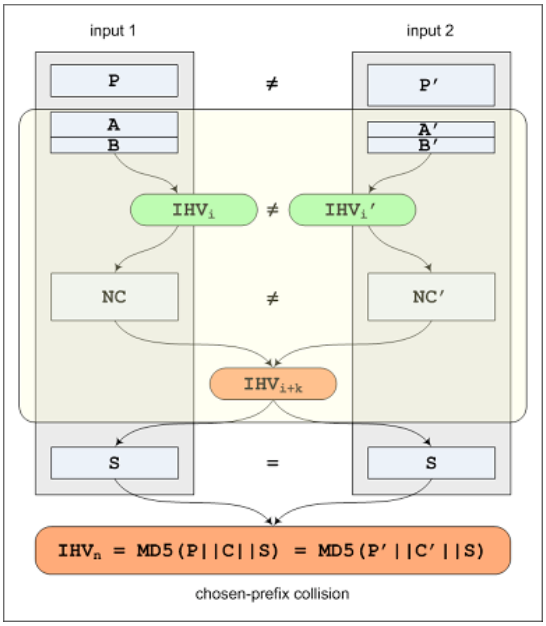
$$
\begin{aligned}
IHV_0 &= IHV_0' \\
IHV_1 &= CF(IHV_0,M_1) \neq IHV_1' = CF(IHV_0',M_1') \\
IHV_2 &= CF(IHV_1,M_2) = IHV_2' = CF(IHV_1',M_2')
\end{aligned}
$$



Due to the iterative structure of MD5 and to the fact that IHV_0 can have any 128 bit value, such collisions can be combined into larger inputs. Namely, for any given prefix P and any given suffix S a pair of "collision blocks" {C,C'} can be computed such that MD5(P||C||S) = MD5(P||C'||S). We use the term "collision block" for a specially crafted bit string that is inserted into another bit string to achieve a collision. One collision block may consist of several input blocks, even including partial input blocks. The collision blocks of [WY] consist of precisely two consecutive input blocks.

In his 2007 MSc thesis [S] (see also [SLW]), Marc Stevens extended this collision construction method to so called "chosen-prefix collisions". This means that collisions can be found between different arbitrary initial IHVs. His method may require more than two input blocks.

Thus, due to the iterative structure of MD5, the chosen-prefix collision construction method is able to produce on input of any pair of chosen prefixes {P,P'} and any suffix S, a pair of collision blocks {C,C'}, such that MD5(P||C||S) = MD5(P'||C'||S).



To be more precise, for given {P,P'} the collision blocks are constructed as follows. First P, P' are padded with bit strings A, A' of any useful size and contents to achieve equal lengths of P||A and P'||A'. Then, using a "birthdaying" step, bit strings B, B' are produced such that the resulting P||A||B and P'||A'||B' have equal length, being a multiple of 512, and such that the resulting IHVs at this point have a prespecified structure. This enables the construction of a pair of "near collision blocks" {NC,NC'} that gives rise to a collision. Each near collision block will consist of a number of 512 bit input blocks. Thus, with C = A||B||NC and C' = A'||B'||NC', the resulting IHVs are identical. Consequently, MD5(P||C) = MD5(P'||C'), and thus also for any suffix S it is the case that MD5(P||C||S) = MD5(P'||C'||S).

The chosen-prefix collision construction method is feasible in practice, but of essentially higher complexity than the method using identical arbitrary initial IHVs. However, in return one gets complete freedom in the choice of both prefixes P and P'.

4. Colliding certificates history

4.1. Meaningfulness

The main problem in trying to abuse the type of collisions for MD5 that can be produced with the available methods is that one does not have sufficient control over the input blocks that are calculated to let the IHVs collide. This means that no methods are known to obtain collision blocks with a prescribed formatting, which would be needed to make the collision blocks fit into a meaningful file.

For a number of different file types, applications of collisions and chosen-prefix collisions have been described in the literature solving the problem of meaningfulness in various ways, see e.g. "The story of Alice and her Boss". We now concentrate on the applications to certificates.

4.2. Different public keys

In 2005 Arjen Lenstra, Benne de Weger and Xiaoyun Wang [LW] showed how collisions of the identical initial IHV type could be built into a pair of X.509 certificates. For an example, see "Colliding X.509 Certificates based on MD5-collisions". The main idea was to hide the random-looking (but carefully engineered) collision blocks in the public key. This will not raise any suspicion, not even to a careful observer who scrutinizes all bits of a certificate looking for signs of fraud, as public keys usually look perfectly like random data anyway. It turned out to be possible to hide collision blocks inside RSA moduli while even assuring the security of the pairs of moduli as being both products of sufficiently large primes. Thus it was shown how to produce a pair of different X.509 certificates with identical MD5 hash values of the "to-be-signed" parts, and therefore with identical CA signatures. This is a violation of fundamental principles of the certificate infrastructure. However, as the prefixes of the colliding inputs had to be equal, the certificates were showing identical identities, severely limiting the practical abuse potential.

4.3. Different identities

In 2007, due to the work of Marc Stevens [S], chosen-prefix collisions for MD5 had become available. This meant that it had become possible to choose any pair of prefixes in the certificates before the RSA moduli, and get certificates with different chosen identities and different public keys (hiding the collision blocks), and with identical MD5 hash values of the "to-be-signed" parts, thus with identical CA signatures. For an example, see "Colliding X.509 Certificates for Different Identities". This is already a much more interesting application. However, a number of limitations complicate this attack when applied to real Certification Authorities.

The attacker has to choose the entire prefix before the collision generation process can begin. This includes fields that the attacker does not directly control, such as the validity period and serial number of the signed certificate. Succesfully predicting those fields requires significant control over the CA's operational procedures, which was thought to be hard to achieve. In addition, the colliding certificates in "Colliding X.509 Certificates for Different Identities" used 8192 bit RSA keys, which are not accepted by all Certification Authorities.

Nevertheless, both the collision construction method of 2005 and the much better chosen-prefix collision construction method of 2007 on certificates based on MD5 were a firm warning to the security community that MD5 should no longer be used for digital signatures in certificates. As attacks only get better with time, it should not come as a surprise that in 2008 we were able to improve upon the previous work on colliding certificates and obtain a rogue CA certificate signed by a commercial Certification Authority.

---

5. Attack details

5.1. Introduction

In July 2008 we set out to investigate if the colliding certificates attack could be applied to a real Certification Authority. The first step was to identify the CAs that still used MD5. Unfortunately it is not possible to determine the hash function a CA uses from the CA certificate. We had to look at (website) certificates issued by the CAs instead. Over the course of a week we spidered the web and collected more than 100,000 SSL certificates, of which about 30,000 were signed by CAs trusted by Firefox. There were six CAs that had issued certificates signed with MD5 in 2008:

- RapidSSL
  C=US, O=Equifax Secure Inc., CN=Equifax Secure Global eBusiness CA-1
- FreeSSL (free trial certificates offered by RapidSSL)
  C=US, ST=UT, L=Salt Lake City, O=The USERTRUST Network, OU=http://www.usertrust.com, CN=UTN-USERFirst-Network Applications
- TC TrustCenter AG
  C=DE, ST=Hamburg, L=Hamburg, O=TC TrustCenter for Security in Data Networks GmbH, OU=TC TrustCenter Class 3 CA/emailAddress=certificate@trustcenter.de
- RSA Data Security
  C=US, O=RSA Data Security, Inc., OU=Secure Server Certification Authority
- Thawte
  C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc, OU=Certification Services Division, CN=Thawte Premium Server CA/emailAddress=premium-server@thawte.com
- verisign.co.jp
  O=VeriSign Trust Network, OU=VeriSign, Inc., OU=VeriSign International Server CA - Class 3, OU=www.verisign.com/CPS Incorp.by Ref. LIABILITY LTD.(c)97 VeriSign

Out of the 30,000 certificates we collected, about 9,000 were signed using MD5, and 97% of those were issued by RapidSSL.

It was quite surprising that so many CAs are still using MD5, considering that MD5 has been known to be insecure since the first collisions were presented in 2004. Since these CAs had ignored all previous warnings by the cryptographic community, we felt that it would be appropriate to attempt a practical attack to demonstrate the risk they present to everybody using a web browser that includes their root CA certificates.

We set out to obtain a legitimate website certificate from this CA for which we would be able to predict the entire contents of the "to-be-signed" part. This certificate would contain a specially crafted collision block in the RSA modulus, which would result in a MD5 collision with a second specially crafted certificate. An ignorant observer, no matter how carefully the certificate is inspected, would most likeley fail to notice anything suspicious about it.

Once we have obtained a valid signature from this CA, we would copy it into our second certificate. Since the "to-be-signed" parts of both certificates have the same MD5 hash, the signature would be valid for the second certificate. Thus the real CA, without knowing it, would provide us with a valid signature for a certificate it has never seen, and should never have signed.

The potential of this attack scenario is even greater than just obtaining a rogue certificate for a single secure website. This is because our rogue certificate doesn't have to be a website certificate, but it could be an intermediary CA certificate. Although the certificate originally signed by the real CA has in the "basic constraints" field the flag "CA = FALSE", indicating that this certificate cannot be used to validate other certificates in a certificate chain, our rogue certificate has the same flag set to "CA = TRUE". We are in possession of the private key corresponding to the public key in this rogue CA certificate. As a result we are able to issue any number of certificates to anybody we choose, and they will be recognized as valid certificates by anybody trusting the real CA, which is all Internet users using one of the common web browsers.

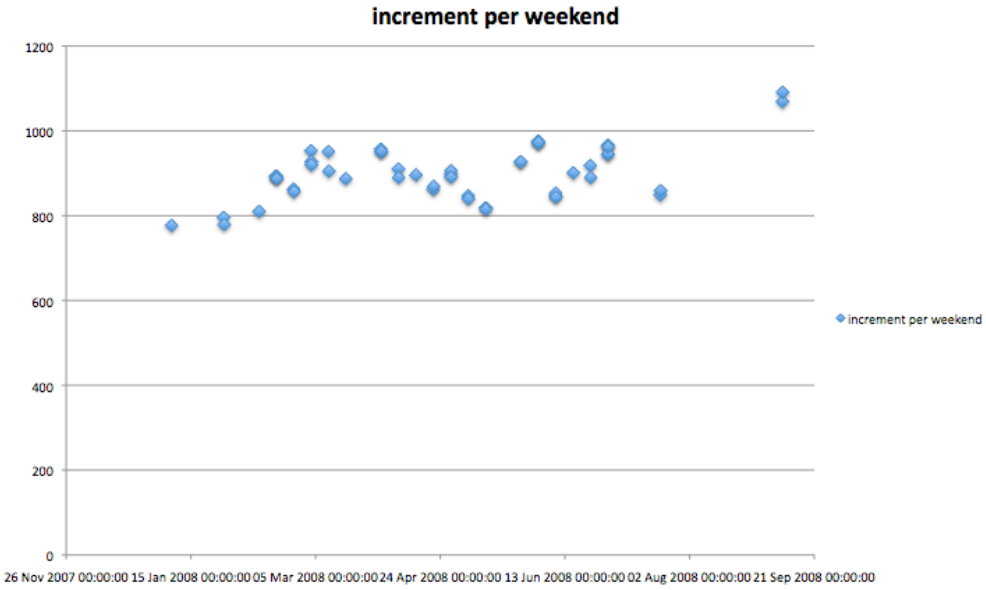
5.2. Validity period and serial number prediction

The first big challenge was to figure out a way to predict the validity period and serial numbers of the certificates issued by a Certification Authority, since these are the only two fields in the chosen prefix that the attacker does not control directly.

Predicting the validity period of the certificate turned out to be an easy task. The system used by RapidSSL and FreeSSL for issuing certificates is fully automated and each certificate is issued exactly 6 seconds after the user clicks on the final "Ok" button to complete the purchase. Since the timestamps in the certificate validity period field use granularity in seconds, we could easily click this button at an arbitrary time T-6 seconds and get a certificate with a validity from time T to T+(1 year). With full control over the validity period, we move on to solving the serial number prediction problem.

We estimated that it would take us 3 days to generate our pair of colliding certificates, which meant that we needed to predict the serial number of the certificate three days in advance. Most of the certificates in our data set had serial numbers that looked random and thus hard to predict, but we noticed that RapidSSL and FreeSSL were using sequential serial numbers.

One can think of the sequential serial number as a remote counter. Each certificate we purchase from the CA reveals the current value of the counter and increments it by one. During the normal operation of the Certification Authority, the counter is incremented based on the number of the issued certificates. Statistical analysis of the serial numbers from the 9,251 certificates in our RapidSSL data set revealed that on average this number has a pretty low variance. In a three day period from Thursday night to Sunday night, the CA typically issues between 800 and 1000 certificates:



The anomaly with the last data point in the graph above is a result of the 100 certificates we purchased during one particular weekend at the end of September. Without our involvement, the increment during that weekend would have been a little less than 1000.

With this information, we decided on the following plan: On Thursday night, we would purchase a certificate to get the current value of the serial number S. We would predict that on Sunday night (time T) the serial number would be a little less than S+1000. A few hours before time T, we would start purchasing certificates to push up the serial number closer and closer to the target number, finally getting it to S+999 about 30 seconds before time T. If we were lucky and the CA received no other certificate request during those last 30 seconds, we would be able to send a request at time T and get our predicted serial number.

5.3. Constructing colliding certificates

In this section a detailed description is given of the contents of the two certificates with identical signatures that we now have. The certificate we obtained from the commercial CA will hereafter be called the "real certificate". Its sibling certificate is our "rogue CA certificate", and it is a sibling in the sense that the "to-be-signed" parts of these certificates were constructed together, to constitute a collision for the MD5 hash function.

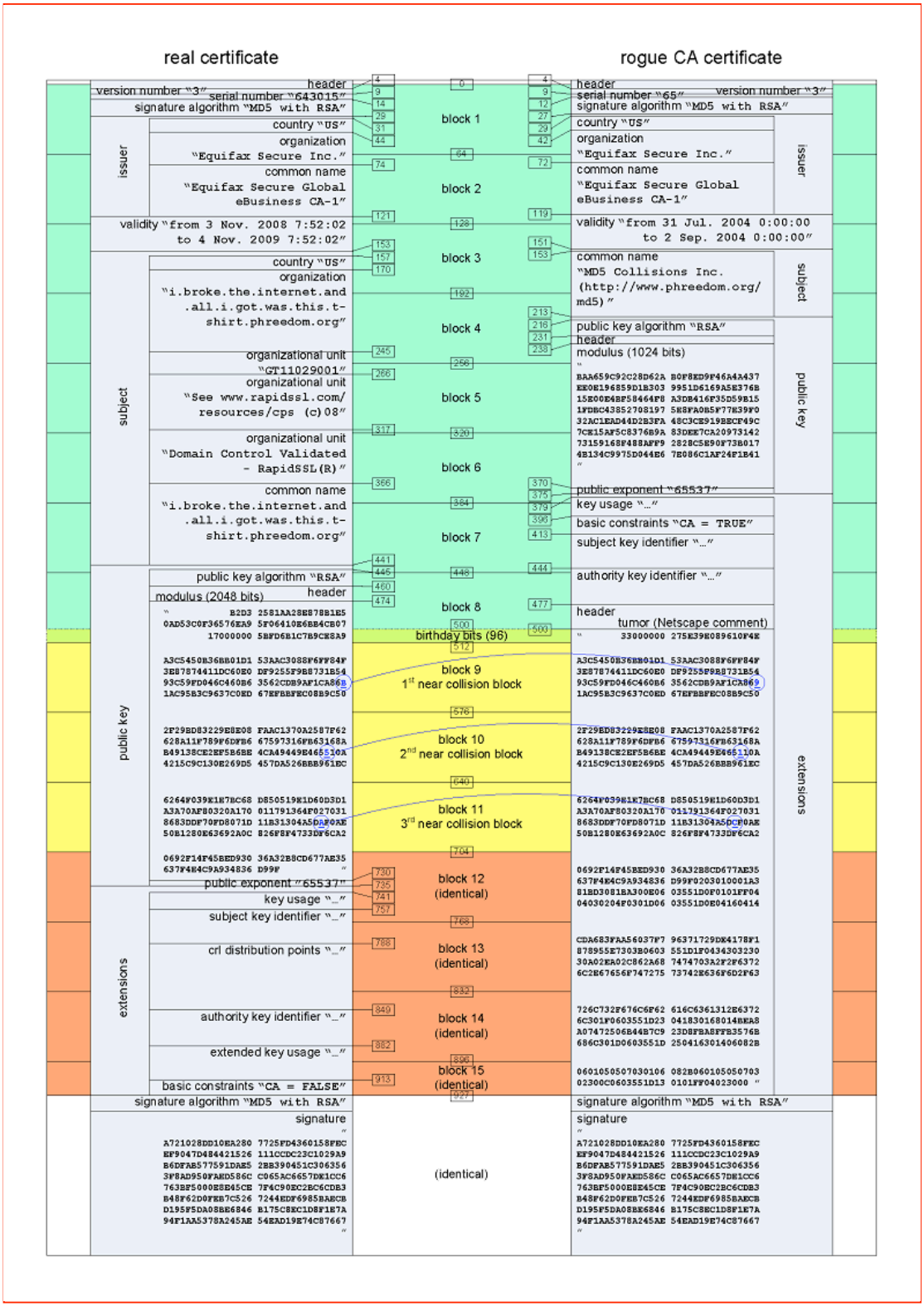
The real certificate was requested from the CA by sending it a "Certificate Signing Request" (CSR). The initial engineering task we had to do is to construct the contents of the CSR and the structure and contents of the rogue CA certificate in such a way that the resulting real certificate (constructed by the CA) and the rogue CA certificate (constructed by us) will be perfectly aligned. The second, and much more intricate, engineering task

was to predict the contents of various fields in the real certificates, and based on this, construct the contents of various other fields in both certificates, in such a way that we would be able to obtain an MD5 collision.

Complying with the X.509 standard [HPFS], each of the two certificates consists of:

- a header of 4 bytes,
- a so called "to-be-signed" part of 927 bytes,
- a so called "signature algorithm" field of 15 bytes,
- a "signature" field of 131 bytes.

The picture below shows in great detail the alignment and most of the contents of the two certificates. The colors green, yellow and red respectively indicate the chosen prefixes, the collision blocks and the suffixes. Click on picture to get it in better resolution in pdf format in a new window.



5.3.1. The real certificate and the CSR

The structure of the "to-be-signed" part of the real certificate is prescribed by the CA. We have control over contents and length of a few fields only. The structure is as follows:

byte 0 - 3: header

byte 4 - 8:	version number (3, default value)
byte 9 - 13:	serial number (643015, set by CA and predicted by us)
byte 14 - 28:	signature algorithm ("md5withRSAEncryption", set by CA)
byte 29 - 120:	issuer Distinguished Name (CA default value)
byte 121 - 152:	validity ("from 3 Nov. 2008 7:52:02 to 4 Nov. 2009 7:52:02", set by CA and predicted by us)
byte 153 - 440:	subject Distinguished Name (Country = "US", Organisation = "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org", Organisational Unit (3 fields set by the CA) Common Name = "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org", Country, Organisation and Common Name set by us in the CSR)
byte 441 - 734:	subject public key info:
	byte 441 - 444: header
	byte 445 - 459: public key algorithm ("RSAEncryption", set by us in the CSR)
	byte 460 - 468: headers
	byte 469 - 729: RSA modulus (2048 bit value, set by us in the CSR)
	byte 730 - 734: RSA public exponent (65537, set by us in the CSR)
byte 735 - 926:	extensions:
	byte 735 - 740: headers
	byte 741 - 756: key usage ("digital signature, nonrepudiation, key encipherment, data encipherment", set by CA)
	byte 757 - 881: subject key identifier, crl distribution points, authority key identifier (uninteresting fields, set by CA)
	byte 882 - 912: extended key usage ("server authentication, client authentication", set by CA)
	byte 913 - 926: basic constraints ("CA = FALSE, Path Length = None", set by CA)

To obtain this certificate we constructed a Certificate Signing Request (CSR), according to the required PKCS#10 format. This request contains the following data:

- "to-be-signed" part:
  - version number (1)
  - subject Country = "US"
  - subject Common Name = "i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org" (used by the CA for the fields Organisation and Common Name)
  - public key algorithm ("RSAEncryption")
  - public key (consisting of the RSA modulus and the RSA exponent)
- signature algorithm ("md5withRSAEncryption")
- signature (2048 bit value, computed with our private key over the "to-be-signed" part, this signature can be verified with the public key inside the request, in this way the CA can be certain that the requestor is in possession of the corresponding private key)

The requirements for the construction of the key pair for the real certificate were as follows:

- the modulus should be a 2048 bit value,
- as public exponent we conveniently chose 65537, as usual,
- the corresponding private key is required to be known by us, because with it the CSR has to be signed,
- the collision blocks are hidden in the modulus.

The alignment turned out such that the 256 byte modulus starts at byte 474. This means that we have 38 bytes of the current 512 bit MD5 input block left. The first 26 bytes of those were simply chosen at random. The last 12 bytes of those were used as "birthday bits", see below for further explanation.

Then we had three "near collision blocks" of 512 bits (i.e. 64 bytes) each. The entire "collision block" thus consists of 96 + 3 x 512 = 1632 bits, i.e. 204 bytes.

At the end of the last near collision block the MD5 collision is established. The collision block was computed by the collision finding method described below. That means that we do not have much control over its contents, we simply have to accept the random looking data that comes out of the cluster of game consoles.

Then we have 256 - 26 - 204 = 26 bytes (= 208 bits) left in the modulus. We used them to make sure that the modulus will have the form of a product of two primes, known to us, in such a way that we can also compute the corresponding private exponent. This is done as follows.

We start with a fixed bit string B of 1840 bits (230 bytes, namely 38 bytes to reach alignment with MD5 input blocks, plus 3 x 64 bytes for 3 near collision blocks). We now want to append another bit string S of 208 bits (26 bytes) such that the resulting 2048 bit $B||S$ represents an integer which is a reasonably secure RSA modulus, i.e. a number $n=p\times q$ with p,q prime, that withstands current factoring methods. In order to find such S, we pick a random 224 bit integer q, and then hope that there is an integer p such that $B||S$ represents the number $n=p\times q$. Then we hope that both p,q are prime. If our hopes turn out to be false, we simply try another random value q, until we succeed.

This method seems coarse, but we don't know how to do it better, and it works well in practice. The reason is as follows. The value of n we are looking for lies in the interval $B\times 2^{208} \le n < (B+1)\times 2^{208}$. For a given q of k bits this means that the interval from $(B/q)\times 2^{208}$ to $((B+1)/q)\times 2^{208}$ should contain an integer p, and furthermore that the integers p,q both are prime. As the width of this interval is $2^{208}/q$, which is approximately 2^{208-k}, we have to try approximately 2^{k-208} times before we find such an integer p lying in the proper interval, irrespective of primality. The probability that q is prime is approximately 1 in $\log(2^k)=k\times\log(2)$. Next, note that p has approximately $2048-k$ bits. The probability that such a number is prime is approximately 1 in $\log(2^{2048-k})=(2048-k)\times\log(2)$. This means that we have to generate approximately $2^{k-208}\times k\times(2048-k)\times(\log(2))^2$ numbers q of k bits before we can expect one hit. In only $k\times(2048-k)\times(\log(2))^2$ cases we have to test a k bit number for primality, and in only $(2048-k)\times\log(2)$ cases we have to test a $2048-$ bit number for primality. We chose $k=224$, which turned out to be doable in practice. The time required for computing this secure modulus was a couple of minutes only on a standard PC.

The resulting RSA modulus thus is rather unbalanced: its prime factors have 1824 and 224 bits respectively. This is estimated to be just out of reach of current factoring methods, in particular ECM.

Note that the private key corresponding to the real certificate is used only for signing the CSR, and that the real certificate itself is used only for providing us with a valid signature. This certificate and corresponding key pair will never be used in some real application, though such use would in principle not harm anybody. An interesting consequence of this is that the creation of a secure RSA modulus is not necessary at all, we could have taken any number for n of which we know the factorisation, such that $\phi(n)$ is coprime to our public exponent 65537. The reason that we need the factorisation is that we need to compute the private exponent, as we have to sign the CSR. There is even no need for n to be a product of two primes, having a prime or any kind of factorisation would also do. Nevertheless we chose to be careful here too, and to make it a secure RSA modulus of which the private key will be kept secret as well.

An interesting point to make is that the "path length" field in the "basic constraints" field of the CA's root certificate is not set. The path length field is the maximum number of intermediate CA levels in the CA hierarchy that this CA will allow between its own level and the end user level. If the CA would have set the path length to 0, then applications checking for the path length would have rejected certificates signed by our rogue CA, as there the path length is 1: our rogue CA is at the level between the end user certificate and the root CA certificate.

For the collision construction it is absolutely necessary that we are able to predict at bit level the entire contents of the "to-be-signed" part of the certificate up to the point where the collision block starts. The only fields that are not either constant, or directly controlled by us are the validity period and the serial number. However, using the techniques described in section 5.2, we were able to predict the values of these two fields.

5.3.2. The rogue CA certificate

The structure of the "to-be-signed" part of the rogue CA certificate is chosen by us (but of course within the limits of the X.509 standard), to facilitate the collision construction process. Here the only field prescribed by the CA is the issuer Distinguished Name. The structure of the rogue CA certificate is as follows:

byte 0 - 3:	header
byte 4 - 8:	version number (3, default value)
byte 9 - 11:	serial number (65, arbitrary choice)
byte 12 - 26:	signature algorithm ("md5withRSAEncryption", set by us in the CSR)
byte 27 - 118:	issuer Distinguished Name (CA default value)
byte 119 - 150:	validity ("from 31 Jul. 2004 0:00:00 to 2 Sep. 2004 0:00:00")
byte 153 - 212:	subject Distinguished Name (Common Name = "MD5 Collisions Inc. (http://www.phreedom.org/md5)")
byte 213 - 374:	subject public key info:
	byte 213 - 215: header
	byte 216 - 230: public key algorithm ("RSAEncryption")

	byte 231 - 237:	headers
	byte 238 - 369:	RSA modulus (1024 bit value)
	byte 370 - 374:	RSA public exponent (65537)
byte 375 - 926:	extensions:	
	byte 375 - 378:	headers
	byte 379 - 395:	key usage ("digital signature, nonrepudiation, certificate signing, offline CRL signing, CRL signing")
	byte 396 - 412:	basic constraints ("CA = TRUE, Path Length = None")
	byte 413 - 476:	subject key identifier, authority key identifier (uninteresting fields)
	byte 477 - 926:	tumor (Netscape extension)

The field that makes this certificate a "CA certificate" is the "basic constraints" fields, with the value "CA = TRUE".

The RSA key pair, of which the public key is inside the rogue CA certificate, was generated by OpenSSL in a standard key pair generation process. The private key is kept in a safe place and we're not planning on releasing it publicly. This private key is required for issuing end user certificates signed by the rogue CA, so this is quite sensitive.

To minimize this danger, the validity period of the rogue CA certificate is set to a relatively short period in the year 2004 (the month August 2004, when Xiaoyun Wang presented the first MD5 collision at the Crypto Conference in Santa Barbara). This backdating is done on purpose. It means that even if the private key to our rogue CA certificate falls into the wrong hands, the certificate would be rejected by all common browsers because of its expiration date, except when the system clock of the user's computer would be set back to within this short period in 2004. The probability that this will happen unintentionally is negligible.

Note that the serial number we have chosen is a rather low one. We do not know whether an actual certificate issued by this CA with this serial number was already in existence. We took a low number because that most probably implies that a possible other certificate with that serial number has already expired a long time ago.

5.3.3. Alignment

The bytes 0 - 473 in the real certificate (the fields up to the modulus, and the first 5 bytes of the modulus field which are a predictable header) are pretty much fixed by CA requirements. Those 474 bytes form the "chosen prefix" on the real certificate's side. For this certificate we chose to have a 2048 bit RSA key. The main reason for this size is the fact that we have to hide the collision block in there. Our collision construction method enables us to make collision blocks of 1632 bits, so 2048 seems a reasonable choice. Moreover 2048 bit RSA moduli are quite common, so no suspicion is raised.

At the side of the rogue certificate we could not use the public key modulus for hiding the collision block. The reason for this is that we want to set the CA flag in the "basic constraints" field to the value "TRUE", contrary to the corresponding field in the real certificate. The problem is that this "basic constraints" field occurs after the public key, and with the type of collisions we have, we cannot allow any bit difference after the collision block. So in the rogue CA certificate, both the public key and the extensions up to the "basic constraints" field have to be placed before the collision block starts.

To accomplish this, it was helpful that the subject Distinguished Name in the real certificate has considerable length, that can easily be stretched by choosing an appropriately sized Common Name. At the same time we took the subject Distinguished Name in the rogue CA certificate as short as possible, to allow sufficient space for a full 1024 bit public key and the "basic constraints" field, all this to be placed in the area corresponding to the subject Distinguished Name field of the real certificate. In this way all necessary fields belonging to the "chosen prefix" of the rogue CA certificate could be placed in the first 477 bytes of the "to-be-signed" part.

The next problem that had to be solved is that we need a field in the extensions of the rogue CA certificate in which we have to hide at least $3 \times 512 + 96 = 1632$ bits (204 bytes) of the collision block. This is random looking data, over which we do not have much control, as this data comes out of the collision construction method, and we do not control that enough to be able to make such a block with some real world interpretation. And we have one more problem, as all data after the public key from the real certificate needs to be copied bitwise into the rogue CA certificate. This data does have an interpretation relevant to the real certificate, but this interpretation most probably is not relevant, even disturbing, for the rogue CA certificate. Together those fields have a considerable size, namely 427 bytes.

These two problems together ask for a place in the rogue CA certificate to hide 427 bytes of data in, some of which are random looking, and some of which have an interpretation that should not be shown in certificate viewers. The solution we adopted is to define a so called "Netscape Comment" block. This is a proprietary extension in which any data can be stored, which will be ignored by most certificate processing applications,

including the major browsers. There is a small problem here in that formally the contents of this field must be of type IA5String, and our content is not of that form. An ASN.1 parser that strictly follows the standard will complain about this, as happens e.g. with Peter Gutmann's program "dumpasn1". But as most application software ignores the extension anyway, the certificate with this standard violating field in it will still be accepted in practice. It is conceivable that the 427 bytes could have been hidden in a different extension field (there is even an X.509 certificate with a movie hidden inside).

The Netscape Comment extension requires a header of 23 bytes. So the contents of the extension will start at byte 500, and that now can become exactly the spot where the birthday bits start, in both certificates. This explains the fact that in the real certificate, where the modulus starts at byte 474, there are first 26 random bytes to fill up the space. In a technical sense the "chosen prefixes" of the two certificates end at byte 499. The collision block starts at byte 500. This collision block consists, as said before, of 96 + 3 × 512 = 1632 bits = 204 bytes, so it ends at byte 703.

So from byte 500 until byte 926 there are 427 bytes inside the rogue CA certificate that are quite meaningless to the certificate itself. We call this big, seemingly unnecessary block of data a "tumor". From our point of view it's a benign one.

We start the modulus of the public key of the real certificate at byte 474. As explained above, for alignment with the rogue CA certificate we need to fill up the first 26 bytes somehow, so we just chose 26 random bytes to start the modulus with. We have to think in terms of blocks of 64 bytes, as MD5 splits up the input in 512 bit blocks. So we now have 12 bytes left until the next 512 bit block starts with a new `IHV` value.

So at byte 500 of the "to-be-signed" part we have reached the stage where in the real certificate we are inside the modulus, and in the rogue CA certificate we are inside the tumor.

The MD5 collision construction method we used consists of two stages: a "birthdaying" stage and a "near collision" stage. The birthdaying stage is meant to prepare the next difference in `IHV`s between the two certificates, at the border between two 512 bit blocks. Thus we have 12 bytes, i.e. 96 bits, available for those birthday bits. This is, due to improvements in the birthdaying procedure, more than enough. In fact, we needed only 72 bits for our new technique of birthdaying, so of the 12 available bytes three were simply set to 0. Note that, by their very nature, these 72 birthday bits are completely different in both certificates, and look completely random. These birthday bits were computed by our collision finding programs, for details see below.

Then there are three MD5 input blocks of 512 bits each, i.e. 3 × 64 bytes, byte numbers 512 to 703. Each of the three input blocks was constructed by our collision finding method to cancel out certain differences in the `IHV`s. Contrary to the birthday bits, these near collision blocks differ only in one or two bits. At the end of the third block the difference has vanished entirely, thus constituting a collision for the MD5 compression function. From byte 704 on the certificates are exactly, i.e. bitwise, identical.

In the real certificate we now have 26 + 12 + 3 × 64 = 230 bytes of the modulus. The remaining 26 bytes of the modulus were computed to ensure that it's a secure RSA modulus, as explained above.

Then in the real certificate the public exponent and all "version 3 extensions" follow. In total this required 197 bytes. They were generated in the real certificate by the CA, and then simply copied into the tumor in the rogue CA certificate. One of the fields in there is the "basic constraints" field, set to "CA = FALSE". Interestingly enough, the bytes of this field are also present in the tumor, but the certificate processing software will not recognize them as having any interpretation attached to them, as they are part of the tumor.

After the extensions in the real certificate and the tumor in the rogue CA certificate have ended, at byte 926, the "to-be-signed" part of the certificates are finished, and the MD5 hash function can be called on these 926 byte long strings to produce the colliding value.

What finally follows inside the certificate are the "signature algorithm" field and the "signature" field. Naturally, those fields are identical in both the certificates.

For download: binary files containing both "to-be-signed" parts:

- "to-be-signed" part of real certificate, binary file
- "to-be-signed" part of real certificate, human-readable hexadecimal file
- "to-be-signed" part of rogue CA certificate, binary file
- "to-be-signed" part of rogue CA certificate, human-readable hexadecimal file

To check that the two "to-be-signed" parts collide, run an MD5 program on the binary files. To check that the two "to-be-signed" parts differ, run e.g. a SHA-1 program on the binary files.

Here are the MD5 Intermediate Hash Values:

IHV	real certificate	rogue CA certificate
IHV_0	0123456789ABCDEFFEDCBA9876543210	0123456789ABCDEFFEDCBA9876543210
IHV_1	058484A77F07A36382AAECF2DFE207A2	713F764E78B5C9B03F8878F7A440551B
IHV_2	D52743425C3DAC23A9E62C6C9670622E	2AC9681DDB3B72D29A1422A515C9E4F4
IHV_3	7789E58E3B45621A3E46A64CA9D7AC3A	104DD09F9F651E554C528578AC1F6885
IHV_4	CDA2CB5673D3D32092C7F1EF80CE5729	15ADC95447929A2AC0EACF9E618E14EB
IHV_5	F08E24604482508B959A0B5762207A3F	D6D6E59C0BDB1F701CB04C29A0573EA0
IHV_6	A83EA6CCCC50B41A4BFADBC6D856B338	3AAB0CE98F1E9B2AC270A5A2C60FF605
IHV_7	0B42EAAB4258AACA8C30BDB8192A1BC0	DE3CCC11526732CA0FD8B9F5992A7673
IHV_8	D21CED8CC56726B6BF2AE4A93D742C3A	D21CED8CCE3D7EB4C82ADCA94674243A
IHV_9	DC1EDBFFF3C3E9E7BCEB3F9E2D0705BD	DC1EDBFFF49941E8BDEB379E2E07FDBC
IHV_{10}	F0D655805A71A74EF8A6A630D11977D8	F0D655805A479F4EF8A69E30D1196FD8
IHV_{11}	9808B5471E7130CC5A30A2ABF2BE4B4D	9808B5471E7130CC5A30A2ABF2BE4B4D
IHV_{12}	AA1F57B21A8732130CB0CAEF4BB9C746	AA1F57B21A8732130CB0CAEF4BB9C746
IHV_{13}	151754FA2FCC5914E72B71B4300B6485	151754FA2FCC5914E72B71B4300B6485
IHV_{14}	271EECDC4DAC9E9C471C34C833917E26	271EECDC4DAC9E9C471C34C833917E26
IHV_{15}	9ED7B966BD815C141B899DC64B528564	9ED7B966BD815C141B899DC64B528564
MD5	9ED7B966BD815C141B899DC64B528564	9ED7B966BD815C141B899DC64B528564

Here are the XOR differences of the MD5 Intermediate Hash Values:

IHV	XOR difference
IHV_0	00000000000000000000000000000000
IHV_1	74BBF2E907B26AD3BD2294057BA252B9
IHV_2	FFEE2B5F8706DEF133F20EC983B986DA
IHV_3	67C43511A4207C4F7214233405C8C4BF
IHV_4	D80F02023441490A522D3E71E14043C2
IHV_5	2658C1FC4F594FFB892A477EC277449F
IHV_6	9295AA25434E2F30898A7E641E59453D
IHV_7	D57E26BA103F980083E8044D80006DB3
IHV_8	000000000B5A5802770038007B000800
IHV_9	00000000075AA80F010008000300F801
IHV_{10}	00000000003638000000380000001800
IHV_{11}	00000000000000000000000000000000
IHV_{12}	00000000000000000000000000000000
IHV_{13}	00000000000000000000000000000000
IHV_{14}	00000000000000000000000000000000
IHV_{15}	00000000000000000000000000000000
MD5	00000000000000000000000000000000

The SHA-1 hash values respectively are `1AA8050F29EECAAFE1F6815487FF164783129E63` and `5D23230D1B36C8DDD21CB5ABDAED46967D1ED4AD`.

The differences in the MD5 `IHV`s are depicted in the following image.

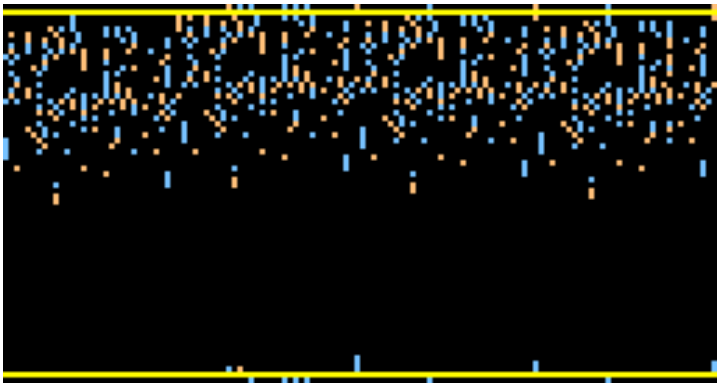
Each line represents the 128 bits of one IHV difference (this time not the XOR difference but the difference in non-adjacent form for 32 bit words; a black pixel stands for no difference, a blue pixel stands for a +1 difference, an orange pixel stands for a -1 difference). It is clear that the IHV difference looks random in IHV_1 to IHV_7, gets special in IHV_8 (this is due to the birthdaying), and then gets canceled out piece by piece in the next three input blocks, so that it has completely vanished in IHV_{11}.

5.3.4. Collision construction

The chosen-prefix collision construction method is basically described in our EuroCrypt 2007 paper [SLW]. However, some crucial improvements to this method have been developed that made the present application possible. Details of those improvements will be published in a forthcoming academic paper. The basic idea of the method is twofold. In the second part of the method differential paths are constructed that are capable of eliminating special bit differences in the IHVs with certain probabilities. This can be called a search for near collisions. In a few consecutive "near collision blocks" thus special combinations of bit differences can be eliminated. For each bit difference a fresh differential path has to be constructed, so it is crucial to have an automated way to produce differential paths.

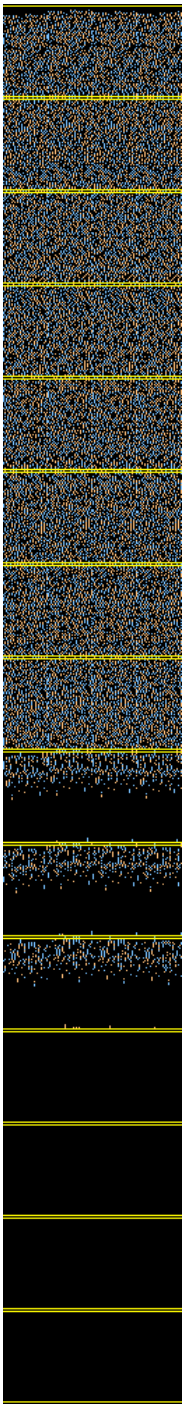
The first part of the method prepares the IHV difference so that it becomes suitable for the second part. This is simply done by a birthday search for an input pair that has an IHV difference with the proper structure. Optimization of the trade-off between the number of birthday bits and the number of near collision blocks is another item that will be discussed in the forthcoming paper. In the case at hand we decided to choose for 72 birthday bits and 3 near collision blocks. The computational bottleneck is the birthday search.

To illustrate the collision construction we have made some nice pictures of bit differences in the internal states of MD5.



The entire inputs for MD5 consist of pairs of 15 input blocks of 512 bits each. In each compression function call the input IHV is put into an internal state of 128 bits. The compression function performs 64 rounds, in each round updating the state, using some bits from the input block. After the 64th round the internal state is XORed with the input IHV to produce the output IHV.

In the pictures above and to the right the differences between IHVs and the 64 internal states in between the rounds are shown. Here each time the difference between the two input files are taken. These differences are computed in non-adjacent form for 32 bit words. Each line in the pictures shows the 128 bits of an IHVor an internal state difference. In the animated picture above the first line, above the yellow division line at the top, shows the input IHV difference, the last line, below the yellow division line at the bottom, shows the output IHV difference, and the 64 lines in between the yellow division lines show the 64 internal state differences after each round. the animation shows the processing of the 15 compression function calls one at a time. The picture to the right shows the same data in one big row rather than animated. Note that each output IHV is the input IHV for the next compression function call.



A single attempt for constructing a chosen-prefix collision costs about a little more than a day. The first stage consisting of the birthday search is computationally the most expensive. Luckily it is also very suited for the special SPU cores of the Cell Processor that the Sony PlayStation 3 uses. We had about 200 PS3s at our disposal, located at the "PlayStation Lab" of Arjen Lenstra at EPFL, Lausanne, Switzerland (see the picture). The birthdaying takes about 18 hours on the 200 PS3s using 30GB of memory that was equally divided over the PS3s. The second stage computes the 3 collision blocks that eliminate the `IHV` differences left after the first stage and costs in total about 3 to 10 hours on a high-end quadcore pc. This part is not suited for the PS3s SPU cores due to the large memory demands and the high number of branches in the software execution flow. Since the two stages are performed on different hardware, we were able to pipeline several attempts, so that within one weekend (72 hours) we could easily perform 3 attempts (3 × 18 + 10 = 64 hours). The total complexity of the collision construction can be estimated at $2^{51.x}$ MD5 compression function calls, when 30 GB of memory is available.

5.4. Real life execution

For one single attempt to obtain a CA signature we had to guess the serial number and the validity period the CA was going to use a few days in advance. This information is crucial to get the chosen prefix for the real certificate's "to-be-signed" part, and this is input for the collision construction method. Once a collision has been found, the public key modulus has to be computed. Then the Certificate Signing Request could be composed and sent to the CA at the precise time, and then we had to wait for the CA to return a signed certificate, and see if the serial number and validity period were guessed correctly.

We performed our attemps over the weekend to take advantage of the lower load of the CA. Each weekend we would generate from one to three collisions with serial numbers within the predicted range and purchase enough certificates to increment the serial number up to the desired value. Unfortunately, the first three tries failed due to timing issues.

We received a couple of certificates signed only a second too late and a number of times requests by other customers of the CA interfered with our attempts to get the right serial number. Finally, on the fourth weekend we succeeded in getting a certificate with the right serial number signed at the right time.

Each certificate we purchased from RapidSSL cost us USD 45. However, the CA allowed us to reissue each certificate up to 20 times for free, which meant that a single certificate request cost us only USD 2.25. In total, we spent USD 657 on purchasing certificates for this project.

5.5. Results

The results of executing our scenario are two certificates. We make them available for download in the common .cer and .pem formats. Moreover we give the output of the "dumpasn1" program applied to the certificates, to presents the certificate structure in a more readable form.

- real certificate (.cer)
- real certificate (.pem)
- real certificate (output of dumpasn1)
- rogue CA certificate (.cer)
- rogue CA certificate (.pem)
- rogue CA certificate (output of dumpasn1)
- example website certificate signed by the rogue certificate (.cer)
- example website certificate signed by the rogue certificate (.pem)
- example website certificate signed by the rogue certificate (output of dumpasn1)

The private key for the certificate is not available for download, but this should not prevent anybody from verifying the authenticity of our certificate.

5.6. Demonstration

To see our rogue CA certificate in action, set your computer's system clock to August 2004, and then click https://i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org/. Do not forget to reset your system clock afterwards.

---

6. Impact

A short summary of our result is that we have come in possession of a "rogue" Certification Authority whose certificate will be accepted by default by most browsers. Thus we are able to issue SSL certificates to any website we like, including rogue websites claiming to be legitimate ones.

This has been possible by exploiting the following weaknesses:

- an efficient method to construct "chosen-prefix collisions" for the MD5 hash function,
- there is at least one commercial Certification Authority that:
  - issues certificates with a signature created using the MD5 hash function,
  - processes online requests for certificates in an automated way,
  - does not check for anomalous requests,
  - allows predicting with reasonable probability of success a valid combination of serial number and validity period,
  - has no technically enforced limit on the length of a chain of certificates.

Any website, whether it is secure (i.e. uses SSL) or not, whether it has an MD5-based, SHA-1-based, SHA-256-based, or any other type of certificate, irrespective of which Certification Authority issued the certificate, can be impersonated, in particular not only genuine websites that have an MD5-based certificate are vulnerable.

The rogue Certification Authority we have has been set up for demonstration purposes only. Its certificate has on purpose been backdated, so that it will be recognized by browsers as having expired more that 4 years ago. Moreover the corresponding private key, needed to issue certificates with, will be kept in a safe place to prevent abuse. Thus it is unlikely that with this particular rogue Certification Authority harm can be done.

However, we have shown that it is possible in practice, with some non-negligible but doable effort, to set up such a rogue Certification Authority. If we can do it now, others will be capable of doing it as well in the foreseeable future. In theory it is even conceivable that other groups or individuals already have developed similar skills and knowledge as we have, and are already able to impersonate websites at their will. We have no indication that this is actually the case. Because of this we believe it is important to stop using MD5 as soon as possible.

In combination with known weaknesses in the Domain Name System (DNS) protocol such as Kaminsky's "DNS Flaw" [K2] (see also [OMM]), the vulnerability we exposed opens the door to virtually undetectable phishing attacks. Without being aware of it, users can be redirected to malicious sites that appear exactly the same as the trusted banking or e-commerce websites they believe to be visiting. User passwords and other private data can fall into wrong hands.

We do not want to help persons with criminal intent. Therefore we will for the time being not release the full details of how we have been able to obtain the rogue Certification Authority certificate. It should however be noted that the basic principles of constructing "chosen-prefix collisions" for MD5 were published already in May 2007, see [SLW], and the "Chosen-prefix collisions" website. To make our present results possible these publicly known techniques have been improved at crucial points. These improvements will be published in a forthcoming academic paper. Apart from that, somebody who wants to redo our work has to do considerable implementation and optimization efforts. For the time being we do not plan to release such implementation details.

Other applications than secure web communication using SSL might be vulnerable as well. Every Certification Authority that will honor requests for MD5-based certificates and that has sufficiently predictable serial numbers and validity periods, may be vulnerable to similar attacks. This may include Certification Authorities in the areas of e-mail signing and encryption, software signing, non-repudiation services, etc.

We feel safe to say that the impact of our work is very limited in the sense that we think it is highly unlikely that our rogue Certification Authority can be abused by anyone (including ourselves!). We have released sufficient detail of our methods to show that we have a realistic attack scenario. The proof of the pudding is our colliding certificates, available for download on this website, so that everyone can verify our claims, and in the demonstration we have provided. Our intent is to raise awareness that MD5 is broken so drastically that its continued use in digital signature schemes and certificates poses realistic threats.

Our desired impact is that Certification Authorities will stop using MD5 in issuing new certificates. We also hope that use of MD5 in other applications will be reconsidered as well. Proper alternatives are readily available.

6.1. Revocation issues

One interesting observation from our work is that the rogue certificate we have created is very hard to revoke using the revocation mechanism available in common browsers. There are two protocols for certificate revocation, CRL and OSCP. Until Firefox 3 and IE 7, certificate revocation was disabled by default. Even in the latest versions, the browsers rely on the certificate to include a URL pointing to a revocation server. Our rogue CA certificate had very limited space and it was impossible to include such a URL, which means that by default both Internet Explorer and Firefox are unable to find a revocation server to check our certificate against.

As a workaround, system admistrators can configure the browsers to always query a company-specific OSCP

server for revocation information, but this option is not easily usable by individual end users. This points out an important problem that still needs to be resolved, in case similar attacks result in certificates that need to be revoked in the future.

Another interesting scenario is that our result opens possibilities for Denial of Service attacks on existing Certification Authorities. If an attacker wants to have somebody's certificate revoked (e.g. of a competing web shop), he can use our techniques to obtain a rogue certificate from the same Certification Authority, which has the same serial number as the target certificate. As soon as this rogue certificate gets published, the Certification Authority has no choice but to revoke it. Since revocation happens only on the basis of the serial number of the certificate, at the same time the legitimate certificate of the victim will be revoked. This attack will become extremely powerful if the root certificate of the Certification Authority is targeted. An attacker can thus force revocation of this root certificate, and with that of the entire certificate tree depending on it.

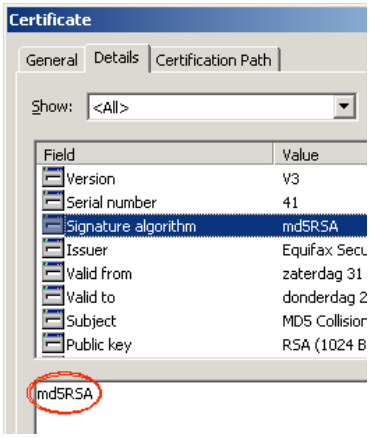
7. Countermeasures

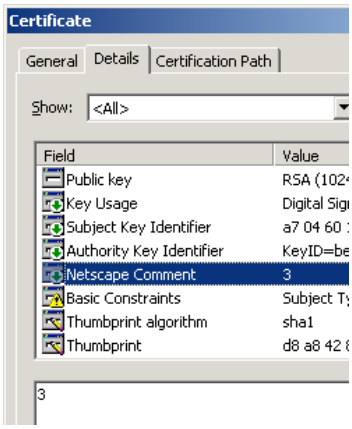
First and foremost, there is no proper excuse for continued use of a broken cryptographic primitive when sufficiently strong alternatives are readily available, for example SHA-2.

Secondly, there is no substitute for security awareness. Openness about security problems, vulnerabilities and technical possibilities is invaluable to make the Internet a safer place. Advice from experts should be taken seriously and early in the process. In this case, MD5 should have been phased out soon after 2004.

This being said, we continue discussing countermeasures that can be taken by different stakeholders.

Users browsing the web might be redirected to a rogue website that has obtained an erroneously trusted SSL certificate. A user cannot do anything to prevent this from happening. She might try to detect it, but we realize that this is somewhat cumbersome for the average web user. The user may click on the proper button or menu item in the browser, to view details of the certificate for the website she is visiting. The used hash function is visible in the "Signature algorithm" field, see the picture to the right, where "md5RSA" means that MD5 was used for signing the certificate. When all certificates in the chain up to the root CA certificate use other hash functions than MD5 such as SHA-1, our attack has not been used.



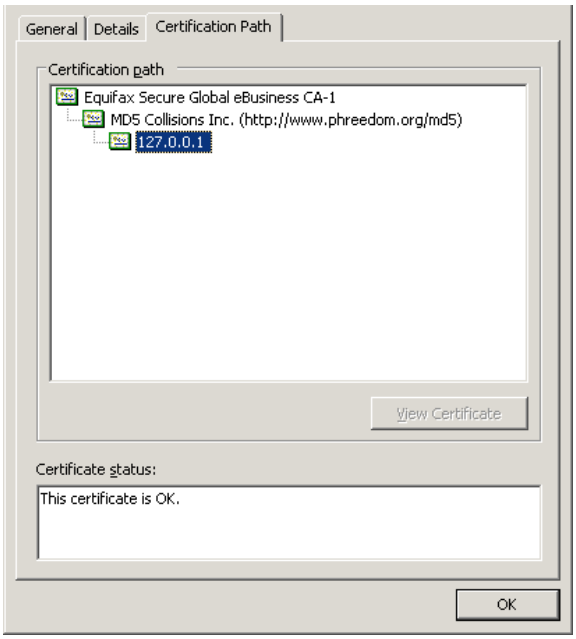


When MD5 has been used, fraud may be detected by inspection of the certificate at bit level. When the certificate has a "tumor" inside, this may or may not be shown in a user friendly certificate viewer. See the picture on the left, where there is a strange "Netscape comment" field with the value 3. Actually this is a 427 byte field that for technical reasons cannot be shown on screen in this viewer. An expert who can inspect the certificate with tools such as "dumpasn1", will be able to identify such a strange field in the certificate. However, a standard user will likely not notice anything. Therefore inspection of certificates is not a strong countermeasure.

Preventive countermeasures are much more powerful. We now describe the preventive countermeasures that CAs and browser vendors can take, but users cannot.

Certification Authorities are recommended to stop using MD5 altogether. To prevent chosen prefix attacks, they can add a sufficient amount of randomness to the certificate fields, preferably as far to the start of the certificate as possible. The serial number is a good field to use for this, as it comes very close to the beginning of the certificate, and it allows for 20 bytes of randomness. Many Certification Authorities seem to use already random serial numbers, albeit probably for different reasons. The German Signature Law prescribes random serial numbers for qualified SHA-1 certificates (see [B]) to provide some randomness in the hash input. We do note however that this use of randomness in the serial number is a workaround, made possible by lucky choices in the X.509 standard. It is not a bad idea in general to add randomness to a hash input when a possible attacker is able to choose the input. A much more reliable, since designed, solution is to use randomized hashing, see [HK]. Such a solution introduces ramdomization as a "mode of operation" for hash functions, which is a much more fundamental approach to the problem than relying on features that happen to be present in existing standards for non-security reasons, or for no reason at all.

Other measures a Certification Authority may take



are to make active use of the "pathlength" parameter in the "basic constraints" field. When a CA has the policy of only issuing end user certificates, this policy can be enforced technically by setting the path length to 0. Our specific attack scenario can be detected because there is a three-level hierarchy (see the picture). This detection can however be avoided by a small variant of our attack scenario, where the rogue colliding certificate is not a CA certificate but a website certificate in itself. We note that such a rogue colliding end user certificate may be constructed without a tumor: the collision block can then be hidden again inside the public key. This makes detection much more difficult. Other ways of formatting the certificate may be possible.

An interesting question is whether CAs should revoke existing certificates signed using MD5. One may argue that the present attack scenario has in principle been possible since May 2007, and that therefore all certificates (or all CA certificates) signed with MD5 that have been issued after this date may have been compromised. Whether they really have been compromised is not relevant. What is relevant is that the relying party who needs to trust the certificate does not have a proper way of checking whether the certificate is to be trusted or not. One may even argue that all older certificates based on MD5 should be revoked, as for an attacker constructing rogue certificates it is easy to backdate them to any date in the past he likes, so any MD5-based certificate may be a forgery. On the other hand, one may argue that the likelihood of these scenarios is quite small, and that the cost of replacing many MD5-based certificates may be substantial, so that therefore the risks of continued use of existing MD5-based certificates may be seen as acceptable. Regardless, MD5 should no longer be used for new certificates.

Finally, Certification Authorities can monitor the flow of Certificate Signing Requests that they receive for abnormal series of requests, such as many requests by the same user in quick succession.

Browser and Operating System vendors such as Microsoft (vendor of Windows and Internet Explorer) and Mozilla (vendor of Firefox) can implement pop-up warnings to the users when an MD5-based certificate is encountered. Blocking MD5-based certificates is also possible, but rather drastic. Browser vendors can implement path length checking. Furthermore, it is the browser vendors who determine which Certification Authorities are present in the trust lists inside the browsers or operating systems. This puts them in a good position to put pressure on the Certification Authorities to adopt proper procedures and use strong cryptographic primitives. We have contacted the mentioned browser vendors so that they are aware of the problem.

Website owners can check whether their Certification Authority has proper procedures, notably does not use unacceptable hash functions such as MD5. Website owners can ask their CAs to switch to more secure hash functions such as SHA-2.

7.1. Responses by Verisign (the owner of RapidSSL) and Microsoft [added Dec. 31]

Verisign, the owner of the RapidSSL brand, has immediately responded when our work became public. See the announcement "This morning's MD5 attack - resolved" by Tim Callan. Some interesting quotes from this blog:

- "We applaud security research of this sort and are glad that white hats like the "MD5 Collision Inc." group make a point of investigating online security."
- "We have discontinued using MD5 when we issue RapidSSL certificates, and we've confirmed that all other SSL Certificates we sell are not vulnerable to this attack. We'll continue on our path to discontinue MD5 in all end entity certificates by the end of January, 2009."
- "... any customer who would like to do so can replace any MD5-hashed certificate free of charge."

We are happy with this very quick and adequate response.

Microsoft issued a Security Advisory (961509): "Research proves feasibility of collision attacks against MD5".

---

8. Frequently asked questions

Question. Can your proof of concept rogue CA certificate be misused?
Answer. This is highly unlikely. We intentionally set the expiration date of our rogue CA certificate to August 2004. Any certificate signed by our rogue CA will similarly be invalid after August 2004. This allows us to test our proof of concept certificate by setting the system date back to August 2004, but prevents the certificate from

being misused if it ever falls into the wrong hands. If your system date is correct, your web browser will notify you that the certificate is invalid because it has expired, effectively warning you that that website should not be trusted. The validity period of the certificate is sufficiently short to make it most unlikely that the system clock of your computer is accidentally set back to this time.

Question. Are all digital certificates/signatures broken?
Answer. No. When digital certificates and signatures are based on secure cryptographic hash functions, our work yields no reason to doubt their security. Our result only applies when digital certificates are signed using the hash function MD5, which is known to be broken. With our method it is only possible to copy digital signatures based on MD5 between two specially constructed digital certificates. It is not possible to copy digital signatures based on MD5 from digital certificates unless the certificates are specially constructed. Even so, our result shows that MD5 is NOT suited for digital signatures. Certification Authorities should stop using the known broken MD5 and move to the widely available, more secure alternatives, such as SHA-2.

Question. What is the status of collision attacks for SHA-1?
Answer. As far as we know the only group seriously working on collision attacks for SHA-1 is in Graz, Austria, see their website "SHA-1 Collision Search".
Status of the theory: at the Rump Session of Crypto 2007 they estimated the complexity of their attack for collisions with identical initial IHVs to be $<2^{61}$ calls to the compression function. For chosen-prefix collisions they estimated in 2006 a complexity of just below the birthday bound of 2^{80}. Improvements on the latter result are probably possible, but nobody has looked into this.
Status of experiments: Since mid 2007 the people in Graz are running a distributed SHA-1 Collision Search BOINC project, aiming at finding a SHA-1 collision with identical initial IHVs. They estimate (private communication) that now (December 2008, so after 1.5 years) they have covered less that 10% of the search space. So they need your spare cycles.

Question. Suppose that a criminal creates by our method his own rogue Certification Authority that is trusted by all browsers.
Are then only websites with certificates from the CA whose signature he used vulnerable to impersonation attacks?
Are only websites with certificates based on MD5 vulnerable to impersonation attacks?
Answer. No. When a criminal uses redirection attacks in combination with a rogue but trusted CA certificate, all websites are equally vulnerable.

Question. What should websites do that have digital certificates signed with MD5?
Answer. Nothing at this point. Digital certificates legitimately obtained from all CAs can be believed to be secure and trusted, even if they were signed with MD5. Our method required the purchase of a specially crafted digital certificate from a CA and does not affect certificates issued to any other regular website.

Question. Where does the title "MD5 considered harmful today" of this website come from?
Answer. In December 2004, shortly after MD5 collisions were found, Dan Kaminsky came with an exploit for them. His paper is entitled "MD5 to be considered harmful someday", see [K1]. We think this day has come.

Question. When and where will the paper with details on the chosen-prefix collision construction method be published?
Answer. The paper is not yet finished. It will be submitted for publication to a peer reviewed academic journal. Early informal release on the web may be expected at some time in the first half year of 2009. When that happens the place of release will be announced here.

Question. Will source and/or executable code of the chosen-prefix collision construction method be made public?
Answer. For the time being we have no such plans.

Question. How do we know that the CAs still using MD5 haven't already been attacked by somebody else?
Answer. Given the time and resources we needed to construct our proof of concept and the advanced mathematics required to implement our method, it is our opinion that it is unlikely this has happened already. However, because our proof of concept shows that it is possible, there is no 100% guarantee.

Question. What are the known weaknesses in DNS that you refer to in the press release?
Answer. The DNS spoofing vulnerability discovered by Dan Kaminsky in 2008 allows an attacker to redirect requests for any website to a web server controlled by them. For more information, please see [K2] and [OMM]. The SSL protocol used for secure websites is supposed to detect this kind of attack because the attacker would not have a legitimate digital certificate for the hijacked website. However, the MD5 collision attack we discovered allows attackers to create arbitrary trusted digital certificates and allows them to hijack any secure website on the Internet if the user's DNS servers are not patched against the DNS spoofing vulnerability.

Question. How much did your proof of concept research cost?
Answer. It took a few months to design and implement our method, based on a lot of knowledge and skills that we have developed over the last two years. We spent about USD 700 on purchasing test certificates from a CA. The computations needed for our work were done on a cluster of about 200 PlayStation 3 game consoles in the cryptanalytic lab at EPFL.

Question. Why were game consoles used? What other hardware is suitable?
Answer. Game consoles use hardware specialized for the computational needs of the detailed 3D graphics in games. This hardware is also very suited for the basic arithmetic used in cryptographic algorithms and greatly outperforms general purpose computers on brute-force computations. We have found that one PlayStation 3 game console is equivalent to about 40 modern single core processors. The most computationally intensive part of our method required about 3 days of work with over 200 game consoles, which is equivalent to 32 years of computing on a typical desktop computer. Common graphic cards have been used by some for MD5 cryptanalysis as well.

Question. How long would it take for someone else to try and imitate you?
Answer. The basic theory behind our work was published in 2007, but we have made a lot of improvements that are not yet in the literature. We estimate that a team of highly skilled researchers and programmers who have prior experience in this area would need at least a month to duplicate our work. Starting from scratch without prior understanding of the MD5 collision techniques would be far more challenging and would take significantly longer. In addition to cryptographic research and development expertise, access to significant computational resources is needed. Regardless of how long it would take to reproduce our work, MD5 should no longer be used for certificates.

Question. What is the best way to ensure that the attack scenario we developed is not possible in the future?
Answer. Stop using MD5 as soon as possible, and migrate to more secure cryptographic hash functions.

Authors

left to right: Benne, Arjen, Marc, Jake, David, Alex (missing: Dag Arne)
(photo: Alexander Klink)

Alexander Sotirov
Independent security researcher,
New York, USA
http://www.phreedom.org/

Marc Stevens
Centrum Wiskunde & Informatica (CWI),
Amsterdam, The Netherlands
http://homepages.cwi.nl/~stevens/
News at the CWI website: English, Nederlands.

Jacob Appelbaum
Noisebridge, The Tor Project,
San Francisco, USA
http://www.appelbaum.net/

Arjen Lenstra
LACAL - LAboratory for Cryptologic ALgorithms,
EPFL - École Polytechnique Fédérale de Lausanne,
Lausanne, Switzerland
http://people.epfl.ch/arjen.lenstra

David Molnar
Computer Science Division,
University of California at Berkeley,
Berkeley, USA
http://www.cs.berkeley.edu/~dmolnar/

Dag Arne Osvik
LACAL - LAboratory for Cryptologic ALgorithms,
EPFL - École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
http://people.epfl.ch/dagarne.osvik

Benne de Weger
EiPSI - Eindhoven Institute for the Protection of Systems and Information,
TU/e - Eindhoven University of Technology,
Eindhoven, The Netherlands
http://www.win.tue.nl/~bdeweger/

Information

For press and general inquiries, please email md5-collisions@phreedom.org.

There is a video of Marc explaining the work:

For more information about this project, see the following project websites:

- http://www.win.tue.nl/hashclash/rogue-ca/ (this site)
- http://www.phreedom.org/research/rogue-ca/ (Alex's site)
- https://i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom.org/ (demo site, works only if you set your computer's system clock to August 2004)

Links

- The IAIK SHA-1 Collision Search BOINC project, and their "SHA-1 Collision Search" website.
- "The story of Alice and her Boss", Magnus Daum and Stephan Lucks, 2005.
- Peter Gutmann's dumpasn1 program, there's a Windows GUI version and an online version as well. Peter Gutmann's X.509 Style Guide, and a certificate with an MPEG movie inside.
- The "Colliding X.509 Certificates based on MD5-collisions" website, Arjen Lenstra, Benne de Weger, Xiaoyun Wang, March 2005.
- The "Colliding X.509 Certificates for Different Identities" website, Marc Stevens, Arjen Lenstra and Benne de Weger, October 2006.
- The "Chosen-prefix collisions" website, Marc Stevens, Arjen Lenstra and Benne de Weger, February 2007.

- Verisign's response "This morning's MD5 attack - resolved" by Tim Callan.
- Microsoft's Security Advisory (961509): "Research proves feasibility of collision attacks against MD5", and a Microsoft Technet blog item Information regarding MD5 collisions problem by Damian Hasse.

References

[B] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen,
"Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen)",
http://www.bundesnetzagentur.de/media/archive/14953.pdf,
November 2008.

[HK] Shai Halevi and Hugo Krawczyk,
"Strengthening Digital Signatures via Randomized Hashing",
http://www.ee.technion.ac.il/~hugo/rhash/rhash.pdf,
January, 2007.

[HPFS] R. Housley, W. Polk, W. Ford, D. Solo,
"Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile",
IETF RFC 3280,
April 2002,
available from http://www.ietf.org/rfc/rfc3280.txt.

[K1] Dan Kaminsky,
"MD5 to be considered harmful someday"
http://www.doxpara.com/md5_someday.pdf,
December 2004.

[K2] Dan Kaminsky,
"Black Ops 2008: It's the end of the cache as we know it"
http://www.doxpara.com/DMK_BO2K8.ppt,
August 2008.

[LW] Arjen Lenstra and Benne de Weger,
"On the possibility of constructing meaningful hash collisions for public keys",
In: Colin Boyd and Juan Manuel Gonzalez Nieto (editors),
"Information Security and Privacy - Proceedings of ACISP 2005",
volume 3574 of Lecture Notes in Computer Science, pages 267-279,
Springer Verlag, Berlin, 2005.

[OMM] Matthew Olney, Patrick Mullen, Kevin Miklavcic,
"Dan Kaminsky's 2008 DNS Vulnerability",
Snort Whitepaper, available at http://www.snort.org/vrt/docs/white_papers/DNS_Vulnerability.pdf,
July 2008.

[S] Marc Stevens,
"On collisions for MD5",
MSc Thesis, Eindhoven University of Technology,
June 2007,
available from http://www.win.tue.nl/hashclash/.

[SLW] Marc Stevens, Arjen Lenstra and Benne de Weger,
"Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities",
In: Moni Naor (editor), "Advances in Cryptology - EUROCRYPT 2007",
volume 4515 of Lecture Notes in Computer Science, pages 1-22,
Springer Verlag, Berlin, 2007.

[WY] Xiaoyun Wang and Hongbo Yu,
"How to Break MD5 and Other Hash Functions"
In: Ronald Cramer (editor), "Advances in Cryptology - EUROCRYPT 2005",
volume 3494 of Lecture Notes in Computer Science, pages 19-35,
Springer Verlag, Berlin, 2005.

Latest Modification December 31, 2008

Vanity Counter 43936