
Indoor WLAN Design

Part VII: More on WPA

Kjell Jørgen Hole

UiB

Last updated 12.02.08

Mail: Kjell.Hole@ii.uib.no

URL: www.kjhole.com

Outline: Improvements in WPA

KJhole.com

- Incorrect use of RC4 encryption keys in WEP leads to successful data and key recovery attacks
- ★ **Improvement:** WPA utilizes re-keying and *one-way hash function* to obtain per-packet encryption keys
- *Message Integrity Code (MIC)* in WEP allows packet forgeries
- ★ **Improvement:** WPA introduces new MIC, called **Michael**
- WEP authentication is broken
- ★ **Improvement:** WPA uses EAP/EAPOL messages and a four-way handshake protocol for authentication

Re-Keying and Per-Packet Key Mixing

creates a new key for every packet

WEP Allows Data and Key Attacks

- WEP (Wired Equivalency Protocol) has many design flaws. Two of the major flaws are:
 - The reuse of IV (Initialization Vector) values, which leads to reuse of RC4 key streams, which leads to a *data recovery attack*
 - A correlation between the combination of the IV and RC4 encryption key with the first RC4 key stream byte, which leads to a *key recovery attack*

1. WPA ensures that a shared key is updated before the IV space is exhausted, i.e., no more than one packet is encrypted under a given key with the same IV value
2. **Per-packet key mixing:**
 - In WEP, it is necessary to generate a different RC4 encryption key for each packet from a pre-shared key. WEP therefore concatenates the IV and the pre-shared key
 - WPA improves the per-packet key generation by preprocessing the shared key and the (extended) 48-bit IV using a *one-way hash function*

Hash function Environment

- The hash function operates in an environment where the encryptor and decryptor
 - share a 128-bit secret key, called the *Temporal Key* (**TK**)
 - ensure that no IV is used more than once with each TK
 - use the RC4 cipher
- The *Transmitter Address* (TA), a 48-bit MAC address, is mixed into the TK to ensure that the various parties encrypting with the TK use *different* key streams

Temporal Key Hash Function

KJhole.com

A two-phase processing of the TK determines the per-packet encryption key:

Phase 1 Utilizes S-box to mix TK with TA and the 32 most significant bits (msb) of the IV, denoted IV32. Output may be cached. It can be reused to process subsequent packets associated with the same TK, TA and IV32.

Phase 2 Utilizes S-box to mix output of phase 1 with TK and the 16 least significant bits (lsb) of the IV, denoted IV16. The IV16 must be different for each packet encrypted under the TK

- An S-box is a non-linear substitution
- The same S-box is used in both Phase 1 and Phase 2
- The S-box substitutes one 16-bit value with another 16-bit value
- The S-box can be implemented as a table look up (see standard)

Two-Phase Process

KJhole.com

The two-phase process may be summarized as

P1K = Phase1 (TK, TA, IV32)
RC4KEY = Phase2 (P1K,TK, IV16)

Phase 1

- The inputs to phase 1 are TK, TA, and IV32. The output is denoted by P1K
- TK and TA are treated as arrays of 8-bit values, while P1K is represented by an array of 16-bit values:

$$\text{TA}[0..5]: 8 \cdot 6 = 48\text{bits}$$

$$\text{TK}[0..15]: 8 \cdot 16 = 128\text{bits}$$

$$\text{P1K}[0..4]: 16 \cdot 5 = 80\text{bits}$$

Step 1 of Phase 1

- Functions used in Phase 1:
 - Lo16() returns the 16 lsb of a 32-bit input value, and Hi16() returns the 16 msb
 - Mk16(X,Y) = 256*X+Y, 16-bit value from two 8-bit values
- Step 1 of Phase 1:
 - P1K[0] = Lo16(IV32)
 - P1K[1] = Hi16(IV32)
 - P1K[2] = Mk16(TA[1],TA[0])
 - P1K[3] = Mk16(TA[3],TA[2])
 - P1K[4] = Mk16(TA[5],TA[4])

Step 2 of Phase 1

The exclusive-or (\oplus) operation is used to obtain the index to the S-box:

```
FOR i=0 to 7
```

```
BEGIN
```

```
    j = 2*(i & 1) // bit-wise and (&) operation
```

```
    P1K[0] = P1K[0] + S[P1K[4]  $\oplus$  Mk16(TK[1+j],TK[0+j])]
```

```
    P1K[1] = P1K[1] + S[P1K[0]  $\oplus$  Mk16(TK[5+j],TK[4+j])]
```

```
    P1K[2] = P1K[2] + S[P1K[1]  $\oplus$  Mk16(TK[9+j],TK[8+j])]
```

```
    P1K[3] = P1K[3] + S[P1K[2]  $\oplus$  Mk16(TK[13+j],TK[12+j])]
```

```
    P1K[4] = P1K[4] + S[P1K[3]  $\oplus$  Mk16(TK[1+j],TK[0+j])] + i
```

```
END
```

- The inputs to the second phase are the output P1K of the first phase, TK, and IV16
- The phase 2 output is a per-packet key , called RC4KEY consisting of 128 bits
 - Since the RC4KEY must conform to the WEP standard, the 24 first bits will be transmitted in plaintext
- P1K is treated as an array of 16-bit values, and the RC4KEY is treated as an array, RC4KEY[0..15], of 8-bit values

Step 1 of Phase 2

KJhole.com

PPK[0] = P1K[0]

PPK[1] = P1K[1]

PPK[2] = P1K[2]

PPK[3] = P1K[3]

PPK[4] = P1K[4]

PPK[5] = P1K[4] + IV16

Step 2 of Phase 2

```
PPK[0] = PPK[0] + S[PPK[5] ⊕ Mk16(TK[1],TK[0])]
PPK[1] = PPK[1] + S[PPK[0] ⊕ Mk16(TK[3],TK[2])]
PPK[2] = PPK[2] + S[PPK[1] ⊕ Mk16(TK[5],TK[4])]
PPK[3] = PPK[3] + S[PPK[2] ⊕ Mk16(TK[7],TK[6])]
PPK[4] = PPK[4] + S[PPK[3] ⊕ Mk16(TK[9],TK[8])]
PPK[5] = PPK[5] + S[PPK[4] ⊕ Mk16(TK[11],TK[10])]
```

// RotR1() rotates 16 bits one bit to the right

```
PPK[0] = PPK[0] + RotR1(PPK[5] ⊕ Mk16(TK[13],TK[12]))
PPK[1] = PPK[1] + RotR1(PPK[0] ⊕ Mk16(TK[15],TK[14]))
PPK[2] = PPK[2] + RotR1(PPK[1])
PPK[3] = PPK[3] + RotR1(PPK[2])
PPK[4] = PPK[4] + RotR1(PPK[3])
PPK[5] = PPK[5] + RotR1(PPK[4])
```

Step 3 of Phase 2

The bit-wise or operation, denoted $|$, and the right bit shift operation, \gg , are used in step 3:

```
RC4KEY[0] = Hi8(IV16)
RC4KEY[1] = (Hi8(IV16) | 0x20) & 0x7F
RC4KEY[2] = Lo8(IV16)
RC4KEY[3] = Lo8( (PPK[5]  $\oplus$  Mk16(TK[1],TK[0]))  $\gg$  1 )
FOR i = 0 to 5
BEGIN
    RC4KEY[4+(2*i)] = Lo8(PPK[i])
    RC4KEY[5+(2*i)] = Hi8(PPK[i])
END
```

- Implementations can achieve a significant performance improvement by caching the output of the first phase
- Consider the case where an MS communicate with a BS
 - the Phase 1 output is the same for $2^{16} = 65536$ consecutive packets from the same TK
 - the MS will perform Phase 1 using its own address (TA). The output will then be cached and used to generate each RC4KEY

New MIC
called Michael

WEP Allows Packet Forgeries

KJhole.com

- One major WEP flaw is
 - The linear un-keyed MIC which allows modification of packets and injection of new packets (the MIC provides hardly any authentication functionality)
- The *Temporal Key Integrity Protocol* (TKIP) was designed to “wrap around” WEP to fill the major security gaps
 - It defeats message forgeries by adding the new MIC Michael

Comment: MIC and MAC

KJhole.com

- In 802.11 the acronym MAC refers to *Media Access Control*. To cryptographers, the acronym MAC refers to the completely unrelated term *Message Authentication Code*
- To avoid confusion, we use the the acronym MIC (Message Integrity Code) instead of MAC

Michael's Design Criteria

KJhole.com

- TKIP, and hence Michael, had to be able to run on early 802.11 (a/b/g/h) devices
- MIC should be computed over the message and then appended
- MIC should utilize 64-bit keys

MIC Key Requirements

- The MIC keys must satisfy the following requirements (details not discussed here):
 - A MIC key must be obtained from TK
 - Different MIC keys should be used for the two directions of traffic
 - MIC keys should be computationally independent of encryption keys
 - MIC keys should be unknown to, and unpredictable for, any third party

Required Security Level

KJhole.com

Definition A MIC is said to have a *security level* of k bits if the probability of a successful forgery on the first attempt is no more than 2^{-k}

- If a MIC has a security level of k bits, then the attacker is expected to need 2^{k-1} failed attempts before a successful attempt
- It is required that Michael has a security level of at least 20 bits

Design Limitation

KJhole.com

- It is not feasible to create a fast and highly secure MIC because only very limited computer power is available
- Forced to design a MIC that is *much weaker* than desirable
- ! MIC must be combined with countermeasures that limit the attacker's possibilities

Attack and Countermeasures

KJhole.com

Attack Send many fraudulent packets in hope that at least one of them passes the MIC test

What to do Detect attack and take active countermeasure:

Detection Assume active attack any time there is a MIC verification failure. Alternatively, ignore first MIC failure, and assume attack after second MIC failure

Countermeasures (1) Delete authentication and encryption keys. (2) Log event as a security-relevant matter. (3) Shut down device if rate of MIC failures is more than one per minute

WARNING

KJhole.com

- The MIC can *only* be used with a secure encryption system
 - When the MIC is securely encrypted, the attacker cannot learn anything from the wireless traffic about the MIC value or the MIC key
- This rules out the use of the MIC without rapid re-keying or without per-packet RC4 key mixing

Michael (1)

- **Michael** is a 8-byte MIC with a 64-bit key and a designed security level of 20 bits
- The MIC key may be viewed as an 8-byte value k_7, \dots, k_0 where k_0 is the least significant byte. The key is converted to two key words K_0 and K_1 of 32 bits each
 - all conversions between bytes and 32-bit words are done using the least-significant-byte-first convention
 - $K_0 = k_3, k_2, k_1, k_0$ and $K_1 = k_7, k_6, k_5, k_4$

- The **message** consists of n bytes m_0, \dots, m_{n-1} . It is
 - first padded at the end with a byte with value 0x5a and then between 4 and 7 zero bytes
 - the number of zero bytes is chosen such that the total number of bytes is a multiple of 4
 - message is then converted to a sequence of 32-bit words M_0, \dots, M_N where $N := \lceil (n + 5) / 4 \rceil$
 - M_N is always zero (because of the 0 byte padding)

Algorithm 1

KJhole.com

Algorithm 1: Michael message processing

Input: Key (K_0, K_1) and padded message M_0, \dots, M_{N-1}

Output: MIC value (V_0, V_1)

MICHAEL($(K_0, K_1), (M_0, \dots, M_{N-1})$)

```
1   $(L, R) \leftarrow (K_0, K_1)$ 
2  for  $i = 0$  to  $N - 1$ 
3       $L \leftarrow L \oplus M_i$ 
4       $(L, R) \leftarrow b(L, R)$ 
5  return  $(L, R)$ 
```

Notation

KJhole.com

- \ll denotes the rotate-left operator on 32-bit values
- \gg denotes the rotate-right operator on 32-bit values
- XSWAP denotes a function that swaps the position of the two least significant bytes and the position of the two most significant bytes in a 32-bit word

Algorithm 2

Algorithm 2: Michael block function

Input: (L, R)

Output: (L, R)

$b(L, R)$

```
1   $R \leftarrow R \oplus (L \ll 17)$ 
2   $L \leftarrow (L + R) \bmod 2^{32}$ 
3   $R \leftarrow R \oplus \text{XSWAP}(L)$ 
4   $L \leftarrow (L + R) \bmod 2^{32}$ 
5   $R \leftarrow R \oplus (L \ll 3)$ 
6   $L \leftarrow (L + R) \bmod 2^{32}$ 
7   $R \leftarrow R \oplus (L \gg 2)$ 
8   $L \leftarrow (L + R) \bmod 2^{32}$ 
9  return  $(L, R)$ 
```

The Sins of Michael

KJhole.com

- Designing a new cryptographic primitive
- Using a new structure for the primitive
- Designing a primitive with marginal security
- fielding an *untested* design
- Relying on other system properties to achieve the security goals

Authentication

EAP/EAPOL messages and the four-way handshake

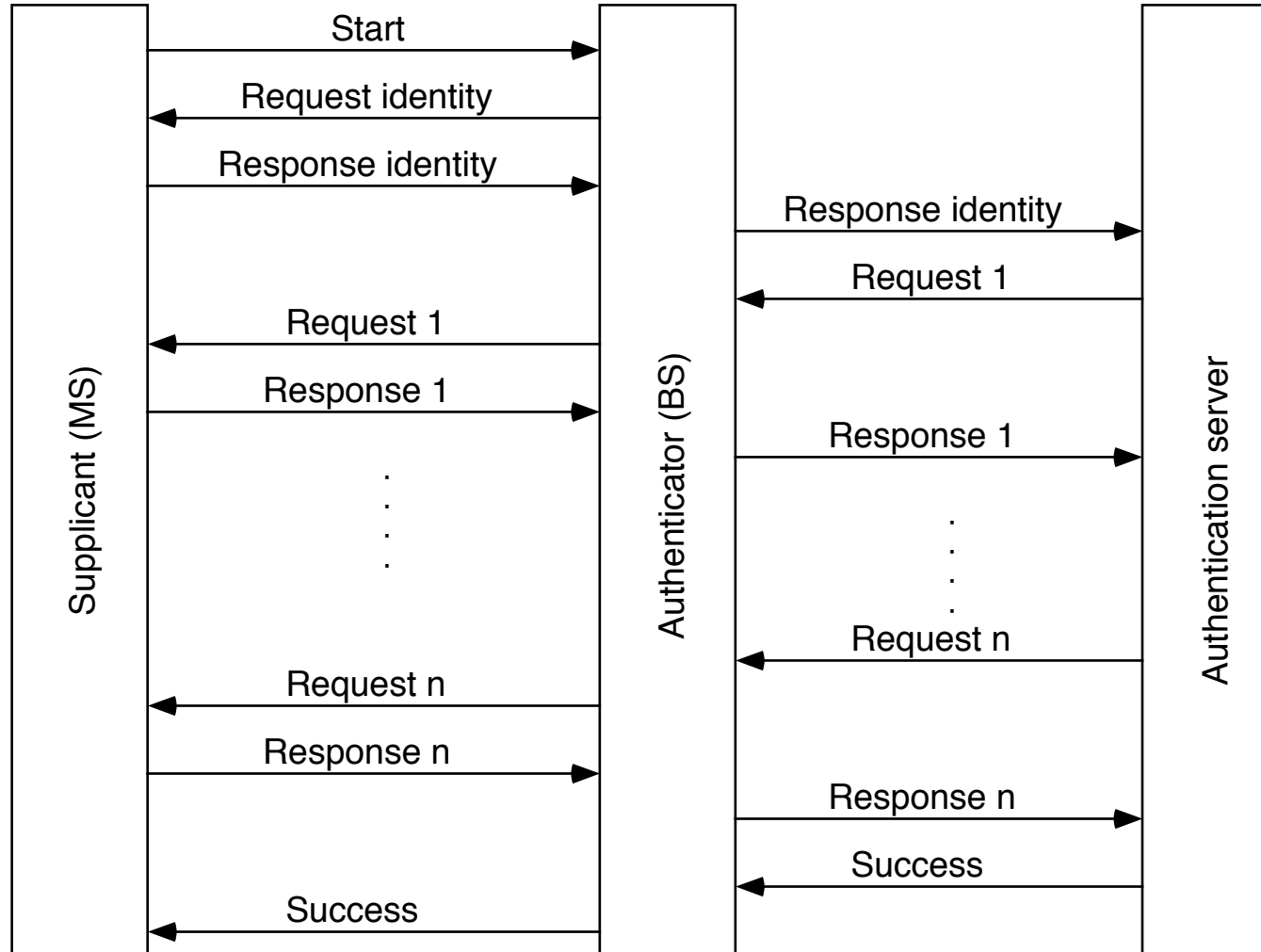
- EAP does not specify how messages should be passed around
- 802.1x defines the protocol **EAP over LAN** (EAPOL) to move EAP messages
- Four different EAPOL messages were introduced in Part VI:
 - EAPOL-Start
 - EAPOL-Key
 - EAPOL-Packet
 - EAPOL-Logoff

EAP Message Flow

1. When an MS (supplicant) wants to connect to a BS (authenticator), it sends an EAPOL-Start message
2. BS responds with an EAP-Request/Identity message
3. MS answers with an EAP-Response/Identity message
4. BS contacts authentication server to find out whether the MS is to be let in
5. An ULA protocol is used to obtain mutual authentication of MS and authenticates server

Encapsulated in EAPOL

Encapsulated in RADIUS



Details: MS-BS Authentication

KJhole.com

- Assume that ULA and EAP/EAPOL were used to mutually authenticate the MS and the authentication server, and that a Pair-wise Master Key (PMK) was generated as part of this process
- Assume also that the connection between the authentication server and BS is secure, i.e., the BS and authentication server share their own secret
 - then the BS can only receive the PMK from the server
- The BS can prove to the MS that it is trusted by the authentication server by showing that it knows the PMK
- An 802.1x four-way handshake lets the MS authenticate the BS

Four-Way Handshake (1)

- **Message (A): BS→MS**

- EAPOL-Key message is sent from the BS (authenticator) to the MS (supplicant)
- Message A contains ANonce and is not encrypted or protected from tampering
- Tampering with the value simply makes the handshake invalid
- Once the MS receives message A it can compute the temporal keys, it already has the PMK (see Part VIII for details)

Four-Way Handshake (2)

- **Message (B): MS→BS**
 - MS sends SNonce to BS
 - Message B is not encrypted, but contains MIC (Message Integrity Code). EAPOL MIC key is used
 - Once the BS receives message B it can compute the temporal keys, it already has the PMK
 - Message B enables BS to verify that MS knows the PMK because the MIC check will fail if the PMKs at the BS and MS are different

Four-Way Handshake (3)

- **Message (C): BS→MS**

- Message C sent from BS to tell the MS that it is ready to start using the new keys for encryption
- Message is not encrypted, but it contains MIC so the MS can verify that BS has matching PMK

- **Message (D): MS→BS**

- Message D is sent from MS to BS to acknowledge completion of the four-way handshake
- Message is sent unencrypted

Review of Four-Way Handshake

KJhole.com

1. ANounce and SNonce values have been exchanged
2. Temporal keys have been computed
3. MS has proved knowledge of the PMK
4. BS has proved knowledge of the PMK
5. Both MS and BS have synchronized and turned on encryption of unicast packets

Remarks: Group key hierarchy must also be generated (not discussed). Once the BS has received an EAP-Success, it lets the MS access the controlled port

WPA introduces mechanisms to mitigate the vulnerabilities in WEP:

- re-keying and packet key mixing
- new MIC, called Michael
- EAP/ULA provides mutual authentication of MS and RADIUS server
- four-way handshake for mutual authentication of MS and BS