Cryptographic Hash Functions
Recent Results on Cryptanalysis and their Implications on System Security

Rüdiger Weis[1] and Stefan Lucks[2]

[1] Technical University of Applied Sciences Berlin
[2] University of Mannheim

Abstract. Recently, several severe attacks against cryptographic hash functions where discovered. This includes attacks against MD4 and MD5, two rather old hash functions. Very unfortunately, MD5 is still widely used in practice in many software integrity schemes, including, but not limited to the most popular Linux packet formats. This may pose a serious security problem. In this paper we discuss the current state of research on hash functions and some first-aid workarounds. We briefely discuss the trouble "Trusted" Computting has got into, by trusting the SHA-1 hash function which now has been broken.

A spectre is haunting IT-security – the spectre of hash function cryptanalysis. Hash functions, the cryptographic workhorses for data authenticity and integrity, are broken one by one. There are three kinds of attacks:

– structural weaknesses,
– analytical weaknesses, and
– exploits of both.

In the current paper, we briefly explain how current hash functions work, which attacks cryptographic hash functions should defend against, and what had gone wrong now (Section 1). More specifically, we describe how to exploit the currently known weaknesses, and discuss their practical consequences (Section 2), including the trouble for "Trusted" Computing, due to the "hardwired" use of SHA-1, and the risks of still using MD5 (3). We further describe the immediate consequences for the "users" of hash functions (Section 4) and an outlook on future development (Section 5). Finally, we give a short summary (Section 6).

1 Hash Functions and Attacks

A cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ maps an infinite set of inputs to the finite set of n-bit hash values. Informally, a real hash function H should behave like an ideal one (i.e., like a random oracle). This would not be useful for a formal definition, though. Instead, one considers somewhat simpler security goals for $H : \{0,1\}^* \rightarrow \{0,1\}^n$.

While collisions (inputs $X \neq Y$ with $H(X) = H(Y)$) necessarily exist, as there are more inputs than outputs, a hash function should be *collision resistant*: given H, it should be infeasible for an adversary to *actually find* any collisions.

Additionally, a hash function should resist preimage and 2nd preimage attacks: Given Z, it should be infeasible for an adversary to *find any* X with $H(X) = Z$, and given Y, it should be infeasible to find Y with $H(X) = H(Y)$.

Classes of Attacks

A bit more formally, we distinguish the following classes of attacks:

Collision attack: Find two messages $M \neq M'$ with $H(M) = H(M')$.

Preimage attack: Given a random value $Y \in \{0,1\}^n$, find a message M with $H(M) = Y$.

2nd preimage attack: Given a message M, find a message $M' \neq M$ with $H(M) = H(M')$.

K-collision attack for $K \geq 2$**:** Find K different messages M^i, with $H(M^1) = \cdots = H(M^K)$.

K-way (2nd) preimage attack for $K \geq 1$**:** Given Y (or M with $H(M) = Y$), find K different messages M^i, with $H(M^i) = Y$ (and $M^i \neq M$).

The last two are natural generalisations of the more ‘traditional” collision attacks (2-collisions) and ‘1-way” preimage and 2nd preimage attacks. We compare the resistance of a given hash function against these attacks with the resistance a random oracle would provide:

Fact: *Model $H : \{0,1\}^* \rightarrow \{0,1\}^n$ as a random oracle. Finding a K-collision for H takes time $\approx 2^{(K-1) \cdot n/K}$, and finding a K-way preimage or a K-way 2nd preimage for H takes time $\approx K \cdot 2^n$.*

Special case: Finding a "normal" (2-)collision for a random oracle H takes time $\approx 2^{n/2}$ and finding a preimage or a 2nd preimage takes time $\approx 2^{2n}$.

1.1 Hash Functions and Digital Signatures

Why is it so important for a hash function to be 2nd preimage resistant and even collision resistant? Why is it so dangerous if a hash function does not defend against finding 2nd preimages or collisions?

As a motivating example, consider digital signature schemes. A message M has to be signed. The signature scheme is broken, if the adversary can forge a signature for another message M'. In practise, cryptographic signature schemes follow the hash-and-sign paradigm: Instead of signing the (potentially very long) message M, one actually computes a short "fingerprint" $H(M)$ and signs X.

Given a signature for some message M, and adversary able to compute a 2nd preimage M' with $H(M) = H(M')$ is able to forge a signature for M', as the signature for M is valid as well for M'.

So much about the importance of 2nd preimage resistance. But why is collision resistance a big deal?

2nd preimage resistance is fine – if we always trust the party who chooses the message M to be signed. But if we don't, we need collision resistance! Otherwise, the adversary could compute the collision $M \neq M'$, sign M or present M to be signed, and later claim the signature on M'.

1.2 Structural Weaknesses of Today's Hash Functions

The design of today's cryptographic hash functions, including but not limited to MD4, MD5, SHA-0, SHA-1, SHA-2 family, RIPEMD, RIPEMD-160, Whirlpool, Tiger, ... ubiquitously follows the MD structure by Merkle and Damgård (MD) [19, 5] for hash functions.

As a hash function $H : \{0,1\}^* \rightarrow \{0,1\}^n$ has to handle arbitrarily long inputs[1], the idea is to design an internal compression function $C : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$ with fixed input size, to split the given message M into m-bit blocks $M[1], \ldots, M[L]$ (with some padding in

[1] In practise, the input size of of all these hash function is limited to some fixed upper bound. As this bound is at least 2^{64} bit, this is hardly a practical limitation.

the final block $M[L]$), and to compute the hash by straightforwardly iterating the compression C, as shown in Figure 1.

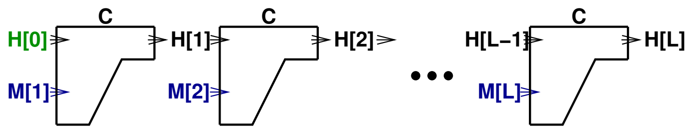


Fig. 1. The iterated Merkle-Damgård (MD) structure for hash functions

As shown in [19, 5], the hash function h is collision resistant, if the compression function is. This is the strong point of the MD structure.

Unfortunately, there is a weak point, as well: the MD structure is rather *failure-unfriendly* – if computing a compression function collision is somehow feasible, all the bets are off. In fact, it had turned out that these hash functions failed worse than expected.

Length Extension Attacks

The *length extension* property is a well-known generic weakness of the MD hash (see e.g. [8, Section 6.3.1]):
Given

$$H = H(M),$$

it is straightforward to compute M' and H', such that

$$H' = H(M||M')$$

– even *for unknown* M. (For the correct padding of the final message block, one needs to know the length of M.)

2nd Collision Attacks

Once you have found a collision, it is trivial to produce more such collisions:
If

$$H(M) = H(N)$$

for any two messages M, N, then

$$H(M||S) = H(N||S)$$

for all $S \in \{0, 1\}^n$.

Multiple Collisions

Finding *multiple collisions* should be way more expensive than finding plain (2-)collisions – but Joux [15] disproved this for the MD design. He described a generic attack to find 2^k-Collisions for a MD hash H in time $\approx k \cdot 2^{n/2}$, instead of $\Omega(2^{n(2^k-1)/2^k})$:
For i in $1 \ldots, k$:

→ Find a local collision $M_i^0 \neq M_i^1$ with

$$H_i = C(H_{i-1}, M_i^0) = C(H_{i-1}, M_i^1).$$

All the 2^k messages

$$(M_1^0, \ldots, M_k^0), (M_1^0, \ldots, M_{k-1}^0, M_k^1), \ldots, (M_1^1, \ldots, M_k^1)$$

hash to the same value H_k. (Note that all messages are of the same (not too large) size of k blocks.)

Using Joux' multicollision attack as a tool

As Joux pointed out [15], the adversary's ability to find large K-collisions opens the door for further attacks. Below, we sketchily describe them.

Attacking cascaded hash functions.

Let a hash $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be defined as

$$H(M) = H^1(M)||H^2(M)$$

with two independent n-bit hashes H^1 and H^2.

If both H^1 and H^2 are independently defined as random oracles, then finding collisions for H takes time 2^n.

If, however, either is constructed as a MD hash, finding a collision for H only takes time

$$\approx (n/2) * 2^{n/2}.$$

W.l.o.g., let H^1 be the MD hash:

→ Find $2^{(n/2)}$-collisions for H^1 (in $(n/2) * 2^{n/2}$ units of time). Statistically, one such collision also collides for H^2 (and thus for H).

Finding multiple (2nd) preimages.

Given a target $Y \in \{0,1\}^n$, the attack proceeds as follows:

$\rightarrow$ Generate 2^k colliding k-block messages $M^1, \ldots, M^{2^k}$ with

$$H_k = H(M^1) = \cdots = H(M^{2^k}).$$

$\rightarrow$ Find a message chunk M_{k+1}, such that $C(H_k, M_{k+1}) = Y$.

This provides a 2^k-way preimage. The first step takes time $k * 2^{n/2}$, which is marginal, compared to the second step. This takes about the time for a single preimage attack, i.e., $\approx 2^n$. For a 2nd preimage message attack with the target message M, just set $Y := H(M)$.

More Structural Weaknesses

Our list of structural weaknesses is not at all complete – other attacks have been described, e.g., by Kelsey and Schneier [23] and Kelsey and Kohno [21].

1.3 Analytical Attacks

The design of cryptographic primitive operations, such as a hash function, depends on both art and science. Studying the internal structure of a hash function is science. Finding a good design for the main building block – namely the compression function – is more art than science. Some would say, it is like black magic.

If a compression function of a hash function is as strong as it should be, attacks such as Joux' would need more than $2^{n/2}$ units of time – impractical for large n (e.g., $n > 160$ for today; perhaps $n > 200$ for the next 25 years). Thus, for sufficiently large n the attacks would be entirely theoretical – as would the exploits, we describe below.

As it turns out, however, the designers of modern hash functions have not been such good "black magicians" as they have hoped for. A variety of analytical attacks against hash functions – i.e., of attacks against the compression functions – have been found recently [34–38, 41, 3]. As finding collisions is the "simplest" task for the attacker, most of these attacks are collision attacks, but even 2nd preimage attacks have been presented [41].

SHA-1 is probably the cryptographic hash function with the largest usage base – at least in the context of digital signatures. Until early 2005, no attacks against SHA-1 have been known (though some attacks against weakened variants of SHA-1 have been studied). In February 2005, however, another breakthrough in the analysis of hash functions has been announced: A collision attack against SHA-1.

There was no need to panic, yet: with its expected running time of 2^{69} hash operations, the attack would only be feasible for an extremely well-funded and highly motivated adversary. The entire attack has been published in August 2005 [37]. At the same conference, an improved collision attack with an expected running time of 2^{63} hash operations has been claimed [38]. If this attack is confirmed, this would be *very bad news* for anyone still using SHA-1: finding collisions for SHA-1 would be within the practical range for a distributed parallel collision search.

2 Exploits – how Practical are Collision Attacks?

With very few exceptions – namely for attacking MD4 – the attacks against hash functions are restricted to finding collisions, and these collisions are more or less random. For cryptographers, these results are exciting - but many so-called "practitioners" turned them down as *practically irrelevant*. The attacks seem to lack two important qualities

– They only find collisions, i.e., the algorithm produces two messages M and M' with $H(M) = H(M')$. Practical attack scenarios would seem to require 2nd preimages, i.e., the algorithm receives M as its input and produces M' with $H(M) = H(M')$.
– The adversary has no control over M and M' – with overwhelming probability both M and M' look like random garbage. As it may appear, typical attack scenarios assume the adversary to find a *meaningful* message M' for a given (harmless) message M. [2]

[2] E.g., consider a document M. being digitally signed by someone. A 2nd preimage attacker may replace M by M'. If the hash function H is used in the signature

As you cannot exercise control over colliding messages M and M', these collisions are theoretically interesting but harmless, right? In the past year, we have met quite a few people who indeed thought so. We argue, however, that **this believe is dangerously wrong!**

2.1 Making the Attack Practical

In practise, all collision finding algorithms

– can take an initial value $H[0]$ as their input

and then compute two colliding messages M and M' with the following two properties:

– M and M' are both of the same length, and
– M and M' are short (e.g., 1024 bit).

Now the Merkle/Damgård construction for hash functions, see Figure 1, allows us to easily construct longer messages $(S||M||T)$ and $(S||M'||T)$ which also collide. Assume S to consist of i blocks $S[1], \ldots, S[i]$. Compute the intermediate hash $H[i]$. Now call the collision finding algorithm, using $H[i]$ as its input, instead of the original initial value $H[0]$. The result are two (partial) messages M and M'. By the properties of the Merkle/Damgård construction, we now have

$$H(S||M) = H(S||M').$$

For any postfix T, we further get

$$H(S||M||T) = H(S||M'||T).$$

So, this is the order of doing things:

1. Choose the prefix S, and compute the input $H[i]$ for the collision finding algorithm.
2. Ask the collision finding algorithm for M and M'.
3. Choose the postfix T.

scheme and $H(M) = H(M')$, the signature for M would as well be valid for M', i.e., we did forge a signature for M'. To benefit from the forgery, however, the attacker would likely need a meaningful M', instead of a random sequence of bits.

How can this be useful for the attacker?

Consider writing some executable. The prefix S ends with some forward-jump into T. The postfix T references some constants, which are conveniently located at the address positions where M or M' are stored. Accordingly, the executable $(S||M||T)$ may behave entirely different from its hash collision $(S||M'||T)$ – and when writing T, its author already knows M and M' and can completely determine what the executables are doing. E.g., running $(S||M||T)$ may serve some reasonable purpose for you, while running $(S||M'||T)$ may open a back door to your the system.

2.2 Known Exploits

Early after the first collisions for MD5 have been found, Kaminski [16] and Mikle [20] played games with executables. Mikle's executables were self-extracting archives, extracting different stuff. One of Kaminski's executables was harmless, while the other one harmfully opened a backdoor.

Lenstra, Wang and de Weger [17, 18] where the first to demonstrate a similar trick with non-executables: they constructed colliding X.509 certificates.

Later, Daum and Lucks demonstrated colliding MD5 PostScript documents [6], using the PostScript if-then-else construct. Similar attacks are even possible for documents description languages without if-then-else, such as PDF or TIFF. This has been shown by Gebhardt, Illies, and Schindler [9, 10], who played tricks with colour tables and similar document-information.

Thus, the weaknesses of the MD hash function structure allows to turn the ability of finding "random" collisions into very well-targeted and meaningful collisions.

2.3 MC Fuzzing

The ability of generating big amounts of colliding messages can be used as a powerful tool to mount Fuzzing Attacks. The main idea of Fuzzing is to test applications by sending random data. This had a long history in software testing but is also in use in automated attacking tools.

An attacker could get a signature for a harmless packet having huge amounts of 'signed' random packets ready for a Fuzzing test.

3 Practical Schemes in Trouble

As should have become clear from the above, given the current attack algorithms it would be easy to generate, e.g., two "meaningful" boot images which collide, i.e., have the same hash value. Also, it would be infeasible. to distinguish malicious packets containing random data from packets which are used, e.g., in a cryptographic context (i.e., are encrypted).

3.1 SHA-1 collisions and "Trusted" Computing?

The central building block of the Trusted Computing Group proposals is the so called Trusted Platform Module (TPM) (dubbed as "Fritz" Chip). This is a bit like a smartcard being hardwired into a computer. Future implementations may even integrate the TPM into the main processor (see also Intel LaGrande).

Ignoring warnings in the past few years (e.g. [39, 40]) the TPM is using SHA-1 for allmost all operations. In other words, SHA-1 is "hardwired" into the current "Trusted" Computing Standards. As it seems, the authors of the "Trusted" Computing standards did unconditionally trust into the security of the SHA-1 hash function. Has it been impossbile for them to even imagine that that SHA-1 could be broken?

In any case, after the first cryptanalytic resuls on the full SHA-1, and also due to the general weaknesses of the MD structure, the consequences for "Trusted" Computing can be severe – several security-critical building blocks appear to be **compromised**, namely:

– integrity measurements (using SHA-1),
– digital signatures (using SHA-1), and
– PKIs (using SHA-1).

A huge practical problem is also the use of SHA-1 for "Trusted" secure booting. Since the generation of SHA-1 collision is practically feasible, checking the boot sequence can not avoid 'Evil Twins Attacks'.

A remedy would be to replace SHA-1 by another (hopefully secure) hash function. The standards would have to be rewritten, all the TPMs out there today would become obsolete, ... at the end, such a change would certainly be possible, but quite costly, both for the TCG and for its customers, the early adopters of "Trusted" Computing.

3.2 MD5 is still widely used

Even today (April 2006) many applications and packet format continue to use MD5 for security-critical authenticity and integrity mechanisms. This gives the authors of the current paper the creeps. We found MD5 being used in, e.g.,

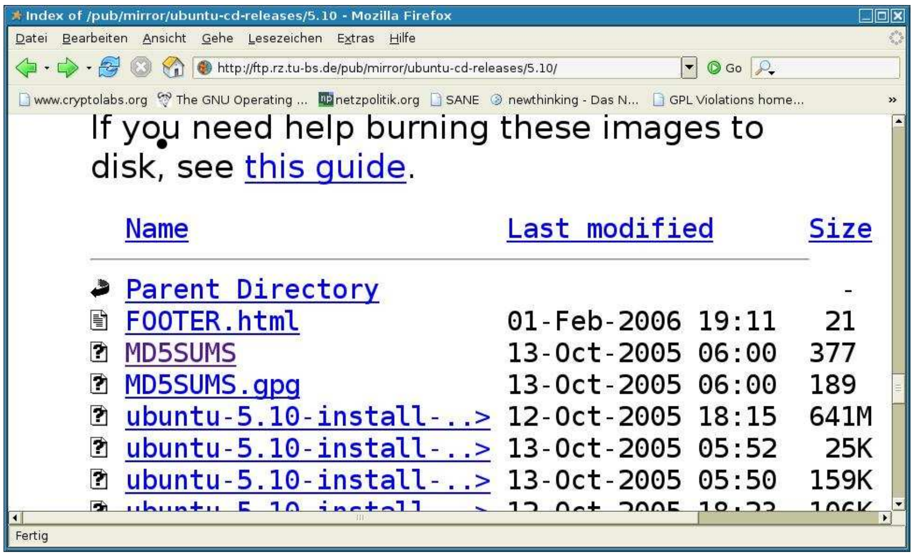
- the Debian Packet Format
- the RPM Packet Format
- Open BSD
- mySQL
- old PGP Keys
- CD checksums
- ...

Recent results on the analysis of MD5 reduce the effort to find collisions to a few minutes on an old notebook. Thus, collision and multi-collision attacks on MD5 are feasible and practical. As discussed earlier the ability to produce multiple colliding messages with a negligible effort can be used to generate Fuzzing packets.

Hotfix Security improvements

Some packet formats sign the a MD5 list of files using gpg (Figure 2). This technique does neither help against MD5 nor SHA collisions. It would be a big security improvement to sign the whole packet (including the MD5 list, if necessary). Our recommendation:

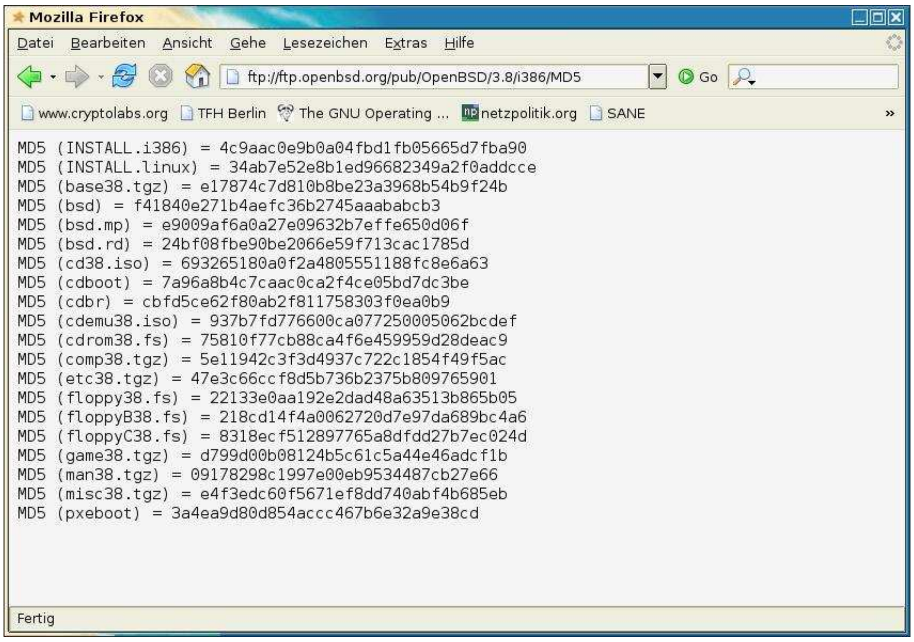
- Migrate directly from MD5 to SHA-2.
- Add SHA-2 hashes to existing MD5 lists.


Index of /pub/mirror/ubuntu-cd-releases/5.10 - Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
http://ftp.rz.tu-bs.de/pub/mirror/ubuntu-cd-releases/5.10/
Go
www.cryptolabs.org
The GNU Operating ...
netzpolitik.org
SANE
newthinking - Das N...
GPL Violations home...
Fertig

If you need help burning these images to disk, see this guide.

```
    Name                              Last modified      Size
  ------------------------------------------------------
    Parent Directory                                      -
    FOOTER.html                       01-Feb-2006 19:11   21
    MD5SUMS                           13-Oct-2005 06:00  377
    MD5SUMS.gpg                       13-Oct-2005 06:00  189
    ubuntu-5.10-install-..>           12-Oct-2005 18:15  641M
    ubuntu-5.10-install-..>           13-Oct-2005 05:52   25K
    ubuntu-5.10-install-..>           13-Oct-2005 05:50  159K
```

Fig. 2. MD5 signed with gpg using SHA-1


Mozilla Firefox
Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe
ftp://ftp.openbsd.org/pub/OpenBSD/3.8/i386/MD5
Go
www.cryptolabs.org
TFH Berlin
The GNU Operating ...
netzpolitik.org
SANE
Fertig

```
MD5 (INSTALL.i386) = 4c9aac0e9b0a04fbd1fb05665d7fba90
MD5 (INSTALL.linux) = 34ab7e52e8b1ed96682349a2f0addcce
MD5 (base38.tgz) = e17874c7d810b8be23a3968b54b9f24b
MD5 (bsd) = f41840e271b4aefc36b2745aaababcb3
MD5 (bsd.mp) = e9009af6a0a27e09632b7effe650d06f
MD5 (bsd.rd) = 24bf08fbe90be2066e59f713cac1785d
MD5 (cd38.iso) = 693265180a0f2a4805551188fc8e6a63
MD5 (cdboot) = 7a96a8b4c7caac0ca2f4ce05bd7dc3be
MD5 (cdbr) = cbfd5ce62f80ab2f811758303f0ea0b9
MD5 (cdemu38.iso) = 937b7fd776600ca077250005062bcdef
MD5 (cdrom38.fs) = 75810f77cb88ca4f6e459959d28deac9
MD5 (comp38.tgz) = 5e11942c3f3d4937c722c1854f49f5ac
MD5 (etc38.tgz) = 47e3c66ccf8d5b736b2375b809765901
MD5 (floppy38.fs) = 22133e0aa192e2dad48a63513b865b05
MD5 (floppyB38.fs) = 218cd14f4a0062720d7e97da689bc4a6
MD5 (floppyC38.fs) = 8318ecf512897765a8dfdd27b7ec024d
MD5 (game38.tgz) = d799d00b08124b5c61c5a44e46adcf1b
MD5 (man38.tgz) = 09178298c1997e00eb9534487cb27e66
MD5 (misc38.tgz) = e4f3edc60f5671ef8dd740abf4b685eb
MD5 (pxeboot) = 3a4ea9d80d854accc467b6e32a9e38cd
```

Fig. 3. OpenBSD MD5 checksums

4 What should be done?

From a cryptographer's standpoint, SHA-1 should be replaced as soon as possible.

4.1 Restrict the Life-Time of Algorithm Certificates

In the context of the German Signature law (similar laws exist in other European countries), cryptographic algorithms, including hash functions, need an official certification. Usually, this is granted for a six-year period. As the current state of knowledge about hash functions is so much at flux, the period should be much shorter, no more than three years.

4.2 Don't use MD4 and MD5 any more!

In the context of digital signatures, the usage of MD5 (or, even worse, MD4) can no longer be justified. As William Burr from the us-american National Institute of Standards and Technology (NIST) pointed out in February 2005:

> "If by some chance you are still using MD5 in certificates or for digital signatures, you should stop." [29]

It may even be prudent to re-consider the validity of older MD5-based (or MD4-based) digital signatures, if these have not already been re-signed using a more up-to-date hash function. Even in security contexts without digital signatures, the usage of MD4 and MD5 should be avoided.

4.3 Migrate away from SHA-1, soon!

Currently, exploiting the SHA-1 attacks would only be feasible for well-funded and motivated adversaries. There is no need to panic, but there are extremely good good reasons for migration to other hash functions soon, wherever possible.

As finding collisions for SHA-1 seems to be within practical range, the usage of SHA-1 for new digital signatures should be stopped as soon as possible. As we will explain below, some alternative choices exist. In contrast to MD4/MD5, the security of signatures already generated does not need to be questioned – assuming no want-to-be forgers new details of the attack before the public did know.

Note that the news about SHA-1 pose problems for the Trusted Computing Group (TCG), with its specifications for a "trusted computing platform", because the current specifications (including the recent version 1.2) do not allow an alternative to SHA-1.

4.4 Use Available Alternatives : SHA-2 and RIPEMD-160

The recent NIST-standard SHA-2 [28] defines several functions with an extended hash size from 224 bit (SHA-224) to 512 bit (SHA-512). Unfortunately, the security of these functions have not been studied much, except for [11], who provided the following results:

- The good news: The SHA-2 hash functions are quite resistant against attack techniques which have been used to attack MD4, MD5, and SHA-1.
- The bad news: The design of SHA-2 is fragile: even marginal modifications of the hash functions turned out to generate major weaknesses.

The SHA-2 functions are a possible short term alternative to SHA-1. No attack against the un-modified SHA-2 functions have been found, so far. Another alternative is the European hash function RIPEMD-160 [7]. As with SHA-2, no attack against RIPEMD-160 is known.

Note that both the SHA-2 family and RIPEMD-160 follow the MD design and, accordingly, suffer from its structural weaknesses.

SHA-2 Drop-In-Replacement

Considering the actual (April 2006) cryptanalytic analyses using 160 bit from SHA-2 output as drop-in replacement for SHA-1 seems to provide a security improvement.

4.5 Randomised Hash

One apparent option to fix the problem with the lack of collision resistance of a hash function H is to randomise the message:

$\rightarrow$ Choose a random string R (of some pre-determined length),
$\rightarrow$ Use $(R, H(R||M))$ as the randomised hash output.[3]

The rationale is that it should be harder to find M, M' such that

$$H(R||M) = H(R||M')$$

for random R with some significant probability, than to find M, M' with

$$H(M) = H(M')$$

– assuming the choice of the random string R is outside the scope of the attacker.

This tweak may actually be useful for some applications, but entirely useless for others: If the party choosing R would not benefit from a collision, but others would, randomised hashing can help. But *if the party who chooses R would benefit from a collision, randomised hashing does not provide any defence.*

5 What is in Store for the Future?

5.1 Medium Term Alternatives

As we mentioned above, RIPEMD-160 and the SHA-2 hash functions also are members of the MD4 family of hash functions. The predecessor RIPEMD (no "-number") of RIPEMD-160 has also been broken recently [34].

Thus, it seems better to avoid the MD4 family altogether ([31, 32]), and even modify the Merkle-Damgård design principle [24–26].

Unfortunately, the ideas behind MD4 have been so influential that all practical hash functions known today have followed them. Hardly any expert did bother to propose or analyse the security of alternative designs. Thus we do not recommend any non-descendant of MD4 for immediate use.

Tiger [1] and Whirlpool [2] are two alternative hash functions. Unfortunately, they also follow the MD construction and thus suffer from the abovely described structural weaknesses. But at least, Tiger and Whirlpool employ compression functions very different

[3] E.g., a signature application would sign $(R, H(R||M))$ instead of $H(M)$.

from those used in the MD4 family. Thus, one may hope that Tiger and Whirlpool escape from the analytical attacks which so many of the hash functions from the MD4 family are vulnerable against.

As, the security of Tiger and Whirlpool have hardly been analysed by anyone, our advice is not to use them *now*. But as Tiger and Whirlpool have started to receive a lot of attention, they might become decent alternatives to RIPEMD-160 and the SHA-2 family in a few years.

As far as we know, there are no published analytical results on Whirlpool so far. A first and still preliminary attack on Tiger has been published recently [22], raising some questions about the security of Tiger. But note that [22] did only break a weakened version of Tiger – the security of the full Tiger still remains open.

5.2 In the Long Term – Searching for a Solution

To summarise: the current state of available secure cryptographic hash functions is poor, if not lousy. The hash functions in practical usage have been broken, the unbroken short term alternatives are rather similar to the broken hash functions, and some still unstudied proposals are waiting to attract the attention of cryptanalysts.

Clearly, cryptographers must improve the state of analysing and developing secure hash functions. Two obvious goals for the science of cryptography are

– to analyse the remaining hash functions, and
– to study new structures as potential alternatives to the MD structure (as done, e.g., in [24, 25, 4, 26, 14]).

This would be a preparation for a long term remedy, such as, e.g., a public competition for the design and analysis of new hash functions, similar to the AES competition for block ciphers which ended in standardising one of the block cipher candidates as the "Advanced Encryption Standard" (AES). Such a competition would necessarily take a few years. Until then, one should use the short and mid term alternatives we described.

6 Summary – What does that mean for YOU?

From a "user's point of view" (the "users" of a hash function are system architects, developers, system administrators, . . .), the current situation is extremely unsatisfactory: *The current state of cryptography does not provide trustworthy and secure hash functions "users" actually need!* It will take quite a few years before this situation is likely to improve (assuming it will improve at all). Accordingly, you should

1. be aware of the fragility of current hash functions (e.g. [12]),
2. avoid the usage of hash cryptographic hash functions, when other options are available[4], and
3. wherever you need to use hash functions, keep an eye on the continuing process of discovering vulnerabilities (attacks) and developing new hash functions. Be prepared to change the hash function you are using!

Conclusion

Cryptographic hash functions are a valuable tool to ensure the integrity and authenticity of executables, libraries, software packages, . . . – but if a hash functions does not satisfy its security goal of being collision resistant, security is at risk, not just theoretically, but in a very practical sense.

All hash function based mechanisms to detect the manipulation of software must be considered insecure if the hash function is broken!

Specifically, we find the fact that the broken hash function MD5 is still widely used to defend software against manipulatins more than alarming! Namely, the Free Sofware community should act now. Our recommendation:

Wherever MD5 is still used to detect the manipulation of data or software, it must be replaced urgently!

This paper suggests some alternatives to MD5 and other broken hash functions.

[4] E.g., for message authentication, cryptography provides hash function based solutions such as the HMAC [13], and block cipher based solutions, such as the OMAC/CMAC [30].

References

1. R. Anderson, E. Biham. Tiger: A Fast New Hash Function. Fast Software Encryption, FSE'96, LNCS 1039, Springer-Verlag, 1996.
2. P. Barreto, V. Rijmen, The Whirlpool Hashing Function, First open NESSIE Workshop, Leuven, Belgium, 13-14 November 2000.
3. E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, W. Jalby. Collisions of SHA-0 and reduced SHA-1. Eurocrypt 2005, LNCS 3494, 36–57.
4. J. Coron, Y. Dodis, C. Malinaud, P. Punyia. Merkle-Damgård revisited: how to construct a hash function. Crypto 2005.
5. I. Damgård. A design principle for hash functions. Crypto 89, LNCS 435, 416–427.
6. M. Daum, S. Lucks. The story of Alice and her boss. Presentation at rump session of Eurocrypt 05. `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions/`
7. H. Dobbertin, A. Bosselaers, B. Preneel. RIPEMD-160, a strengthened version of RIPEMD. Fast Software Encryption 1996, LNCS 1039, pp. 71-82.
8. N. Ferguson, B. Schneier. Practical Cryptography. Wiley Publishing, 2003.
9. M. Gebhardt, G. Illies, W. Schindler. Reusable hash collisions for special file formats, presentation at Rump session of Crypto.
10. M. Gebhardt, G. Illies, W. Schindler. A note on the pracitcal value of single hash collisions for special file formats. Jana Dittmann (ed.), proceedings of Sicherheit 2006, Lecture Notes in Informatics LNI Volume P-77, ISBN 3-88579-171-4, ISSN 1617-5486.
11. H. Gilbert, H. Handschuh. Security Analysis of SHA-256 and Sisters. Selected Areas in Cryptography (SAC '03), LNCS 3006, pp. 175-193.
12. The Hashing Function Lounge. `http://paginas.terra.com.br/informatica/paulobarreto/hflounge.html`
13. The HMAC Papers. `http://www-cse.ucsd.edu/ mihir/papers/hmac.html`
14. S. Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. FSE 2006.
15. A. Joux. Multicollisions in iterated hash functions, application to cascaded constructions. Crypto 04, LNCS 3152, pp. 306-316.
16. D. Kaminski. MD5 to be considered harmful someday. Dec. 2004. `http://www.doxpara.com/md5_someday.pdf`
17. A. Lenstra, B. de Weger. On the possibility of constructing meaningful hash collisions for public keys. ACISP 2005.
18. A. Lenstra, B. de Weger, X. Wang. Colliding X.509 certificates. Cryptology eprint archive report 2005/067. `http://eprint.iacr.org/2005/067/`
19. R. Merkle. One-way hash functions and DES. Crypto 89, LNCS 435, 428–446.
20. O. Mikle. Practical attacks on digital signatures using MD5 message digest. Cryptology eprint archive report 2004/356. `http://eprint.iacr.org/2004/356/`
21. J. Kelsey, T. Kohno. Herding Hash Functions and the Nostradamus Attack. Eurocrypt 2006.
22. J. Kelsey, S. Lucks. Collisions and Near-Collisions for Reduced-Round Tiger. Fast Software Encryption (FSE) 2006.

23. J. Kelsey, B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2n Work. Eurocrypt 2005.
24. S. Lucks. Design Principles for Iterated Hash Functions. Cryptology ePrint Archive: Report 2004/253.
25. S. Lucks. A Failure-Friendly Design Principle for Hash Functions. Asiacrypt 2005.
26. M. Nandi, W. Lee, K. Sakurai, S. Lee. Security analysis of a 2/3-rate double length compression function in the black box model. FSE 2005, LNCS 3557, 243–254.
27. National Institute of Standards and Technology (NIST), FIPS180-1, SECURE HASH STANDARD,
28. NIST, "FIPS 180-2: Secure Hash Standard (SHS)", August 2002 (change notice: February 2004).
29. F. Olsen. NIST moves to stronger hashing, Federal Computer Week, Feb. 7, 2005.
30. The OMAC page. `http://crypt.cis.ibaraki.ac.jp/omac/omac.html`
31. R. Weis. Cryptographic Protocols and Algorithms for Distributed Multimedia Systems, PhD Thesis, University of Mannheim, 2000.
32. R. Weis. Hash Problems, CCC Datenschleuder, 2005.
33. R. Weis, S. Lucks. Hash Funktionen gebrochen,. Datenschutz und Datensicherheit, Dud 2005.
34. X. Wang, X. Lai, D. Feng, H. Cheng, X. Yu. Cryptoanalyisis of the hash functions MD4 and RIPEMD. Eurocrypt 2005, LNCS 3494, 1–18.
35. X. Wang, H. Yu. How to break MD5 and other hash functions. Eurocrypt 2005, LNCS 3494, 19–35.
36. X. Wang, H. Yu, Y. L. Yin. Efficient collision search attacks on SHA0. Crypto 2005.
37. X. Wang, Y. L. Yin, H. Yu. Finding collisions in the full SHA1. Crypto 2005.
38. X. Wang, A. Yao, F. Yao. New Collision Search for SHA-1. Presentation at rump session of Crypto 2005 (communicated by A. Shamir).
39. R. Weis, A. Bogk. TCPA - resistance is futile?, Chaos Communication Congress, Berlin, 2002.
40. R. Weis, S. Lucks, A. Bogk. TCG 1.2 - fair play with the 'Fritz' chip?, SANE2004, Amsterdam 2004.
41. H. Yu, G. Wang, G. Zhang, X. Wang. The Second-Preimage Attack on MD4. CANS 2005.