Key recovery attacks on MACs based on properties of cryptographic APIs

Karl Brincat*[1] and Chris J. Mitchell[2]

[1] Visa International EU, PO Box 253, London W8 5TE, UK
brincatk@visa.com
[2] Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK
C.Mitchell@rhul.ac.uk

Abstract. This paper is concerned with the design of cryptographic APIs (Application Program Interfaces), and in particular with the part of such APIs concerned with computing Message Authentication Codes (MACs). In some cases it is necessary for the cryptographic API to offer the means to 'part-compute' a MAC, i.e. perform the MAC calculation for a portion of a data string. In such cases it is necessary for the API to input and output 'chaining variables'. As we show in this paper, such chaining variables need very careful handling lest they increase the possibility of MAC key compromise. In particular, chaining variables should always be output in encrypted form; moreover the encryption should operate so that re-occurrence of the same chaining variable will not be evident from the ciphertext.
Keywords: Message Authentication Code, cryptographic API, cryptanalysis

1 Introduction

MACs, i.e. *Message Authentication Codes*, are a widely used method for protecting the integrity and guaranteeing the origin of transmitted messages and stored files. To use a MAC scheme it is necessary for the sender and recipient of a message (or the creator and verifier of a stored file) to share a secret key K, chosen from some (large) keyspace. The data string to be protected, D say, is input to a MAC function f, along with the secret key K, and the output is the MAC. We write

$$\text{MAC} = f_K(D).$$

The MAC is then sent or stored with the message, i.e. the string which is transmitted or stored is $D||f_K(D)$ where $x||y$ denotes the concatenation of data items x and y.

In this paper we are concerned with a particular class of cryptographic APIs, namely those providing access to the functions of a cryptographic module. Such

* The views expressed in this paper are personal to the author and not necessarily those of Visa International

modules will typically provide a variety of cryptographic operations in conjunction with secure key storage, all within a physically secure sub-system. We moreover assume that the API is designed so that use may be made of the cryptographic functions without access being given to internally stored keys. We are concerned here with attacks which may, in certain circumstances, enable a user with temporary access to the cryptographic API to use its functions to discover an internally stored key.

2 APIs, MACs, and chaining variables

Most cryptographic modules have relatively limited amounts of internal memory. They also typically require the data they process to be passed as parameters in an API procedure call, since they typically will not have the means to directly access memory in their host system. As a result they can normally only compute a MAC on a data string of a certain maximum length. Thus provisions are often made in the cryptographic API for the module to compute a 'part MAC' on a portion of a data string. Computation of a MAC on the complete data string will then require several calls to the module.

Given that it is always desirable to minimise the stored state within the module, it is therefore necessary to provide the means to input/output 'partial MAC computation state information' to/from the module. This is normally achieved by providing for the input/output of 'Chaining variables' within the MAC processing procedure calls of the API. That is, it should be possible to input and output the value of H_i for a certain i, as defined in Section 3 below.

Based on this idea, we can identify four different types of MAC computation call to a cryptographic module:

- *Type A: ('all')* where the entire data string to be MACed is passed in a single call and a MAC is output (no chaining variables are input or output),
- *Type B: ('beginning')* where the first part of a data string is passed to the module (and a chaining variable is output but not input),
- *Type M: ('middle')* where the central part of a data string is passed to the module (and chaining variables are input and output), and
- *Type E: ('end')* where the last part of a data string is passed to the module (and a chaining variable is input but not output, and a MAC is output).

3 On the computation of MACs

MACs are most commonly computed using a block cipher in a scheme known as a CBC-MAC (for *Cipher Block Chaining* MAC). There are a number of variants on the basic CBC-MAC idea, although the following general model (see [1, 2]) covers most of these variants.

The computation of a MAC on a data string D, assumed to be a string of bits, using a block cipher with block length n, is performed with the following steps.

1. *Padding and splitting.* The data string D is subjected to a padding process, involving the addition of bits to D, the output of which (the *padded string*) is a bit string of length an integer multiple of n (say qn). The padded string is divided (or 'split') into a series of n-bit blocks, $D_1, D_2, \ldots, D_q$.
2. *Initial transformation.* An Initial transformation I, possibly key-controlled, is applied to D_1 to yield the first *chaining variable* H_1, i.e. $H_1 = I(D_1)$.
3. *Iteration.* Chaining variables are computed as $H_i = e_K(D_i \oplus H_{i-1})$ for i successively equal to $2, 3, \ldots, q$, where K is a block cipher key, and where $\oplus$ denotes bit-wise exclusive-or of n-bit blocks.
4. *Output transformation.* The n-bit *Output block* $G = g(H_q)$, where g is the output transformation (which may optionally be key-controlled).
5. *Truncation.* The MAC of m bits is set equal to the leftmost m bits of G.

The relevant international standard, namely ISO/IEC 9797-1, [1], contains six different CBC-MAC variants. Four are based on combinations of two Initial and three Output transformations. The various Initial and Output transformations have been introduced to avoid a series of attacks possible against the 'original' CBC-MAC (using Initial transformation 1 and Output transformation 1).

- Initial transformation 1 is: $I(D_1) = e_K(D_1)$ where K is the key used in the Iteration step, i.e. it is the same as the Iteration step.
- Initial transformation 2 is: $I(D_1) = e_{K''}(e_K(D_1))$ where K is the key used in the Iteration step, and $K'' \neq K$.
- Output transformation 1 is: $g(H_q) = H_q$, i.e. the identity transformation.
- Output transformation 2 is: $g(H_q) = e_{K'}(H_q)$, where $K' \neq K$.
- Output transformation 3 is: $g(H_q) = e_K(d_{K'}((H_q))$, where $K' \neq K$.

These options are combined in the ways described in Table 1 to yield four of the six different CBC-MAC schemes defined in ISO/IEC 9797-1, [1].

Table 1. CBC-MAC schemes defined in ISO/IEC 9797-1

Algorithm number	Input transformation	Output transformation	Notes
1	1	1	The 'original' CBC-MAC scheme, [3].
2	1	2	K' may be derived from K.
3	1	3	CBC-MAC-Y, [4]. The values of K and K' shall be chosen independently.
4	2	2	K'' shall be derived from K' in such a way that $K' \neq K''$.

Finally note that three Padding Methods are defined in [1]. Padding Method 1 simply involves adding between 0 and $n-1$ zeros, as necessary, to the end of the data string. Padding Method 2 involves the addition of a single 1 bit at the end of the string followed by between 0 and $n-1$ zeros. Padding Method 3

involves prefixing the string with an n-bit block encoding the bit length of the string, with the end of the string padded as in Method 1. When using a MAC algorithm it is necessary to choose one of the padding methods and the degree of truncation.

4 Attacks on CBC-MACs

There are two main types of attack on MAC schemes. In a *MAC forgery* attack, an unauthorised party is able to obtain a valid MAC on a message which has not been produced by the holders of the secret key. Typically the attacker will use a number of valid MACs and corresponding messages to obtain the forgery. A *key recovery* attack enables the attacker to obtain the secret MAC key. The attacker will typically need a number of MACs to perform such an attack, and may require considerable amounts of off-line computation. Note that a successful key recovery attack enables the construction of arbitrary numbers of MAC forgeries. In this paper we are exclusively concerned with key recovery attacks.

All attacks require certain resources (e.g. one or more MACs for known data strings). Clearly the less resources that are required for an attack, the more effective it is. As a result we introduce a simple way of quantifying an attack's effectiveness.

Following the approach used in [1], we do this by means of a four-tuple which specifies the size of the resources needed to the attacker. For each attack we specify the tuple $[a, b, c, d]$ where a is the number of off-line block cipher encipherments (or decipherments), b is the number of known data string/MAC pairs, c is the number of chosen data string/MAC pairs, and d is the number of on-line MAC verifications. The reason to distinguish between c and d is that, in some environments, it may be easier for the attacker to obtain MAC verifications (i.e. to submit a data string/MAC pair and receive an answer indicating whether or not the MAC is valid) than to obtain the genuine MAC value for a chosen message.

5 Key recovery attacks

In order to understand why the large number of CBC-MAC variants exist, we need to discuss some elementary key recovery attacks.

MAC algorithm 1 can be attacked given knowledge of one known message/MAC pair (assuming that $m > k$). The attacker simply recomputes the MAC on the message with every possible key, until the key is found giving the correct MAC. This attack has complexity $[2^k, 1, 0, 0]$, which is feasible if k is sufficiently small. E.g., if the block cipher is DES then $k = 56$, and it has been shown, [5], that a machine can be built for a few hundred thousand dollars which will search through all possible keys in a few days.

MAC algorithm 2 is subject to a similar attack. However, MAC algorithm 3 (CBC-MAC-Y) is not subject to this attack, which is one reason that it has been adopted for standardisation. In fact the best known key recovery attack

on this MAC algorithm has complexity $[2^{k+1},2^{n/2},0,0]$, as described in [6]. An alternative key recovery attack, requiring only one known MAC/data string pair, but a larger number of verifications, is presented in [7]; this attack has complexity $[2^k,1,0,2^k]$.

Whilst there are many situations where MAC Algorithm 3 is adequate, in some cases the above-referenced key recovery attacks pose a threat to the secrecy of the MAC key. One example of a scenario where this might be true is where users of a service (e.g. banking or mobile telephony) are issued with tamper-resistant devices containing unique MAC keys. It might be possible for an individual responsible for shipping these devices to consumers to interrogate devices for a considerable period of time before passing them to their intended user. This interrogation might involve generating and/or verifying large numbers of MACs. If such an interrogation could be used to obtain the secret key, then the security of the system might be seriously compromised. This is precisely the case for GSM implementations using the COMP128 algorithm, where, as described in recent postings on the web, [8], access to a SIM (Subscriber Identity Module) by a retailer could enable the authentication key to be discovered prior to a SIM being issued to a customer.

This motivates the development of MAC algorithm 4 (first described in [7]), which was designed to offer improved security relative to MAC algorithm 3 at the same computational cost. Unfortunately, as shown in [9], MAC algorithm 4 only offers a significant gain in security with respect to key recovery attacks if Padding Method 3 is used.

6 Chaining variable protection

6.1 The need for encryption

Suppose the chaining variable H_i is passed to and from the module in unencrypted form. Suppose also that an attacker has temporary access to the API of the cryptographic module, and wishes to recover the MAC key.

We first consider the case of MAC algorithm 3, where we suppose that an independent pair of keys (K,K') is used. A single use of the 'Type B' call to the MAC computation function will return a chaining variable which depends only on the first key K (the chaining variable is essentially a MAC as generated by MAC algorithm 1). Knowledge of this one chaining variable, and of the data string used to yield it, will be sufficient to recover K in an attack of complexity $[2^k,1,0,0]$. Given an additional MAC/data string pair, the other key K' can be recovered with a similar 'brute force' search, and hence we have recovered the entire key at a total complexity of $[2^{k+1},2,0,0]$. That is, MAC algorithm 3 is now no more secure than MAC algorithms 1 and 2.

The same attack applies to MAC algorithm 4, and thus, for APIs supporting MAC algorithms 3 and/or 4, encryption of the chaining variable is essential.

6.2 Simple encryption is not enough

We next see that, for MAC algorithm 4, encryption is not always sufficient. As we noted above, MAC algorithm 4 is best used in conjunction with Padding Method 3. However, although use of this padding method prevents the attacks described in [9], it does not prevent attacks made possible by the availability of encrypted chaining variables. This is based on the assumption that if the same chaining variable is output twice, then the encrypted version of that chaining variable will be the same on the two occasions. This will enable us to find a 'collision', which can be used as part of a key recovery attack. We next describe such an attack.

The attacker first chooses two n-bit blocks: D_1 and D_1'. These will represent the first blocks of padded messages, and hence they will encode the bit length of the messages (since Padding Method 3 is being used). Typically one might choose D_1 to be an encoding of $4n$ and D_1' to be an encoding of $3n$, which will mean that D_1 will be the first block of a 5-block padded message and D_1' will be the first block of a 4-block padded message. For the purposes of the discussion here we suppose that D_1 encodes a bit-length resulting in a $(q+1)$-block padded message, and D_1' encodes a bit-length resulting in a q-block padded message ($q \geq 4$).

The attacker now acquires $2^{n/2}$ 'Type B' MAC computation calls to the cryptographic module for a set of $2^{n/2}$ two-block 'part messages' of the form D_1, X, where X varies at random. The attacker also acquires a further $2^{n/2}$ 'Type B' calls for the $2^{n/2}$ two-block 'part messages' of the form D_1', Y, where Y varies at random. By routine probabilistic arguments (called the 'birthday attack', see [10]), there is a good chance that two of these 'part messages' will yield the same chaining variable. We are assuming that this will be apparent from the encrypted versions of the chaining variables.

The attacker now has a pair of n-bit block-pairs: (D_1, D_2) and (D_1', D_2') say, which should be thought of as the first pair of blocks of longer padded messages, with the property that the 'partial MACs' for these two pairs are equal, i.e. so that if

$$\begin{aligned} H_1 &= e_{K''}(e_K(D_1)), \\ H_2 &= e_K(D_2 \oplus H_1), \\ H_1' &= e_{K''}(e_K(D_1')), \quad \text{and} \\ H_2' &= e_K(D_2' \oplus H_1'), \end{aligned}$$

then $H_2 = H_2'$.

The remainder of the attack is a modified version of Attack 1 from [9]. The attacker next acquires $2^{n/2}$ 'Type A' calls to the cryptographic module for a set of $2^{n/2}$ $(q+1)$-block padded messages of the form

$$D_1, D_2, X_1, X_2, \ldots, X_{q-1},$$

where $X_1, X_2, \ldots, X_{q-1}$ can be arbitrary. As previously, there is a good chance that two of these messages will yield the same MAC. Suppose the two padded

strings are $D_1, D_2, E_3, E_4, \ldots, E_{q+1}$ and $D_1, D_2, E'_3, E'_4, \ldots, E'_{q+1}$, and suppose that the common MAC is M.

Now submit the following two padded strings for MACing, namely:

$$D'_1, D'_2, E_3, E_4, \ldots, E_q$$

and

$$D'_1, D'_2, E'_3, E'_4, \ldots, E'_q.$$

If we suppose that the MACs obtained are M' and M'' respectively, then we know immediately that

$$d_{K'}(M') \oplus E_{q+1} = d_K(d_{K'}(M)) = d_{K'}(M'') \oplus E'_{q+1}.$$

Now run through all possibilities L for the unknown key K', and set $x(L) = d_L(M')$ and $y(L) = d_L(M'')$. For the correct guess $L = K'$ we will have $x(L) = d_{K'}(M')$ and $y(L) = d_{K'}(M'')$, and hence $E_{q+1} \oplus x(L) = E'_{q+1} \oplus y(L)$. This will hold for $L = K'$ and probably not for any other value of L, given that $k < n$ (if $k \geq n$ then either a second 'collision' or a larger brute force search will probably be required).

Having recovered K', we do an exhaustive search for K using the relation $d_{K'}(M') \oplus E_{q+1} = d_K(d_{K'}(M))$ (which requires 2^k block cipher encryptions). Finally we can recover K'' by exhaustive search on any known text/MAC pair, e.g. from the set of $2^{n/2}$, which again will require 2^k block cipher encryptions.

It follows that the above attack has complexity [2^{k+2},0,3.$2^{n/2}$,0], which is not significantly greater than the complexity of the best known key recovery attacks against MAC algorithm 3.

Thus it is important that the encryption is performed in such a way that if the same chaining variable is output twice then this is not evident from the ciphertext. This can be achieved in a variety of ways. One possibility is to encrypt the chaining variable using a randomly generated session key, and to encrypt this session key with a long term key held internally to the cryptographic module. The encrypted session key can be output along with the encrypted chaining variable.

6.3 Message length protection

If Padding Method 3 is used, then it is clearly necessary to inform the cryptographic module of the total message length when the first 'Type B' call is made, so that the Padding Block can be constructed and used to perform the first part of the MAC calculation. We next show that, along with the chaining variable, it is important for the on-going MAC calculation to 'keep track' of this message length, and of the amount of data so far processed.

Suppose this is not the case. Then a simpler variant of the attack described in Section 6.2 above becomes possible. This operates as follows.

The attacker arranges for the part MAC to be computed on the padded two-block 'part message' (D_1, D_2) using a 'Type B' MAC computation call to

obtain the resulting chaining variable H_2, which may or may not be an encrypted version of the chaining variable resulting from the MAC operation. The attacker then assembles approximately $2^{n/2}$ distinct message strings $X_1, X_2, \ldots, X_{q-1}$ ($q \geq 3$) and arranges for their MACs to be computed using as many 'Type M' and 'Type E' calls as necessary, and with H_2 as the initial chaining variable input to the first 'Type M' call. The attacker effectively finds the MACs for the message strings $D_1, D_2, X_1, \ldots, X_{q-1}$. There is a good chance that two of these messages will yield the same MAC. Suppose that the two padded strings are $D_1, D_2, E_3, \ldots, E_{q+1}$ and $D_1, D_2, E'_3, \ldots, E'_{q+1}$. Denote the common MAC by M.

Since we are assuming that the cryptographic module does not keep track of the message length, it is not important whether the contents of block D_1 match the true length of the rest of the message. The MAC values obtained may not be true MACs of the (padded) input strings using Padding Method 3. Our objective is to recover the secret keys and not to present verifiable MAC forgeries. The fact that the calculated MAC values are not true MACs is unimportant and bears no consequence on the key recovery attack to be described.

The attacker now submits as many 'Type M' and 'Type E' calls as necessary to obtain the 'MAC' value for the two strings $D_1, D_2, E_3, \ldots, E_q$ and $D_1, D_2, E'_3, \ldots, E'_q$ using the partial MAC H_2 for the input pair (D_1, D_2) as chaining variable input to the first 'Type M' call in each case. Denote the final 'MAC' outputs by M' and M'' respectively. As before, we know immediately that

$$d_{K'}(M') \oplus E_{q+1} = d_K(d_{K'}(M)) = d_{K'}(M'') \oplus E'_{q+1}.$$

The attack now proceeds as in the previous case to recover K', K and finally K''. The complexity of this version of the attack is $[2^{k+2},0,2^{n/2},0]$, which is less than the complexity of the attack described previously.

The main conclusion we can draw from this attack is that, when Padding Method 3 is used, a further variable should be input and output along with the chaining variable. This variable should indicate the number of data bits remaining to be processed. In addition, integrity protection should be deployed over the entire set of input/output variables, to prevent modifications being made.

7 Other issues

The attacks described above are only examples of possible weaknesses of APIs when 'partial' MAC calculations are performed. We now mention one other area where problems may arise.

In some applications it is desirable to have a cryptographic module which can exist in two operational modes. In one mode MAC calculations are possible, but in the other mode only MAC verifications are possible. Indeed, by implementing such a scheme, it is possible to gain some of the desirable properties of a digital signature, simply by using a MAC.

To implement such a scheme will require two different sets of API procedure calls, one for MAC computation and one for MAC verification. This is simple enough, except when we consider the issue of verifying a MAC on a data string which is too long to be handled in one call to the module. In such a case it will be necessary to implement 'Types B, M and E' procedure calls for MAC verification. However, it should be clear that the 'Type B' and 'Type M' calls will be indistinguishable from the corresponding calls for a MAC computation. This poses significant risks to the separation of MAC computation and verification, and will need to be analysed carefully in the design of any module API.

8 Summary and conclusions

There exist Cryptographic APIs and Cryptographic Modules following the model discussed in this paper. Sometimes Cryptographic APIs do not pass a chaining variable value but rather a pointer to the memory location of the chaining variable (eg [11]). It is not clear whether this memory is accessible or not by a potential attacker – if this memory is part of the cryptographic module, then presumably it is not accessible; otherwise, the attacks described here are possible. It has not always been possible to obtain the necessary detail about the format of the chaining variables produced by the investigated cryptographic modules and APIs (eg [12, 11, 13, 14]). This is understandable as this information may be regarded as proprietary and sensitive by the manufacturers. The recent publication of [1] means that MAC algorithm 4, as described there, has probably not yet been implemented by many manufacturers, and thus the comments and attacks presented here should be considered as illustrations of what could happen, rather than actual attacks on any APIs and cryptographic modules in use today. However, the general comments on block-cipher based MACs may be relevant even to products in use today, especially if the chaining variables are not encrypted or their integrity is not protected.

When chaining variables have to be imported and exported from a cryptographic module, it is important that they be output in encrypted form. Moreover, the encryption should operate in such a way that if the same chaining variable is ever output twice, then this will not be evident from the ciphertext.

Moreover, when Padding Method 3 is used, a further variable should be input and output along with the chaining variable. This variable should indicate the number of data bits remaining to be processed. In addition, integrity protection should be deployed over the entire set of input/output variables, to prevent modifications being made.

References

1. International Organization for Standardization Genève, Switzerland: ISO/IEC 9797–1, Information technology — Security techniques — Message Authentication Codes (MACs) — Part 1: Mechanisms using a block cipher. (1999)

2. Preneel, B., van Oorschot, P.: On the security of iterated Message Authentication Codes. IEEE Transactions on Information Theory **45** (1999) 188–199
3. American Bankers Association Washington, DC: ANSI X9.9–1986 (revised), Financial institution message authentication (wholesale). (1986)
4. American Bankers Association Washington, DC: ANSI X9.19, Financial institution retail message authentication. (1986)
5. Electronic Frontier Foundation: Cracking DES: Secrets of encryption research, wiretap politics & chip design. O'Reilly (1998)
6. Preneel, B., van Oorschot, P.: A key recovery attack on the ANSI X9.19 retail MAC. Electronics Letters **32** (1996) 1568–1569
7. Knudsen, L., Preneel, B.: MacDES: MAC algorithm based on DES. Electronics Letters **34** (1998) 871–873
8. Wagner, D.: GSM cloning. `http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html` (1999)
9. Coppersmith, D., Mitchell, C.: Attacks on MacDES MAC algorithm. Electronics Letters **35** (1999) 1626–1627
10. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1997)
11. IBM: (IBM PCI Cryptographic Coprocessor) `http://www-3.ibm.com/security/cryptocards/html/overcca.shtml`.
12. Baltimore: (KeyTools Overview) `http://www.baltimore.com/keytools/`.
13. Microsoft: (CryptoAPI Tools Reference) `http://msdn.microsoft.com/library/psdk/crypto/cryptotools_0b11.htm`.
14. RSA Laboratories: PKCS#11 Cryptographic Token Interface Standard. (1997) Version 2.01, `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11`.