Authenticated Encryption

Case study: TLS

The TLS Record Protocol (TLS 1.2)

$k_{b \to s}$, $k_{s \to b}$

$k_{b \to s}$, $k_{s \to b}$

Unidirectional keys: $k_{b \to s}$ and $k_{s \to b}$

Stateful encryption:

- Each side maintains two 64-bit counters: $ctr_{b \to s}$, $ctr_{s \to b}$
- Init. to 0 when session started. ctr++ for every record.
- Purpose: replay defense

TLS record: encryption (CBC AES-128, HMAC-SHA1)

$k_{b \to s} = (k_{mac}, k_{enc})$

Browser side **enc($k_{b \to s}$, data, $ctr_{b \to s}$) :**

not transmitted in packet

step 1: tag ← S(k_{mac}, [$++ctr_{b \to s}$ || header || data])

step 2: pad [header || data || tag] to AES block size

step 3: CBC encrypt with k_{enc} and new random IV

step 4: prepend header

TLS record: decryption (CBC AES-128, HMAC-SHA1)

Server side **dec($k_{b\to s}$, record, $ctr_{b\to s}$)** :

step 1: CBC decrypt record using k_{enc}

step 2: check pad format: send bad_record_mac if invalid

step 3: check tag on [++$ctr_{b\to s}$ || header || data]

send bad_record_mac if invalid

Provides authenticated encryption

(provided no other info. is leaked during decryption)

Bugs in older versions (prior to TLS 1.1)

IV for CBC is predictable: (chained IV)

IV for next record is last ciphertext block of current record.

Not CPA secure. (a practical exploit: BEAST attack)

Padding oracle: during decryption

if pad is invalid send decryption failed alert

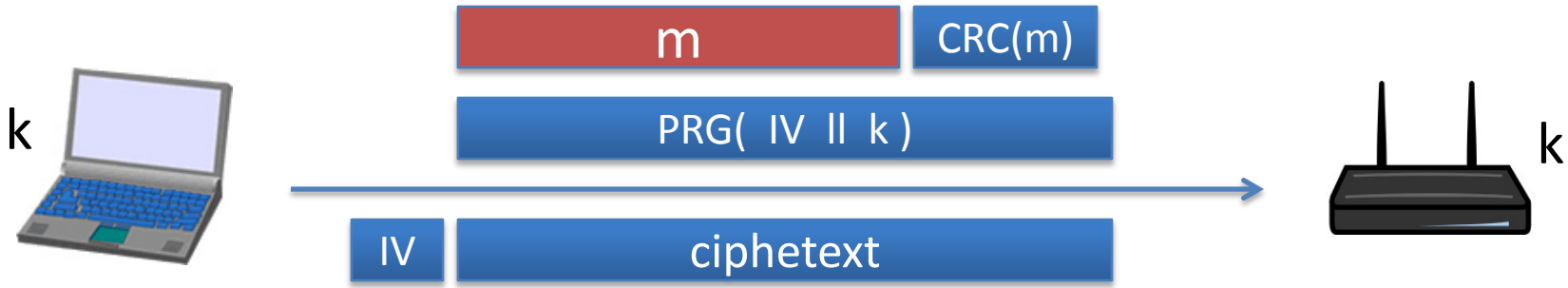
if mac is invalid send bad_record_mac alert

⇒ attacker learns info. about plaintext (attack in next segment)

Lesson: when decryption fails, do not explain why

802.11b WEP: how not to do it

802.11b WEP:



Previously discussed problems:

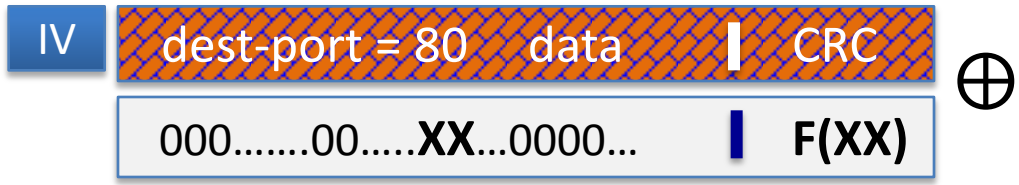
two time pad and related PRG seeds

Active attacks

Fact: CRC is linear, i.e. $\forall m,p$: **$CRC(m \oplus p) = CRC(m) \oplus F(p)$**

WEP ciphertext:

attacker:



XX = 25⊕80

Upon decryption: CRC is valid, but ciphertext is changed !!

End of Segment