Minding Your MAC Algorithms*

Helena Handschuh, Gemplus, France
Bart Preneel, K.U.Leuven, Belgium

Abstract. *In spite of the advantages of digital signatures, MAC algorithms are still widely used to authenticate data; common uses include authorization of financial transactions, mobile communications (GSM and 3GPP), and authentication of Internet communications with SSL/TLS and IPsec. While some MAC algorithms are part of 'legacy' implementations, the success of MAC algorithms is mainly due to their much lower computational and storage costs (compared to digital signatures). This article describes a list of common pitfalls that the authors have encountered when evaluating MAC algorithms deployed in commercial applications and provides some recommendations for practitioners.*

1 Introduction

MAC algorithms compute a short string as a complex function of a message and a secret key. In a communications setting, the sender will append the MAC value to the message (see also Fig. 1). The recipient shares a secret key with the sender. On receipt of the message, he recomputes the MAC value using the shared key and verifies that it is the same as the MAC value sent along with the message. If the MAC value is correct, he can be convinced that the message originated from the particular sender and that it has not been tampered with during the transmission. Indeed, if an opponent modifies the message, the MAC value will no longer be correct. Moreover, the opponent does not know the secret key, so he is not able to predict how the MAC value should be modified.

* The work described in this paper has been partly supported by the European Commission under contract IST-2002-507932 (ECRYPT). The information in this paper is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

The main security properties of a MAC algorithm is that one should not be able to forge MAC values, that is, to predict values on new messages without knowing the secret key. A second requirement is that it should be computationally infeasible to recover the MAC key by exhaustive search, since an exhaustive key search allows for arbitrary forgeries.

Historically the first commercial deployment of MAC algorithms has been made by the financial sector (wholesale and retail banking, including electronic purses, debit and credit cards). On the Internet, MAC algorithms are used for authenticating data at the transport layer (TLS or Transport Layer Security, the successor of SSL) and at the network layer (IPsec, both in the AH and the ESP transforms). The GSM operators use a proprietary algorithm in the smart cards (SIMs or Subscriber Identification Modules) to authenticate the users. For 3rd generation systems (3GPP) the Milenage algorithm has been proposed as the example algorithm for user authentication; signalling information is also authenticated using the KASUMI block cipher.

The most popular MAC algorithms are the variants of CBC-MAC which are based on a block cipher; in the past this has been mostly DES or triple-DES and currently AES is becoming more popular. Since the mid 1990s, constructions based on hash functions such as HMAC have been introduced on the Internet. All these algorithms have been included in a large number of standards (IETF, FIPS, ANSI, ISO, ...); the most comprehensive is the 3-part standard IS 9797 published by ISO/IEC [10].

In the following, the key length in bits of a MAC algorithm is denoted with k and the length in bits of the MAC result is denoted with m.

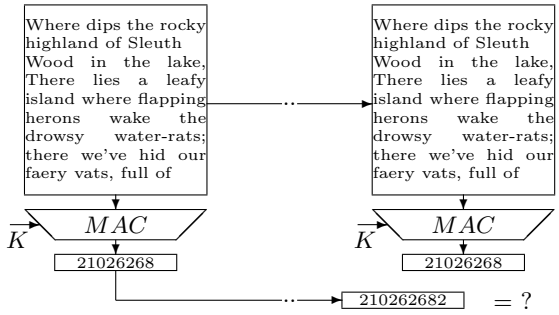

Fig. 1. Using a MAC Algorithm for data authentication.

2 Common Pitfalls

Throughout this paper, we analyze problems which we encountered while evaluating MAC algorithms. Several of these algorithms were flawed and as a result, we present simple recommendations on constructions that should be avoided.

2.1 Recommendation 1: Don't use proprietary algorithms without additional evaluation

Kerckhoffs' principle, which dates back to the 19th century, states that the secrecy of a cryptographic algorithm should only rely on the secrecy of the key (and not on that of the algorithm itself). Nevertheless, it is a common belief in industry that cryptographic algorithms should be kept secret. While this might be the only solution for constrained environments in which there are insufficient resources to deploy standard cryptographic algorithms (for example not enough gates for hardware designs or memory cells for embedded devices), most secret designs are far less secure than published algorithms. Many proprietary designs have not been subjected to extensive cryptanalysis; when they leak out, flaws are discovered which would have been identified immediately if the algorithms would have been published before implementation. This does not necessarily mean that all proprietary designs are flawed; numerous designs are based on sound publicly known building blocks such as AES or SHA-1, but security by obscurity has rarely proven successful. As an example, we consider two algorithms which leaked out in the last years: alleged SecureID [5] and alleged Comp128 [9]. The first algorithm is used for authenticating users to their corporate VPN. It computes a one-time password as a MAC value computed on the current time using a secret key. This MAC value is provided to the VPN server along with the username and password to open a secure session. The second algorithm is used in a challenge-response protocol to authenticate a user to the GSM network. It derives a MAC value from a random challenge provided by the network and a secret key stored in the SIM card of the user. The MAC value is sent back to the network which verifies it before giving access to the network. Both algorithms were cryptanalyzed immediately after publication; they are both vulnerable to collision attacks. It takes a few hours only to recover the secret key of a SIM card and a few months to find the key for a SecurID token. Therefore, secret designs should not be trusted unless sufficient independent evaluation of the algorithms has been performed and documented.

2.2 Recommendation 2: Don't use short MAC lengths

The length in bits of a message authentication code is directly related to the number of trials that an adversary has to perform before a message (and its associated MAC value) is accepted. For a MAC value of bit-length m, the adversary has to perform on average 2^{m-1} random on-line MAC verifications before his strategy succeeds. Therefore MAC values of 16 bits require about 32 000 trials, which

is rather straightforward in a fully automated setting (for example over the internet). Therefore, 16-bit MAC values should not be used. The minimum reasonable length for a MAC is 32 bits; this corresponds to about 2 billion or $2 \cdot 10^9$ trials. For high value applications 64 bits are more appropriate.

2.3 Recommendation 3: Don't use single DES or short keys

Another common pitfall is to use strong publicly known algorithms with too short keys. For example the use of truncated 40-bit keys or even single DES, which has a 56-bit key, are strongly discouraged. Ten years ago, these algorithms would have given adequate protection for low-security applications. However, today they are vulnerable to straightforward exhaustive search attacks, in which an attacker simply tries out every possible value for the key and compares the computation results to the valid MACs available to him. For a key of bit-length k, 2^{k-1} trials on average are required to find the correct key. The current record for exhaustive key search is less than a day for a single 56-bit DES key on specialized hardware combined with additional distributed computing. With a 1 million US$ key search machine, a DES key could be recovered in about 1 minute, which means that the cost per key is about 1 US$ [22]. It is therefore recommended that keys of at least 80-bits should be used. An exhaustive search for an 80-bit key would take 2^{24} (about 16 million) times more effort, which is in 2004 out of range for a normal attacker.

It has often been claimed that keys of MAC algorithms can be shorter than keys for encryption algorithms: in most cases authenticity is verified immediately, while confidentiality needs to be protected for 5 to 10 years (or even more). However, it should be pointed out that in the case of a MAC algorithm, an opponent can keep searching for keys and restart the key search machine every time a new text-MAC pair is observed: therefore, one should *not* consider the lifetime of a single MAC key, but the lifetime of the system, which can often be close to 10 years as well.

2.4 Recommendation 4: Don't use unkeyed hash functions as MAC algorithms

Hash functions are functions which transform a variable-length message into a fixed-length scrambled image of it. In addition, they cannot be inverted (i.e., one cannot recover the message from the hash value) and it is very hard to find two different messages which hash to the same value (this is called a collision). These functions are sometimes thought to have good properties for generating MACs. This may well be the case, but they should never be used without a key: anybody can forge a new valid MAC on a new message by simply computing the hash value on this message and appending it.

Now let's examine what happens when the hash function is used in combination with a secret key. Most popular hash functions are based on the simple iteration of a compression function f, that takes inputs of fixed size; they are known as iterated hash functions. These hash functions divide the message M into blocks $M = M_1, M_2, \ldots, M_t$ and apply the following recursive formula to compute $h(M)$:

$$H_i = f(H_{i-1}, M_i)\,,$$

where H_0 represents a fixed public initial value and the hash value $h(M) = H_t$. If an iterated hash function is used and the key K is prepended to the message M, i.e., the MAC value is given by $MAC = h(K||M)$ where $||$ is the symbol for concatenation, then it is straightforward to compute the MAC of a different message $M||M'$ from MAC as follows: $MAC' = h(K||M||M') = f(MAC, M')$. Note that one does not need to know the secret key to perform this computation, which shows that this method is not secure. Truncating the MAC to fewer bits than the output length of the hash function might help, but the security of this scheme remains doubtful. It has been shown that even a construction which computes a MAC on message M with key K as $h(K||M||K)$ may be insecure [19, 20]; the attack does not violate the security proof of [2], but illustrates its limitations when key recovery attacks are considered.

The recommended constructions for MACs based on unkeyed hash functions are HMAC [3] and MDx-MAC [18]. The HMAC function is a generic transformation that is (roughly) computed as $h(K||h(K'||M))$; MDx-MAC is more complex since it modifies the internals of the hash function.

2.5 Recommendation 5: Don't use a public or secret CRC

In some communication systems, a CRC (Cyclic Redundancy Check) is appended to a message before it is sent over a noisy channel to allow for detection of transmission errors. However these codes do not protect the integrity of a message or authenticate the origin of the message. They don't withstand malicious attacks, as they can only spot errors which occur randomly on the channel. Let us examine the binary case. A CRC consists of m bits that are appended to a message such that the resulting bit-string interpreted as a polynomial is a multiple of the CRC polynomial g of degree m. This shows that if one adds to a message with CRC a multiple of this polynomial (for example the polynomial g itself), this modification will not be detected. A CRC is usually implemented in the form of an LFSR (Linear Feedback Shift Register) of length m; the feedback coefficients of the LFSR correspond to the coefficients of the polynomial g. Thus for a given message, every bit of the resulting code is a linear combination of bits of the original message. If the formula of the CRC is public, this linear combination is known as well, because it does not depend on the message, only on the feedback polynomial g of the LFSR. If the CRC is secret, this combination is not known to the attacker.

If the CRC is public, it is easy to compute the MAC for a given message by simply computing the associated CRC for it, as there is no secret involved here. An example of a system which used such a weak method for data integrity is the IEEE 802.11 WEP standard (Wireless Equivalent Privacy) to secure Wireless LANs. This protocol computes a 32-bit CRC on a 128-bit nonce before encrypting the result with an additive stream cipher (RC4). This is highly insecure and allows for packet spoofing as pointed out by Borisov et al. [8].

If the CRC formula is secret, one can forge a new MAC from two known MACs. Assume that we have two equal-length messages M and M' with associated MAC values MAC and MAC'. As every output bit of the LFSR is the same linear combination of the message input bits, it is clear that the MAC value of the message $M \oplus M'$ equals $MAC \oplus MAC'$. Furthermore, the feedback polynomial g may be recovered using only a few chosen messages for which the corresponding MACs are obtained. Typically, $m + 1$ chosen messages of length m bits are required for an m-bit MAC or LFSR. Therefore public CRCs or even secret CRCs do not provide the security features expected from a MAC algorithm and should not be used for data integrity.

2.6 Recommendation 6: Don't even use a *keyed* public or secret CRC

One may believe that the weaknesses pointed out in the previous section can be solved by using a secret key, somehow prepended, exored or appended to the message before the CRC computation. This is not the case. Assume that the MAC value is computed on a message M as $CRC(K||M)$. Recall that for a public CRC, each bit m_i of the MAC is defined as a known linear combination of message bits M_i and key bits K_i. Let l be the length of the message in bits, then

$$m_i = \bigoplus_{j=1}^{l} a_j \cdot M_j \oplus \bigoplus_{j=1}^{k} b_j \cdot K_j$$

where all binary coefficients a_j and b_j are known. Then it is easy to flip a few bits in M and to compute the associated MAC for the new message: the linear combinations of key bits have not changed, and the linear combinations of message bits can be computed from the previous ones. For a secret CRC, the linear combinations are not known to the attacker; however, they can be discovered using a randomly chosen message M of length l bits, a carefully chosen message $M' = M||0 \ldots 0$ of length $l+m$ bits, and their respective m-bit MACs. These two

MACs can be concatenated to form a $2m$-bit sequence from which the feedback polynomial of the shift register can be reconstructed using a variant of the Berlekamp-Massey algorithm (see for example [15, pp. 200–202]. With another method, the feedback polynomial can be recovered using about $m + 1$ carefully chosen messages and their associated MACs.

2.7 Recommendation 7: Don't use simple CBC-MAC on variable-length inputs

CBC-MAC is an iterated MAC algorithm based on a block cipher. The encryption and decryption with the block cipher E using the key K will be denoted by $E_K(.)$ and $D_K(.)$ respectively. The block length of the block cipher is denoted with n, hence $E_K(.)$ and $D_K(.)$ are permutations on n-bit strings. The input M of the MAC algorithm is divided into t n-bit blocks denoted $M_1, M_2, \ldots, M_t$ which are processed as follows:

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

The initial value H_0 is equal to the all zero string (see also Recommendation 9). Often one applies an output transformation g to obtain the MAC value, or $\mathrm{MAC}_K(M) = G(H_t)$. If no output transformation is applied, one calls this scheme simple CBC-MAC. Figure 2 with IV the all zero string represents CBC-MAC with output transformation G keyed by a second key K'.

It has been proved by Bellare et al. [4] that if the block cipher E is a secure block cipher, CBC-MAC is a secure MAC algorithm if it is applied on input strings of fixed length. However, if CBC-MAC is used on variable-length inputs this proof does not apply; even worse, it is known that simple CBC-MAC on variable length messages is insecure. In the following M_1, M_1', M_2, and M_2' denote n-bit strings. Consider for simplicity a text consisting of a single block M_1 and $m = n$ (the attack applies in a more general case as well). Assume that one knows $\mathrm{MAC}_K(M_1)$; it follows immediately that

$$\mathrm{MAC}_K(M_1 \| (M_1 \oplus \mathrm{MAC}_K(M_1))) = \mathrm{MAC}_K(M_1)\,.$$

This implies that one can construct a new message with the same MAC value, which is a forgery. Note that this attack applies even if a MAC algorithm key is used only once. If one knows $\mathrm{MAC}_K(M_1)$ and $\mathrm{MAC}_K(M_1')$, a similar calculation shows that

$$\mathrm{MAC}_K(M_1 \| (M_1' \oplus \mathrm{MAC}_K(M_1))) = \mathrm{MAC}_K(M_1')\,.$$

If one knows $\mathrm{MAC}_K(M_1)$, $\mathrm{MAC}_K(M_1 \| M_2)$, and $\mathrm{MAC}_K(M_1')$, one knows that

$$\mathrm{MAC}_K(M_1' \| M_2') = \mathrm{MAC}_K(M_1 \| M_2)$$

if $M_2' = M_2 \oplus \mathrm{MAC}_K(M_1) \oplus \mathrm{MAC}_K(M_1')$. Again this allows for a forgery, as an adversary can forge the MAC on $M_1' \| M_2'$ given knowledge of three other text-MAC pairs.

In order to preclude these attacks, EMAC should be used (cf. Sect. 3), in which the output of simple CBC-MAC is encrypted using a different key, as represented in Fig. 2 (when the block cipher E is used as output transformation G). This variant has a security proof which is also valid for messages of variable-input length [17]. Another solution is OMAC [11].

Note that a common choice for the output transformation G is the selection of the leftmost $m/2$ bits. However, Knudsen has shown that for the common parameters $n = 64$, $m = 32$ one can forge a MAC based on only 130 000 text-MAC pairs, which is much less than anticipated [12].

2.8 Recommendation 8: Don't use the same key for MAC and encryption

We consider here constructions where a MAC is computed on a message that is also encrypted with the same key. The main reason for using different keys for encryption and MAC computation is that these keys may have completely different life cycles. As an example, the key for encryption may need to be backed up (for later recovery), while a MAC key may be thrown away after a message has been received and verified.

The other reason is that this combination may result in trivial attacks on the MAC algorithm. For

example, if the attacker knows a plaintext-ciphertext pair $(\tilde{X}, \tilde{Y})$ from the encryption, he can easily append in CBC-MAC the block $\text{MAC}_K(X) \oplus \tilde{X})$ and he knows that the resulting MAC value will be $\tilde{Y}$. A second trivial example is if one computes CBC-MAC on the plaintext and subsequently appends the MAC value to the plaintext. Encrypting the result in CBC-mode with initial value equal to 0 will result in an encrypted MAC value that is always equal to $E_K(0)$, independent of the plaintext.

As a clarification, this recommendation does not hold when a mode for authenticated encryption is used such as OCB [21], since such a mode is specifically designed to protect both authenticity and confidentiality in a transformation with a single key.

2.9 Recommendation 9: Don't use an IV with CBC-MAC

In this section we address the case of CBC-MAC used in combination with an initial value denoted by IV in Fig 2.

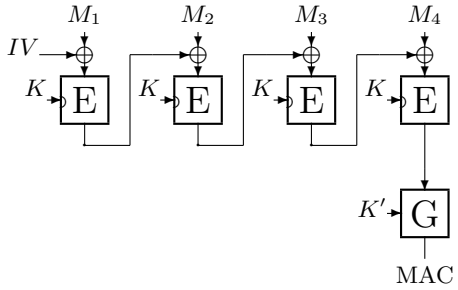

Fig. 2. CBC-MAC with initialization vector

Depending on how the IV itself is transmitted and protected in the scheme (encrypted and/or protected with a MAC, and/or synchronized with the receiver), this solution may be extremely insecure. Assume that the IV is sent in clear over the line and is not itself protected by a MAC nor synchronized with the receiver. Then the first block M_0 of each message can trivially be replaced by an active attacker by any other block N_0, provided that the IV is also changed to $\text{IV} \oplus M_0 \oplus N_0$. If this is the case, the value which is encrypted in the first encryption function is still equal to $\text{IV} \oplus M_0$. Therefore, the final MAC value is still valid, and the forged message will be accepted by the receiver. This remains true even if the IV is generated at random by the sender. A more subtle case is when the IV equals the previous MAC value $MAC*$. This value cannot be tampered with. However, as the attacker gets to collect messages M^i and their associated MACs MAC_i, he can forge new messages simply by adding the value $MAC* \oplus MAC_{i-1}$ to the first block of any pair (M^i, MAC_i) he keeps in his list. All these forged messages and their original MACs will be accepted as valid by the receiver.

2.10 Recommendation 10: Be careful when using Triple-DES

Recommendation 3 states that CBC-MAC based on DES (with a 56-bit key) should not be used. One year after the publication of DES in 1977, a strengthened version of DES was proposed called triple-DES (with two keys or with three keys). In 1986, the ANSI retail MAC algorithm was published, which is a CBC-MAC algorithm based on two-key triple-DES [1]: this particular scheme uses single DES throughout the calculation and triple-DES for the last block only; in most applications the output is truncated to 32 bits. The expectation was that the security would be increased significantly at the cost of only two additional encryptions. We will now show that this expectation has only been reached partially.

Forging a 32-bit ANSI retail MAC requires 2^{32} attempts. A more sophisticated forgery attack is known in which the opponent is sure that the forgery will succeed; it needs 2^{32} known text-MAC pairs and 2^{33} chosen text-MAC pairs; while this is not better than CBC-MAC based on single DES, this seems to be an acceptable security level. One the other hand, one would expect that a key recovery attack requires at least 2^{80} encryptions. However, it turns out that with 2^{32} chosen text-MAC pairs, the effort to recover the key can be reduced to either 2^{57} encryptions (only twice the effort of breaking single DES) or to 2^{48} MAC verifications.

In some applications, the strength of the ANSI retail MAC is further reduced by sending the two DES keys along with the message encrypted in ECB mode; this opens an avenue for a subtle attack in which the number of MAC verifications can be further reduced to 2^{32}. This means that the security against key search is reduced to the effort of forging a single MAC by brute force and exhaustive search of a single DES key.

Another example of subtle weaknesses of MAC algorithms based on triple-DES is the NIST proposal for RMAC based on three-key triple-DES [16]; it turns out that the 168-bit triple-DES key in the output transformation can be recovered using about 2^{56} chosen text-MAC pairs and 2^{57} encryptions [13].

3 Recommended MAC Algorithms

In view of the large number of well-analyzed and standardized MAC algorithms, there should be no reason to use insecure MAC algorithms that violate one or more of the recommendations put forward in this paper.

We recommend to use standardized MAC algorithms that can be found in the International Standard ISO/IEC 9797. This is a 2-part international standard: Part 1 describes MAC algorithms based on block ciphers and Part 2 describes MAC algorithms based on hash functions. Part 1 contains – in addition to some simpler CBC-MAC schemes for legacy purposes that are not recommended – the ANSI retail MAC as well as MacDES [14] for use with triple-DES and EMAC [17] that is recommended for use with AES. Recall that EMAC applies a simple CBC-MAC and encrypts its output using a different key. The standard contains a comparison of the security of all the schemes that have been included. It is likely that the next edition will also contain OMAC [11] (see Fig. 3).

Part 2 of the standard contains HMAC, MDx-MAC and a variant of MDx-MAC that is efficient for short messages. HMAC is clearly the most popular scheme, since it has also been standardized by the IETF and by NIST. We recommend using HMAC with hash functions such as RIPEMD-160 and SHA-1. They process the message in blocks of 512 bits. The resulting MAC value is computed as follows:

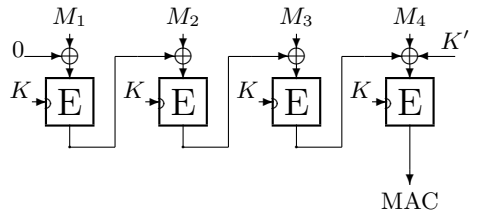

Fig. 3. The OMAC construction. The key K' is related to the original key K and takes one of two different values: exactly which one depends on the last message block requiring padding bits or not.

$$\text{MAC} = h((K \oplus opad) \| h((K \oplus ipad) \| x))$$

where K is the key padded with zeroes to a full block of 512 bits and *opad* and *ipad* are constant 512-bit strings (equal to 64 times the hexadecimal values '$\texttt{36}_\texttt{x}$' and '$\texttt{5c}_\texttt{x}$' respectively). It is equivalent to compute and store the partial hash value of the two padded key blocks $(K \oplus ipad)$ and $(K \oplus opad)$ first, and to compute the MAC value of the message starting from these pre-computed secret initial values.

Newer MAC algorithms such as PMAC [7] that allows for parallel operation and UMAC [6], that offers a very good performance for long messages are possible alternatives; so far these schemes have not been included in the existing standards, but their security has been evaluated thoroughly.

References

1. ANSI X9.19 *"Financial Institution Retail Message Authentication,"* American Bankers Association, August 13, 1986.
2. M. Bellare, R. Canetti, H. Krawczyk, "Pseudorandom functions revisited: The cascade construction and its concrete security," *Proceedings 37th Annual Symposium on the Foundations of Computer Science, IEEE*, 1996,

pp. 514–523. Full version `http://www.cs.ucsd.edu/users/mihir/papers/cascade.html`.

3. M. Bellare, R. Canetti, H. Krawczyk, "Keying hash functions for message authentication," *Advances in Cryptology, Proceedings Crypto'96, LNCS 1109*, N. Koblitz, Ed., Springer-Verlag, 1996, pp. 1–15. Full version `http://www.cs.ucsd.edu/users/mihir/papers/hmac.html`.
4. M. Bellare, J. Kilian, P. Rogaway, "The security of cipher block chaining," *Journal of Computer and System Sciences,* Vol. 61, No. 3, Dec. 2000, pp. 362–399. Earlier version in *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 341–358.
5. A. Biryukov, J. Lano, B. Preneel, "Cryptanalysis of the Alleged SecurID Hash Function," *Selected Areas in Cryptography, Proceedings SAC 2003, LNCS 3006*, M. Matsui, R. Zuccherato, Eds., Springer-Verlag, 2004, pp. 130–144.
6. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway, "UMAC: Fast and secure message authentication," *Advances in Cryptology, Proceedings Crypto'99, LNCS 1666*, M.J. Wiener, Ed., Springer-Verlag, 1999, pp. 216–233.
7. J. Black, P. Rogaway, "A block-cipher mode of operation for parallelizable message authentication," *Advances in Cryptology, Proceedings Eurocrypt'02, LNCS 2332*, L. Knudsen, Ed., Springer-Verlag, 2002, pp. 384–397.
8. N. Borisov, I. Goldberg, D. Wagner, "Intercepting mobile communications: the insecurity of 802.11," *Proceedings MOBICOM 2001*, pp. 180–189.
9. H. Handschuh, P. Paillier, "Reducing the collision probability of alleged Comp128," *Smart Card Research and Applications, Proceedings CARDIS '98, LNCS 1820*, J.-J. Quisquater, B. Schneier, Eds., Springer-Verlag, 2000, pp. 366–371.
10. ISO/IEC 9797 *"Information technology – Security techniques – Message Authentication Codes (MACs). Part 1: Mechanisms using a block cipher,"* 1999, *"Part 2: Mechanisms using a dedicated hash-function,"* 2002.
11. T. Iwata, K. Kurosawa, "OMAC: One key CBC MAC," *Fast Software Encryption, LNCS 2887*, T. Johansson, Ed., Springer-Verlag, 2003, pp. 129–153.
12. L. Knudsen, "Chosen-text attack on CBC-MAC," *Electronics Letters*, Vol. 33, No. 1, 1997, pp. 48–49.
13. L. Knudsen, T. Kohno, "Analysis of RMAC," *Fast Software Encryption, LNCS 2887*, T. Johansson, Ed., Springer-Verlag, 2003, pp. 182–191.
14. L. Knudsen, B. Preneel, "MacDES: MAC algorithm based on DES," *Electronics Letters*, Vol. 34, No. 9, 1998, pp. 871–873.
15. A.J. Menezes, P.C. van Oorschot, S. Vanstone, *"Handbook of Applied Cryptography,"* CRC Press, 1997.
16. NIST Special Publication 800-38B, *"Draft Recommendation for Block Cipher Modes of Operation: The RMAC Authentication Mode,"* October 2002.
17. E. Petrank and C. Rackoff, "CBC MAC for real-time data sources," *Journal of Cryptology*, Vol. 13, No. 3, 2000, pp. 315–338.
18. B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Advances in Cryptology, Proceedings Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.
19. B. Preneel, P.C. van Oorschot, "On the security of two MAC algorithms," *Advances in Cryptology, Proceedings Eurocrypt'96, LNCS 1070*, U. Maurer, Ed., Springer-Verlag, 1996, pp. 19–32.
20. B. Preneel, P.C. van Oorschot, "On the security of iterated Message Authentication Codes," *IEEE Trans. on Information Theory*, Vol. IT–45, No. 1, 1999, pp. 188–199.
21. P. Rogaway, M. Bellare, J. Black, "OCB: A block-cipher mode of operation for efficient authenticated encryption," *ACM Trans. Information and System Security (TISSEC)*, Vol. 6, No. 3, Aug. 2003, pp. 365–403.
22. M.J. Wiener, "Efficient DES key search," *Presented at the Rump Session of Crypto'93*. Reprinted in *"Practical Cryptography for Data Internetworks,"* W. Stallings, Ed., IEEE Computer Society, 1996, pp. 31–79.