Paper IV: Weaknesses in the Temporal Key Hash of WPA

Weaknesses in the Temporal Key Hash of WPA

Vebjørn Moen, Håvard Raddum, and Kjell J. Hole

This article describes some weaknesses in the key scheduling in Wi-Fi Protected Access (WPA) put forward to secure the IEEE standard 802.11-1999. Given a few RC4 packet keys in WPA it is possible to find the Temporal Key (TK) and the Message Integrity Check (MIC) key. This is not a practical attack on WPA, but it shows that parts of WPA are weak on their own. Using this attack it is possible to do a TK recovery attack on WPA with complexity $\mathcal{O}\left(2^{105}\right)$ compared to a brute force attack with complexity $\mathcal{O}\left(2^{128}\right)$.

1 Introduction

The IEEE standard 802.11-1999 [1] is a set of protocols defining a communication channel inspired by Ethernet, but using unlicensed radio spectrum bands instead of wires. Since radio is used to communicate, eavesdropping can be done by anyone with a radio receiver, and anyone with a radio transmitter can write to the channel. This shows the need for built-in security in the WLAN design. The 1999 standard includes a security protocol called Wired Equivalent Privacy, or WEP. The goal was to achieve the same level of security as wired Ethernet.

It has been shown that the WEP design has many basic flaws and does not fulfill the design goals. It does not defend properly against packet forgery or replay, which allows an attacker to use the 802.11 infrastructure to launch attacks on the WEP encryption key. In addition, WEP uses the RC4 encryption algorithm in a way that makes it possible to mount plaintext recovery attacks and key recovery attack using public domain software, *e.g.* AirSnort [2].

To correct these design flaws, the 802.11 Working Group (WG) has chartered work to find a new security protocol. First the WG has defined WPA, which is a WEP wrapper design, to fix all the known problems with WEP. It has been established that WPA cannot fulfill the original WEP design goals, because the available CPUs on existing hardware are too limited. Therefore, the WG is also working on a new protocol based on the Advanced Encryption Standard (AES) that can meet the original design goals, but it will require new hardware.

The rest of this paper is organized as follows: Section 2 describes the algorithms that are relevant to the security in WPA, Section 3 describes the attack, Section 4 discusses the practicality and impact of the results, and Section 5 gives a summary of this paper.

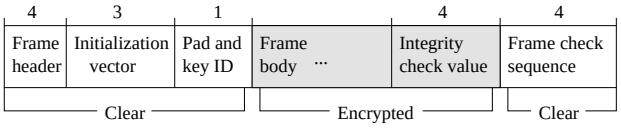

Figure 1: WEP frame. Length of fields measured in bytes.

2 Algorithms

2.1 WEP

WEP was suggested in the IEEE standard 802.11-1999 to provide security equivalent with that of a wired Ethernet. The WEP algorithm should insure confidentiality and integrity of the frames on the wireless network. A Cyclic Redundancy Check (CRC) is used to compute an Integrity Check Value (ICV) on the message. The ICV is then concatenated on the message before encrypting with the stream cipher RC4. The WEP-frame is illustrated in Figure 1.

RC4 is a symmetric cipher, *i.e.*, the same key encrypts and decrypts the data. The encryption key is a per-packet key which is obtained by concatenating an *Initialization Vector* (IV) with the user key. Because of export regulations the standard specifies 64-bit keys where 24 bits are the IV, but many vendors have also implemented 128-bit keys where 24 bits are the IV.

2.1.1 Security issues

The CRC used for the ICV can catch single-bit alterations with high probability, but it is not cryptographically secure. The CRC is a linear function of the message, and Borisov *et al.* [3] showed that it is possible to make controlled modifications to a ciphertext without disrupting the checksum.

The standard ignores the issue of key management. Most vendors do not implement any key distribution mechanism, this means that keys must be statically entered into either the driver software or firmware. All the mobile stations accessing the same access point use the same pre-shared key and can therefore decrypt each others packets. Since the key needs to be manually distributed and typed into a device, it is not likely that the key will be changed very often. The IV is only 24 bits long, which implies that the same key and IV will be reused, this is known as the two-time pad [3].

Fluhrer *et al.* [4] also found a correlation between the combination of the IV and user key with the first RC4 key stream byte, which leads to a practical key recovery attack.

2.2 Wi-Fi Protected Access

Because WEP has been shown to be totally insecure, the 802.11 WG has suggested a new security protocol. The protocol is called Wi-Fi Protected Access (WPA). The goal for this protocol is to fix all known security flaws in WEP and it was designed to be deployed as a software patch on existing hardware.

WPA includes a key hash function [5] to defend against the Fluhrer *et al.* [4] attack, a Message Integrity Code (MIC) [6] and a key management scheme based

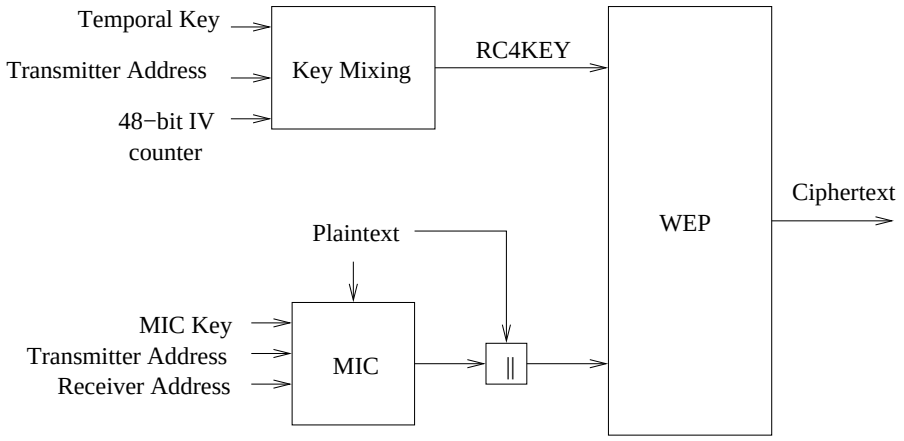

Figure 2: WPA encapsulation process

on 802.1X [7] to avoid key reuse and to ease the key distribution. Figure 2 shows the encapsulation process.

The 16-byte *Temporal Key* (TK) is obtained from the key management scheme during the authentication, and goes into the key hash function together with the 6-byte *Transmitter Address* (TA) and a 48-bit IV, often called the TKIP sequence counter. The key hash function outputs a 16-byte RC4 key where the three first bytes are derived from the IV. This key is used only for one WEP frame, since the IV is implemented as a counter which increases after each package, and the key is therefore often called a per-packet key. The IV counter also works as a defense against replay attacks, the receiver will not accept packets with smaller or equal IV to previously received packets.

Integrity of the message is insured by the MIC. This function takes as input a MIC key, TA, receiver address, and the message, and outputs the message concatenated with a MIC-tag. If necessary this output is fragmented before it enters WEP.

This means that WPA is a wrapper for WEP insuring that a <TK,IV> pair is only used once by a sender, and improving the integrity of WEP frames by applying a non-linear message integrity function. More details about the MIC can be found in [6].

2.2.1 Key mixing

The key mixing function is described by Housley *et al.* [5], this function is also called a *temporal key hash*. As shown on Figure 2, this function takes as input the TK, the TA and the 48-bit IV, and outputs a 128-bit WEP key where 24 bits are derived from the IV. The least significant 16 bits of the 48-bit IV are denoted IV16, and 32 most significant bits are denoted IV32. The key mixing is a two-phase process which may be summarized as:

$$\begin{aligned} P1K &= \text{Phase1}(TK, TA, IV32) \\ RC4KEY &= \text{Phase2}(P1K, TK, IV16) \end{aligned}$$

```
PHASE1_STEP1:
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])

PHASE1_STEP2:
FOR i = 0 to 7
BEGIN
   j = 2*(i & 1)
   P1K[0] = P1K[0] + S[P1K[4] ⊕ Mk16(TK[ 1+j],TK[ 0+j])]
   P1K[1] = P1K[1] + S[P1K[0] ⊕ Mk16(TK[ 5+j],TK[ 4+j])]
   P1K[2] = P1K[2] + S[P1K[1] ⊕ Mk16(TK[ 9+j],TK[ 8+j])]
   P1K[3] = P1K[3] + S[P1K[2] ⊕ Mk16(TK[13+j],TK[12+j])]
   P1K[4] = P1K[4] + S[P1K[3] ⊕ Mk16(TK[ 1+j],TK[ 0+j])] + i
END
```

Algorithm 1: Phase1 of Temporal Key hash

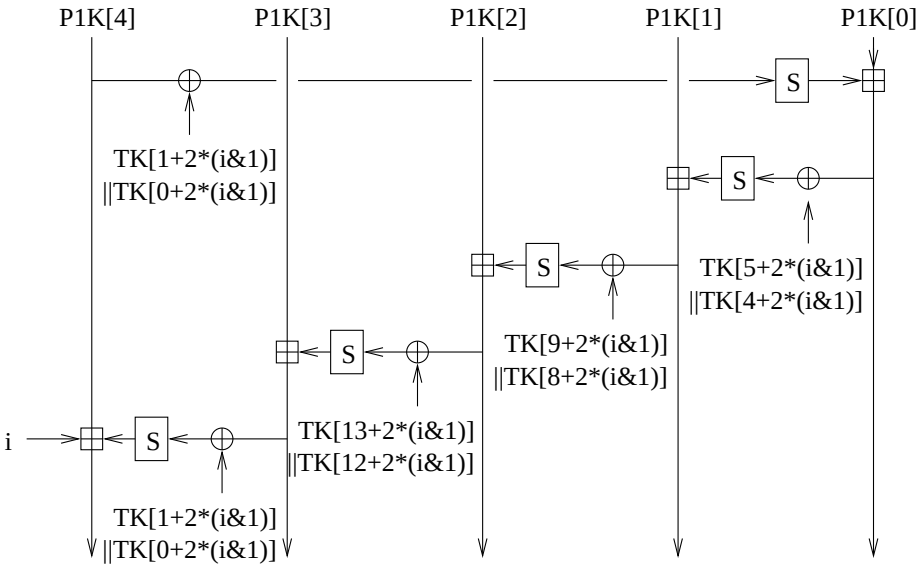

Figure 3: One of 8 rounds of the first phase of temporal key hash. This round is repeated for $i = 0..7$.

Phase 1 is shown in Algorithm 1 and in Figure 3. This part is usually only done once every 2^{16} packets and cached. It takes TK, TA and IV32 as input and outputs P1K used as input in Phase 2.

```
PHASE2_STEP1:
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

PHASE2_STEP2:
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])]

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])

PHASE2_STEP3:
RC4KEY[0] = Hi8(IV16)
RC4KEY[1] = (Hi8(IV16) | 0x20) & 0x7F RC4KEY[2] = Lo8(IV16)
RC4KEY[3] = Lo8((PPK[5] ⊕ Mk16(TK[1],TK[0])) >> 1)

FOR i = 0 to 5
BEGIN
   RC4KEY[4+(2*i)] = Lo8(PPK[i])
   RC4KEY[5+(2*i)] = Hi8(PPK[i])
END
```

Algorithm 2: Phase2 of the temporal key hash

Phase 2 takes the output from Phase1, TK and IV16 as input, and outputs the 128-bit WEP key. Phase 2 is described in Algorithm 2 and in Figure 4. Note that the TK is viewed as an array [0..15] of 8-bit bytes.

The S-box is a bijective nonlinear function defined by a table lookup in [5], it takes a 16-bit input and outputs a 16-bit value. The Mk16 function takes two 8-bit inputs and produces a 16-bit word, such that Mk16(X,Y) $= 256 * X + Y$ which is equivalent to Mk16(X,Y) $= X||Y$. Hi16 takes a 32-bit input and returns the most significant 16 bits, Lo16 takes a 32-bit input and returns the least significant 16 bits. Hi8 and Lo8 are similar but with input size 16 bits and output size 8 bits. Furthermore, & denotes bit-wise logical AND, and | represents bit-wise logical OR. Also, note that + in the Algorithms 1 and 2, and

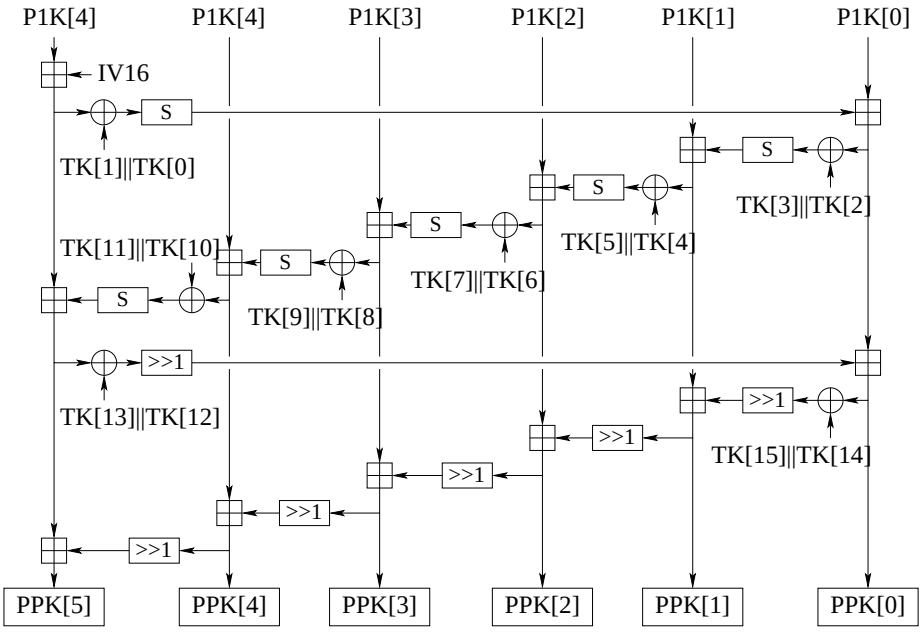

Figure 4: Phase 2 of the temporal key hash. The 96-bit PPK is used together with 16 bits of the IV and 8 bits of the TK to create the 128-bit WEP key.

$\boxplus$ in Figures 3 and 4 are addition modulo 2^{16}; $\oplus$ represents bit-wise exclusive OR. Both RotR1 and $>>1$ denote right circular shift by 1. P1K and PPK are treated as arrays of 16 bit words and RC4KEY is treated as an array of 8-bit bytes.

3 Attacking Temporal Key Hash

This section describes an attack on the temporal key hash described above. It is assumed that the attacker has knowledge of a few (less than 10) RC4-keys computed under the same IV32. Whether this is a realistic assumption or not will be discussed in Section 4.

Under this assumption we show that the attacker can easily compute the TK, and thus decrypt any packet the same way the legitimate receiver does. The attack has a complexity of about 2^{32} simple operations, and takes a few minutes to execute on a normal modern PC. The attack is basically done by computing backwards through Phase 2, guessing on parts of the TK. We can check if a guess is right or wrong since we know that the P1K-values do not change before IV32 changes.

The attack makes use of the fact that eight bits of TK can be computed directly from an RC4-key. The PPK-values output from Phase 2 are known from the RC4-key, in particular PPK[5] is known. By looking at Step 3 of Phase 2 in Algorithm 2 we see how RC4KEY[3] is computed It is then easy to reveal the least significant bit of TK[1], and the seven most significant bits of TK[0].

The rest of this section describes the attack in detail, showing how we can

break the rest of the TK into six parts, and guess on one part at the time. In the diagrams below, thick lines indicates that we know the values carried on them, and dotted lines indicate that there are two choices for the values on the lines.

3.1 Finding TK[10] and TK[11]

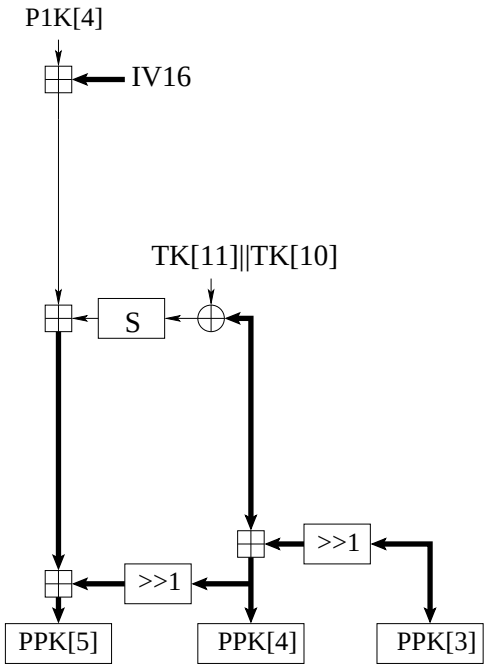

Figure 5: Part of Phase 2 needed to compute TK[10] and TK[11] .

Figure 5 is cut out from Figure 4 of Phase 2 of the key hash. The attack is based on the divide-and-conquer technique as illustrated in the figure. The idea is to find some bytes of the TK at a time, and Figure 5 shows the parts of PPK and TK which are needed to calculate backwards to P1K[4]. Since PPK[3], PPK[4] and PPK[5] are known we can start backtracking Phase 2. The arrows show what is input and output.

The first thing we need to do is to right shift PPK[3] and PPK[4] by one, and this is straight forward. The inverse of addition modulo 2^{32} is subtraction modulo 2^{32}. Using this we can compute backwards up to the point where the values depend on TK[10] and TK[11]. Now we guess on the value of TK[11]||TK[10], which allows us to backtrack through the XOR and S-box and two additions modulo 2^{32}. Remember that the IV is a known value, sent in the clear in the WPA packet. For each guess of TK[11]||TK[10] we get a suggestion for P1K[4]. We repeat the above procedure for each RC4-key that we are using in the attack, and if different RC4-keys give different P1K[4]-values, the guess was wrong. Using two or three RC4-keys should be enough to eliminate all but the correct values of TK[10] and TK[11].

3.2 Finding TK[8] and TK[9]

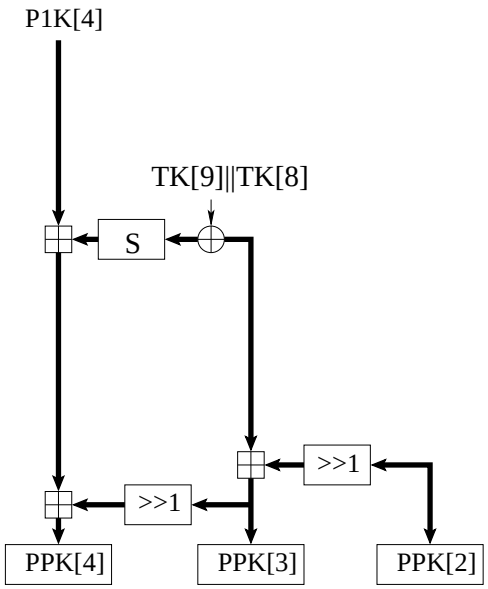

Figure 6: Part of Phase 2 needed to determine TK[8] and TK[9].

Figure 6 shows the part of Phase 2 necessary to compute TK[8] and TK[9]. Note that when we found TK[10] and TK[11] during the previous step, we also found the correct value of P1K[4]. Therefore, TK[8] and TK[9] can be computed directly, without any guessing.

3.3 Finding TK[6] and TK[7]

As Figure 7 shows, TK[6] and TK[7] can be found exactly the same way as TK[10] and TK[11] were found.

3.4 Finding TK[0], TK[1], TK[12] and TK[13]

Consider the part of Phase 2 shown in Figure 8. Here we take advantage of the fact that only eight bits of TK[0] and TK[1] are unknown. In order to compute the value of P1K[0], we can again make use of the now known value of P1K[4], but this time it is necessary to guess on TK[12], TK[13], and the eight unknown bits of TK[0] and TK[1] to reconstruct a candidate for P1K[0], a total of 24 bits. Again, if we don't get the same value of P1K[0] for all RC4-keys, we can discard the current guess as wrong.

At this stage, a subtle point comes into play. Assume we take the correct values of TK[0], TK[1], TK[12] and TK[13], but flip the least significant bit of TK[12]. This is a wrong guess, and should be discarded given sufficiently many

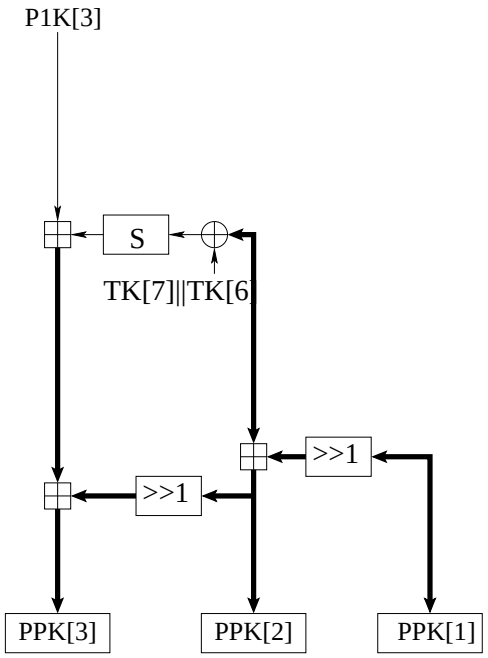

Figure 7: Part of Phase 2 needed to calculate TK[6] and TK[7]

RC4-keys. We will compare the values obtained in Figure 8 using this guess, to the values obtained with the correct guess. The guess for TK[0] and TK[1] is the correct one, so the values on the horizontal wire on the top will be equal in both cases. Going through the rotation in the bottom half, the values will differ only in the most significant bit. This is the same as saying that they differ by $2^{15} \pmod{2^{16}}$. Since the remaining operations for computing P1K[0] are subtraction mod 2^{16}, the computed values for P1K[0] with this wrong key guess will differ in the most significant bit from the correct value of P1K[0]. In particular, the P1K[0]-values computed with the wrong guess will all be equal! This means that the least significant bit of TK[12] can not be determined at this point using our method, however, it is easily determined later. It also means that P1K[0] is not determined completely, we only know the low 15 bits.

3.5 Finding TK[2], TK[3], TK[14] and TK[15]

Figure 9 shows the most expensive part of the attack. Here we will guess on TK[2], TK[3], TK[14] and TK[15] at the same time. For each guess we will compute the values of P1K[1] and check if they are equal. The matter is slightly complicated by the fact that we do not know P1K[0] completely. P1K[0] can take one of two values, so we have to do the check on P1K[1] for both values. For the correct guess of TK[14] and TK[15], there will be two values of TK[3]||TK[2] suggested, one for each P1K[0].

The problem with the least significant bit occurs here too, we can not find the least significant bit of TK[14]. This means we do not need to guess on it

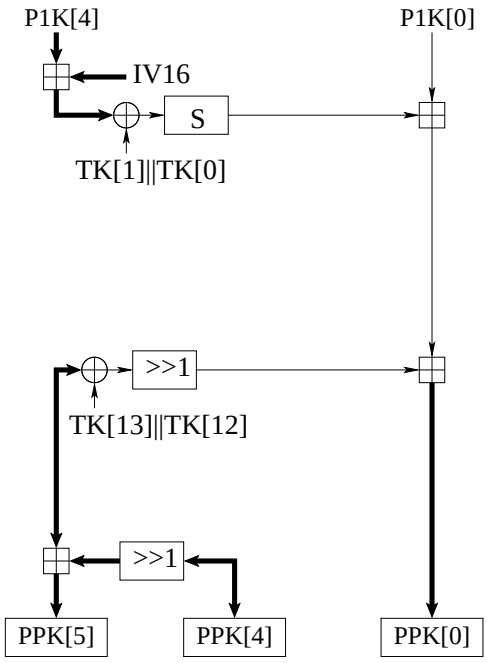

Figure 8: Part of Phase 2 needed to compute TK[0], TK[1], TK[12] and TK[13]

either, so there will be a total of 31 bits guessed. For each guess we will have to do the check on P1K[1] twice, once for each of the two possible values of P1K[0], so the overall complexity is 2^{32} checks. This step dominates the overall complexity of the attack. After this we are left with four sets of possible values of TK[2], TK[3], TK[12], and TK[14].

3.6 Finding TK[4] and TK[5]

Finally, Figure 10 shows how TK[4] and TK[5] are found by checking on the P1K[2]-values. Each guess also includes guessing on the least significant bit of TK[14]. For each of the two possibilities of TK[14] there will be one TK[5]- and TK[4]-value suggested as correct.

3.7 Finding the least significant bits of TK[12] and TK[14]

After completing all six steps described above, we are left with four possible values for the whole TK. Each possible TK has its corresponding P1K-value. The correct TK can now be found by running Phase 1 for each of the TK candidates, and see which one that gives its corresponding P1K as output. The probability that a wrong TK results in its corresponding P1K is $4 * 2^{-80} = 2^{-78}$ since P1K is 80 bits.

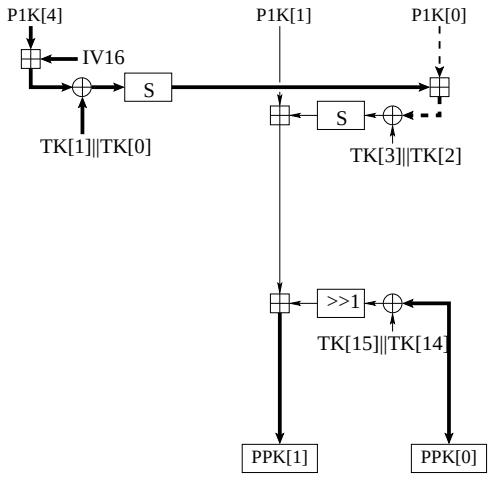

Figure 9: Part of Phase 2 needed to calculate TK[2], TK[3], TK[14] and TK[15]

3.8 Attacking with only two RC4-keys

When the attacker only has two RC4-keys, she only has a 16-bit condition for eliminating wrong guesses at each step. In the step with TK[0], TK[1], TK[12] and TK[13], we are guessing on 23 bits, so we expect to have about 2^7 candidates suggested for these four bytes. In the step with TK[2], TK[3], TK[14] and TK[15] we are guessing on 31 bits, so we expect to be left with 2^{15} candidates for this part of the TK. In the other steps we are guessing on 16 bits, so we expect to be only left with the correct TK-parts. With the four possible variations of the least significant bit of TK[12] and TK[14], this gives us $4 \cdot 2^7 \cdot 2^{15} = 2^{24}$ candidates for the whole TK. Each candidate has its corresponding set of P1K-values. Now we can run Phase 1 for each candidate and see which one has matching suggestions for the P1K value from both Phase1 and Phase2. The probability of a wrong TK to result in corresponding P1K values is $2^{24} \cdot 2^{-80} = 2^{-56}$, so with high probability only the correct TK will pass this test. Total work with only two texts is approximately $\mathcal{O}\left(2^{38}\right)$, since we need to guess on 31 bits for each of the 2^7 suggestions for TK[0], TK[1], TK[12] and TK[13].

3.9 Temporal Key recovery attack on WPA

The results in this paper imply that it is possible to mount a Temporal Key recovery attack on WPA with time complexity $\mathcal{O}\left(2^{105}\right)$ compared to simply brute force search on the TK, which has time complexity $\mathcal{O}\left(2^{128}\right)$. The idea of this attack is simply to brute force two distinct RC4 keys with 104 unknown bits in each and then apply the attack described in this paper to recover the 128-bit Temporal Key and the 64-bit message authentication key with additional $\mathcal{O}\left(2^{38}\right)$ effort. This attack has time complexity $\mathcal{O}\left(2^{105}\right)$ which still is not practical, but it is a significant reduction.

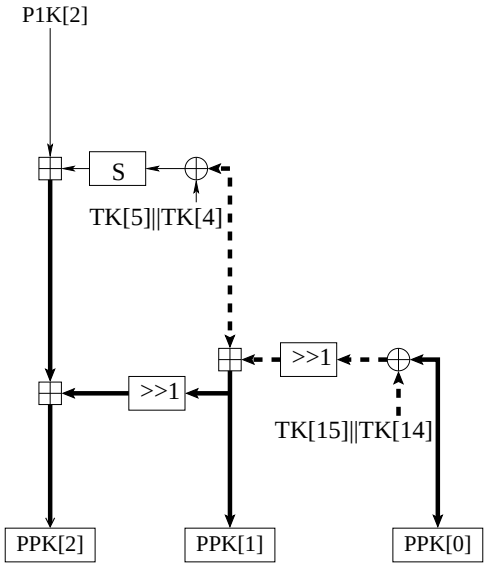

Figure 10: Part of Phase 2 needed to compute TK[4] and TK[5].

4 Discussion of the attack

The attack described in the previous section has been implemented on a computer to verify its correctness. The processor used is an Intel Pentium 4 2.53 GHz, and it takes about 6-7 minutes to recover the TK given four or more RC4-keys. Given only 2 texts the work is a factor 2^7 bigger, which gives a running time of approximately 15 hours.

This makes it a highly practical attack, but if the results of this paper shall have any impact, the important question is: "How likely is it that an attacker gets two RC4-keys generated under the same IV32?" The answer to this question depends on the implementation. The main contribution of this paper is therefore to highlight the weak spots of WPA.

4.1 Loss of RC4-key = total loss of security

The most important thing to keep in mind for an implementor is that RC4-keys used to encrypt packets, and the TK are equally important to keep secret. As this paper shows, the loss of a few RC4-keys allows the attacker to recover the long term secret TK, and not only the contents of the compromised packets.

In [6], describing the details of the MIC part of WPA, it is stressed that the integrity of a packet relies on the fact that each packet is encrypted. The author points out that the loss of one RC4-key allows the attacker to recover the MIC-key, and thus produce a valid MIC-tag to a packet of her choice. However, the attacker needs to block the receiver from receiving this and any subsequent packets until the modified packet has been inserted on the channel. Also, the attacker can only modify the particular compromised packet, since the other packets are protected by encryption using other keys. This seems to limit the threat to that of an active adversary with capabilities of blocking the receiver, being able to modify compromised packets.

This paper shows that the compromise of two or more RC4-keys is much more serious. The attacker may recover all secret keys the user has, and can therefore perform any action the user can do. In particular, WPA provides no forward secrecy since the attacker can construct earlier RC4-keys, as well as future ones, once some keys have leaked.

4.2 Cut-and-paste cryptographic primitives

It is quite common to take (parts of) existing cryptographic primitives and construct new cryptographic algorithms from them. For example, WPA uses RC4 for encryption, and parts of the AES round function in the temporal key hash. Ferguson [6] warns that the MIC function used in WPA is only secure in this particular setting. We would like to issue the same warning when it comes to temporal key hash: it is no good as a hash function, but only as a key generator.

4.3 Possible attack scenario

It is possible to imagine a situation where a leader of a group is communicating on behalf of the whole group. Some parts of the information the leader receives he wants to keep to himself, while other parts should be broadcast to all the group members. For instance, a review form for conference submissions often contains the fields "comments to program chair only" and "comments to program committee". Since WPA is a wireless network, all members of the group can receive the packets broadcast from the access point, but only the leader holding the TK can decrypt the packets. Some of the packets are for the whole group to read. The leader can choose to broadcast the contents of these packets to the group, but since the group members can receive the encrypted versions of these packets themselves, a cheaper way would be to just broadcast the RC4-keys for the packets in question, and let each member do the decryption himself.

Someone not aware of the shortcomings of the temporal key hash might opt for this solution, not knowing this allows the whole communication to be read by everyone.

5 Summary

We have shown the whole security in WPA relies on the secrecy of all packet keys. Given one packet key it is possible to find the MIC key and given two packet keys with the same IV32 an attacker can do anything the legitimate user can, for the duration of the TK.

Since these packet keys are supposed to be kept secret, the attack in this paper does not imply that WPA is broken, but it underlines the importance of keeping each and every packet key secret.

References

[1] IEEE 802.11 WG, *802.11b: Standards for Local and Metropolitan Area Networks: Wireless Lan Medium Access Control (MAC) and Physical Layer (PHY) Specifications.* IEEE, 1999.

[2] Airsnort, last visited: June 12th, 2006. [Online]. Available: airsnort.shmoo.com

[3] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting mobile communications: The insecurity of 802.11," in *MOBICOM 2001*, 2001.

[4] S. Fluhrer, I. Mantin, and A. Shamir, "Weaknesses in the key scheduling algorithm of RC4," *Lecture Notes in Computer Science*, vol. 2259, pp. 1–24, 2001. [Online]. Available: citeseer.ist.psu.edu/fluhrer01weaknesses.html

[5] R. Housley, D. Whiting, and N. Ferguson, "Alternate temporal key hash," IEEE doc. 802.11-02/282r2, 2002.

[6] N. Ferguson, "Michael: an improved MIC for 802.11 WEP," IEEE doc. 802.11-2/020r0, 2002.

[7] IEEE 802.1 WG, *802.1x: Standards for Local and Metropolitan Area Networks: Port-Based Access Control.* IEEE, 2001.