A Note on the Fragility of the "Michael" Message Integrity Code

Avishai Wool, *Senior Member, IEEE*

***Abstract*—The IEEE 802.11 wireless local area network standard did not incorporate a cryptographic message integrity code into its wired equivalent privacy (WEP) protocol, and relied upon CRC-32 for message integrity. This was shown to be completely insecure since WEP uses a stream cipher (RC4) for encryption.**

The latest IEEE 802.11i draft addresses this, and other, weaknesses discovered in WEP. IEEE 802.11i suggests three new modes of operation: two based on the Advanced Encryption Standard cipher and one [temporal key integrity protocol (TKIP)] still based on RC4. The TKIP mode is intended for use on legacy hardware, which is computationally weak. TKIP uses a new, keyed, 64-b, message integrity code called Michael. In this letter, we highlight a weakness in Michael and suggest a simple fix.

***Index Terms*—Message authentication code, wireless security.**

I. Introduction

A. Background

NETWORK security is often seen as an application layer issue, to be addressed at the highest levels of the protocol stack. However, wireless local area network (WLAN) protocols require security components within layer 2, to protect both data confidentiality and access to the network. Specifically, hosts and access points need to be authenticated and traffic needs to be encrypted. Unfortunately, these security requirements are not always identified early enough in the design process, leading to standards and products whose security is weaker than it should be. This is precisely the case with the IEEE 802.11 WLAN standard [8], which uses the wired equivalent privacy (WEP) protocol for data confidentiality.

Millions of devices based on the IEEE 802.11 have been sold. However, over the last two years, several significant security problems have been discovered in the standard. WEP data integrity is vulnerable to attack [4] and its authentication mechanisms may be defeated [2]. Moreover, the encryption protocol used in WEP has been severely compromised [7], [12], and WEP-cracking software is widely available off the Internet.[1] See [14] for one possible reason why so many problems are encountered.

The upcoming IEEE 802.11i draft [9] addresses these weaknesses. IEEE 802.11i suggests three new modes of operation: two modes (WRAP and CCMP) based on the Advanced Encryption Standard (AES) cipher [1], and one mode [temporal key integrity protocol (TKIP)] still based on RC4 [10]. The rationale behind the multiple choices is that the more secure AES-based modes will require new hardware with more computation power. The TKIP mode is intended to be a temporary fix, whose purpose is to prolong the life of the fielded, computationally-weak, hardware. However, given the huge installed base of existing IEEE 802.11 equipment, it is likely that TKIP will remain in use for several years, and, therefore, it needs to be made as secure as possible, subject to the existing hardware's constraints.

B. Michael Message Integrity Code

One of the central ingredients in TKIP is a new message-integrity code (MIC) called Michael [5]. This is a new keyed cryptographic hash function, which produces a 64-b output. The MIC is designed to have a 20-b cryptographic strength.[2] In most applications, such an MIC would be considered much too weak for use.

The security of Michael relies on the assumption that the communication, and crucially, the MIC value, is encrypted and unknown to the attacker. This is a weak, and rather unusual, assumption to make. Cryptographic MICs are designed to, at the very least, resist known-plaintext attacks, in which both the message text and MIC value are assumed to be known to the cryptanalyst. Usually much stronger attacks (such as chosen plaintext, timing attacks, power/side-channel attacks, etc.) are considered. Apparently, this weak design choice was dictated to the Michael creator when his work was commissioned, and its goal was to keep the MICs CPU requirements down to a bare minimum.

Cryptographers consider fielding new and yet-unknown cryptographic functions to be highly risky, even reckless. This is even stated in the Michael design document [5]. In fact, it is surprising that the TGi working group is willing to adopt such a new design, when well understood, and very efficient, MIC designs (e.g., HMAC with SHA-1 or HMAC with MD5 [3]) are available, albeit with 128–160-b block lengths.

C. Contributions

In this letter, we highlight a surprising property in Michael: not only is it not one-way, in fact it is *invertible*. There exists a simple function, InvMichael, which can recover the secret MIC key k_{mic} given a single known message M and its matching MIC value $m = \mathrm{Michael}(M, k_{\mathrm{mic}})$.

In TKIP, the MIC value is encrypted by a stream cipher, RC4, which is seeded by a long-term key combined with a per-message initialization vector (IV). Thus, supposedly, even if the message text is known to the attacker, the MIC value is not. As long as the MIC value is kept secret, the InvMichael function cannot be applied to compute the MIC key.

However, the TKIP frame format *appends* the MIC value after the message body. This fact, coupled with the invertibility of Michael, lead to a related-message passive attack that breaks

Manuscript received Januay 13, 2003; accepted April 28, 2003. The editor coordinating the review of this paper and approving it for publication is S. Sarkar.

The author is with the Department of Electrical Engineering Systems, Tel Aviv University, Ramat Aviv 69978, Israel (e-mail: yash@acm.org).

Digital Object Identifier 10.1109/TWC.2004.833470

[1]The AirSnort Homepage; http://airsnort.sourceforge.net; 2002

[2]The MIC's creator claims that it has an effective 30-b cryptographic strength.

the MIC key k_{mic} if two messages with different lengths are ever encrypted using the same key stream.

The invertibility of Michael was certainly known to the MIC designer, though not explicitly mentioned in [5]. In fact, he even alludes to the related-message attack in a remark [5, p. 14]. However, the IEEE 802.11i standard draft does not suggest specific counter-measures for this type of attack. The draft relies solely on key-stream material not being reused, which is ensured through a careful selection of the IV values. We show that a minimal change to the TKIP message format can protect the MIC key from being broken, even if an incorrect TKIP implementation does occasionally reuse IV values. Such a change can be seen as a "defense in depth" design—using it, the MIC key is protected by two separate mechanisms.

This letter is organized as follows. In Section II, we describe the problems with WEP data integrity and introduce the details of Michael. In Section III, we show how to invert Michael, explain how a related-message attack can break the MIC key, and suggest a fix. We conclude with Section IV.

II. Data Integrity in WEP and in TKIP

A. *Compromised WEP Data Integrity*

The existing IEEE 802.11 standard does not have a cryptographic mechanism to defend its data's integrity. In WEP, data integrity is only protected by CRC-32. Unfortunately, as noted by [4], CRC-32 provides absolutely no data integrity against a malicious attacker, even when both the data and CRC are covered by WEP encryption. CRC-32 is an unkeyed linear function of the data. WEP encrypts by XORing the RC4 key-stream with the data, and XOR is a linear operator as well. Thus, an attacker that sniffs a WEP-encrypted message M can easily flip any bit positions of her choice in M, and she can adjust the message's CRC-32 code to match the modified message. This can be done, through the RC4 encryption, by XORing easily computed bit strings with M.

B. *Details of Michael*

The TKIP mode of operation in the IEEE 802.11i draft introduces a new cryptographic mechanism for data integrity protection, in the form of the Michael MIC. Michael was designed specifically for use in IEEE 802.11i and has never been deployed in any crypto-system. The design document for Michael is [5].

Michael accepts a 64-b key, in the form of two 32-b words, and an arbitrarily long message, and produces a 64-b MIC value. The message is padded at the end with a single byte with hexadecimal value $5a$, followed by between 4 and 7 zero bytes. The number of zero bytes is chosen so that the overall length of the padded message is a multiple of 4.

Michael keeps a 64-b internal state, which is initialized by the MIC key. Each message word, including the padding, is XOR-ed into the state, after which a mixing function $b()$ is applied to the state. The function b is an unkeyed four-round Feistel structure, which uses rotations, XORs, and additions modulo 2^{32}, as its building blocks (see Fig. 1 for details).

```
Procedure Michael($(K_0, K_1), (M_0, \ldots, M_{N-1})$)
Input: Key $(K_0, K_1)$ and padded message (as 32-bit words)
$M_0, \ldots, M_{N-1}$
Output: MIC value $(V_0, V_1)$
  $(l, r) \leftarrow (K_0, K_1)$
  for $i = 0$ to $N-1$ do
    $l \leftarrow l \oplus M_i$
    $(l, r) \leftarrow b(l, r)$
  return $(l, r)$

Procedure $b(l, r)$
  $r \leftarrow r \oplus (l \hookleftarrow 17)$
  $l \leftarrow (l + r) \bmod 2^{32}$
  $r \leftarrow r \oplus \mathrm{XSWAP}(l)$
  $l \leftarrow (l + r) \bmod 2^{32}$
  $r \leftarrow r \oplus (l \hookleftarrow 3)$
  $l \leftarrow (l + r) \bmod 2^{32}$
  $r \leftarrow r \oplus (l \hookrightarrow 2)$
  $l \leftarrow (l + r) \bmod 2^{32}$
  return $(l, r)$
```

Fig. 1. Michael procedure. $\hookrightarrow$ indicates a 32-b right rotation. $\hookleftarrow$ indicates a left rotation. XSWAP is a function that swaps the position of the two least significant bytes and the position of the two most significant bytes in a word, i.e., XSWAP$(ABCD) = BADC$ where A, B, C, and D are bytes.

```
Procedure InvMichael($(V_0, V_1), (M_0, \ldots, M_{N-1})$)
Input: MIC value $(V_0, V_1)$ and padded message
$M_0, \ldots, M_{N-1}$
Output: The key $(K_0, K_1)$
  $(l, r) \leftarrow (V_0, V_1)$
  for $i = N-1$ downto 0 do
    $(l, r) \leftarrow b^{-1}(l, r)$
    $l \leftarrow l \oplus M_i$
  return $(l, r)$

Procedure $b^{-1}(l, r)$
  $l \leftarrow (l - r) \bmod 2^{32}$
  $r \leftarrow r \oplus (l \hookrightarrow 2)$
  $l \leftarrow (l - r) \bmod 2^{32}$
  $r \leftarrow r \oplus (l \hookleftarrow 3)$
  $l \leftarrow (l - r) \bmod 2^{32}$
  $r \leftarrow r \oplus \mathrm{XSWAP}(l)$
  $l \leftarrow (l - r) \bmod 2^{32}$
  $r \leftarrow r \oplus (l \hookleftarrow 17)$
  return $(l, r)$
```

Fig. 2. InvMichael procedure.

III. The Fragility of Michael

A. *Inverting Michael*

The first observation we make is that, since the mixing function b is a Feistel structure, which does *not* depend on the key, it can be inverted. The inverse function b^{-1} simply works backward through the rounds, alternating between XOR and subtraction modulo 2^{32} operations. The pseudocode for b^{-1} appears in Fig. 2.

This observation lets us invert the Michael MIC. Given a single known message and its MIC value, we can reveal the secret MIC key. We start with the MIC value, and then repeatedly apply b^{-1}, and XOR a message word (from last to first), until we arrive at the secret MIC key (see Fig. 2 for details).

We emphasize that this attack works 100% of the time, does not need to enumerate keys, and does not need any special conditions on the message or MIC value to hold. Any message will do. Since an attacker can capture messages with known contents with relative ease, we see that the security of Michael critically depends on the secrecy of the MIC values.

Note that the Michael design document [5] does not explicitly show how to invert the MIC, but clearly the designer was aware of Michael's sensitivity to MIC value exposure. The need to keep the MIC values secret is stated several times in [5].

B. Related-Message Attack

Proposition III.1: Suppose the attacker is able to capture two TKIP message frames (ciphertexts), M_1 and M_2, and the following conditions hold:

1) M_1 and M_2 are encrypted using the same encryption key and same IV;
2) $\mathrm{len}(M_1) \geq \mathrm{len}(M_2) + 8$;
3) plaintext P_1 of the longer message M_1 is known to the attacker.

Then, the secret MIC key k_{mic} can be computed in linear time.

Proof: Since both messages are encrypted with the same encryption key and same IV (condition 1), the RC4 pseudorandom generator produces the same key-stream K for both messages. By condition 3, the plaintext P_1 is known to the attacker, but the 8-B MIC value $\mathrm{Michael}(P_1, k_{\mathrm{mic}})$, which is appended to the plaintext, is unknown. Nevertheless, the attacher can recover the first $\mathrm{len}(M_1) - 8$ RC4 key-stream bytes by computing $K[i] = M_1[i] \oplus P_1[i]$ for $i = 1, \ldots, \mathrm{len}(M_1) - 8$. By condition 2, the amount of recovered key-stream is enough to decrypt all of the shorter plaintext P_2 and its MIC value m_2, which immediately follows the plaintext P_2 in the frame M_2. Given the plaintext P_2 and the MIC value $m_2 = \mathrm{Michael}(P_2, k_{\mathrm{mic}})$, the attacker can use the InvMichael function and compute k_{mic}. ■

Remarks:

- The attack can mostly work even without condition 3 (i.e., when neither plaintext is known to the attacker). Since both messages are encrypted by the same key-stream, the plaintext of both messages, up to the length of the shorter message, can be broken using techniques of classical cryptanalysis (cf., [11]). This provides the plaintext of the shorter message. To obtain the MIC value the attacker would then need to guess the next 8 B of plaintext in the longer message.
- Both conditions 2 and 3 are easy for the attacker to fulfill, since IEEE 802.11 frames do not have a fixed length, and since higher-level protocols generate highly structured and predictable traffic. Thus, the only TKIP mechanism protecting the MIC key k_{mic} works by negating condition 1. TKIP is designed to ensure that IV values are not reused, so that an attacker will not be able to capture related packets during the lifetime of k_{mic}.

C. Defense in Depth: Additional Counter-Measures

The original IEEE 802.11 standard used 24-b IVs and did not include significant requirements on the choice of IV values. This meant that finding related messages, encrypted using the same WEP key and same IV, was fairly straightforward [4]: after at most 2^{24} transmitted frames, IV values would start being reused. Some WEP implementations are reportedly much worse than others in this respect, including one that allegedly alternates between two IV values (cf., [12]).

The TKIP mode of IEEE 802.11i includes a much stronger IV selection mechanism, which uses a 48-b "extended IV." The IV is incremented as a counter which should not overflow without a key renegotiation. Thus, a correct TKIP implementation will not produce the related messages that the attack needs. Nevertheless, a slightly incorrect TKIP implementation might somehow repeat RC4 seed values—and recall that the attack only needs one pair of related messages. In the following, we provide two variants of an extremely simple counter-measure that protects the MIC key even if RC4 seed values are repeated.

1) MIC Value Before the Message Body: The first variant we suggest is straight-forward: Place the MIC value in a fixed position in the frame. Specifically, we suggest to place the MIC value *before* the message body, within the frame headers, as in [13].

If the MIC value is in a fixed location in the frame, then even if all the conditions of Proposition III.1 hold, the MIC values are not exposed. This is because the key-stream bytes that are exposed through the attacker's knowledge of the plaintext of one message do not encrypt the MIC value in any other message. Thus, even if key-stream material is reused by an incorrect TKIP implementation, the MIC values remain unknown to the attacker, and the InvMichael procedure cannot be applied.

A fortuitous side effect of placing the MIC value before the message body is that it helps defend against the [7], [12] WEP-cracking attacks. This is because the RC4 weakness of [7] relies on having known plaintext in the first byte positions. Placing the pseudorandom MIC value in the first byte positions makes the [7] weakness harder to exploit.

A disadvantage of placing the MIC before the message body is that it not well suited for a hardware or firmware implementation. Firmware implementations often accept the message word by word, update the MIC value for each word, encrypt the word and transmit it. When the last message word is transmitted, the firmware makes the final transformations on the MIC value, encrypts and transmits it. Such a firmware implementation would not be able to compute the MIC and transmit it before processing (and transmitting) the whole message body. Nevertheless, placing the MIC in the headers is very easy in a software implementation, that builds the message frame in a memory buffer.

Note that having the MIC in a fixed location in the frame significantly reduces the risk of having the MIC key broken due to IV reuse, but does not eliminate the risk completely. Given a pair of messages encrypted by the same RC4 key, one could certainly mount a brute-force attack against the MIC (enumerate all 2^{64} values for k_{mic}, compute a proposed MIC value for the first message, and check using the second message). Apparently, more efficient differential attacks are also possible in such a scenario [6], but those are sophisticated chosen-plaintext attacks, which are much harder to mount than the passive known-plaintext attack of Section III-A.

2) MIC Value After the Message Body: To allow for firmware implementations that compute the MIC on the fly, the MIC value has to be appended after the message body. Therefore, we suggest a second variant, in which the MIC value trails the body.

The idea is to store the first 8 B of the RC4 key-stream, $K[0],\ldots,K[7]$ in a buffer, KM. The message body is then encrypted using the RC4 key-stream bytes $K[8],\ldots,K[N+7]$ for an N-byte message. The MIC value can be computed on-the-fly during encryption, or it can be computed in advance if the whole message is available in a buffer. After the MIC value is computed, it is encrypted using the eight key-stream bytes stored in KM, and the encrypted MIC value is appended after the message body.

Note that this variant protects the MIC values exactly like the simpler variant of Section III-C1, since in both variants the MIC value is always encrypted by key-stream bytes $K[0],\ldots,K[7]$. Its main advantage is that a firmware implementation can compute the MIC value on the fly, at the cost of maintaining an 8-B buffer.

IV. Conclusion

The latest IEEE 802.11i draft proposes a mode of operation called TKIP, which incorporates a new MIC called Michael. In this letter, we showed that the security of the MIC completely breaks down if a single message with its MIC value is exposed. We also showed how a related message attack can can expose the MIC value if a TKIP implementation ever reuses IV values. Finally, we showed that simple changes to the frame format can protect the MIC even in the face of IV reuse. The changes we propose follow a "defense in depth" paradigm; they are unnecessary if IV values are never reused, and they protect the secret MIC key if the first line of defense fails and IV values are reused.

Acknowledgment

M. Rafael suggested the idea of buffering the first key-stream bytes and using them to encrypt the MIC when it is after the message body. The author would like to thank M. Jalfon for providing him with the latest IEEE 802.11i draft and N. Ferguson for valuable feedback on an earlier version of this manuscript.

References

[1] *Advanced Encryption Standard*, NIST FIPS PUB 197, Nov. 2001.
[2] W. A. Arbaugh, N. Shankar, and Y. C. J. Wan, "Your 802.11 wireless network has no clothes," in *Proc. IEEE Int. Conf. Wireless LANs and Home Networks*, 2001, pp. 131–141.
[3] M. Bellare, R. Canetti, and H. Krawczyk. (1997, Feb.) "HMAC: Keyed hashing for message authentication". Internet Engineering Task Force RFC 2104. [Online]. Available: http://www.ietf.org/rfc/rfc2104.txt
[4] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: The insecurity of 802.11," in *Proc. 7th ACM Conf. Mobile Computing and Networking*, Rome, Italy, 2001, pp. 180–189.
[5] N. Ferguson. (2002) "Michael: An Improved MIC for 802.11 WEP". [Online]. Available: http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/2-020.zip
[6] ——, private communication, 2003.
[7] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," in *Proc. 8th Workshop Selected Areas in Cryptography*, LNCS 2259, 2001, pp. 1–24.
[8] *Wireless LAN medium access control (MAC) and Physical layer (PHY) specifications*, ANSI/IEEE Standard 802.11, 1999.
[9] *Draft supplement to standard for telecommunications and information exchange between systems—LAN/MAN Specific requirements—Part 11: Wireless medium access control (MAC) and physical layer (PHY) specifications: Specification for enhanced security*, IEEE Std 802.11i/D3.0, Nov. 2002.
[10] R. L. Rivest, "The RC4 encryption algorithm," RSA Data Security Inc., (proprietary), 1992.
[11] B. Schneier, *Applied Cryptography*, 2nd ed. New York: Wiley, 1996.
[12] A. Stubblefield, J. Ioannidis, and A. Rubin, "A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP)," *ACM Trans. Inform. Syst. Security*, vol. 7, no. 2, pp. 319–332, 2004.
[13] A. Wool. (2002) Lightweight key management for IEEE 802.11 Wireless LAN's with key refresh and host revocation. IEEE 802.11 TGi working group. [Online]. Available: http://grouper.ieee.org/groups/802/11/Documents/DocumentHolder/2-411.zip
[14] A. Wool, "Why security standards sometimes fail," *Commun. ACM*, vol. 45, no. 12, p. 144, Dec. 2002.