MACs and Hash Functions: State of the Art

Bart Preneel[1]*, Department Electrical Engineering-ESAT, Katholieke Universiteit Leuven, Belgium*

This paper gives a survey of MAC algorithms and hash functions. It describes the main properties, summarizes the most important constructions and reports on their security.

Introduction

While secrecy protection and encryption are as old as writing itself, information authentication is a problem of the information age, created by the decoupling of information and its physical bearer; electronic information cannot be protected by seals or manual signatures. During the last twenty years cryptology has created a number of tools to address this problem; the most important ones are digital signatures based on public-key algorithms[2] such as RSA, ElGamal, and DSA.

While there is no doubt that digital signatures and certificates are the cornerstone for applications such as electronic commerce, there has been also a strong need for conventional (or symmetric) algorithms for information authentication. The main reason is that digital signatures are too computation intensive to apply them to bulk data (documents of 100 Kbyte or more) or to each packet of a packet-switched network such as the Internet. For the latter application, the signature size (between 40 and 128 bytes) poses problems as well.

The need to digitally sign large amounts of information has led to the invention of cryptographic hash functions (also known as MDCs or Manipulation Detection Codes). Hash functions reduce an input document of arbitrary size to a short string (typically 8 to 20 bytes), which is the 'fingerprint' or 'digest' of this document. There are of course many inputs corresponding to a single output string, but hash functions are constructed in such a way that it is computationally infeasible to find one or more such inputs. The digital signature is then applied to the short hash result, rather than to the large input document. This results in an improved performance, as hashing is typically much faster than signing. Other advantages are that signatures are shorter and harder to forge. For some recent signature schemes the hash function has even become an integral part of the construction. Hash functions have been used as a building block in many other applications, such as commitment to information without revealing it, protecting of pass-phrases, tick payments, key derivation, and key agreement. It should be pointed out that hash functions have often been used in applications that require new security properties, for which they have not been evaluated.

For some applications, both the computational overhead and the size of a digital signature are too large; examples are packet-switched networks (e.g., IP security) and inexpensive electronic purses

[1] F.W.O. postdoctoral researcher, sponsored by the National Fund for Scientific Research – Flanders (Belgium).

[2] W. Diffie, M.E. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, Vol. IT–22, No. 6, 1976, pp. 644–654.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

(e.g., Mondex and Proton). In that case signatures can be replaced by Message Authentication Codes or MACs. A MAC algorithm computes a short string (typically 4 to 16 bytes) as a function of the input and a secret key. The sender appends the MAC to the information; the receiver recomputes the MAC and compares it to the transmitted version (see also Figure 1). He accepts the information as authentic only if both values are equal. The eavesdropper can modify the message, but does not know how the update the MAC accordingly. This procedure requires that sender and receiver are privy to the same secret key. MACs provide weaker guarantees than signatures, as they can only be used in a symmetric setting, where the parties trust each other. In technical terms, they do not provide non-repudiation of origin. On the other hand, MACs are much more efficient than digital signatures in terms of computation, storage, and key size.

The rest of this paper is organised as follows. First definitions are given for hash functions, attacks on hash functions are discussed, and a survey is presented of the most important constructions. Then MACs are treated in a similar way. Finally some concluding remarks are formulated.

Definitions of hash functions

Hash functions were introduced in cryptography by W. Diffie and M. Hellman together with the concept of digital signatures. A hash function is a function that takes an input of arbitrary size and reduces it to an output of fixed length (m bits). Moreover, this computation should be 'efficient'. One also requires that the hash function is publicly known and does not require any secret parameter. For cryptographic applications, one imposes three security requirements, which can be stated informally as follows:

preimage resistance: for essentially all outputs, it is 'computationally infeasible' to find any input hashing to that output;

2nd-preimage resistance: it is 'computationally infeasible' to find a second (distinct) input hashing to the same output as any given input;

collision resistance: it is 'computationally infeasible' to find two colliding inputs, i.e., x and $x' \neq x$ with $h(x) = h(x')$.

A hash function that is preimage resistant and 2nd-preimage resistant is called *one-way*; a hash function that satisfies the three security properties is called *collision resistant*. Note that for one-way functions (as opposed to one-way hash functions), one typically requires only preimage resistance.

Collision resistance is required for digital signature to avoid the following scenario: Bob creates two messages with the same hash value, one of which says 'Alice owes Bob 100£' and the other one 'Alice owes Bob 100 000£'. Then Bob asks Alice to sign the first message. Later he claims that Alice has signed the 2nd message instead (which has the same hash value and thus the same signature). Alice could preclude this attack by randomising the message before signing it, but this is not always a solution (for example, Alice could still attack her own signature).

Not all applications need collision resistance; loosening the requirements results in savings in terms of computation and storage (cf. infra). However, most designers aim for this property to have a tool that can be used in all the applications, and to be on the safe side. It is important to note that some applications need different properties, for example that it is computationally infeasible to find x from $h(x\|y)$ and y (here $\|$ denotes concatenation).

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

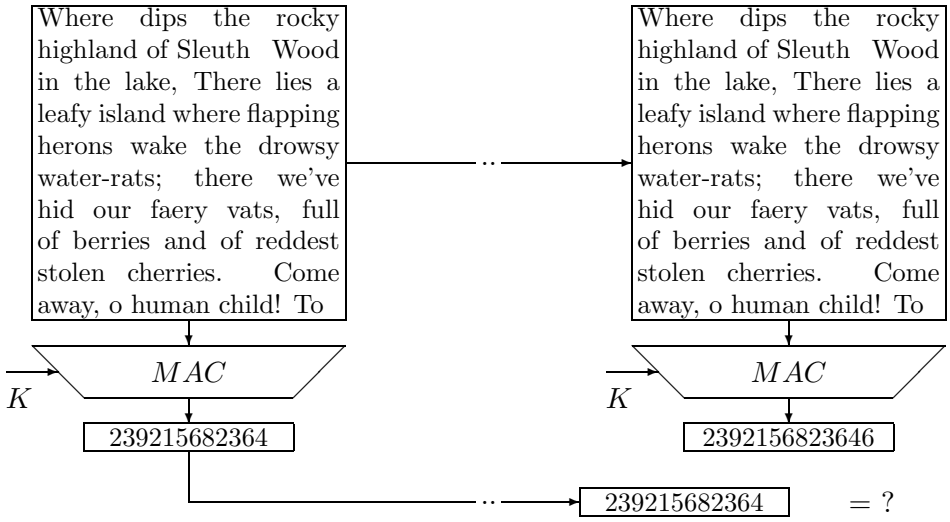

Figure 1: Using a Message Authentication Code for data integrity.

Security of hash functions

This section discusses attacks that depend only on the size of the parameters of the hash functions, and not on their internal structure.

Brute force (2nd) preimage search

In order to find (2nd) preimages, on can just pick an arbitrary input and evaluate h. The success probability of a single trial is $1/2^m$, with m the number of bits in the hash result, which implies that on average 2^{m-1} attempts are required. This attack can be applied off-line and in parallel. If it is sufficient to find a (2nd) preimage for one out of t given values, the success probability is increased with a factor of t. This can be avoided by parameterising the hash function for each input. This attack can be precluded by choosing values of m between 64 bits (marginally secure) to 128 bits (security for 20 years or more even against a determined opponent).

Brute force collision search

Surprisingly, finding collisions is much easier than finding preimages: one needs only about $\sqrt{2^m} = 2^{m/2}$ evaluations of the hash function (as this results in about 2^m pairs of hash values). This phenomenon is related to the 'birthday paradox,' which states that in a group of 23 people, the odds are 1:2 that there are two people with the same birthday. Note that it is easy to impose that the colliding message are restricted to a small set with a prescribed format, and that one can implement this attack in parallel with minimal storage requirements[3]. Collision resistance requires that m is between 128 bits (marginally secure) and 160 bits (20 years or more).

Constructions for hash functions

Almost all hash functions are iterative processes, that repeatedly apply a simple com-

[3] P.C. van Oorschot, M.J. Wiener, "Parallel collision search with application to hash functions and discrete logarithms," *Proc. 2nd ACM Conference on Computer and Communications Security*, ACM, 1994, pp. 210–218

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

pression function f. The input x is padded to a multiple of the block size, and is then divided into t blocks denoted x_1 through x_t. The intermediate result is stored in an n-bit ($n \geq m$) *chaining variable* denoted with H_i:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(H_{i-1}, x_i), \qquad 1 \leq i \leq t \\ h(x) &= g(H_t)\,. \end{aligned}$$

Here g denotes the *output transformation*.

The goal of the designer is to derive the security properties of the hash function h from those of the compression function f. In order to exclude trivial attacks, it is important to fix the value of IV and to precode the message. The simplest way of coding is to append an extra block at the end with the length of the input in bits. R. Merkle and I. Damgård showed that under these conditions, collision resistance of f is sufficient for collision resistant of h. Note that it is not a necessary condition, i.e., collisions for the compression function do not necessarily lead to collisions for the hash function. X. Lai and J. Massey proved a similar property for (2nd) preimage resistance; in this case the converse holds (in a weaker form).

The large number of broken hash functions serves as a warning to users to be careful in adopting a hash function, and to allow for a modular design such that hash functions can be replaced whenever necessary. In the remainder of this section, an overview is given of constructions for hash functions based on block ciphers, hash functions using algebraic structure, and custom-designed hash functions.

Hash functions based on block ciphers

The popularity of these constructions is in part historical, as designers tried to use the DES also for hashing. This can lead to compact implementations and to a transfer of the trust in the block cipher to the hash function. However, the use of a block cipher in this application requires slightly stronger properties from the block cipher, and this construction is only efficient if the key scheduling of the block cipher is not too slow. In this section it will be assumed that the block cipher has block length and key length of m bits.

The earlier constructions are *single block length* hash functions, i.e., the size of the hash result is equal to the block size of the block cipher. The first secure constructions were the scheme by Matyas, Meyer, and Oseas, which has been included in ISO/IEC 10118-2:1994, and its dual, widely known as the Davies-Meyer scheme[4]. The Davies-Meyer scheme has the following compression function:

$$H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}\,.$$

Here $E_K(x)$ denotes the encryption of plaintext x using the key K. It was shown that 12 'secure' schemes exist of this type, which can be derived by an affine transformation of variables on two basic schemes.[5] It is conjectured that for these schemes no shortcut attacks exist, which implies that collisions require about $2^{m/2}$ operations and a (2nd) preimage about 2^m operations. However, since most current block ciphers have a block length of $m = 64$ bits, collisions can be found in only 2^{32} operations and hash functions with a larger hash result are needed.

The aim of *double block length* hash functions is to achieve a higher security level against collision attacks. The *hash rate* of a hash function based on an m-bit block cipher is the number of m-bit message blocks

[4]The real inventors are probably S.M. Matyas and C.H. Meyer.

[5]B. Preneel, R. Govaerts, J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Advances in Cryptology, Proc. Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 368–378.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

hashed per encryption. For all constructions with rate 1 of a large class it was shown that collisions for the hash function can be found in at most $2^{3m/4}$ operations[6]. Well known constructions are MDC-2 and MDC-4; they have hash rates 1/2 and 1/4 respectively. MDC-2 has been included in ISO/IEC 10118-2:1994. For both hash functions a collision attack is believed to require at least 2^m operations (for DES this corresponds to 2^{55} encryptions). MDC-2 and MDC-4 do not have a collision resistant compression function. Two constructions have a 'proof of security' for the collision resistance of the compression function:

- the constructions by R. Merkle[7] achieve a security level of 2^{56} for DES and rates up to 0.27;
- the constructions proposed by L.R. Knudsen and B. Preneel,[8] which offer better trade-offs between rate and security level; moreover the security level can be higher than 2^m.

Hash functions based on algebraic structures

Another natural construction for hash functions uses modular arithmetic which has proved very useful for public-key cryptography. Certain applications have a fast processor for such operations, and could make an optimal use for this by employing it also for hashing. For performance reasons, one prefers modular squaring as basic operation. Moreover, computing square roots modulo an RSA modulus N (the product of two large primes) is equivalent to factoring. Note that recently an efficient way has been developed to construct an RSA modulus between three or more parties in such a way that none of the parties knows its factors; this is particularly useful for hashing applications. However, it seems to be quite hard to construct a hash function based on modular squaring. This has been illustrated by the many failed attempts (including the informative Annex D of the 1989 edition of CCITT X.509, which is the same as ISO/IEC 9594-8:1989).

The most promising approach is based on the following function:

$$H_i = (\tilde{x}_i \oplus H_{i-1})^2 \bmod N \ \oplus H_i \,.$$

Some modifications are required to avoid several attacks. In the current draft proposal (ISO/IEC CD 10118-4, July '97), m denotes the length of the modulus, and n denotes the largest multiple of 16 strictly larger than n; H_i and $\tilde{x}_i$ are strings of length n, and x_i has length $n/2$. The first four of the n bits that are input to the modular squaring are set to 1, and after the squaring only the least significant n bits are kept. The bytes of $\tilde{x}_i$ consist of four 1-bits followed by four consecutive bits of x_i. At the end, an output transformation is defined that requires an additional eight iterations. The above proposal is denoted MASH-1; MASH-2 replaces the exponent 2 by 257.

While it is possible to construct hash functions for which finding collisions is as hard as solving a number-theoretic problem believed to be difficult (factoring, discrete logarithm), most of these are not practical. M. Bellare and D. Micciancio proposed the following construction:[9] hash

[6] L.R. Knudsen, X. Lai, B. Preneel, "Attacks on fast double block length hash functions," *Journal of Cryptology*, in print.

[7] R. Merkle, "One way hash functions and DES," *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 428–446.

[8] L.R. Knudsen, B. Preneel "Fast and secure hashing based on codes," Advances in Cryptology, Proc. Crypto'97, LNCS 1294, B. Kaliski, Ed., pp. 485–498.

[9] M. Bellare, D. Micciancio, "A new paradigm for collision-free hashing: incrementality at reduced cost," *Advances in Cryptology, Proc. Eu-*

individual parts of a message, represent the hash results as elements over a group of large prime order, and combine them to one hash result by multiplication in the group. They show that, under certain assumptions, breaking the larger hash function is equivalent to solving the discrete logarithm problem in this group. Moreover, the hash function is incremental: if only part of the message is changed, recomputing the hash function requires very little effort.

Many attempts have been made to construct efficient hash functions based on additive and multiplicative knapsacks. This research is supported by interesting theoretical perspectives. However, it seems that for practical parameters, most of these hash functions have been broken.

Custom designed hash functions

Also in this class there is no lack of proposals and attacks. Hash functions for which collisions have been found (or at least for the compression function) include Snefru by R. Merkle (with less than 8 rounds), FFT-Hash I and II by C.P. Schnorr, and MD2, MD4, extended-MD4, and MD5 by R.Rivest. As noted above, collisions for the compression function by itself are not sufficient to discredit a hash function, as they might be easier to find than collisions for the hash function. However, a collision resistant compression function is a useful design criterion. A second aspect is that applications sometimes have highly redundant messages, which makes finding collisions in this set much harder. Nevertheless, some of the attacks leave room for further improvement, and it is prudent to take that into account.

The MD4-family seems to make optimal use of current 32-bit microprocessors, which makes it very popular for applications. The powerful attack by H. Dobbertin on MD4[10] suggest that it should no longer be used. MD5 is very fast, but randomly looking collisions have been found for the compression function (again by H. Dobbertin). One can expect that it is currently feasible to extend this attack to the complete hash function, and to find more realistic colliding messages. The protection offered against brute force collision search is only marginal. It seems thus prudent to upgrade current applications of MD5.

Because of an undisclosed attack (which was claimed to be not very serious), the US FIPS 186 standard SHA has been replaced by SHA-1 (FIPS 186-1). Both versions have a 160-bit result. A partial attack (two rounds out of three) has led to an upgrade of RIPEMD (proposed by the European Consortium RIPE). Its successors are RIPEMD-128 (with a 128-bit result) and RIPEMD-160 (with a 160-bit result).[11] The latter two schemes, together with SHA-1 have been included in ISO/IEC 10118-3:1997. On a 90 MHz Pentium, RIPEMD-160 and SHA-1 run at about 50 Mbit/s, while RIPEMD achieves 75 Mbit/s. This should be compared to DES, which runs at 17 Mbit/s (with a fixed key).

For the future, there is certainly a need for hash functions optimised for 64-bit architectures, and for fast parameterised one-way hash functions with a result of 80, 96 or 112 bits. None of the known hash functions is really suited for the 8-bit and 16-bit processors found in smart cards.

rocrypt'97, LNCS 1233, W. Fumy, Ed., Springer-Verlag, 1997, pp. 163–192.

[10] H. Dobbertin, "Cryptanalysis of MD4," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 53–69.

[11] H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160, a strengthened version of RIPEMD," *Fast Software Encryption, LNCS 809*, R. Anderson, Ed., Springer-Verlag, 1994, pp. 71–82.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

Definitions of MACs

The banking world used MACs already in the seventies. A MAC is a hash function that takes as input a second parameter, the secret key. The main security property is that for someone who does not know the key, it should be computationally infeasible to predict the MAC value corresponding to a given message.

The attack model is as follows: an opponent can choose a number of inputs x_i, and obtain the corresponding MAC value $h_K(x_i)$ (his choice of x_i might depend on the outcome of previous queries, i.e., the attack may be adaptive). Next he has to come up with an input x ($\neq x_i, \forall i$) and the value $h_K(x)$, which has to be correct with probability significantly larger than $1/2^m$, with m the number of bits in the MAC result. If the opponent succeeds in finding such a value, he is said to be capable of an *existential forgery*. If the opponent can choose the value of x, he is said to be capable of a *selective forgery*. If the success probability of the attack is close to 1, the forgery is called *verifiable*. An alternative strategy is to try to recover the secret key K from a number of message/MAC pairs. A *key recovery* is more devastating than a forgery, since it allows for arbitrary selective forgeries.

A MAC is said to be secure if it is computationally infeasible to perform an existential forgery under an adaptive chosen text attack. Note that in many applications only known text attacks are feasible. For example, in a wholesale banking application one could gain a substantial profit if one could choose a single message and obtain its MAC. Nevertheless, it seems prudent to work with a strong definition.

Security of MACs

The attacks discussed in this section depend only on the size of the parameters of the MACs, and not on their internal structure: brute force key search, guessing of the MAC, and a birthday forgery attack.

Brute force key search

This attack consists of running through the key space and checking whether a key corresponds to the known message-MAC pairs. About k/m values are sufficient to determine the key uniquely; for most applications this value lies between 1 and 4. The expected computational effort is 2^{k-1} MAC evaluations, where k is the number of (effective) key bits.

Brute force key search can be precluded by choosing a sufficiently large key. A value of 56 bits offers only marginal security, while 75 to 90 bits is sufficient for 15 years or more. Currently finding a 56-bit key in a period of 1 year requires an investment of 50 000$, and it can be done in a few months by using idle cycles on the Internet, as was demonstrated in the Spring of 1997; with a 1 million US$ investment, the search time can be reduced to a few hours. Moreover, one also has to take into account the empirical observation that the computing power for a given cost is multiplied by four every 3 years (Moore's 'Law').

It is important to recover a MAC key within its active lifetime, which can be very short in communications applications. Outside that period, the key is completely useless. However, one can mount the attack during the complete lifetime of the system; it is sufficient to feed to the search machine the current text/MAC pairs. In order to assess the feasibility of this attack, one has to compare the cost of an attack with the profit which can be made by recovering one or more keys during the lifetime of the system.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

Guessing the MAC

Another attack strategy is to pick an arbitrary input and guess the MAC value by choosing an m-bit string uniformly at random. For a good MAC algorithm, one expects that the success probability equals $1/2^m$ (here the probability is taken over all keys). A related strategy consists in guessing the value of the key, and computing the MAC value. Its success probability is $1/2^k$. The success probability of the combined attacks is equal to $1/2^{\min(k,m)}$. The value of k is typically larger than that of m to preclude a brute force key search. Strictly speaking this is not a forgery attack, as the success probability is very small (the forgery is certainly not verifiable). However, one should take this attack into account when selecting a MAC algorithm. The value of m depends on the expected profit of a successful attempt, as well as on the number of trials that are allowed by the system, i.e., the way the system reacts to false alarms. For most applications $m = 32 \ldots 64$ is sufficient to render this attack uneconomical.

Birthday forgery attack

Similar to hash functions, MACs are often built using an iterated compression function (cf. supra). The secret key may be introduced in the IV, in the compression function f, and/or in the output transformation g.

B. Preneel and P.C. van Oorschot describe a generic forgery attack that applies to all iterated MACs.[12] The basic idea behind the attack is the birthday paradox. Its feasibility depends on the bitsizes n of the chaining variable and m of the MAC result, and the nature of the output transformation g. If $n = m$, and g is a permutation, the attack requires 1 chosen text and about $2^{m/2}$ known message-MAC pairs. If $n > m$, an additional number of about 2^{n-m} chosen message-MAC pairs is required. The attack requires at least one chosen text. Hence in applications where any access to the MAC algorithm with the correct key is precluded, it should not be considered a problem. Also, the forged message is of a special form: if the MAC is known for three messages of the form x, x', and $x\|y$, one can forge the MAC for $x'\|y$.

This attack motivates the use of a MAC result which is smaller than the size of the internal memory n. It can be precluded by appending a sequence number at the *end* of every message. Such a sequence number if useful to prevent replay attacks as well[13]. Alternatives are prepending the length or randomising the output transformation.

For the simplest case where $n = m$, the attack goes as follows. After observing $2^{m/2}$ text-MAC pairs, an opponent expects to find two texts (say x and x') with the same MAC result (cf. the birthday paradox). If g is a permutation, this implies that $H_t = H'_t$. If the MAC algorithm is deterministic, it follows that $h_K(x \,\|\, y) = h_K(x' \,\|\, y)$ for any string y. Hence, if an opponent obtains $h_K(x \,\|\, y)$ with a chosen text attack, he can perform a forgery on $x' \,\|\, y$.

Note that further optimisations of the attack are possible, which reduce the number of known texts. A more serious concern is that for certain MAC algorithms, the forgery attack can be extended to a key recovery attack.

Constructions for MACs

Compared to the number of block ciphers and hash functions, relatively few MAC algorithms have been proposed. The main

[12] B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Advances in Cryptology, Proc. Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.

[13] D. Davies, W. Price, *Security for Computer Networks*, 2nd ed., Wiley, 1989.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

reason is that MACs have been derived from other primitives (initially from block ciphers and currently from hash functions), which reduces the need for custom designed proposals. During the last years, remarkable improvements have been achieved by information-theoretic secure MACs; the progress in this area is briefly discussed.

MAC algorithms based on block ciphers

The most popular MAC algorithm is certainly CBC-MAC; it has been included in several standards including ANSI X9.9:1986, ANSI X9.19:1986, ISO 8731:1987, and ISO/IEC 9797:1993. It is widely used with DES as the underlying block cipher. CBC-MAC is an iterated MAC, with the following compression function:

$$H_i = E_K(H_{i-1} \oplus x_i)\,, \quad 1 \le i \le t\,,$$

with $H_0 = 0$. An output transformation is needed to preclude the following simple forgery: given $h_K(x)$, $h_K(x\|y)$, and $h_K(x')$, one knows that $h_K(x'\|y') = h_K(x\|y)$ if $y' = y \oplus h_K(x) \oplus h_K(x')$.

One approach is for g to select the leftmost m bits. However, L.R. Knudsen has shown that the simple attack can be extended to this case[14]; it requires then approximately $2^{(n-m)/2}$ chosen texts and 2 known texts.

A stronger and widely used alternative is to replace the processing of the last block by a two-key triple encryption (with keys $K_1 = K$ and K_2); this is commonly known as the ANSI retail MAC, since it first appeared in ANSI X9.19:

$$g(H_t) = E_{K_1}(D_{K_2}(H_t))\,.$$

Here D denotes decryption. This mapping requires little overhead, and has the additional advantage that it precludes an exhaustive search against the 56-bit DES key.

All these variants are vulnerable to the birthday forgery attack, which requires a single chosen message and about 2^{32} known messages (if DES is used with $m = 64$). If $m = 32$, an additional 2^{32} chosen messages are required. Note that with a fast DES implementation on a PC, this number of texts can be collected in a single day. Bellare et al. provide a proof of security for CBC-MAC, i.e., they establish a lower bound to break the system under certain assumptions on the block cipher. It almost matches the upper bound provided by the birthday forgery attack.

For the ANSI retail MAC, 2^{32} known texts are sufficient for a key recovery requiring $3 \cdot 2^k$ encryptions, compared to 2^{2k} encryptions for exhaustive search[15]. If DES is used, this implies that key recovery may become feasible. Another key recovery attack needs only a single known text, but requires about 2^k MAC verifications. Moreover, it reduces the effective MAC size from $\min(m, 2k)$ to $\min(m, k)$.

An alternative to CBC-MAC is RIPE-MAC, which adds a feedforward[16]:

$$H_i = E_K(H_{i-1} \oplus x_i) \oplus x_i\,, \quad 1 \le i \le t\,.$$

It has the advantage that the round function is harder to invert (even for someone who knows the secret key). An output transformation is needed as well.

XOR-MAC is another scheme based on a block cipher[17]. It is a randomized algorithm and its security can again be reduced

[14] L.R. Knudsen, "Chosen-text attack on CBC-MAC," *Electronics Letters*, Vol. 33, No. 1, 1997, pp. 48–49.

[15] B. Preneel, P.C. van Oorschot, "A key recovery attack on the ANSI X9.19 retail MAC," *Electronics Letters*, Vol. 32, No. 17, 1996, pp. 1568–1569.

[16] RIPE, *"Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040),"* *LNCS 1007*, A. Bosselaers and B. Preneel, Eds., Springer-Verlag, 1995.

[17] M. Bellare, R. Guérin, P. Rogaway, "XOR

to that of the block cipher. It has the advantage that it is parallellisable and that it is incremental, i.e., small modifications to the message (and to the MAC) can be made at very low cost. The use of random bits clearly helps to improve security, but it has a cost in practical implementations. Also, the performance is typically 50% slower than CBC-MAC.

Note that cryptanalysis of the underlying block cipher can often be extended to an attack on CBC-MAC, where an attacker typically obtains less information than in conventional cryptanalysis on the ECB mode. Examples can be found in the literature for linear and differential cryptanalysis of DES.

MAC algorithms based on hash functions

The availability of fast dedicated hash functions (mainly of the MD4-family) has prompted several proposals for MAC algorithms based on these functions. The first proposed constructions are the secret prefix and secret suffix methods which can be described as follows: $h_K(x) = h(K\|x)$, $h_K(x) = h(x\|K)$. However, the first one allows for extension attacks, and the second one opens the possibility of off-line attacks[12].

The next proposal is the secret envelope method, which can be described as $h_K(x) = h(K_1\|x\|K_2)$ (for example Internet RFC 1828). For this method, Bellare et al. provide a security proof based on the assumption that the compression function of the hash function is pseudo-random[18]. While this is an interesting result, it should be pointed out that the compression function of most hash functions has not been evaluated with respect to this property. It was shown that the birthday forgery attack, which requires about $2^{n/2}$ known texts, can be extended to a key recovery attack[19]. MDx-MAC, presented by B. Preneel and P.C. van Oorschot[12] extends the envelope method by also introducing secret key material into every iteration. This makes the pseudo-randomness assumption more plausible. Moreover, it precludes the key recovery attack by extending the keys to complete blocks.

HMAC is yet another variant, which uses a nested construction (also with padded keys)[20]:

$$h_K(x) = h(K_2\|h(x\|K_1))\,.$$

HMAC will be used for providing message authentication in the Internet Protocol. The security of HMAC is guaranteed if the hash function is collision resistant for a secret value H_0, and if the compression function itself is a secure MAC for 1 block (with the secret key in the H_i input and the message in the x_i input). While these assumptions are weaker, we believe that the the latter one still requires further validation for existing hash functions.

Custom designed MACs

The most important dedicated MAC algorithm is certainly the Message Authenticator Algorithm (MAA). MAA was designed by D. Davies in 1984 and became an

MACs: new methods for message authentication using block ciphers," *Advances in Cryptology, Proc. Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 15–28.

[18] M. Bellare, R. Canetti, H. Krawczyk, "Pseudorandom functions revisited: the cascade construction and its concrete security," *Proc. 37th Symposium on Foundations of Computer Science,"* IEEE, 1996.

[19] B. Preneel, P.C. van Oorschot, "On the security of two MAC algorithms," *Advances in Cryptology, Proc. Eurocrypt'96, LNCS 1070*, U. Maurer, Ed., Springer-Verlag, 1996, pp. 19–32.

[20] M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology, Proc. Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 1–15.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

ISO banking standard in 1987 (ISO 8731-2). Recently several weaknesses of MAA have been exposed[21]. The birthday forgery attack can be optimized; it requires only 2^{24} messages of 1 Kbyte; a corresponding key recovery attack needs 2^{32} chosen texts consisting of a single message block. The number of off-line multiplications for this attack varies between 2^{44} for one key in 1000 to about 2^{51} for one key in 50. This should be compared to about $3 \cdot 2^{65}$ multiplications for an exhaustive key search. Finally, several classes of weak keys of MAA have been identified.

Several MAC algorithms in use have not been published, such as the S.W.I.F.T. authenticator, and the Swedish algorithm Data Seal. Proprietary MAC algorithms which can process only short messages include Telepass 1, the DECT standard authentication algorithm, and the Sky Videocrypt system of British Sky Broadcasting.

Information-theoretic secure MACs

Information-theoretic secure MACs (also known as authentication codes or A-codes) have been introduced by Simmons in the seventies. In the eighties a large amount of research has been devoted to study these MACs from a theoretical viewpoint. Already in 1981 Carter and Wegman provide a more practical approach, which allows to reduce the key size almost independent of the message size. Building on this approach, major improvements have been found both in terms of speed of computation and the key size. Part of this has been driven by the establishment of connections between error-correcting codes and A-codes. Examples include the work of Halevi and Krawczyk[22] and Johansson[23]. Both constructions achieve speeds which are two to three times faster than any known hash function or MAC algorithm, and require only relatively short keys.

For practical applications, one still has to deal with the problem that the keys can be used only once. This can be avoided by generating the keys using a cryptographically strong pseudo-random string generator; the price paid is that the construction becomes only computationally secure, but the security model is well understood. For the same security level against forgery attacks, the MAC will be about twice as long. Both elements indicate that this approach will mainly be suited for high speed applications with long messages.

Concluding remarks

During the last five years significant progress has been made both in the cryptanalysis and in the design of secure and efficient hash functions and MAC algorithms. For many applications, these primitives will remain a key building block to achieve high performance at a reasonable cost.

When selecting these primitives, it is important to take into account the progress in cryptanalysis and the increasing potential of brute force attacks. For long term security (10 years or more), a collision resistant hash function should have a result of at least 160 bits, and the key size for a MAC algorithm should be at least 80 bits. For present day processors (with the exception of the low-end smart cards), the

[21] B. Preneel, V. Rijmen, P.C. van Oorschot, "A security analysis of the Message Autenticator Algorithm (MAA)," *European Transactions on Telecommunications*, Vol. 8, No. 5, 1997, in print.

[22] S. Halevi, H. Krawczyk, "MMH: Software message authentication in the Gbit/second rates," *Fast Software Encryption, LNCS 1267*, E. Biham, Ed., Springer-Verlag, 1997, pp. 172–189.

[23] T. Johansson, "Bucket hashing with a small key size," *Advances in Cryptology, Proc. Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 149–162.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.

added cost for these parameters is certainly acceptable.

Appeared in *Information Security Technical Report 2(2)*, pp. 33–43, 1997.