Security and Cryptography 2021

Mirosław Kutyłowski

X. WIFI

standards:

- evolution
- little interaction with academic community
- underspecified,
- sometimes not literally implemented, lack of documentation
- sometimes formal security proofs – like for WPA, but nevertheless ... attacks

Learning from early mistakes: WEP

- stream encryption
- PRNG reinitialized frequently, the seed is the frame identifier + shared seed
- problems with PRNG algorithm RC4

RC4 KSA (Key Scheduling Algorithm) for key K

```
for i=0 to 255 do
    S[ i ] := i
end
j := 0
for i= 0 to 255 do
    j:= j+S[i]+K[i mod len(K)] mod 256
    swap(S, i , j )
end
i:=0, j:=0
```

RC4 PRNG, execute the following loop as long as output needed:

```
i:=i + 1 mod 256
j:= j + S[i] mod 256
swap(S, i , j )
return S[ S[ i ] + S[j] mod 256 ]
```

FMS Attack (**Fluhrer, Mantin, Shamir**)

- assumption: the adversary already knows the first l bytes of the key AND the first output byte of RC4 PRNG
  - so the adversary can perform the first l steps of Key Scheduling Algorithm
  - the goal will be to learn one more byte of the key
- assumptions about the state of KSA after l steps for a given initial vector:

→ $S_l[1] < l$

→ $S_l[1] + S_l[S_l[1]] = l$

- validity of the assumption for a given initial vector can be checked by simulation of the first l steps
- **assume that:**
  - $S_l[1], S_l[S_l[1]], S_{l+1}[l]$ did not participate in any swaps during the rest of the KSA (it is likely to occur)
- **then**:
  - for the generation of the first output byte we take

$$S_{n+1}[\,S_{n+1}[1] + S_{n+1}[S_n[1]]\,]$$

  - if the assumption was ok then this is the same as

$$S_{n+1}[\,S_l[1] + S_l[S_l[1]]\,] = S_{n+1}[l] = S_{l+1}[l] = S_l[j_{l+1}]$$

the last equation follows from the fact that at the step $l+1$ there is a swap at positions l and j_{l+1}

- from the output byte we derive the candidate for j_{l+1} (the position of the output byte in S_l)
- on the other hand: $j_{l+1} = j_l + K[l] + S_l[l]$, so we may derive a candidate for $K[l]$
- the first swap of PRNG will swap $S[1]$ and $S[S[1]]$ and this does not change the value of $S[1] + S[S[1]]$,
- HOWEVER the output value would be affected by the swap if $S[1] + S[S[1]] = 1$ or $S[1] + S[S[1]] = S[1]$

→ case: $S[1] + S[S[1]] = 1$

then: since the values has not been changed (assumption), we would have $S_l[1] + S_l[S_l[1]] = 1$ as well. But it is equal to l by another assumption. The case is impossible to occur!

→ case: $S[1] + S[S[1]] = S[1]$

then: $S_l[1] + S_l[S_l[1]] = S_l[1]$, so $S_l[S_l[1]] = 0$. But $S_l[1] < l$ and $S_l[1] + S_l[S_l[1]] = l$, so we must have $S_l[S_l[1]] > 0$. The case is impossible.

so the swap

Krack against WPA2

- attack based on crypto assumption: “no IV used twice”
- works despite “provable security”, but the proofs have not modelled all scenarios
- effects depend on particular implementation. Most cases:
  - decryption due to reuse of the same string in stream cipher
  - or just making mess by replay attack (e.g. against NTP- network time protocol)

4-way handshake

- ”supplicant”= user, “authenticator”=Access Point
- PMK Pairwise Master Key is preshared
- PTK (Pairwise Transient Key) derived as a session key
- PTK=$f(\mathrm{PMK}, \mathrm{ANonce}, \mathrm{SNonce})$, PTK splitted into TK (Temporal Key), KCK (Key Confirmation Key), KEK (Key Encryption Key)
- for WPA2 also GPK (Group Temporal Key) transported to the supplicant (used by AP for broadcast)

- frames: EAPOL consisting of
  - header - determines which message it is in the handshake
  - replay counter – used to detect replayed frames, replay counter will be increased
  - nonce - nonces (of supplicant and authenticator) to generate new keys
  - RSC Receive Sequence Counter - starting packet number of a group key
  - MIC - contains Message Integrity Check created with KCK
  - Key Data - contains group key encrypted with KEK
- encryption schemes used: AES-CCMP, GCM , MAC: Michael (weak), GHASH (from GCM)

handshake:

- notation: after “;” the data are encrypted
- green background = “sometimes”
- Enc^i_K is encryption with key K and IV i

association stage	supplicant		authenticator
		Authentication request →	
		← Authentication response	
4-way handshake		← Msg1(r,Anonce)	
	derive PTK		
		Msg2(r,Snonce) →	
		← Msg3(r+1,GTK)	
			derivePTK
		Msg4(r+1) →	
	install PTK,GTK		install PTK
group key		$\leftarrow \mathrm{Enc}^x_{\mathrm{PTK}}(\mathrm{Group1}(r+2;\mathrm{GTK}))$	
handshake		$\mathrm{Enc}^y_{\mathrm{PTK}}(\mathrm{Group2}(r+2)) \rightarrow$	
	install GTK		install GTK

Table 1.

- state automaton definded, states for the supplicant:

  A PTK-INIT:

    - entered when 4 way handshake started
    - exit to state PTK-START with Msg1 received
    - operations: PMK- preshared master key

  B PTK-START:

    - exit: self loop with MSg1 received, with proper Msg3 to state PTK-NEGOTIATING (proper= MIC correct and no replay)
    - operations:
      - TPTK=CalcPTK(PMK,ANonce,SNonce)
      - Send Msg2(SNonce)

  C PTK-NEGOTIATING:

    - exit: unconditional to PTK-DONE

  - operations:
    - PTK=TPTK
    - Send Msg4

D PTK-DONE:

  - exit: to PTK-START if Msg1 received, to PTK-NEGOTIATING if proper Msg3 received

attack 1 - plaintext retransmission of Msg3

supplicant		adv		authenticator
	← Msg1(r,Anonce)		← Msg1(r,Anonce)	
derive PTK				
	Msg2(r,Snonce) →		Msg2(r,Snonce) →	
	← Msg3(r+1;GTK)		← Msg3(r+1;GTK)	
				derivePTK
	Msg4(r+1) →			
install PTK,GTK				
	$\mathrm{Enc}^1_{\mathrm{PTK}}\{\mathrm{Data}(A....)\}\rightarrow$			
	← Msg3(r+2;GTK)		← Msg3(r+2;GTK)	
	$\mathrm{Enc}^2_{\mathrm{PTK}}\{\mathrm{Msg4}(r+1)\}\rightarrow$			
resinstall PTK,GTK				
			$\mathrm{Enc}^2_{\mathrm{PTK}}\{\mathrm{Msg4}(r+1)\}\rightarrow$	(rejected)
			Msg4(r+1) →	
				install PTK
	$\mathrm{Enc}^1_{\mathrm{PTK}}\{\mathrm{Data}(B....)\}\rightarrow$		$\mathrm{Enc}^1_{\mathrm{PTK}}\{\mathrm{Data}(....)\}\rightarrow$	

mechanism:

- according to the 802.11 standard Msg4(r+1) will be accepted as it is checked that r+1 is a replay counter used before
- the problem is that $\mathrm{Enc}^1_{\mathrm{PTK}}\{\mathrm{Data}(A....)\}$ and $\mathrm{Enc}^1_{\mathrm{PTK}}\{\mathrm{Data}(B....)\}$ use the same IV but security of the encryption modes used collapse in this case

attack 2 - only ciphertext retransmission of Msg3 accepted

CPU		NIC		adv.
	← Msg1(r,Anonce)		← Msg1(r,Anonce)	
	Msg2(r,Snonce) →		Msg2(r,Snonce) →	
			← Msg3(r+1;GTK)	
			← Msg3(r+2;GTK)	
	← Msg3(r+1;GTK)			
	← Msg3(r+2;GTK)			
	Msg4(r+1) →			
	install keys →		Msg4(r+1) →	
		install PTK,GTK		
	Msg4(r+2) →			
	install keys →		$\text{Enc}^1_{\text{PTK}}\{\text{Msg4}(r+2)\} \rightarrow$	
		reinstall PTK,GTK		
	$\text{Data}(....) \rightarrow$			
			$\text{Enc}^1_{\text{PTK}}\{\text{Data}(....)\} \rightarrow$	

mechanism:

- assumption: encryption and decryption offloaded to NIC (network interface controller)

- main CPU does not decrypt messages and always receives messages already decrypted by NIC,
- so it cannot distinguish the case when Msg3(r+2;GTK) has been received as plaintext or already encrypted. In both cases the reaction is the same and asks NIC to install keys
- adversary holds the first Msg3 until the authenticator sends another one because of no response
- finally two ciphertexts created with the same IV
- the problem is that in fact there are two state machines - one for main CPU and one for NIC and collectively they are not equivalent to the original machine from the standard

attack 3 - in some systems (MacOS) the message Msg3 has to be encrypted

attack when refreshing the key

CPU		NIC		adversary
	← Msg1(r,Anonce)		← Msg1(r,Anonce)	
	Msg2(r,Snonce) →		Msg2(r,Snonce) →	
			← Enc$^{x}_{\mathrm{ptk}}$(Msg3(r+1;GTK))	
			← Enc$^{x+1}_{\mathrm{ptk}}$(Msg3(r+2;GTK))	
	← Msg3(r+1;GTK)			
	← Msg3(r+2;GTK)			
	Msg4(r+1) →			
	install keys →		Msg4(r+1) →	
		install PTK,GTK		
	Msg4(r+2) →			
	install keys →		$\mathrm{Enc}^{1}_{\mathrm{PTK}}\{\mathrm{Msg4}(r+2)\}\rightarrow$	
		reinstall PTK,GTK		
	$\mathrm{Data}(....)\rightarrow$			
			$\mathrm{Enc}^{1}_{\mathrm{PTK}}\{\mathrm{Data}(....)\}\rightarrow$	

mechanism:

- the countermeasure was that when refreshing then the Msg3 must be encrypted
- intention was that encryption with the new key so after reinstallation new key used and no problem that the counter starts again from 1
- the mistake is that it is not checked under which key the message has been encrypted

attack 4 - group key reinstallation

challenge:

when to reinstall the key for AP? Options:

a) right after sending information to the supplicants

b) after receiving ack from all supplicants

each scenario leads to problems

attack 4a - group key reinstallation immediately after sending group message

supplicant		adv		authenticator
				refresh GTK
	$\leftarrow \mathrm{Enc}^{x}_{\mathrm{PTK}}\{\mathrm{Group1}(r; \mathrm{GTK})\}$		$\leftarrow \mathrm{Enc}^{x}_{\mathrm{PTK}}\{\mathrm{Group1}(r; \mathrm{GTK})\}$	
install GTK				install GTK
	$\mathrm{Enc}^{y}_{\mathrm{PTK}}\{\mathrm{Group2}(r)\} \rightarrow$			
			$\leftarrow \mathrm{Enc}^{x+1}_{\mathrm{PTK}}\{\mathrm{Group1}(r+1; \mathrm{GTK})\}$	
	$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(\ldots)\}$		$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(\ldots)\}$	
	$\leftarrow \mathrm{Enc}^{x+1}_{\mathrm{PTK}}\{\mathrm{Group1}(r+1; \mathrm{GTK})\}$			
reinstall GTK				
	$\leftarrow \mathrm{Enc}^{1}_{\mathrm{GTK}}\{\mathrm{GroupData}(\ldots)$			

mechanism:

- after key reinstallation one can replay the old message as the index 1 will be accepted again