Tripwire: Part Three – an integrity checker

SAFETY FIRST

KLAUS BOSAU

File system integrity checkers are increasingly popular, but hardly any of their countless users are aware of their forerunner: the hash function. The third instalment in our Tripwire series is intended to close that particular knowledge gap.

The development of the one-way hash function, or message digest function, had its origins in a cryptographic requirement. To speed up the time-consuming process of signing messages by encrypting them with the sender's private key, it made sense to first reduce the content volume to a more manageable size, the fixed-size hash value or message digest.

The special function of this interim step, which is only distantly related to the task of key-based encryption mechanisms such as RSA or ElGamal, leads to a more effective implementation and therefore to a significant increase in speed. Instead of sending the encrypted original of the message, now the original itself and the encrypted message digest are being sent.

First, the recipient decrypts the message digest with the sender's public key. A subsequent comparison of the result with a message digest produced from the original will provide information about the authenticity (and integrity) of the message. In order for modified signatures to be able to fulfil this authentication function, suitable candidates have to meet high standards.

Message blocks

For instance, the function *f* must apply to every part of a message. That is mostly a question of design. A process called padding ensures that different sized messages are treated in the same way. In the case of MD5, a message is padded until its size is an integral multiple of 512 bits, so that the compression function (the heart of the hash function) can be processed in blocks within the next computational step.

The workflow is reminiscent of Unix pipelines. Each 512-bit message block is involved once and

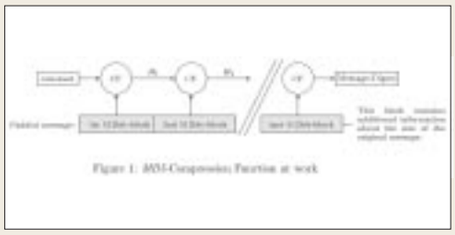
Figure 1: Block schematic of MD5.

If f denotes the compression function and Bi represents the i-th of a total of n message blocks, then this can also be very concisely expressed as a set of equations:

H0 = K

Hi = f(Hi-1,Bi) i=1,2,...,n

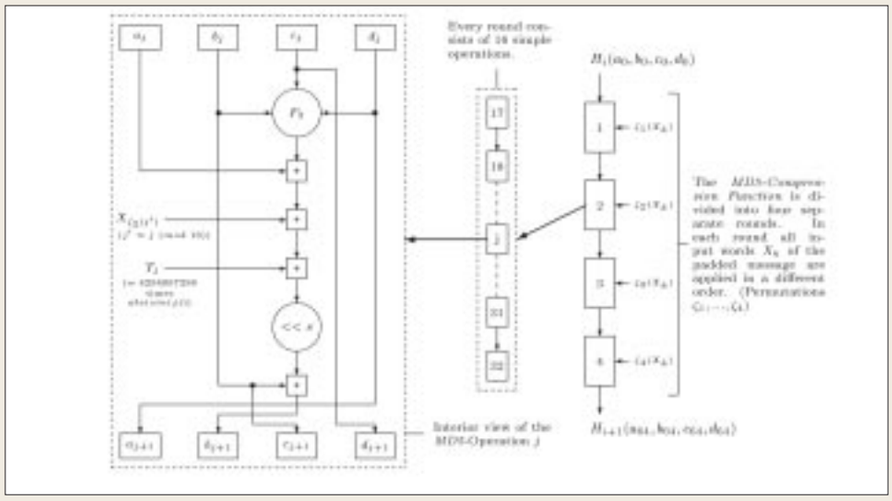
The hash value of the extended message N = B1,...,Bn would then be Hn.



used for the continued modification of a given initial hash value, which is in the form of four 32-bit constants. The 128-bit hash value of the message in question is computed as the final output (see Figure 1). By chaining the interim results *Hi* – called chaining values – every part of the message is adequately included in the calculation. However, in the compression function itself there are large disparities (see Figure 2).

The function must not be invertable, so that it will be impossible to work out the message content from knowing the message digest. This requirement has been pretty well met up to now, since the message digest normally tends to be much smaller than the

Figure 2: Inner workings of the MD5 compression function.



The compression function has been implemented as a hash value pipeline consisting of 64 identically structured individual operations. Each operation processes only one of the total of sixteen 32-bit words Xk in the current message block. It follows therefore that a word is used exactly four times by the pipeline. This happens in successive phases, or rounds. The diagram shows how the four 32-bit chaining variables aj,..., dj, which form the current chaining value, are linked. In this Fn denotes the non-linear function belonging to round n, which is derived from bit operations (example: F2(b,c,d)=(b^d) v(c^ d)), »x« denotes an addition (modulo 2^32) and »<< s« a bit rotation by s positions).

(Tj) is a prepared field of trigonometric origin (sine function), that provides every one of the 64 operations with its own additive constant. Index k and the shift amount s change with each operation: k = k(j), s = s(j).

The expression A = ((A + Fn(B,C,D) + Xk + Tj) < s) + B can be implemented effectively by using four 32-bit registers (A, B, C, D) as a buffer for the chaining variables, one of the reasons for the popularity of MD4.

The functional construction ensures the thorough mixing of the message's data words Xk - hence 'hash'. For example, the multiplicative character of pipelines makes it practically impossible to predict how even the change of one message bit will affect the output, making it equally difficult to discover suitable alterations that would result in a message producing a specific desired hash value.

The crucial problem turns out to be ensuring high speed (leading to a restriction to logical bit operations and register-friendly variable formats) without giving rise to the dangers of analysis through over-simplification (as happened with MD4). How difficult it is to arrive at a design, which combines acceptable speed with sufficient complexity, is evident from the widespread use of the ageing MD4 operation type, which often forms the basis for new developments due to its familiarity. For a detailed description of the structure, please refer to Rivest's original publication.

message itself, and every extreme compression inevitably destroys information. Occasionally some experts manage, after a careful analysis of the functionality, to design an inverse function, mostly for simplified variants. Although this is not able to reproduce the original itself, it can produce a message with the same message digest. The days of that particular hash function are then numbered.

This happened in 1998 for a simplified variant of MD4 with two instead of the usual three rounds. Ironically, a leading player in this was the same Hans Dobbertin of the German Federal Information Security Agency (BSI), who later participated in the development of the European equivalent of SHA-1, the RIPEMD-160.

Unique message digest

It must be impossible to produce for any given message, another message with different content that would result in the same message digest.

Hash functions that achieve this are described as weakly collision-free, because these conditions do not make very high demands of the algorithm itself.

Apart from the danger of algorithm inversion already outlined, which is a special case in this context, the algorithm in question must also be able to resist brute-force attacks. These are a constant threat due to ever increasing computer performance. The offset of a forged message is simply altered until the resulting message digest matches that of the original.

In practise this can be done by appending spaces to the ends of some lines. For 30 lines it does not take much effort to produce 2^30 duplicates in this way. To make this more difficult, the bit pattern of the original length is therefore inserted at the end of the extended message during padding.

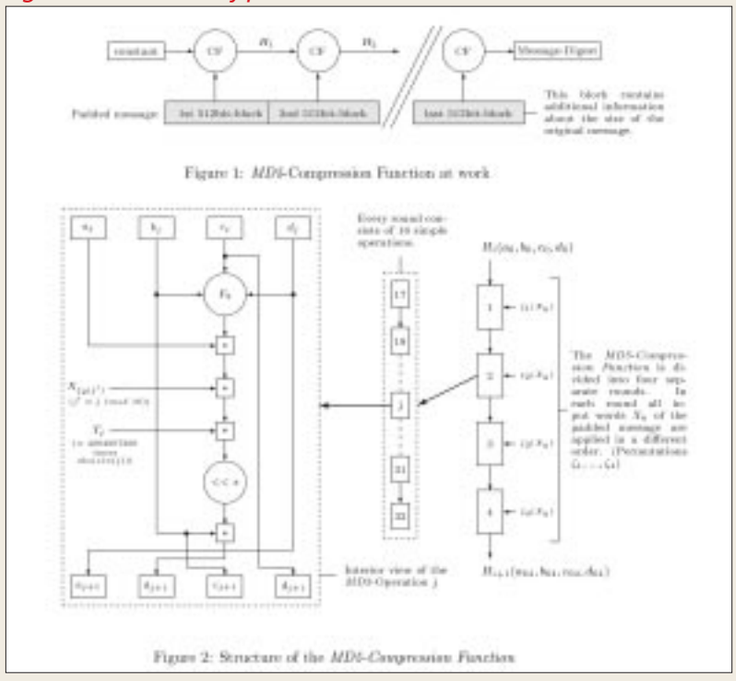
A processor with a performance of 12.5 Mips (very slow by today's standards) can manage this in 0.42 seconds for a 47K file in case of a CRC-16 hash value and in four hours for a CRC-32 one. It is therefore not difficult to produce a forgery of the same size as the original. However, if the format of the hash value is increased, the limits of this method become obvious. The rapidly decreasing probability of matching (of the order of *2-n* for a message digest size of *n* bits) can barely be offset by increased computational performance for formats of 64-bits. The 128-bit format, which is the current standard, should be able to withstand this sort of attack for quite a while yet.

Strongly collision-free functions

It should also not be possible to find any two messages with a different content that just happen to have the same message digest. This condition is much harder to fulfil than the previous one, which is why these functions are called strongly collision-free hash functions.

Empirical investigation shows that collisions are not uncommon for hash value sizes up to 64-bits. That is not surprising, especially since the probability of a collision for a message digest with a size of *n* bits is of the order of *2-n/2*,

Figure 3: The birthday paradox



*The birthday paradox is the name given to the deductive result of the astonishingly large disparity in the statistical probability of two specific events: A birthday party consists of N guests. The probability that one of them shares a birthday with the host is p' = 1-(364/365)N, but p = 1 - 365!/((365-N)! * 365N) for the event of the birthdays of two guests randomly coinciding.*
The implication of this result quickly becomes clear if we calculate the number of guests needed for a 50 per cent probability. It turns out that while the first condition can only be met with 253 people, the second only requires 23. The reason for the large difference is the fact that the increase in the number of possible guest pairs is of course quadratic, while the increase in potential matches with the host is only linear in relation to the number of guests.
The analogy to the probability of random collisions is immediately apparent. It is indeed possible to derive the order 2-n/2p mentioned previously by substituting the number of days in a year with that of all possible hash values with a length of 2n bits. This makes the total number of guest the number of messages required to achieve a certain degree of matching.
*A good approximation of the 50/50 probability is therefore N P 1,2 * 2n/2 (more precisely: N P (1 + [root of](1 + 0,69 * 2n+3))/2). On the other hand, it would take a total of N' P 0,69 * 2n P 0,48 * N2 messages to find a match for a specific hash value (host-birthday)! The result for N obviously provides a useful criterion for the collision resistance of an n-bit hash function.*
For the testing of formats beyond 128-bits, which are barely manageable in terms of computation, handier duplicates with short signatures are used, such as 16 or 32-bits. It is then possible to use the experimentally computed collision frequency – ideally one collision to N hash values – to extrapolate the behaviour of the original. Note: significant disparities can point to an unequal distribution of the hash value probabilities. That would, of course, be unwelcome, as an attacker could take advantage of it.

Info

http://www.faqs.org/rfcs/rfc1321.html
H.Dobbertin: RIPEMD with two-round compress function is not collision-free Journal of Cryptology
H.Dobbertin: Cryptanalysis of MD4, Fast Software Encryption - Cambridge Workshop. Lecture notes in Computer Science vol 1039
B.den Boer and A.Bosselaers: An attack on the last two rounds of MD4. Advances in Cryptology - Crypto '91
CryptoBytes ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto1n3.pdf
B.den Boer and A.Bosselaers: Collisions for the compression function of MD5, Advances in Cryptology, Proc. Eurocrypt '93
ftp://ftp.cert.dfn.de/pub/docs/crypt/ripemd160.ps.gz

which is much greater than in the classic case of a brute-force attack.

The once unrivalled cyclic redundancy checks CRC-16 and CRC-32 fail spectacularly in this respect. Out of 250 000 files tested on various computers 14 000 pairs with identical 16-bit hash values were found, and there were still three pairs with an identical 32-bit hash value. This aspect only loses its significance for hash value formats far beyond the 64-bit mark.

This result is a good example of the birthday paradox (Figure 3). It forms the basis of perhaps the most promising of all attack types: the Birthday Attack. The aim of this brute-force variant is to obtain a digital signature through random collision.

Unhappy birthday

In the Birthday Attack, the attacker produces two documents: a forgery that requires a signature and an innocuous message which the recipient would have no hesitation in signing. Then *2n/2* variants of each with almost identical content are produced, along with the corresponding message digests in *n-bit* format. Minor alterations to the text that convey the same meaning are sufficient.

Now the probability of finding two documents with the same message digest but belonging to different groups must be greater than *1/2*. The attacker then only needs to get the innocuous version digitally signed by the recipient (encryption of the message digest with the private key) and can then send the corresponding signature together with the suitable variant of the forgery. It is now impossible for a recipient to recognise the message as fake. The culprit has achieved his purpose without ever having gained access to the private key of the target.

Known weaknesses...

Consequently only a message digest of sufficient size can offer protection. In this titanic struggle that is not a constant. While the 128-bit message digest was all the rage until recently, some experts are already demanding a 160-bit alternative today.

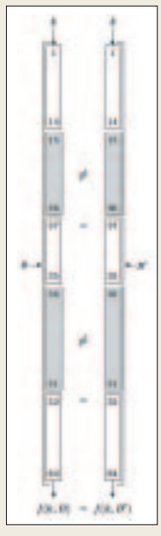
PCs are not a threat in this respect. Even though a 200 MHz Pentium processor only takes nine milliseconds to calculate an SHA hash value for a 50K file, it would have to produce *2^81 P 2.4 * 10^24* variants to find a match. A usable result would take *7 * 10^14* years of computing time, too long for most of us.

Even hardcoding or parallelisation via the Internet are no help for these formats in the foreseeable future, unless it was possible to shorten the computing time through clever choice of process. Such an attempt, which undermined the trust put in MD5 until then, was made in 1996 by the German cryptographer Hans Dobbertin (see Figure 4).

... and the tandem pipeline

This hash function research can practically be transferred to MD5. MD4 was once pestered by an inconspicuous attack from two of Dobbertin's

Figure 4: Example of an attack on MD5.



The cryptographer Dobbertin was only able to discover collisions for the MD5 compression function, i.e. an initial value k and two different 512-bit blocks B and B', for which the following is true: f(k,B) = f(k,B'). However, if the hash value H = H2 of a message N = B, B2 is written in the form f(H1, B2) = f(f(K,B), B2)), the consequences of these results become clear immediately. The different messages N = B, B2 and N' = B', B2 created during padding would produce the same hash value H, if it were possible to match the initial value k of the collision created to the initial value K of the MD5 hash function by suitable alteration of the discovered process.
The attack takes advantage of the process used by the MD5 compression function for reading in the words of the message block. To simplify matters, Dobbertin assumes that the blocks B and B' only differ in one of the 16 words, e.g. X14. Since every word is read exactly once in each of the four rounds, following a set pattern which varies for each round, the four diverging steps in the calculation of the hash values for B and B' can be identified immediately: step 15 (round 1), step 26 (round 2), step 36 (round 3) and step 51 (round 4).
The basic idea is to produce two harmonised inner collisions in the segments 15 to 26 and 36 to 51 by selecting appropriate chaining values. The chaining values of B and B' should therefore match again before steps 27 and 52, ensuring that they would develop identically in the three other segments. The solution found should lead to identical compression values f(k, B) and f(k, B') and therefore to a collision in the compression function.
These assumptions lead to a set of equations in which the chaining values had to be logically regarded as an unknown. This simplification through certain assumptions reduces the problem to the iterative solution of equations that only contained the non-linear functions Fn of the MD5 operation, known quantities ci and [delta]x, as well as an unknown quantity x:

Fn(c1, c2, x) + c3 = Fn(c4, c5, x + [delta]x)

In this way, both the chaining values from step 15 onwards and the message blocks B and B' (functions of the now known chaining values) become accessible to calculation. Details of the calculation methods can be found in RIPEMD and MD4. Finally, the initial value k could be found through simple backwards calculation, starting with step 14. The process was later implemented as a PC application. It took a Pentium processor ten hours to track down a collision for the MD5 compression function.

colleagues. Three years later MD4 was history. In cryptobytes, page 5, the reader can find a specific example of random collision, i.e. two messages that produce the same message digest despite having different contents. The computing time needed was less than one hour.

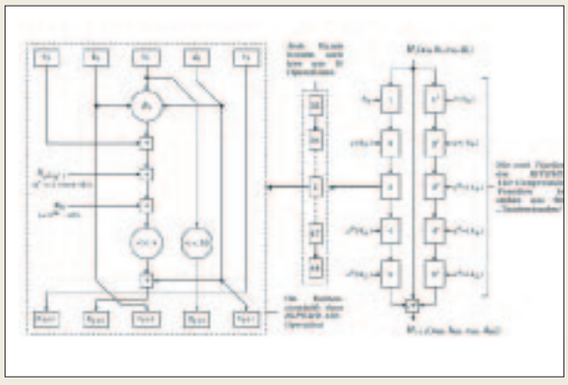
The future will show how long Rivest's MD5 will stand up to the inventiveness of ambitious cryptographers. There are already (more) secure alternatives: the newcomers SHA-1 and RIPEMD-160 exhibit some distinctive features that make attacks of the kind described above considerably harder (see Figure 5).

The one-way hash function has developed into a universal tool. The secure transmission of passwords and digital timestamping (as realised in eternal logfiles) are further applications for this cryptographic star.

After this excursion into the high-flying world of hash functions, the next part will return to more down-to-earth topics and give practical tips for operating and securing Tripwire shields. We will also be introducing an extension that breathes new life into the slightly outmoded Academic Source Release. ■

Figure 5: RIPEMD-160, a state of the art hash algorithm

The structure of RIPEMD-160 is significantly different from that of MD5. The signature format has increased from 128 to 160 bits and a fifth data word was required ej. This is



intended to deter brute-force attacks. The added bit rotation << 10 on the other hand has its origins in the shrewd manoeuvre of two certain gentlemen (Boer and Bosslaers).

In order to avoid specific procedures during the reading of the words for a message block as in MD5, the development team around Hans Dobbertin is relying on the new combination of two conventional pipelines, each of which only contributes to the compression value as a summand. As both of the complementary steps i and i' in a round use a different sequence for reading in (additional permutation [pi](n) = 9n + 5 (mod 16) of the data words for the current message block at i') and also exchange information, conservative attacks using the internal collision model will fail here.

This sort of reinforcement is achieved at the cost of a nine per cent drop in performance as compared to SHA-1, but it is probably the most effective defensive concept against keen analysers. RIPEMD-160 does not come with any licensing conditions, and because of its pipeline structure it is easily applied to double-length signatures.