TECHNICAL FEATURE

HASH WOES

Morton Swimmer and Jonathan A. Poritz
IBM Research GmbH

In a rump session of the August 2004 Crypto conference, where attendees have the chance to give informal (non-refereed) presentations of works in progress, a group of Chinese researchers demonstrated flaws in a whole set of hash functions and the entire crypto community was abuzz. In this article, we will clarify the situation and draw lessons from this incident.

First, we need a little background.

THE NAUGHTY BITS

A hash function h is an algorithm which maps a message (bit string) x of arbitrary length to a digest $h(x)$ of a fixed length – a property we call *compression.* For obvious reasons of practicality, the digest $h(x)$ must be easy to compute for any message x.

One example is the common CRC-32 function. In non-malicious environments, this compression alone can be useful – for example to detect transmission errors on a noisy channel – but in security applications we often use the hash digest as part of an authentication or other cryptographic protocol.

A trivial (but very widespread) kind of authentication using hash functions is the practice, within the open source community, of multiply-posting 'md5 checksums' of software releases, so that a prospective downloader can compare the checksum (actually the hash digest) of the binary which they actually download to the value they have found on various public websites. (Hence a hacker who wishes to Trojanize a software package would have to get his version onto the public servers and *also* get the corresponding hashes to all sites which post these authentication hash values).

More sophisticated uses of hash functions include various (standardized!) digital signature schemes, as well as a cute technique to make what are called 'non-interactive zero-knowledge proofs of knowledge', which are essentially certificates which prove that the issuer has certain knowledge, without revealing all of the details of that knowledge.

CRITERIA

For these security applications, one needs more robust hash functions which satisfy further properties. We will describe the three most common criteria from [1]. (There are other, more elaborate, criteria we could have used, but such precision is not needed for our discussion here.)

For an unkeyed cryptographic hash function h, we generally require one (or all) of the following characteristics:

PRE: *Pre-image resistance* means that, for essentially all given digests y, it is computationally infeasible to find any input x which hashes to that value (so $y=h(x)$). This means we cannot find an inverse function to h that is computationally feasible.

2PRE: *Second-pre-image* resistance means that it is computationally infeasible to find a second input $x' \neq x$ which has the same output $h(x) = h(x')$, given that we know x (and therefore the output $h(x)$).

COLL: *Collision resistance* means that it is computationally infeasible to find distinct inputs $x \neq x'$ that hash to the same outputs, i.e. $h(x) = h(x')$.

Typically, we assume 'computationally infeasible' to mean no better than the brute-force approach, although in formal cryptography this relates to the idea of 'polynomial-time' algorithms (or, rather, algorithms which are *not* polynomial-time).

A candidate hash function h which fails to satisfy PRE or 2PRE is not likely to be useful in security applications: for example, both such failures allow an obvious denial-of-service attack on the simple authentication mentioned above, while failure of PRE usually allows digital signatures using h to be completely forged.

COLL is rather more subtle. If, for example, we have found colliding inputs x and x', and we can get an automated signer to sign $h(x)$, then we can later present the forged x' instead as the signed message; of course, this results merely in a denial-of-service attack (again), unless we have fine control over the collision production.

In fact, there will always be collisions, due to the compression factor of the hash function. What we require of a secure hash is not that there are no collisions, but that it is not possible to generate many collisions with a reasonable amount of computational power.

HISTORY

Modern cryptographic hash history starts with Ron L. Rivest's creation of the MD4 hash algorithm in 1990, which was meant to be an improvement on the slow MD2 algorithm. However, very soon it was suspected that MD4

was not secure enough and in 1992, Rivest supplanted it with MD5, which contained an extra round in the three rounds of the MD4 compression function and changed the functions in some of the rounds. This and other modifications were meant to make the algorithm less symmetric and therefore more secure.

Soon after, MD5 was extended to become HAVEL. Likewise RIPEMD, which appeared a few years later, was also based on MD4.

MD4 BROKEN

Apparently, the suspicions surrounding MD4 were well founded. In 1992, collisions in some parts of the MD4 algorithm were found and by 1995, Hans Dobbertin had found a method of producing meaningful collisions in just a few seconds using the computers of the day [2]. By 1998, Dobbertin had found a way of inverting a reduced strength version of MD4. This means that MD4 completely fails COLL, and its status for PRE and 2PRE is considered essentially broken.

Meanwhile, MD5 was under attack as well. After some successful attacks against the compression function in MD5, Dobbertin extended his MD4 attack to MD5 and was able to produce collisions by modifying the algorithm very slightly.

RIPEMD was also attacked by Dobbertin and collisions were found in a reduced round version of it. In response, the algorithm was modified and RIPEMD-128 (128 bits) and RIPEMD-160 (160 bits) emerged – these are included in the ISO/IEC DIS 10118-3 standard, which was finalized this year.

Meanwhile, in 1993, NIST (National Institute for Standards and Technology) published the Secure Hash Algorithm (SHA, now usually referred to as SHA-0), which was meant to be used with NIST's signature algorithm DSA. Like so many others, it too is based on the now thoroughly broken MD4, so it was little surprise when, in 1995, a modification appeared: SHA-1.

Not only did SHA-1 produce a 160-bit hash code (over the 128 bits of MD4 and MD5), but it included a unique expansion function before the compression, which is attributed with providing greater security. SHA-1, and its sisters, SHA-256, SHA-512 and SHA-384 are now also included in the ISO/IEC DOS 10118-3 standard and are the only hash algorithms allowed by NIST's FIPS 180-2 standard.

THE DOWNFALL OF MD5

After Dobbertin's success with breaking MD4, the focus shifted to breaking MD5 security. A website intending to organize a distributed computer attacking MD5 (as was used in the RSA challenge) was created at http://www.md5crk.com/.

While that effort was focused on a brute-force search for collisions, the more interesting problem of finding a less computationally exhausting attack on the complete algorithm remained elusive. That is, until the presentation given at this year's Crypto conference by Xiaoyun Wang, Dengguo Feng, Xuejia Lai and Honbo Yu [3], all of Chinese research institutes or universities.

Using a multi-processor *IBM pSeries* machine, Wang *et al.* claim to be able to calculate a collision in just over an hour. For the HAVEL-128 extension to MD5, they were able to calculate collisions in 2^6 calculations, as opposed to the brute-force approach requiring 2^{64}. In fact, they claimed that, with their method, collisions can be found by hand calculation for MD4.

Next, they showed two collisions in the original RIPEMD algorithm and finally they mentioned that collisions could be found in the original SHA-0 algorithm in 2^{40} calculations, as well as in the HAVEL-160.

Unfortunately, their presentation (and a paper on the e-print archive site of the International Association for Cryptologic Research) did not provide very much detail about their method. However, they made a convincing argument and it is likely that Wang *et al.* will publish a more detailed paper in the near future so that the crypto community can evaluate it.

Although eclipsed somewhat by the Wang, *et al.* presentation, Antoine Joux (whose ideas the Chinese team used in their work) announced in the same session of the conference the existence of a collision in the original SHA-0 algorithm, effectively lending weight to Wang, *et al.*'s similar statement.

Also in the same session, Israeli cryptographer Eli Biham announced results on the COLL attacks against SHA-1 that he has been waging. So far, collisions have been obtainable in a reduced round version of SHA-1 (40 instead of 80 rounds).

DO WE CARE?

For simple applications of hashes, PRE and 2PRE security is probably sufficient. But before you breathe a sigh of relief, consider this: it is believed that evidence of COLL attacks implies that 2PRE and possibly PRE attacks are also likely to be imminent. This seems to be borne out by our experience with MD4, where first partial, then full COLL attacks preceded useful COLL and then partial PRE attacks. If MD4 were still of interest, there might be a successful PRE attack by now.

But since COLL attacks do not automatically imply PRE and 2PRE attacks, we may be OK for simple uses of hashes. Using MD5 hashes for file integrity or document signing applications will still provide a good level of security until useful collisions can be found.

Applications like SSL may also be secure enough because, while an attacker might try to have a certificate authority sign one version of a certificate and then use its collision later to forge the site's certificate, in practice, signing authorities provide part of the data that then gets signed, which greatly limits the usefulness of a COLL attack in this instance.

However, more complicated cryptographic protocols may rely specifically on a hash function being collision-free. This is apparently the case with ISO/IEC 18033-2 [4], which is still in draft status. These applications will have to look for more secure hash functions.

CONCLUSIONS

MD5 is now considered broken. We can expect folks to be moving away from it in the near future, as the opportunity arises. The same goes for SHA-0, RIPEMD and HAVEL-128.

SHA-1 is now also in doubt, as the crypto community generally believes that COLL attacks are quite possible in the near future, and so also may follow attacks on 2PRE and PRE. Thus it seems prudent to replace SHA-1 with SHA-256 or better in security-critical standards, software, and even in actual digital data which must remain secure for some time.

Furthermore, SHA-1 is hardwired in a whole host of internationally-recognized standards (such as NIST's official digital signature scheme, most of the work of the Trusted Computing Group, *etc.*), and implementations of all such standards must be considered somewhat suspect unless and until each one is examined by experts and concluded even to be safe with a possibly COLL-failing hash function.

Because all of the hash functions we have discussed are based on MD4, it is tempting to lay the blame for this mess on this single fact. However, it is important to note that it has not been proven that COLL-resistant hash functions are possible at all. Also, the entire field of hash functions is at best a black art and is based on the best judgement of a few very talented individuals.

LESSONS

Even a perfect hash function only ever has 'as much security as' half as many bits as its digest size (this notion can be made precise in cryptography). So, in fact, we were *already at risk* before this recent discovery of flaws in MD5 and SHA-1, every time we used such a hash in a protocol with a keyed cryptographic primitive whose key size was more than half the digest size – at risk in the sense that we were getting less security than we thought.

Thus one lesson the current situation should teach us is to be careful about which hash function we use, even simply in terms of its digest size.

We should also take from this incident the fact that we must keep implementations that rely on hash functions flexible. There must be a simple way to replace broken hash function code (or simply increase the size of the digest, when necessary for a certain number of bits of security, as just noted).

Furthermore – and just as important – there must be a clear migration path to the new hash function. Not only must the hash function be modularized, but the storage available for the hash code must be expandable. It may also be necessary to keep the old, defunct hash codes around and space needs to be allocated for that, too.

Standards bodies should make their designs modular in a similar fashion, so that any future advances in the cryptanalysis of hash functions do not invalidate the entire design, but merely require more powerful hashes to be plugged in.

Given the pervasiveness of MD5 usage, it will be interesting to see how long it takes until we have completely migrated away from MD5. There are still some programs in use today that employ MD2!

BIBLIOGRAPHY

[1] Menezes, Alfred; van Oorschot, Paul C.; Vanstone, Scott A., *Handbook of Applied Cryptography*, 1997, CRC Press.

[2] Hans Dobbertin, 'The Status of MD5 After a Recent Attack', *CryptoBytes* vol. 2, no. 2, 1996, ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto2n2.pdf.

[3] Xiaoyun Wang; Dengguo Feng; Xuejia Lai; Honbo Yu, 'Collisions for Hash Functions MD4, MD5, HAVEL-128 and RIPEMD', International Association for Cryptologic Research, 2004, http://eprint.iacr.org/2004/199/.

[4] Working Group 2 of ISO/IEC JTC 1/SC27, 'ISO 18033-2: An Emerging Standard for Public-Key Encryption', 2004, available from http://www.shoup.net/iso/.