Taking a different approach to attack WPA2-AES, or the born of the

CCMP known-plain-text attack

Domonkos P. Tomcsanyi

<domonkos@tomcsanyi.net>

Lukas Lueg

<lukas.lueg@googlemail.com>

April, 2010

Abstract

In this paper we describe a new approach in attacking IEEE802.11 wireless networks protected by the WPA2-AES CCMP encryption and authentication mechanism. Our method uses encrypted data from which some bytes are known to speed up the password recovery process. As far as we know this is the first major breakthrough in Wireless-LAN security history, since the Beck-Tews WPA-TKIP keystream attack[1] in 2008. Our implementation of the attack is not fully optimized yet, but it is already 50% faster than the original attack based solely on the four-way handshake. We also defined a new theoretical maximum for WPA2-AES password recovery speed.

[1] Tews, Erik, and Martin Beck. "Practical Attacks against WEP and WPA." *Aircrack-ng.org*. 8 Nov. 2008. Web. 22 Apr. 2011. <http://dl.aircrack-ng.org/breakingwepandwpa.pdf>.

Introduction

IEEE802.11 networks are really common todays especially in SOHO, but also in enterprise environments providing users an easy way to access the internal network and also the Internet. Hence it is widely used to transmit sensitive information privacy and integrity must be taken care of.

To address this need the IEEE 802.11 standard provided first WEP, but since it had many flaws in it a new standard was created to correct these problems, called WPA. Meanwhile the IEEE started working on a totally new standard to make WiFi secure once and for all. They came up with the standard WPA2. It uses the AES block cipher in Counter Mode to encrypt data and to ensure the integrity of a packet. Today, except 802.1x RADIUS based authentication which is only used in enterprise environment, there is nothing more secure than WPA2-AES. The attack we are proposing, as far as we know is the first weakness found in WPA2-AES which therefore obviously has a serious impact on the safety of all WiFi networks around the globe.

We divided this paper in sections. Section 1 describes thoroughly how WPA2-AES works, by going step-by-step from the password to an encrypted packet (MPDU) and also along that we describe the current attack on the passphrase used to protect the network. Then in Section 2 we give a description of the initial idea of the attack. Last but not least in Section 3 we show the advanced and optimized version of our attack not only in theory but also implemented and confirmed by benchmark results.

Section 1 – How does WPA2-AES CCMP PSK work and what was so far the only attack against it?

First of all a shared secret is needed, that is called Pre-Shared Key or PSK. This is usually a password, which, according to the WPA standard has to be between 8 and 63 characters. After a password was entered the Access Point (AP) computes the master key, from which all other keys will be derived later. This key is called Pairwise Master Key or PMK, and it is simply the entered passphrase salted with the network's name (SSID) and the length of the SSID hashed 4096 times with PBKDF2 – which actually relies on HMAC-SHA1 (Figure 1).

PMK = PBKDF2(passphrase, ssid, ssidLength, 4096, 256)

PBKDF2 (P, S, c, dkLen)

Options:	PRF	underlying pseudorandom function (hLen denotes the length in octets of the pseudorandom function output)
Input:	P	password, an octet string
	S	salt, an octet string
	c	iteration count, a positive integer
	dkLen	intended length in octets of the derived key, a positive integer, at most (2^32 - 1) * hLen
Output:	DK	derived key, a dkLen-octet string

Figure 1[2]

After this operation completed the PBKDF2 function outputs 256 bits (=32 bytes) of pseudo-random bytes which are then used as a master key in every operation.

The next step is to compute keys that are actually used for encryption and securing the integrity of data. This happens when a client wants to connect to the AP. During association a 4-way-handshake happens which has two reasons: firstly it authenticates both the client and the AccessPoint, secondly it triggers the computation of the session keys. Together all the session keys are referred as Pairwise Transient Key or PTK. In computing those keys a whole new set of PRNG-function are used. These functions were explicitly created for being used in WPA/WPA2, each of them incorporates a different text string into the input. "These functions are referred to as PRF-n, where n is the number of bits required. The defined choices are:

- PRF-128
- PRF-256
- PRF-384
- PRF-512

Each function takes three parameters and produces the desired number of random bits. The three parameters are:

1. A secret key (K)
2. A text string specific to the application (e.g., nonce generation versus pairwise key expansion)

[2] RSA Laboratories, Kaliski B. "PKCS #5: Password-Based Cryptography Specification Version: 2.0." *IETF.org*. Sept. 2000. Web. 26 Apr. 2011. <http://www.ietf.org/rfc/rfc2898.txt>.

3. Some data specific to each case, such as nonces or MAC addresses

The notation used for these functions is: PRF-n(K, A, B). So, for example, when we specify that the starting value for the nonce is:

Starting nonce = PRF-256(Random Number, "Init Counter", MAC || Time)

it means that the PRF-256 function is invoked with:

K = Random number

A = The text string "Init Counter"

B = A sequence of bytes formed by the MAC address followed by a number representing time

In a similar way, the computation of the pairwise temporal keys is written:

PRF-512(PMK, "Pairwise key expansion", MAC1||MAC2||Nonce1||Nonce2)

Here MAC1 and MAC2 are the MAC addresses of the two devices where MAC1 is the smaller (numerically) and MAC2 is the larger of the two addresses. Similarly, Nonce1 is the smallest value between ANonce and SNonce, while Nonce2 is the largest of the two values.

The group temporal keys are derived as follows:

PRF-256(GMK, "Group key expansion", MAC||GNonce)

Here, MAC is the MAC address of the authenticator, that is, the access point for infrastructure networks.

We see how all the various keys can be derived by using PRF-n, so how is the PRF-n implemented? Obviously, this has to be carefully specified if we hope to have interoperability.

All the variants of PRF are implemented using the same algorithm based on HMAC-SHA-1.[8] HMAC-SHA-1 is a hashing algorithm, it is approved by the US National Institute for Science and Technology (NIST; www.nist.gov), which publishes the details of the algorithm. It takes in a stream of data and produces a message digest of fixed length, e.g. 20 bytes. The message digest is quite unpredictable (except by using the algorithm) and tells you nothing about the contents of the message that was "digested." Even if you changed one single bit in input message, an entirely new digest would be produced with no apparent connection to the previous value. There's a clue to how we can make a hashing algorithm into a pseudorandom number generator.

[8] SHA stands for secure hash algorithm.

We take a message and hash it using HMAC-SHA-1 to get a 160-bit (20-byte) result. Now change one bit in the input message and produce another 160 bits. We already know that this 160 bits appears unrelated to the first one, so if we put them together, we get 320 bits of apparently random data. By repeating this process, we can generate a pseudorandom stream of almost any number of bits. This is how HMAC-SHA-1 is used to implement the PRF-n functions. Here are the details:

1. Start with the function PRF-n (K, A, B) where n can be 128, 256, 384, or 512.
2. Initialize a single byte counter variable i to 0.
3. Create a block of data by concatenating the following:
    - A (the application-specific text)
    - 0 (a single 0 byte)
    - B (the special data)
    - i (the counter, a single byte value)

    This is written: A|0|B|i

4. Compute the hash of this block of data using key K:

    r = HMAC-SHA-1(K, A|0|B|i)

5. Store the value of r for later in a register called R.
6. Now repeat this calculation as many times as needed to generate the needed number of random bits (because 160 are generated each time, you may get more than you need, whereupon the extra bits are discarded). Before each iteration, increment the counter i by one and after each iteration appending the result bits r to the register R.

After the required number of iterations, you have your random stream of bytes." [3]

After the both the AP and the client got their Nonces from the PRF-256 function the 4-way handshake begins. First the AP sends its random number, A_Nonce to the client. This naturally happens in plain-text since no keys have been set up yet, therefore it is not possible at all to protect the A_Nonce, but it is not necessary because all an attacker can do is to change it to something arbitrary but that would result in an error later as the session key computed by the client will not match the session key of the AP.
Right now the client has everything needed to compute the Pairwise Transient Key, hence it does so by using the PRF-384 function (if it uses WPA2, in WPA the PRF-512 function is used because of the different key-hierarchy). In WPA/WPA2 each key is 128 bit long, so the PTK is cut in three equally long parts. The resulting three keys are used for the following purposes: one is used to encrypt data during the initialization phase, one is used to encrypt data during regular communication and one is used to calculate MessageIntegrityCheck (MIC) values on every packet so that possible tampering could be detected easily.
The AP needs to compute the same three keys to be able to communicate with the client, but it needs the client's random number to do so, so the client now sends S_Nonce to the

[3] Edney, Jon, and William A. Arbaugh. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Boston, MA: Addison-Wesley, 2004. Print.

AP. This is still unencrypted, but the S_Nonce has a MessageIntegrityCheck (MIC) attached to it. The AP can calculate the PTK and then go back and verify the MIC. A successful verification indicates two things: there was no tampering with the data while it was transferred through the insecure channel (air) but also the client has the correct PMK so it is authenticated to connect to the network.
Thirdly the AP sends a message to the client claiming that it is ready to install the keys, this message contains some encrypted data (which is from the point of this paper is not important) and also a MessageIntegrityCheck value. Both the encrypted data and the MIC is used by the client to verify that the AP knows the PMK, so it is authenticated to communicate with the client.
Last, but not least the client sends a message to the AP saying that everything went well and they may start using the keys. Encrypted communication begins.

In WPA2 AES is used to encrypt individual packets. AES is a standardized block-cipher function, which means it encrypts data in blocks. In the case of AES this block size is 128 bits, which is actually equal to the size of the key used during encryption and also to the size of the output produced. To use a block-cipher the easiest and logical way would be to divide the plain-text into 128-bit-long chunks and encrypt each one of them (Electronic Code Book, or ECB mode). There is only one flaw in this idea, namely if the plain-text contains a pattern that pattern would show up in the encrypted data too. For example the encrypted version of a plain-text that contains 64 times the letter “A” would be the same over and over again (Figure 2) which is not recommended at all; an attacker would be able to easily spot this specific pattern in the encrypted data.

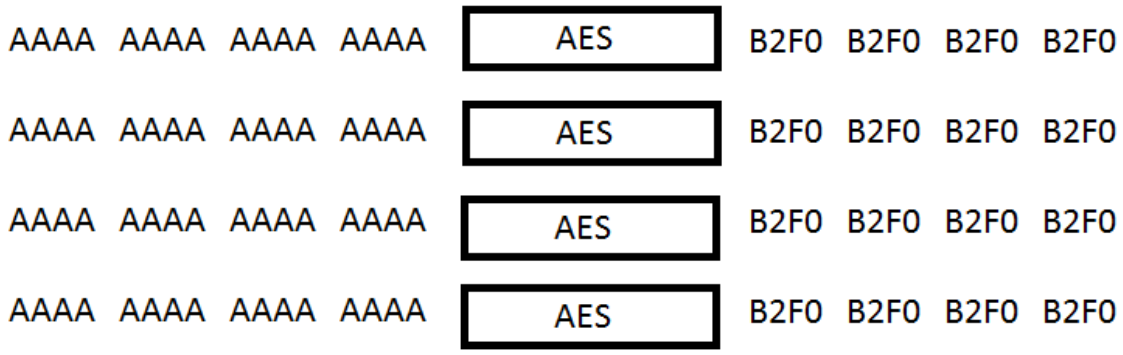

Figure 2

Therefore an other mode was created called the counter-mode. In this not the actual data but a counter is encrypted and then the encrypted version of the counter is XOR-ed to the plain-text. The counter’s value is changed after each block; this ensures that even if the plain-text has a pattern in it that won’t show up in the encrypted data (Figure 3).

Figure 3

Because of the XOR function's special property (A XOR B = C -> C XOR B = A and C XOR A = B) for decryption only the value of the counter and the AES-key is needed. If more than one block of data is encrypted only the first value of the counter has to be known, since the counter used to encrypt any data block *n* could be calculated as *INITIAL_COUNTER+n*.

In WPA2 AES-counter mode is used to encrypt data so it happens like this:
An unencrypted packet (MPDU) arrives from the operating system to the driver stack. First the IEEE80211 MAC header is separated from the packet, because it will be transferred unencrypted. Then an initial counter value is generated by using the PacketNumber, the Flags the packet has and some other data. This value will be included later unencrypted in the packet between the MAC-header and the encrypted payload (also known as the CCMP-header). A MessageIntegrityCheck value is computed over the whole packet (MAC header + CCMP header + payload) and appended to the end of the payload. Next the payload + the MIC gets encrypted by using AES-counter mode, the initial value for the counter being derived from the CCMP header. The MAC-header and the CCMP-header gets prepended to the encrypted payload and the packet is ready for transmission.

Current attack on WPA2-AES CCMP PSK – the Key Confirmation Key attack

The current attack solely relies on the 4-way-handshake. The idea behind the attack is that during the handshake process one key from the PairwiseTransientKey is used to sign the client's random number, S_Nonce. So the attacker does the following: first he needs to capture a full 4-way-handshake. This could be achieved in two ways: either the attacker does passive sniffing on the target network and waits for a client to connect or (if there is already a client connected) it could DeAuthenticate a client from the network (to send DeAuthentication packets to a client the attacker doesn't have to know the key!) forcing the client to do the 4-way-handshake again.

After the handshake is known the attack could be performed without being in the range of the target network. To recover the passphrase the attacker first needs to compute the PairwiseMasterKey using a possible password. This is a really resource-hungry process, since it needs to do all the 4096 rounds of PBKDF2. After that it needs to calculate the PTK using the two captured random numbers, this requires him to do one round of PRF-384. Why? The order of the keys in the PTK is strict, which means the first key is always the EAPOL_integrity

key. Since the first round of SHA-1 outputs 160 bits it is enough to do one iteration of PRF-384 to get the first 128 bits of the PTK, which is again the EAPOL_integrity key.
After that the attacker calculates the MessageIntegrityCheck value for the S_Nonce that once has already been signed (=MessageIntegrityCheck value was calculated above it) by a valid client. If the two of MICs match that means that the used PTK was correct, which means the used PMK was correct which means the original password used was correct (Figure 4).

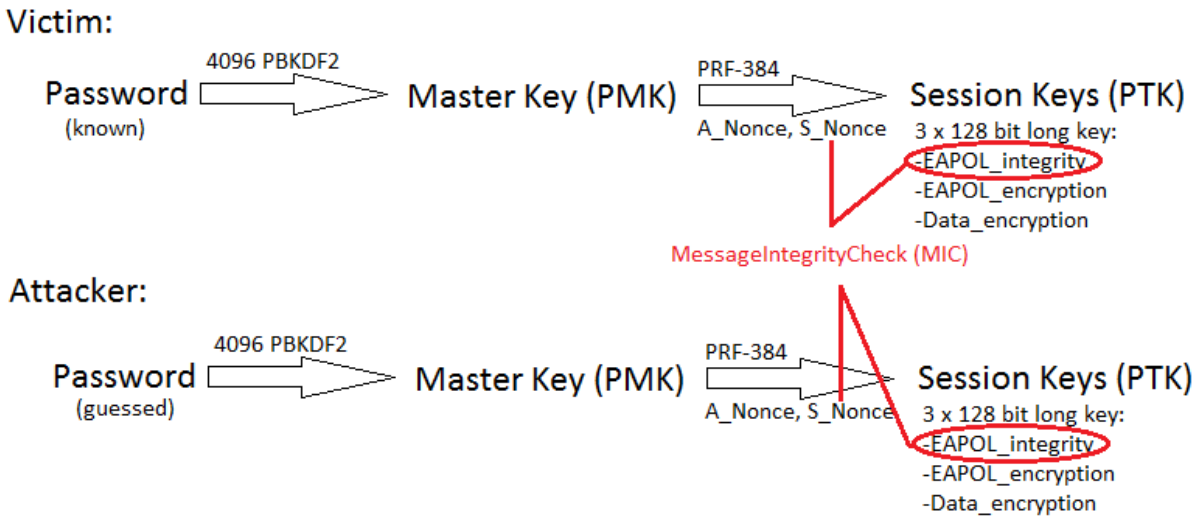

Figure 4

Processing power needed (after the PMK has been computed):
To calculate the first 160 bits of the PTK and to calculate a MessageIntegrityCheck value above the S_Nonce a total of 12-15 SHA-1 operations are needed (EAPOL frames don't have a fixed length, so calculating the MIC does not always require the same amount of SHA-1 iterations).

Section 2 – The idea of a different approach to attack WPA2

The whole research was started by Domonkos Tomcsanyi, section 2 is only a collection of his brainstorming and main idea.

When I first came up with the idea to attack WPA2 somehow differently I really had no other way in my mind, but to read the standards and try to find something that could help me. In the first step I looked at the current cracking process, and immediately saw it could be divided up in two parts: part 1 starts with a password and ends after computing the PMK part 2 is the rest (computing the PTK, calculating a MIC and check it against the captured MIC) – figure 5.

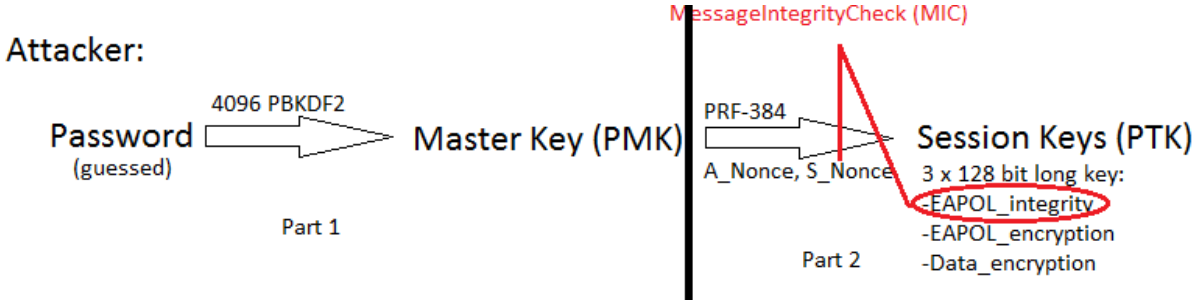

Figure 5

Part one has already been researched and optimized: since only the only variable used to calculate the PMK is the name of the network (SSID) hash databases could be created by using a good list of passwords and some common SSIDs. Hence right now there is nothing else I can do to speed up part one or change anything it, the whole process is standardized, there is no way around it – yet. So I focused on part two namely how the keys from the PTK are used to actually encrypt data.

First thing I noticed that the initial counter value used during encryption is transferred as plain-text. I thought, well there it is, I know the input of the AES function, if only I could figure out what the output was I could already start a new attack by trying different Data_encryption keys and counter values as inputs for AES and checking the output against the real output.

But how to figure out, what the output was? I know that it is the encrypted version of the (known) counter (Figure 6).

Figure 6

One thing is obvious, the encrypted data is created like this:
UNKNOWN_ENCRYPTED_COUNTER_VALUE xor PLAIN_TEXT = ENCRYPTED_DATA

Well, if we formulate this equation a little bit different, like this:

UNKNOWN_ENCRYPTED_COUNTER_VALUE = ENCRYPTED_DATA xor PLAIN_TEXT

Okay, so if we know the plain-text version of a packet, then we can calculate the encrypted version of the counter, which means this equation will only have one changing value, the key:
AES(INITIAL_COUNTER, KEY) = ENCRYPTED_COUNTER_VALUE

How do we figure out what the plain-text was? Well in WiFi it is certainly not a hard task: first of all every single packet contains a so called LLC header and a SNAP header. These 8 bytes are constants and they are always at the beginning of the packet. Well, AES uses 16-byte-long blocks, so we need 8 more bytes. The answer: ARP-packets! ARP-packets always have a constant 8-byte-header and after the header comes a 6-byte-long MAC address which is actually transferred unencrypted in the MAC-header too. So if we add this up we know 22 bytes of an ARP packet:

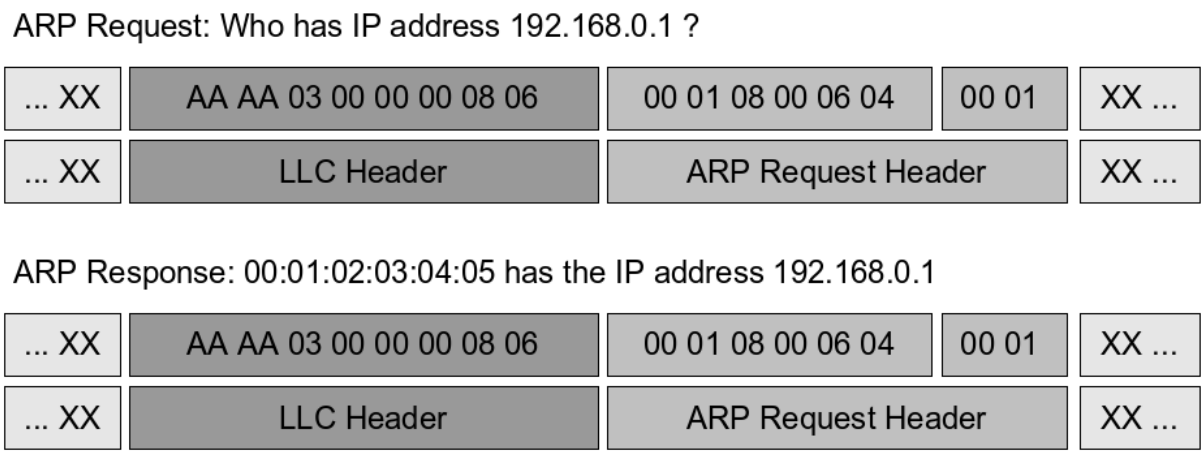

Figure 7[4]

22 bytes are more than enough, so now there is only one more question: how do we distinguish between ARP and IP packets? Well the paper "The fragmentation attack in practice"[5] presented an excellent way to do this: by length! The length of an ARP packet is far less than any IP packet in fact they are the shortest packets of all.

To summarize the initial idea again: we do everything just like with the original (KCK) attack until we capture the 4-way-handshake, but after that we don't stop sniffing; we keep looking for an ARP packet, which could be recognized by its typical (really short) length. After we found one, we re-construct the initial value of the counter from the CCMP-header of the ARP packet, then we XOR the first 16 bytes of the packet with the concatenated string of the LLC/SNAP header and the ARP-header, resulting in the encrypted version of the counter. Now we launch the attack by feeding the initial counter and a key into AES and see if the output matches the encrypted counter we just unveiled in the previous step.

[4] *ARP Header*. Photograph. SarWiki - Humbolt University, Berlin. *SarWiki*. Humboldt University - Berlin, June-July 2007. Web. 25 Apr. 2011. <http://sarwiki.informatik.hu-berlin.de/Image:Arp-header.png>.
[5] Bittau, Andrea. "The Fragmentation Attack in Practice." *Aircrack-ng.org*. 17 Sept. 2005. Web. 26 Apr. 2011. <http://download.aircrack-ng.org/wiki-files/doc/Fragmentation-Attack-in-Practice.pdf>.

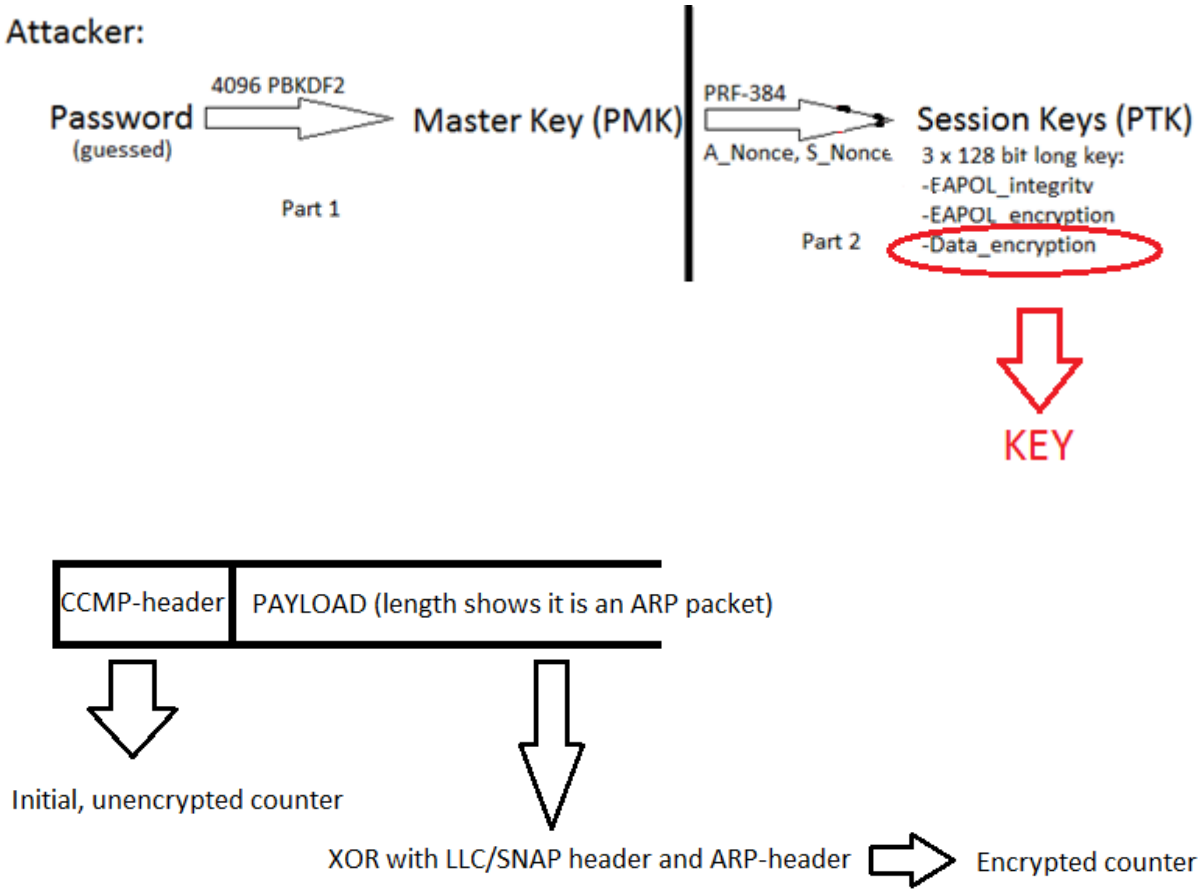

Figure 8

Section 3 – An advanced version of the attack, implementation

After coming up with the idea an implementation was necessary to show, if the new attack is any better in practice than the current KCK-attack, therefore a Lukas Lueg was invited to participate in the project. He did not only successfully implemented the idea, but also modified it to make it simpler and better. Section 3 is based on only his findings.

First of all because of easier and better implementation the original idea was turned “upside down”. From now on all we do is trying to decrypt a packet with a guessed key and see whether the LLC/SNAP and ARP-header show up in the output. If yes, than the key we used is correct, so we were able to recover the used passphrase correctly.

Second of all, the CCMP known plain-text, as Lukas named it later was not truly universal since it heavily relies on an ARP packet which might be hard to sniff in a reasonable time-interval. His idea was that instead of knowing a full 16-byte AES-block of a packet only one half of it would be sufficient. One half means 8 bytes which is exactly the amount we know of EVERY wireless packet – the LLC/SNAP header. Chanced are that we find a key which

actually decrypts the first 8 bytes of a packet correctly, but fails on the other 8 bytes (=on one AES-block) are practically zero.

Now we need to see, if the CCMP known plain-text attack is faster or not than the currently used KCK attack.

Let's see, for the KCK attack we needed one round of PRF-384 to get the EAPOL_integrity key that was used to calculate the MIC-value on the 2nd packet of the 4-way-handshake (S_Nonce). To calculate the MIC we needed to do more rounds of HMAC_SHA1, in total we ended up with 12-15 rounds of SHA-1.

To calculate the Data_Encryption key used to encrypt a packet we need to do fifteen rounds of SHA-1, this is three iteration of PRF-384 because this key is the last one in the PTK. However, as Lukas discovered there are two (as far as we know not yet published/discussed) weaknesses in the PRF-384 function:
1. The output of iteration one is not fed into the input of iteration two, which means different iterations could be calculated independently, since the only difference between two iterations is the last byte, in which a counter value is initialized and changed after each iteration. So to get the results of iteration two and three the value of iteration one does not need to be calculated. We already reduced the number of iteration by one. (Note: we need iteration two and three to calculate the Data_integrity key, because the function outputs 20 bytes per iteration, but a key is only 16 bytes long. This means that the second key in the PTK – EAPOL_encryption – contains 4 bytes from iteration one and twelve from iteration two which ultimately shows that the third key includes 8 bytes of the second and 8 bytes from the third iteration).

2. The counter that changes between iterations is placed at the end of the input string which means we can "re-use the state of the SHA1-algorithm between iteration two and three as the HMAC-key and the first block of the message are the same.

Combining these two completely unnecessary weaknesses allows us to reduce the number of SHA1-rounds required to compute the *Temporal Key* from fifteen to seven. This is still more than the five rounds of SHA1 required to compute the *Key Confirmation Key* but in fact is more than fast enough: The one key-setup plus one AES-round required to confirm the *Temporal Key* can be done much faster than the four to six rounds of SHA1 required to check the *Key Confirmation Key*. This is especially true as we can utilize hardware-based implementations of AES with the new AES-NI instruction-set found in recent processors."[6]

To summarize it: out attack only needs 7 rounds of SHA-1 while the KCK attack needed 12-15 so it should give us around 50% gain in speed.

The CCMP known plain-text attack was implemented in the tool called Pyrit. Pyrit is a complex program, mostly written in Python, which is capable of using GPUs and CPUs to do and also speed up WPA/WPA2 key recovery. The newer version of Pyrit which contains our attack showed on our benchmarks that the estimated 50% gain was correct and possible.

[6] Lueg, Lukas. "Known-plaintext Attack against CCMP." Web blog post. *Pyrit*. Wordpress.com, 16 Apr. 2011. Web. 26 Apr. 2011. <http://pyrit.wordpress.com/2011/04/16/known-plaintext-attack-against-ccmp/>.

Perspective of our research

Looking only at the numbers does not show a huge gain speed – at least not a gain that is practical. However this research was started to find a new and faster way to attack WPA2 in practice it became a more theoretical than a practical research. The main achievement for us is the fact that we actually find a new theoretical maximum for the speed of WPA2 password recovery.

It would be still possible to speed up both the KCK and the CCMP known plain-text attack by for example moving all the SHA-1 operations to GPUs, but right now it is impractical for two reasons:
1. HDDs would bottleneck the whole process, since we are still getting all the PMKs from a pre-computed database that is stored on a hard-drive. 15 million PMKs/sec means around 510 MB/sec. This speed is certainly achievable, but not with today's common HDDs, and still this speed would not be the maximum possible.

2. Using GPU power would require exceptional hardware, not only in HDDs but in graphics card(s) too. Pyrit's main goal is to have a pre-computed database, possibly computed with GPU power and every single optimization possible than make this database available for download, so people without having access to fast GPUs could still do a fast recovery of their WPA/WPA2 passwords by only using CPU power.

Conclusion

We presented a new way to attack WPA2 passphrases by using not only the data we gathered from the 4-way-handshake but also using encrypted data sniffed after the handshake happened. Because of many optimization and two (so far) not discussed problems in the PRF-384 function our attack is 50% faster than its predecessor the KCK attack. It does not require any special modification to drivers, it does not depend on a certain WiFi chipset. Currently there is no way to prevent a CCMP known plain-text attack since the problems discovered are in the standards hence they can't be changed or fixed easily. Still we have to acknowledge the fact that from practical means this attack has not improved the speed of the recovery by dangerously much. Its main achievement is the new theoretical maximum for the speed of a WPA2-passphrase recovery attack.

References – Works Cited

ARP Header. Photograph. SarWiki - Humbolt University, Berlin. *SarWiki*. Humboldt University - Berlin, June-July 2007. Web. 25 Apr. 2011. <http://sarwiki.informatik.hu-berlin.de/Image:Arp-header.png>.

Bittau, Andrea. "The Fragmentation Attack in Practice." *Aircrack-ng.org*. 17 Sept. 2005. Web. 26 Apr. 2011. <http://download.aircrack-ng.org/wiki-files/doc/Fragmentation-Attack-in-Practice.pdf>.

Edney, Jon, and William A. Arbaugh. *Real 802.11 Security: Wi-Fi Protected Access and 802.11i*. Boston, MA: Addison-Wesley, 2004. Print.

Lueg, Lukas. "Known-plaintext Attack against CCMP." Web log post. *Pyrit*. Wordpress.com, 16 Apr. 2011. Web. 26 Apr. 2011. <http://pyrit.wordpress.com/2011/04/16/known-plaintext-attack-against-ccmp/>.

RSA Laboratories, Kaliski B. "PKCS #5: Password-Based Cryptography Specification Version: 2.0." *IETF.org*. Sept. 2000. Web. 26 Apr. 2011. <http://www.ietf.org/rfc/rfc2898.txt>.

Tews, Erik, and Martin Beck. "Practical Attacks against WEP and WPA." *Aircrack-ng.org*. 8 Nov. 2008. Web. 22 Apr. 2011. <http://dl.aircrack-ng.org/breakingwepandwpa.pdf>.