

Introduction to Stream Ciphers

Attacks on CSS and WEP

Vitaly Shmatikov

Stream Ciphers

◆ Remember one-time pad?

$$\text{Ciphertext}(\text{Key}, \text{Message}) = \text{Message} \oplus \text{Key}$$

- Key must be a random bit sequence as long as message

◆ Idea: replace “random” with “pseudo-random”

- Encrypt with pseudo-random number generator (PRNG)
- PRNG takes a short, truly random secret seed and expands it into a long “random-looking” sequence
 - E.g., 128-bit seed into a 10^6 -bit pseudo-random sequence

No efficient algorithm can tell this sequence from truly random

◆ $\text{Ciphertext}(\text{Key}, \text{Msg}) = \text{IV}, \text{Msg} \oplus \text{PRNG}(\text{IV}, \text{Key})$

- Message processed bit by bit, not in blocks

Properties of Stream Ciphers

- ◆ Usually very fast (faster than block ciphers)
 - Used where speed is important: WiFi, DVD
- ◆ Unlike one-time pad, stream ciphers do not provide perfect secrecy
 - Only as secure as the underlying PRNG
 - If used properly, can be as secure as block ciphers
- ◆ PRNG is, by definition, **unpredictable**
 - Given the stream of PRNG output (but not the seed!), it's hard to predict what the next bit will be
 - If $\text{PRNG}(\text{unknown random seed}) = b_1 \dots b_i$, then b_{i+1} is "0" with probability $\frac{1}{2}$, "1" with probability $\frac{1}{2}$

Weaknesses of Stream Ciphers

◆ No integrity

- Associativity & commutativity: $(X \oplus Y) \oplus Z = (X \oplus Z) \oplus Y$
- $(M_1 \oplus \text{PRNG}(\text{seed})) \oplus M_2 = (M_1 \oplus M_2) \oplus \text{PRNG}(\text{seed})$

◆ Known-plaintext attack is very dangerous if keystream is ever repeated

- Self-cancellation property of XOR: $X \oplus X = 0$
- $(M_1 \oplus \text{PRNG}(\text{seed})) \oplus (M_2 \oplus \text{PRNG}(\text{seed})) = M_1 \oplus M_2$
- If attacker knows M_1 , then easily recovers M_2
 - Most plaintexts contain enough redundancy that knowledge of M_1 or M_2 is not even necessary to recover both from $M_1 \oplus M_2$

Stream Cipher Terminology

- ◆ Seed of pseudo-random generator often consists of **initialization vector (IV)** and **key**
 - IV is usually sent with the ciphertext
 - The key is a secret known only to the sender and the recipient, not sent with the ciphertext
- ◆ The pseudo-random bit stream produced by $\text{PRNG}(\text{IV}, \text{key})$ is referred to as **keystream**
 - PRNG must be cryptographically secure
- ◆ Encrypt message by XORing with keystream
 - $\text{ciphertext} = \text{message} \oplus \text{keystream}$

How Random is "Random?"



Cryptographically Secure PRNG

◆ Next-bit test

- Given N bits of the pseudo-random sequence, predict $(N+1)^{\text{st}}$ bit
- Probability of correct prediction should be very close to $1/2$ for any efficient adversarial algorithm

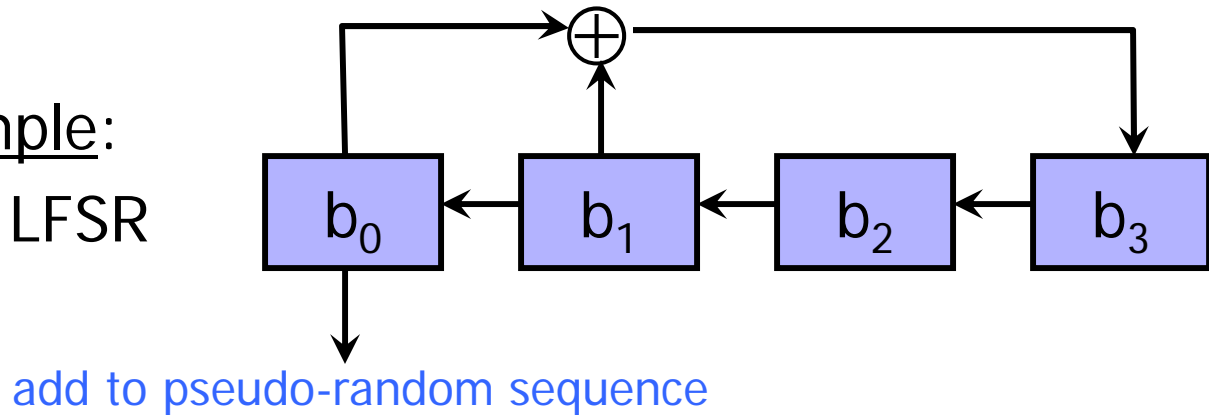
◆ PRNG state compromise

- Even if attacker learns complete or partial state of the PRNG, he should not be able to reproduce the previously generated sequence
 - ... or future sequence, if there'll be future random input(s)

◆ Common PRNGs are not cryptographically secure

Linear Feedback Shift Register (LFSR)

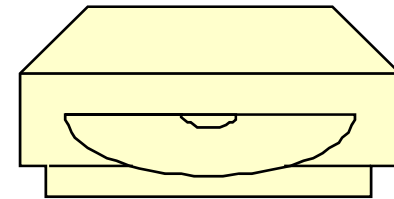
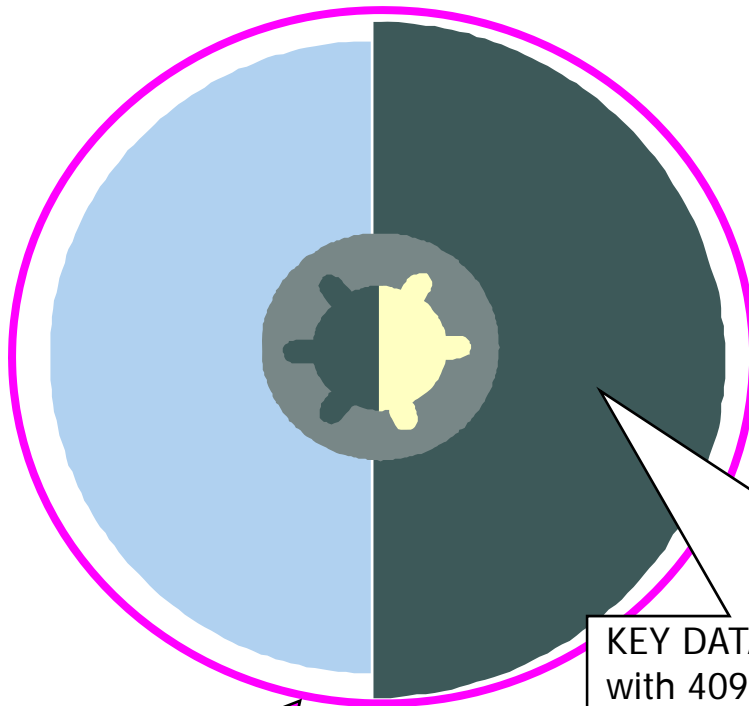
Example:
4-bit LFSR



- ◆ Key is used as the seed
 - For example, if the seed is 1001, the generated sequence is 1001101011110001001...
- ◆ Repeats after 15 bits (2^4-1)

Content Scrambling System (CSS)

◆ DVD encryption scheme from Matsushita and Toshiba



Each player has its own **PLAYER KEY** (409 player manufacturers, each has its player key)

KEY DATA BLOCK contains disk key encrypted with 409 different player keys:

- $\text{Encrypt}_{\text{DiskKey}}(\text{DiskKey})$
- $\text{Encrypt}_{\text{PlayerKey1}}(\text{DiskKey}) \dots \text{Encrypt}_{\text{PlayerKey409}}(\text{DiskKey})$

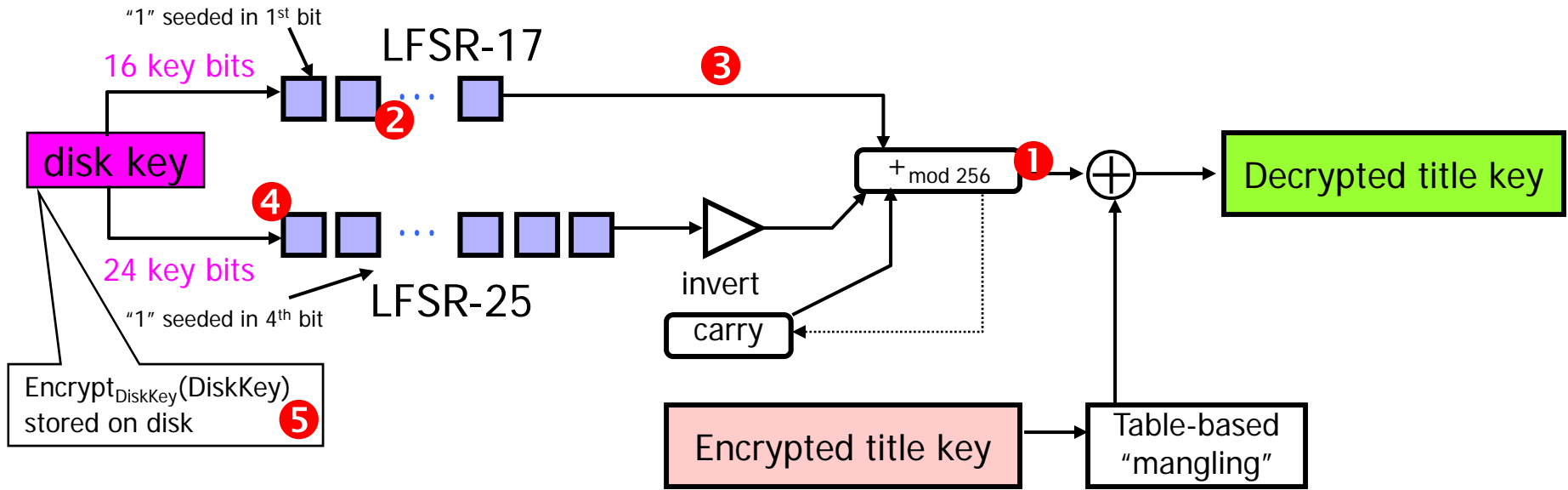
Each DVD is encrypted with a disk-specific 40-bit **DISK KEY**

This helps attacker verify his guess of disk key

What happens if even a single player key is compromised?

Attack on CSS Decryption Scheme

[Frank Stevenson]



- 1 With known ciphertext and plaintext, guess the 8 bits output by XOR; repeat 5 times to recover 40 bits – this takes $O(2^8)$
- 2 Guess 16 bits of the key contained in LFSR-17 – this takes $O(2^{16})$
- 3 Clock out 24 bits out of LFSR-17, use them to determine the corresponding output bits of LFSR-25 (this reveals all of LFSR-25 except the highest bit)
- 4 Clock back 24 bits, try both possibilities – this takes $O(2)$
- 5 Verify the key

This attack takes $O(2^{25})$

DeCSS

- ◆ In CSS, disk key is encrypted under hundreds of different player keys
 - ... including Xing, a software DVD player
- ◆ Reverse engineering object code of Xing revealed its decryption key
 - Recall that every CSS disk contains the master disk key encrypted under Xing's key
 - One bad player \Rightarrow entire system is broken!
- ◆ Easy-to-use DeCSS software

DeCSS Aftermath

- ◆ DVD CCA sued Jon Lech Johansen, one of DeCSS authors (eventually dropped)
- ◆ Publishing DeCSS code violates copyright
 - Underground distribution as haikus and T-shirts
 - “Court to address DeCSS T-Shirt: When can a T-shirt become a trade secret? When it tells you how to copy a DVD.” - From Wired News



RC4

- ◆ Designed by Ron Rivest for RSA in 1987
- ◆ Simple, fast, widely used
 - SSL/TLS for Web security, WEP for wireless

Byte array $S[256]$ contains a permutation of numbers from 0 to 255

$i = j := 0$

loop

$i := (i+1) \bmod 256$

$j := (j+S[i]) \bmod 256$

swap($S[i], S[j]$)

output $(S[i]+S[j]) \bmod 256$

end loop

RC4 Initialization

Divide key K into L bytes

Key can be any length
up to 2048 bits

for i = 0 to 255 do

 S[i] := i

 j := 0

 for i = 0 to 255 do

 j := (j + S[i] + K[i mod L]) mod 256

Generate initial permutation
from key K

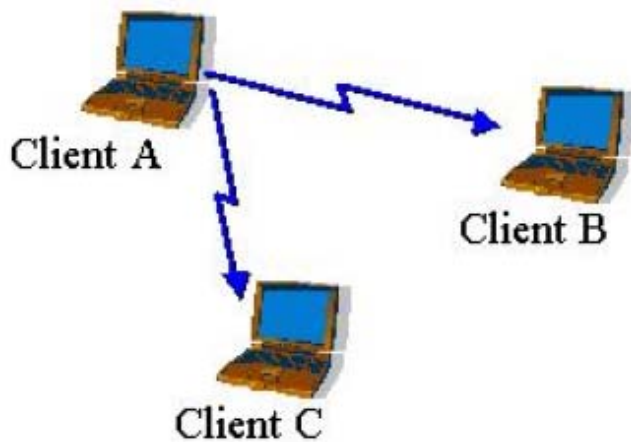
 swap(S[i], S[j])

- ◆ To use RC4, usually prepend **initialization vector** (IV) to the key
 - IV can be random or a counter
- ◆ RC4 is not random enough! 1st byte of generated sequence depends only on 3 cells of state array S. This can be used to extract the key.
 - To use RC4 securely, RSA suggests discarding first 256 bytes

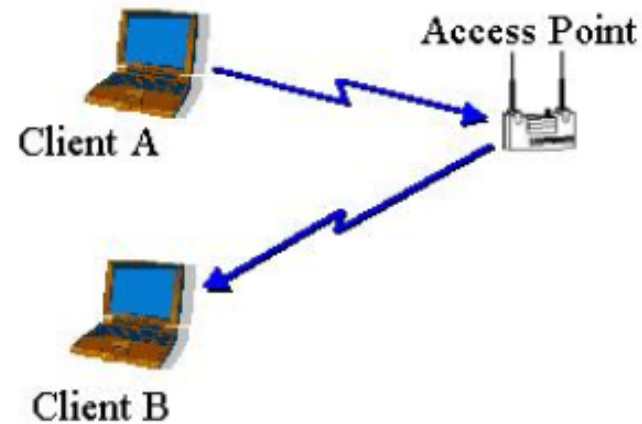
Fluhrer-Mantin-Shamir attack

802.11b Overview

- ◆ Standard for wireless networks (IEEE 1999)
- ◆ Two modes: **infrastructure** and **ad hoc**



IBSS (ad hoc) mode



BSS (infrastructure) mode

Access Point SSID

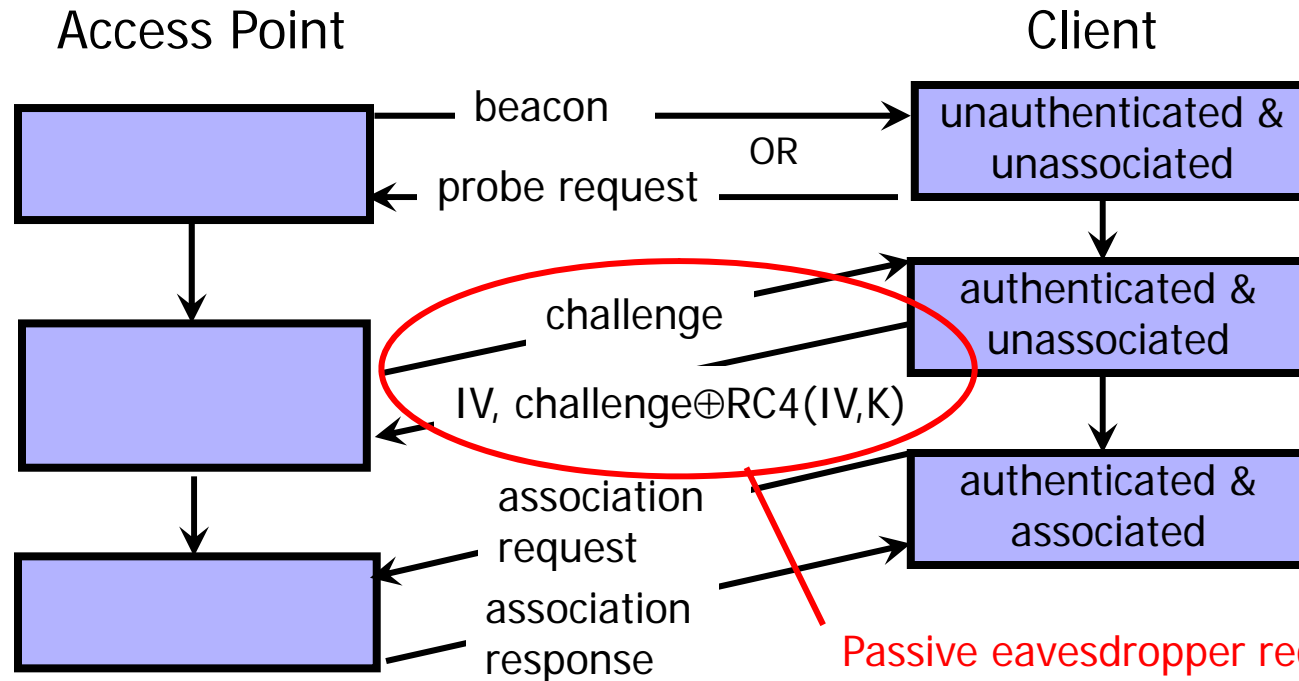
- ◆ Service Set Identifier (SSID) is the “name” of the access point
 - By default, access point broadcasts its SSID in plaintext “beacon frames” every few seconds
- ◆ Default SSIDs are easily guessable
 - Linksys defaults to “linksys”, Cisco to “tsunami”, etc.
 - This gives away the fact that access point is active
- ◆ Access point settings can be changed to prevent it from announcing its presence in beacon frames and from using an easily guessable SSID
 - But then every user must know SSID in advance

WEP: Wired Equivalent Privacy

- ◆ Special-purpose protocol for 802.11b
 - Intended to make wireless as secure as wired network
- ◆ Goals: confidentiality, integrity, authentication
- ◆ Assumes that a secret key is shared between access point and client
- ◆ Uses RC4 stream cipher seeded with 24-bit initialization vector and 40-bit key
 - Terrible design choice for wireless environment
 - In SSL, we will see how RC4 can be used properly

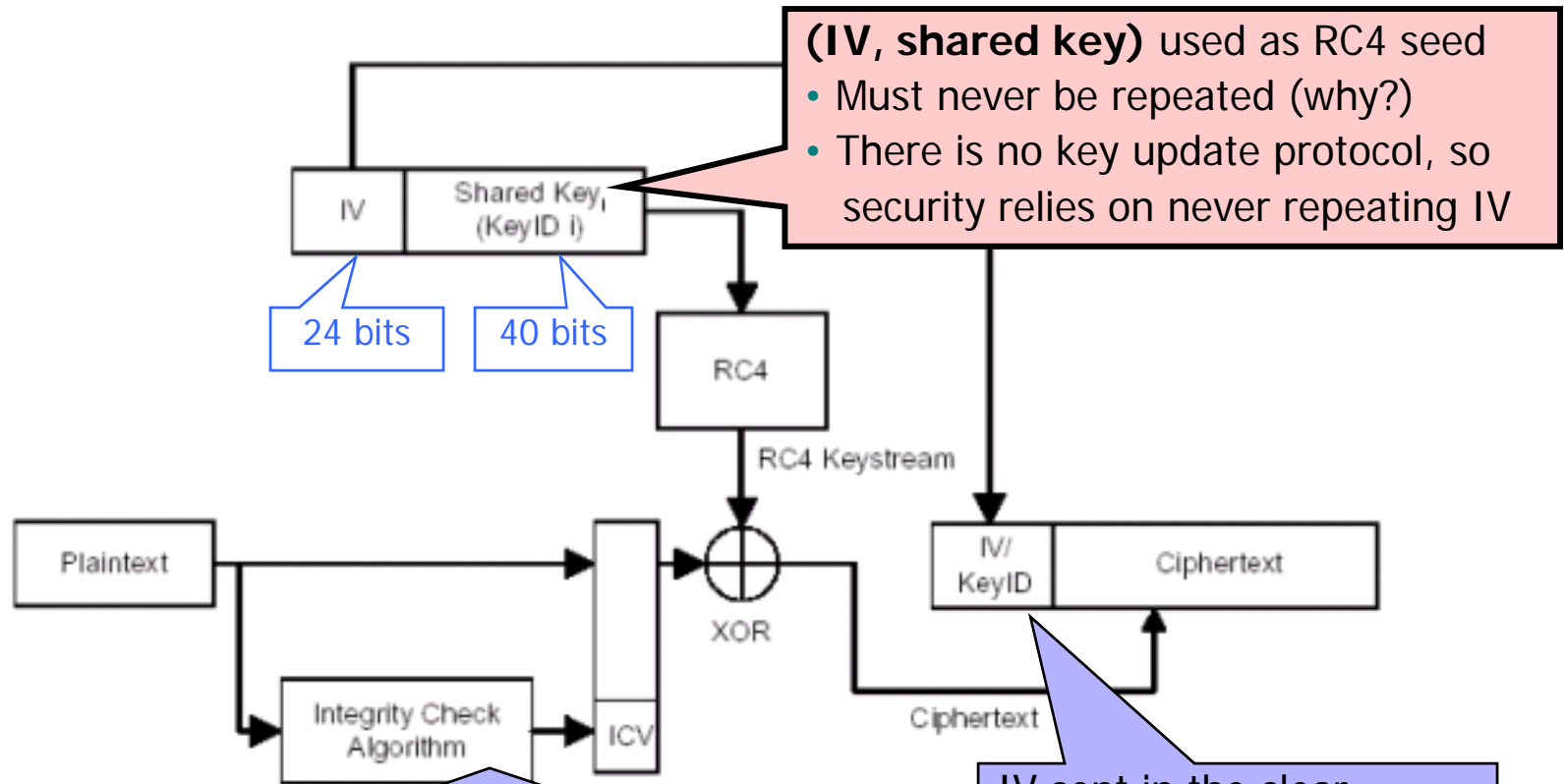
Shared-Key Authentication

Prior to communicating data, access point may require client to authenticate



Passive eavesdropper recovers $RC4(IV, K)$, can respond to any subsequent challenge without knowing K

How WEP Works



(IV, shared key) used as RC4 seed

- Must never be repeated (why?)
- There is no key update protocol, so security relies on never repeating IV

CRC-32 checksum is linear in \oplus :
if attacker flips some plaintext bits, he knows which bits of CRC to flip to produce the same checksum

IV sent in the clear
Worse: changing IV with each packet is optional!

no integrity!

RC4 Is a Bad Choice for Wireless

- ◆ Stream ciphers require synchronization of key streams on both ends of connection
 - This is not suitable when packet losses are common
- ◆ WEP solution: a separate seed for each packet
 - Can decrypt a packet even if a previous packet was lost
- ◆ But number of possible seeds is not large enough!
 - RC4 seed = 24-bit initialization vector + fixed key
 - Assuming 1500-byte packets at 11 Mbps,
 2^{24} possible IVs will be exhausted in about 5 hours
- ◆ Seed reuse is **deadly** for stream ciphers

Recovering Keystream

- ◆ Get access point to encrypt a known plaintext
 - Send spam, access point will encrypt and forward it
 - Get victim to send an email with known content
- ◆ If attacker knows plaintext, it is easy to recover keystream from ciphertext
 - $C \oplus M = (M \oplus \text{RC4}(\text{IV}, \text{key})) \oplus M = \text{RC4}(\text{IV}, \text{key})$
 - Not a problem if this keystream is not re-used
- ◆ Even if attacker doesn't know plaintext, he can exploit regularities (plaintexts are not random)
 - For example, IP packet structure is very regular

Keystream Will Be Re-Used

- ◆ In WEP, repeated IV means repeated keystream
- ◆ Busy network will repeat IVs often
 - Many cards reset IV to 0 when re-booted, then increment by 1 \Rightarrow expect re-use of low-value IVs
 - If IVs are chosen randomly, expect repetition in $O(2^{12})$ due to birthday paradox
- ◆ Recover keystream for each IV, store in a table
 - $(\text{KnownM} \oplus \text{RC4}(\text{IV}, \text{key})) \oplus \text{KnownM} = \text{RC4}(\text{IV}, \text{key})$
- ◆ Wait for IV to repeat, decrypt and enjoy plaintext
 - $(M' \oplus \text{RC4}(\text{IV}, \text{key})) \oplus \text{RC4}(\text{IV}, \text{key}) = M'$

It Gets Worse

- ◆ Misuse of RC4 in WEP is a design flaw with no fix
 - Longer keys do not help!
 - The problem is re-use of IVs, their size is fixed (24 bits)
 - Attacks are passive and very difficult to detect
- ◆ Perfect target for Fluhrer et al. attack on RC4
 - Attack requires known IVs of a special form
 - WEP sends IVs in plaintext
 - Generating IVs as counters or random numbers will produce enough “special” IVs in a matter of hours
- ◆ This results in **key recovery** (not just keystream)
 - Can decrypt even ciphertexts whose IV is unique

Weak Countermeasures

- ◆ Run VPN on top of wireless
 - Treat wireless as you would an insecure wired network
 - VPNs have their own security and performance issues
 - Compromise of one client may compromise entire network
- ◆ Hide SSID of your access point
 - Still, raw packets will reveal SSID (it is not encrypted!)
- ◆ Have each access point maintain a list of network cards addresses that are allowed to connect to it
 - Infeasible for large networks
 - Attacker can sniff a packet from a legitimate card, then re-code (spoof) his card to use a legitimate address

Fixing the Problem

◆ Extensible Authentication Protocol (EAP)

- Developers can choose their own authentication method
 - Passwords (Cisco EAP-LEAP), public-key certificates (Microsoft EAP-TLS), passwords OR certificates (PEAP), etc.

◆ 802.11i standard fixes 802.11b problems

- Patch: TKIP. Still RC4, but encrypts IVs and establishes new shared keys for every 10 KBytes transmitted
 - No keystream re-use, prevents exploitation of RC4 weaknesses
 - Use same network card, only upgrade firmware
- Long-term: AES in CCMP mode, 128-bit keys, 48-bit IVs
 - Block cipher (in special mode) instead of stream cipher
 - Requires new network card hardware