Wired Equivalent Privacy (WEP)

Aldrich Asuncion & Brian Guadalupe

Ma 195J.18: Principles of Cryptography
November 27, 2017

Introduction

General idea

- How to send protocol data units securely from one entity to another?
- Requires both cryptography (for security) and encoding (for data integrity)

What is WEP?

- WEP is a security protocol for wireless networks
- Based on the IEEE 802.11-1997 standard

What is WEP?

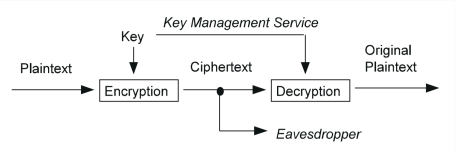
- WEP is a security protocol for wireless networks
- Based on the IEEE 802.11-1997 standard

Goal

To provide security that equates the security mechanisms of wired local access networks (LANs)

What is WEP?

- WEP is a symmetric-key algorithm



- Assumes that there is some Key Management Service which securely provides a copy of the same secret key to both the sender and recipient
- Details of Key Management Service not to be discussed in report, but one such way to allow secure key exchange is the Diffie-Hellman key exchange

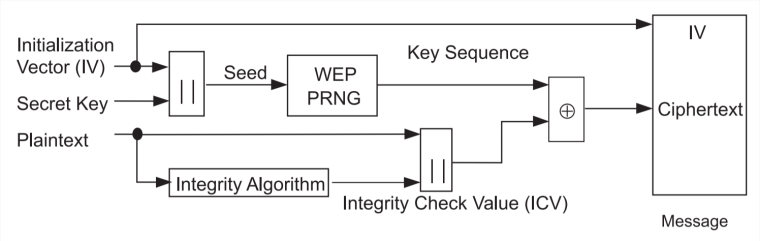
Properties of WEP

The WEP algorithm has the following properties:

1. Reasonably strong
2. Self-synchronizing
3. Efficient
4. May be exportable
5. Optional

Encryption and Encoding

Encryption process



Encryption process

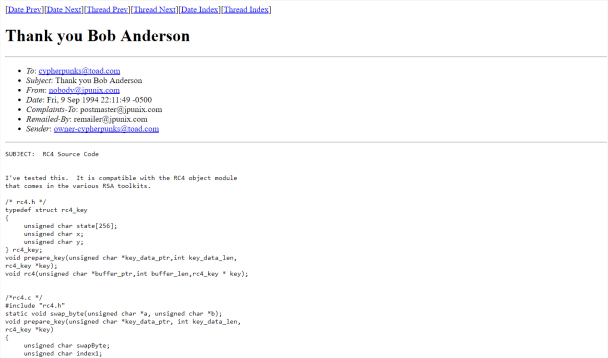
	Known by...		
	Sender	Receiver	Public
Secret Key	Yes	Yes	No
Initialization Vector	Yes	Yes	Yes (since it's useless w/o the secret key)

Pseudorandom number generator (PRNG)

- WEP uses the **RC4** algorithm for its PRNG
- RC4 is a stream cipher designed by Ron Rivest in 1987
- Originally a trade secret, RC4 was leaked and then reverse-engineered in 1994

Pseudorandom number generator (PRNG)

- WEP uses the **RC4** algorithm for its PRNG
- RC4 is a stream cipher designed by Ron Rivest in 1987
- Originally a trade secret, RC4 was leaked and then reverse-engineered in 1994



[Date Prev][Date Next][Thread Prev][Thread Next][Date Index][Thread Index]

Thank you Bob Anderson

- *To*: cypherpunks@toad.com
- *Subject*: Thank you Bob Anderson
- *From*: nobody@jpunix.com
- *Date*: Fri, 9 Sep 1994 22:11:49 -0500
- *Complaints-To*: postmaster@jpunix.com
- *Remailed-By*: remailer@jpunix.com
- *Sender*: owner-cypherpunks@toad.com

SUBJECT: RC4 Source Code

I've tested this. It is compatible with the RC4 object module
that comes in the various RSA toolkits.

```
/* rc4.h */
typedef struct rc4_key
{
     unsigned char state[256];
     unsigned char x;
     unsigned char y;
} rc4_key;
void prepare_key(unsigned char *key_data_ptr,int key_data_len,
rc4_key *key);
void rc4(unsigned char *buffer_ptr,int buffer_len,rc4_key * key);


/*rc4.c */
#include "rc4.h"
static void swap_byte(unsigned char *a, unsigned char *b);
void prepare_key(unsigned char *key_data_ptr, int key_data_len,
rc4_key *key)
{
     unsigned char swapByte;
     unsigned char index1;
```

RC4 overview

- RC4 generates a keystream, which is a pseudorandom stream of bits
- To generate the keystream, the cipher uses an internal state which consists of two parts:
  1. A permutation of all 256 possible bytes S
  2. Two 8-bit index-pointers i and j

RC4 key-scheduling algorithm

```
1: procedure KSA(K)
2:     for i = 0 to 255 do
3:         S[i] = i
4:     end for
5:     j = 0
6:     for i = 0 to 255 do
7:         j = (j + S[i] + K[i mod K.length]) mod 256
8:         Swap(S[i], S[j])
9:     end for
10: end procedure
```

KSA is the RC4 key-scheduling algorithm, which initializes i, j, and the permutation S based on a supplied key K.

RC4 pseudorandom generator

```
1: i = 0
2: j = 0
3: procedure PRG
4:     i = (i + 1) mod 256
5:     j = (j + S[i]) mod 256
6:     SWAP(S[i], S[j])
7:     z = S[(S[i] + S[j]) mod 256]
8:     return z
9: end procedure
```

For as many iterations as needed, the PRG updates the state and produces one byte (modulo 256) of output.

Integrity check algorithm

- The WEP integrity check algorithm uses **CRC-32**
- CRCs are based on cyclic codes
- CRCs are popular due to ease of implementation in hardware

CRC-32 algorithm

Definition

Suppose that the plaintext is expressed as a polynomial $P(x)$. Then

$$\mathrm{CRC}\,(P(x)) = x^{32}P(x) \bmod G(x)$$

where $G(x)$ is the generator polynomial defined by

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

CRC-32 algorithm

Definition

Suppose that the plaintext is expressed as a polynomial $P(x)$. Then

$$\mathrm{CRC}\,(P(x)) = x^{32}P(x) \bmod G(x)$$

where $G(x)$ is the generator polynomial defined by

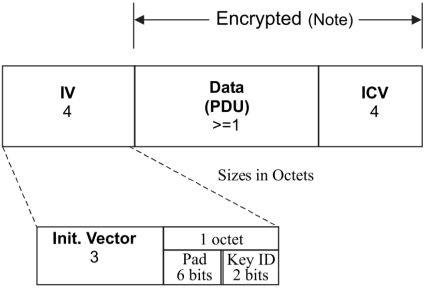
$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Property

$$\mathrm{CRC}\,(P(x) \oplus Q(x)) = \mathrm{CRC}\,(P(x)) \oplus \mathrm{CRC}\,(Q(x))$$

Decryption and Verification

Decryption process

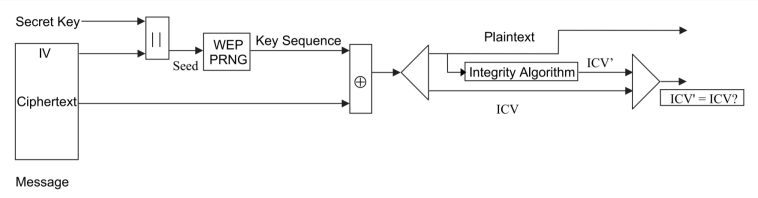


NOTE-The encipherment process has expanded the original MPDU by 8 octets, 4 for the Initialization Vector (IV) field and 4 for the Integrity Check Value (ICV). The ICV is calculated on the Data field only.

In short the receiver receives two values: (Initialization Vector, Ciphertext (contains Plaintext and CRC))

Decryption process

Using the initialization vector, the secret key and RC4, the recipient first decrypts the ciphertext.



The recipient then verifies the recovered plaintext.

Verification process

From §8.2.3 of the IEEE 802.11-1997 standard

Correct decipherment shall be verified by performing the integrity check algorithm (CRC-32) on the recovered plaintext and comparing the output ICV′ to the ICV transmitted with the message. If ICV′ is not equal to ICV, the received MPDU is in error and an error indication is sent to MAC management.

Vulnerabilities

Recap: How WEP encrypts data

- Given the plaintext (as a polynomial) $P(x)$, we first attach the CRC-32 checksum to it:

$$x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right)$$

- Recall that $\mathrm{CRC}\left(P(x)\right) = x^{32}P(x) \bmod G(x)$
- Then the RC4 algorithm generates a key $\mathrm{RC4}_{\text{IV, Secret Message}}(x)$, and we encrypt the plaintext by adding the key to it:

$$C(x) = x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\text{IV, Secret Message}}(x)$$

IV reuse

- If two packets of data use the same (publicly viewable IV), then any outside party can view the message
- Recall that in WEP, the secret message is not frequently changed, only the IV is frequently changed!

IV reuse

- Suppose an attacker acquires the following:

$$C_1(x) = x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\text{IV, Secret Message}}(x)$$
$$C_2(x) = x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\text{IV, Secret Message}}(x)$$

IV reuse

- Suppose an attacker acquires the following:

$$C_1(x) = x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x)$$
$$C_2(x) = x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x)$$

- Note that by simply adding the intercepted blocks together:

$$\begin{aligned} C_1(x) \oplus C_2(x) = {} & x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x) \\ & \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x) \\ C_1(x) \oplus C_2(x) = {} & x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \end{aligned}$$

IV reuse

- Suppose an attacker acquires the following:

$$C_1(x) = x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x)$$
$$C_2(x) = x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x)$$

- Note that by simply adding the intercepted blocks together:

$$\begin{aligned} C_1(x) \oplus C_2(x) = x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x) \\ \oplus\, x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV,\ Secret\ Message}}(x) \\ C_1(x) \oplus C_2(x) = x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \end{aligned}$$

- The packets are now vulnerable to a **known-plaintext attack**. If the attacker knows either $P(x)$ or $Q(x)$ then it is possible to recover the other message.

IV reuse

- Since the initialization vector is only 24 bits long, there are only $2^{24} = 16777216$ choices for the IV
- Because of this, an attacker can build a decryption dictionary for all keystreams

Unauthorized plaintext modification

- Suppose the attacker wants to modify the sent message
- Specifically: Add a polynomial $Q(x)$ to the message
- All the attacker has to do is intercept the message and add $x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right)$ to it

Unauthorized plaintext modification

Proof.

$$x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right)$$

□

Unauthorized plaintext modification

Proof.

$$x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right)$$
$$= \left(P(x) \oplus Q(x)\right)x^{32} \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x)$$

□

Unauthorized plaintext modification

Proof.

$$
\begin{aligned}
& x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \left(x^{32}P(x) \bmod G(x)\right) \oplus \left(x^{32}Q(x) \bmod G(x)\right) \\
&\oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x)
\end{aligned}
$$

□

Unauthorized plaintext modification

Proof.

$$
\begin{aligned}
& x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \left(x^{32}P(x) \bmod G(x)\right) \oplus \left(x^{32}Q(x) \bmod G(x)\right) \\
&\oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \left(\left(x^{32}P(x) \oplus x^{32}Q(x)\right) \bmod G(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x)
\end{aligned}
$$

□

Unauthorized plaintext modification

Proof.

$$
\begin{aligned}
& x^{32}P(x) \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \oplus x^{32}Q(x) \oplus \mathrm{CRC}\left(Q(x)\right) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \mathrm{CRC}\left(P(x)\right) \oplus \mathrm{CRC}\left(Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \left(x^{32}P(x) \bmod G(x)\right) \oplus \left(x^{32}Q(x) \bmod G(x)\right) \\
&\oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \left(\left(x^{32}P(x) \oplus x^{32}Q(x)\right) \bmod G(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x) \\
&= \left(P(x) \oplus Q(x)\right) x^{32} \oplus \mathrm{CRC}\left(P(x) \oplus Q(x)\right) \oplus \mathrm{RC4}_{\mathrm{IV},\ \text{Secret Message}}(x)
\end{aligned}
$$

□

Unauthorized plaintext modification

- The attacker has tampered with the message **and the checksum**!
- The key vulnerability is that the XOR operation used by the RC4 stream cipher is associative, and that the CRC-32 operation is linear

Unauthorized plaintext modification

- The attacker has tampered with the message **and the checksum**!
- The key vulnerability is that the XOR operation used by the RC4 stream cipher is associative, and that the CRC-32 operation is linear

Important!

CRC-32 is **not a cryptographic hash function**, so it doesn't offer the security of being one-way

Attacks on WEP

Attacks on WEP

Attacks based on statistical analysis of attempted packets have been found since 2001.

FMS attack (Fluhrer, Mantin, Shamir) The attack needs to intercept 4 to 6 million packets to succeed in obtaining the full secret key with a success probability of at least 50%.

KoreK attack The number of captured packets is reduced to about 700,000 for 50% success probability.

PTW attack (Pyshkin, Tews, Weinmann) The attack needs just about 35,000 to 40,000 packets for 50% success probability, which can be collected in less than 60 seconds on a fast network. Only a few seconds of CPU time is needed to execute the attack.

Aftermath

- The Wi-Fi Alliance announced that WEP had been superseded by Wi-Fi Protected Access (WPA) in 2003
- WEP was deprecated in 2004, with the ratification of the full 802.11i standard (a.k.a. WPA2)

References

- IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.
  IEEE Std 802.11-1997, pages 1–445, November 1997.
- Jiejun Kong, Mario Gerla, B.S. Prabhu, and Rajit Gadh.
  An Overview of Network Security in WLANs.
  In *Handbook of Wireless Local Area Networks*, pages 476–496. CRC Press, 2005.
- Erik Tews and Martin Beck.
  Practical attacks against WEP and WPA.
  WiSec '09, pages 79–86, Zurich, Switzerland, 2009. ACM Press.