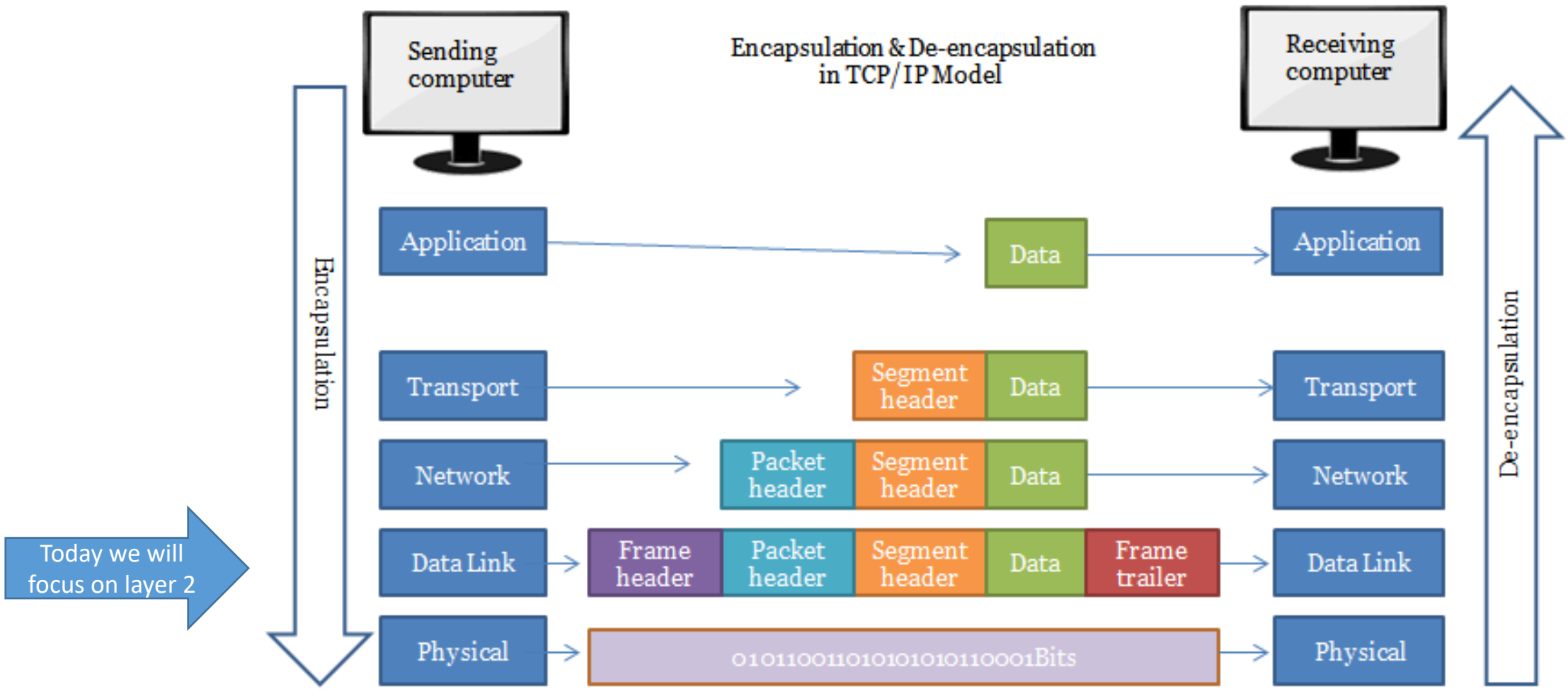
# Ασφάλεια στα ασύρματα δίκτυα WLAN

Χριστόφορος Νταντογιαν

dadoyan@ionio.gr


Encapsulation & De-encapsulation in TCP/IP Model
Sending computer
Receiving computer
Encapsulation
De-encapsulation
Application
Data
Application
Transport
Segment header
Data
Transport
Network
Packet header
Segment header
Data
Network
Data Link
Frame header
Packet header
Segment header
Data
Frame trailer
Data Link
Physical
01011001101010101010110001Bits
Physical
Today we will focus on layer 2

# Overview

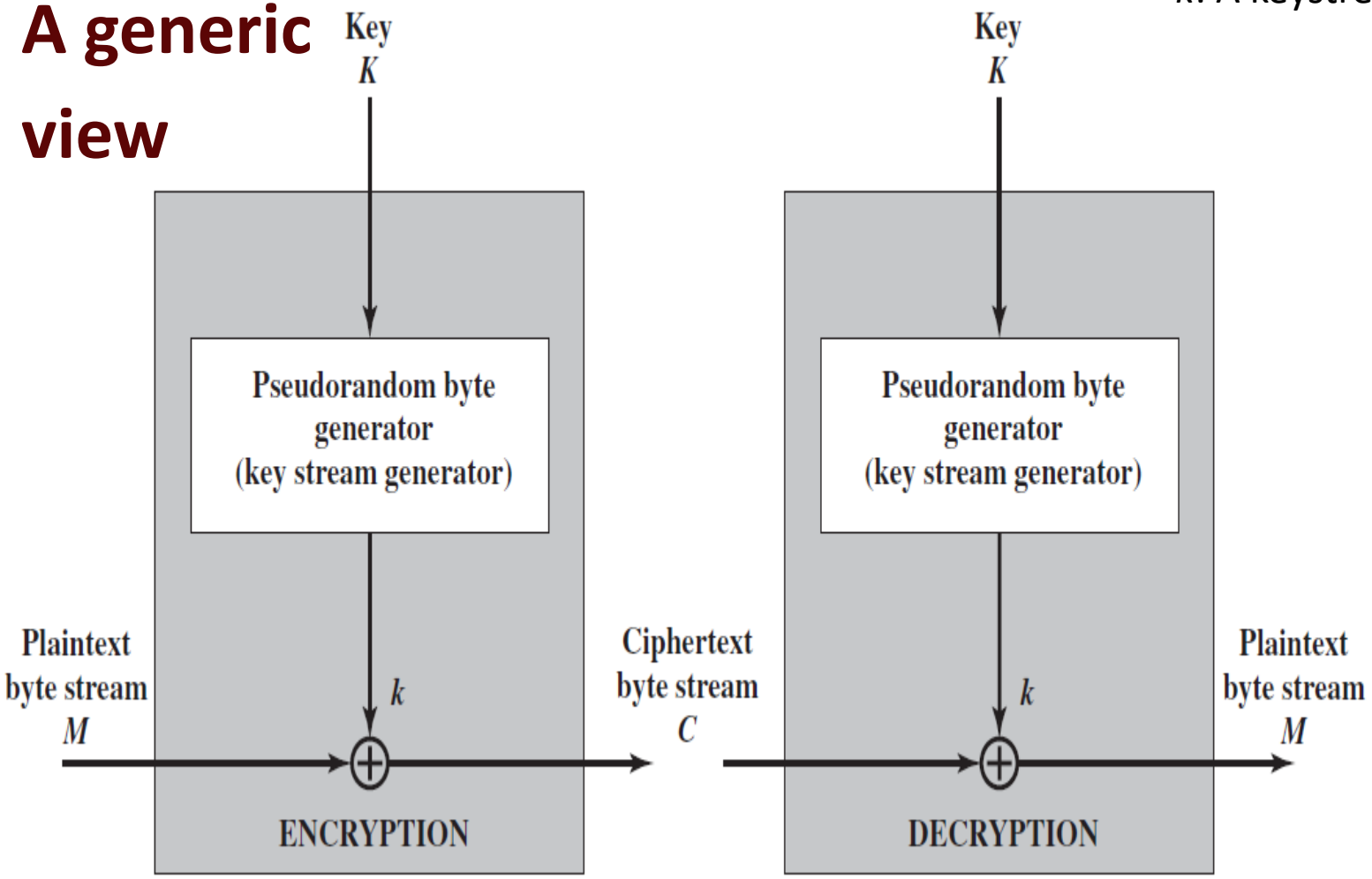
- The IEEE 802.11 WEP protocol was designed in 1999 to provide authentication and data encryption between a host and a wireless access point
- **Out of band** exchange of symmetric key (in reality a password)
- WEP uses 40 bits key and WEP2 uses 104 bits.
- Encryption uses stream cipher RC4 → RC4 is extremely fast when implemented in software.
- However, performance come at the price of lowered security → several weaknesses have been identified in RC4

# The RC4 Algorithm

*K*: The seed usually 256 bytes

*k*: A keystream generated for *M*

## A generic view



# Example of RC4 encryption

| Key | Keystream | Plaintext | Ciphertext |
|---|---|---|---|
| Key | EB9F7781B734CA72A719... | Plaintext | BBF316E8D940AF0AD3 |
| Wiki | 6044DB6D41B7... | pedia | 1021BF0420 |
| Secret | 04D46B053CA87B59... | Attack at dawn | 45A01F645FC35B383552544B9BF5 |

# RC4 and keystream reuse problem

- Assume C1= k ⊕ P1
- Assume C2= k ⊕ P2

- C1 ⊕ C2 = (k ⊕ P1) ⊕ (k ⊕ P2) = (k ⊕ k) ⊕ P1 ⊕ P2 = 0 ⊕ P1 ⊕ P2 = P1 ⊕ P2

- Conclusion: Never use the same key(stream) for encryption

⊕ = XOR

| Input | | Output |
|---|---|---|
| A | B | A xor B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Real life

- RC4 does **not take a separate nonce alongside the key**. This means that if a single long-term key is to be used to securely encrypt multiple streams, a security protocol must specify how to combine the nonce and the long-term key to generate the stream key for RC4.

- One approach to addressing this is **to generate a "fresh" RC4 key by hashing a long-term key with a nonce**. However, many applications that use RC4 simply concatenate key and nonce;

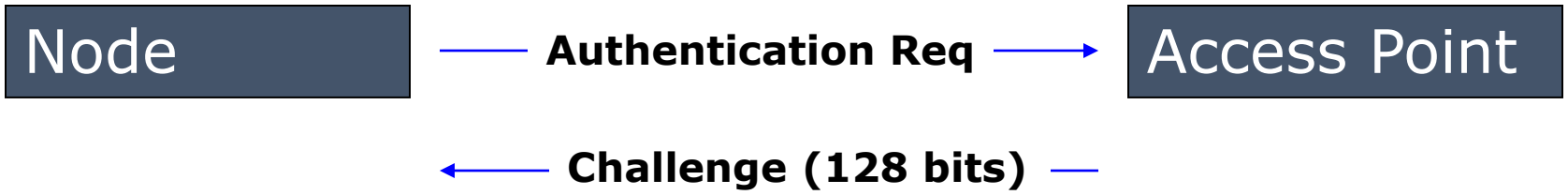
# Authentication in WEP



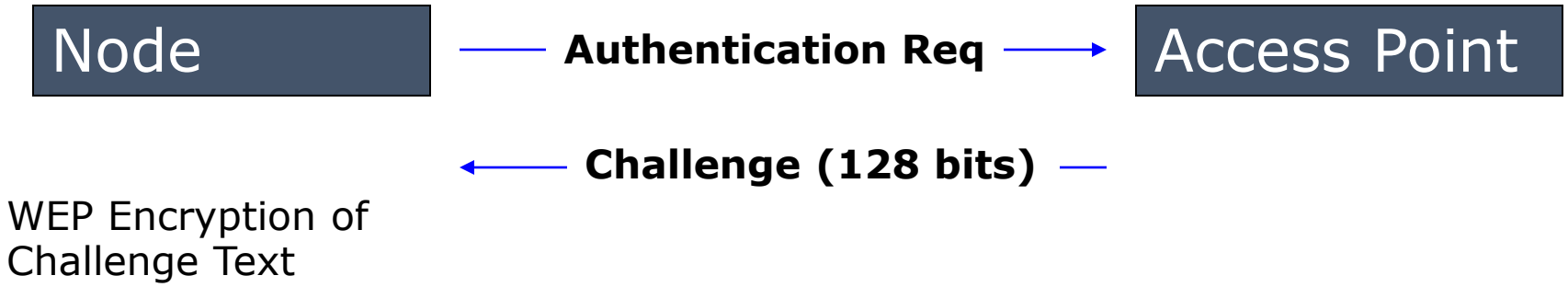
# Authentication in WEP



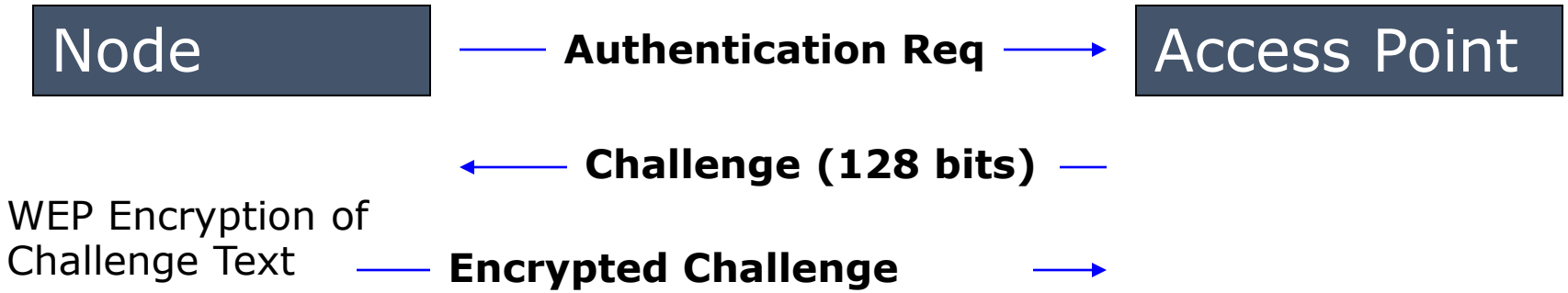
# Shared Key Authentication in WEP



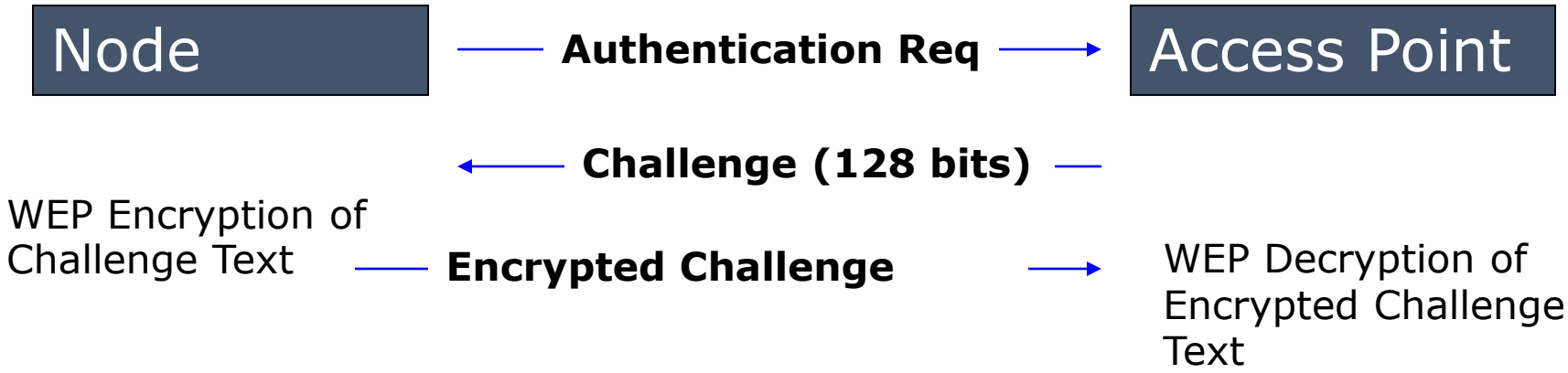
# Authentication in WEP



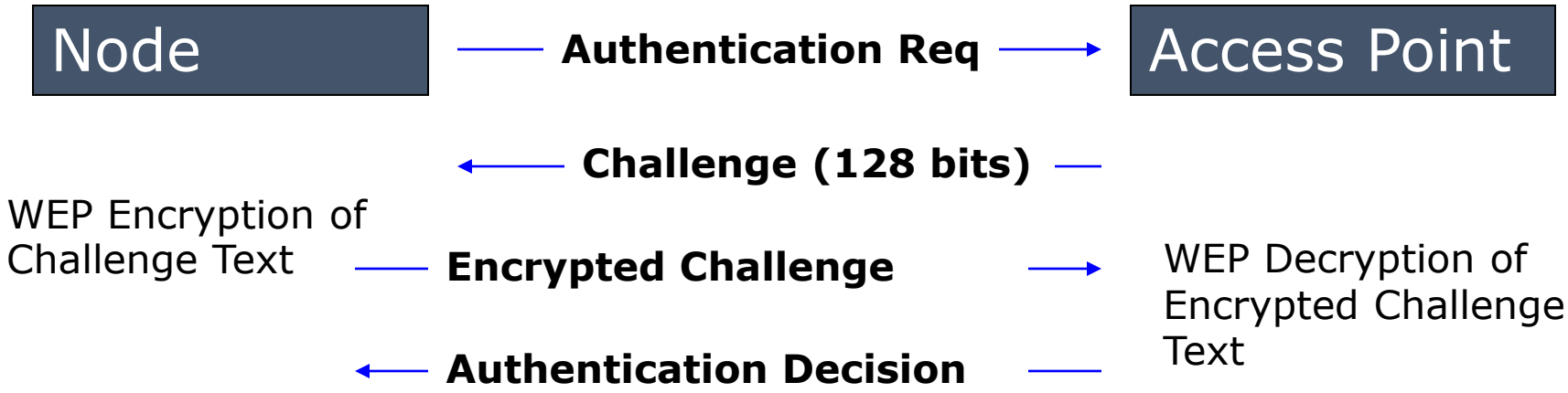
# Authentication in WEP



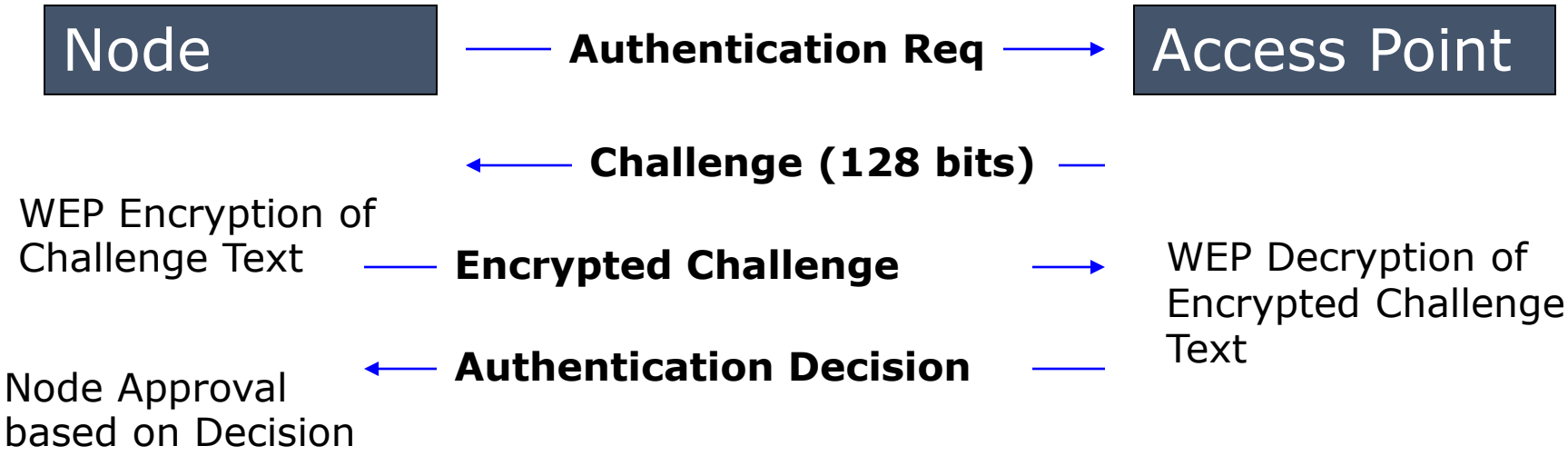
# Authentication in WEP



# Authentication in WEP



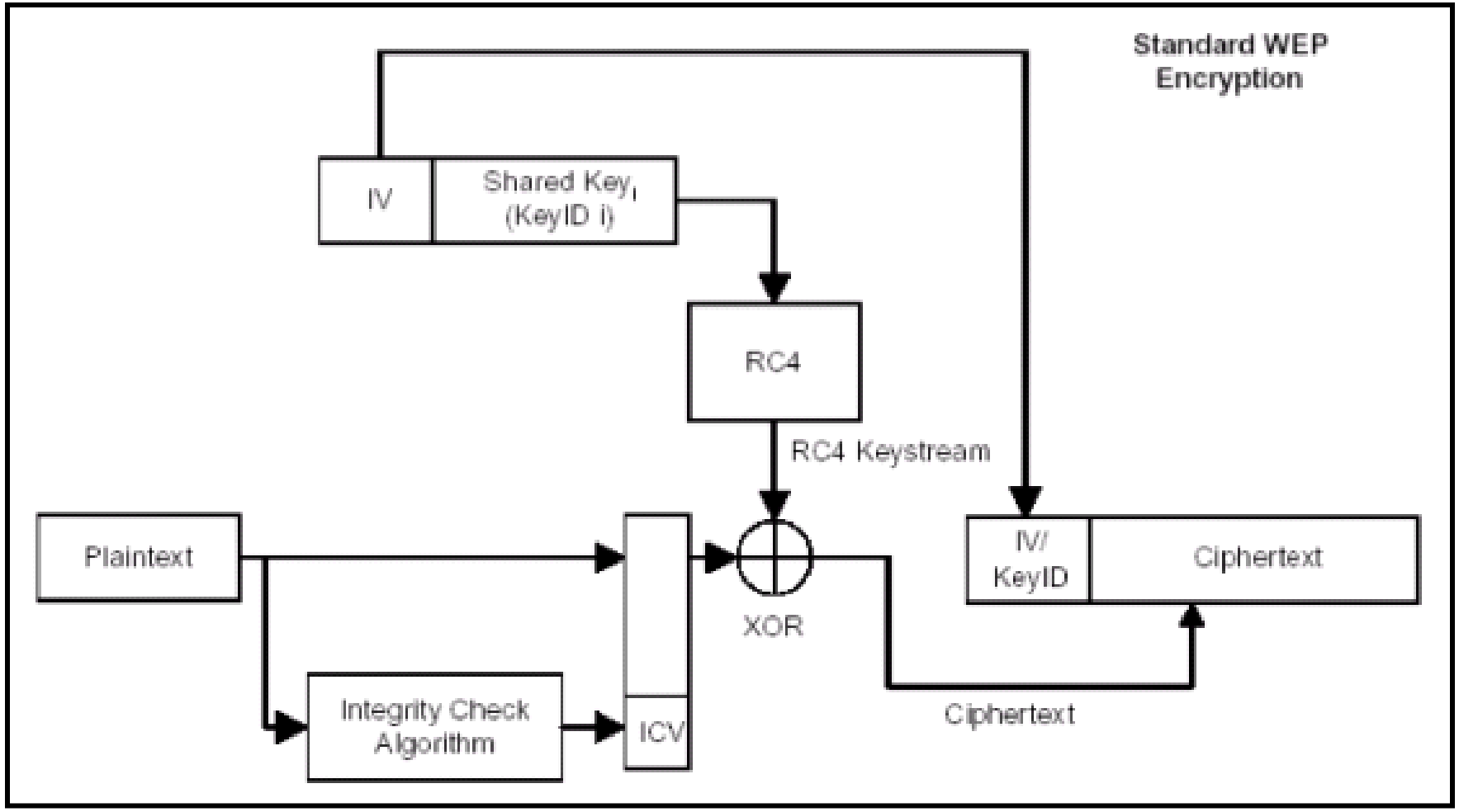
# Authentication in WEP



# RC4 Encryption in WEP

- To prevent these attacks, WEP uses a per-packet IV to vary the keystream generation process for each frame of data transmitted.
- WEP generates the keystream as a function of both the secret key (which is the same for all packets) and a public initialization vector IV (which varies for each packet); this way, each packet receives a different keystream.
- The IV is included in the unencrypted portion of the transmission so that the receiver can know what IV to use when deriving the keystream for decryption.
- The IV is therefore available to attackers as well , but the secret key remains unknown and maintains the security of the keystreamd reusing the same key in RC4,

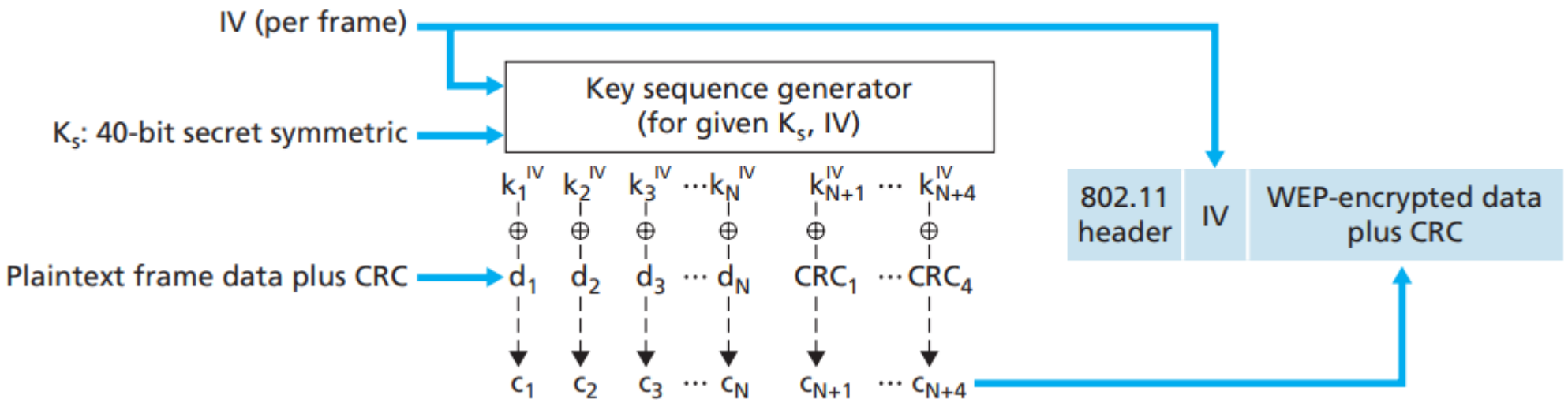
# RC4 Implemented in WEP



# WEP wrong integrity solution

- The integrity check algorithm in WEP was CRC32 → 4 bytes
- A cyclic redundancy check (CRC) is an error-detecting code
- Its not considered a MAC!!!
- It is very easy to generate two messages with the same CRC32

# RC4 Implemented in WEP



# Attacking WEP

- The WEP standard recommends (but does not require) that the IV be changed after every packet transmission…
- **A security design flaw:** The IV field used by WEP is only 24 bits wide, nearly guaranteeing that the same IV will be reused for multiple messages.
  - A busy access point sending 1500 byte packets and achieving an average 5Mbps bandwidth (the full transmission rate is 11Mbps) will exhaust the available space in less than half a day.
- Implementation details can make keystream reuse occur even more frequently. An implementation that uses a random 24-bit IV for each packet, will be expected to incur collisions on average (i.e., 50% probability) after transmitting just $2^{12} = 4096$ packets, due to birthday paradox!!

# Attacking WEP

- Thus, an attacker has access to P1 $\oplus$ P2…
- Since IV's are public, duplicate IV's can be easily detected by the attacker → Easy to find keystream reuse occurence
- Many fields of the message are well known… (e.g., IP).
- A chosen plaintext attack on P1 can reveal P2
  - IP headers, email etc.

# FMS (Fluhrer, Mantin and Shamir) attack

- But a more disturbing attack in RC4 was discovered in 2001 named FMS…
- Preconditions for the FMS attack
  1. Encryption based on **different keys that have a mathematical relationship connecting these keys which is known** to the attacker (this is known as related key vulnerability)
  2. The attacker must be able to read the first byte of the keystream (output of the RC4)
  3. The attacker must be able to read the prefix (at least 3 bytes) of the key

  - Wait! 1 and 3 always are true for WEP → The attacker knows that the second part of the key (40 or 104 bits) is always the same (related keys) → WEP concatenates IV with the key → what about precondition 2? You can apply a known plaintext attack since the first byte of the WEP packet is always 0xAA → 2 also is always true!

# FMS attack

- The FMS attack is based on the discovery that certain IVs create **weak keys** → an attacker knowing **the first byte of the keystream** and the first *m* bytes of the (weak) key can derive the (*m* + 1)th byte of the key with 5% probability
- Attack scenario: the attacker collects packets searching for the use of weak IVs → the attacker tries byte by byte to reveal the whole key using statistics
- Requires a large number of packets (6.000.000)
- Optimized attacks further discovered can reveal the key in seconds

# Countermeasures for the FMS attack

1. Drop the first bytes (e.g., 256) → Costly → However there are some implementations named RC4-Drop[n]

2. Do not use concatenation for IVs

3. Do not use a different key for each packet

SSL uses RC4 in a way that is not vulnerable to FMS attack cause it avoids 2 and 3!

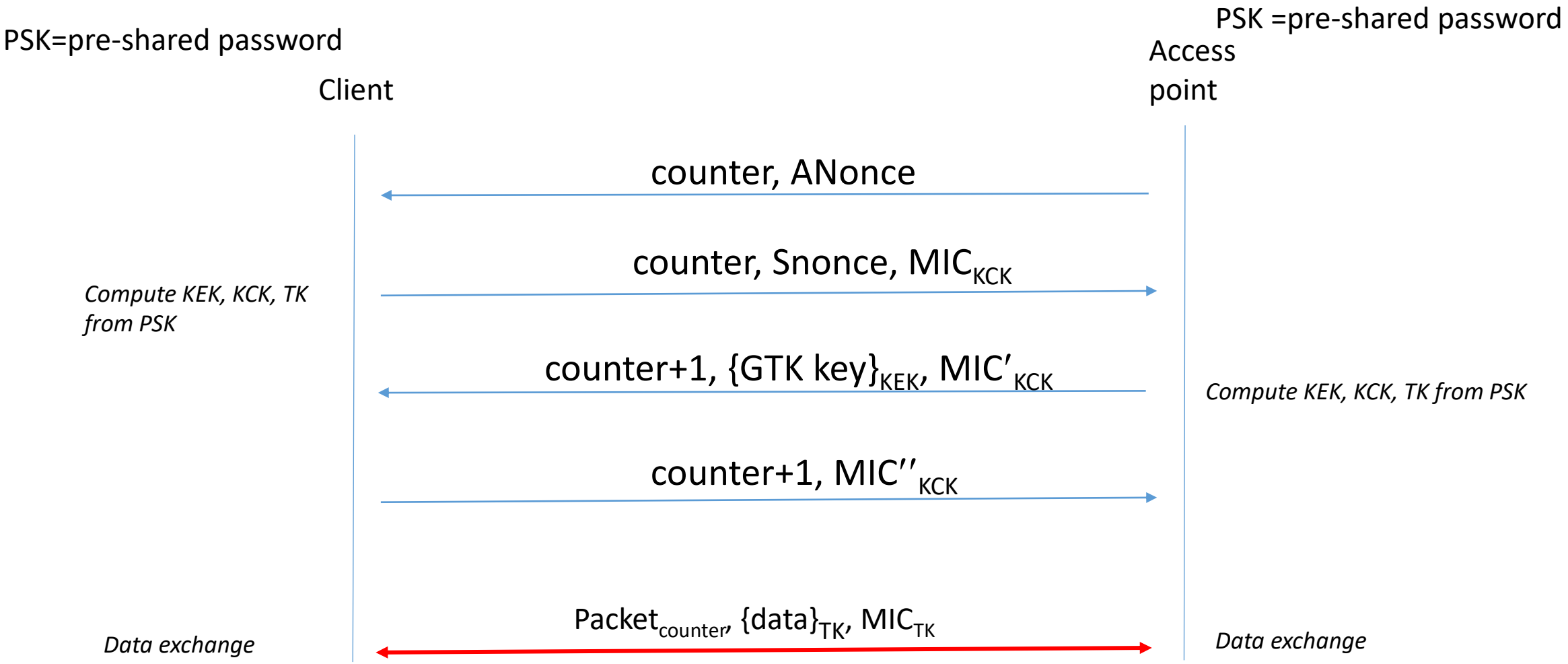
# WPA and WPA2

- IEEE 802.11i standard
- WPA/WPA2 replaces WEP
- WPA is a **software-based** update on WEP that still uses RC4 for encryption
  - Short term solution until new access points supporting WPA2 reach the market.
- WPA2 uses AES for encryption → AES cannot be used on Access Points / ADSL modems that were designed for WEP cause their hardware cannot support it, as AES is not as lightweight as RC4, hence requires new hardware in Aps
- WPA/WPA2 use 802.1X for authentication and key agreement

# Terminology

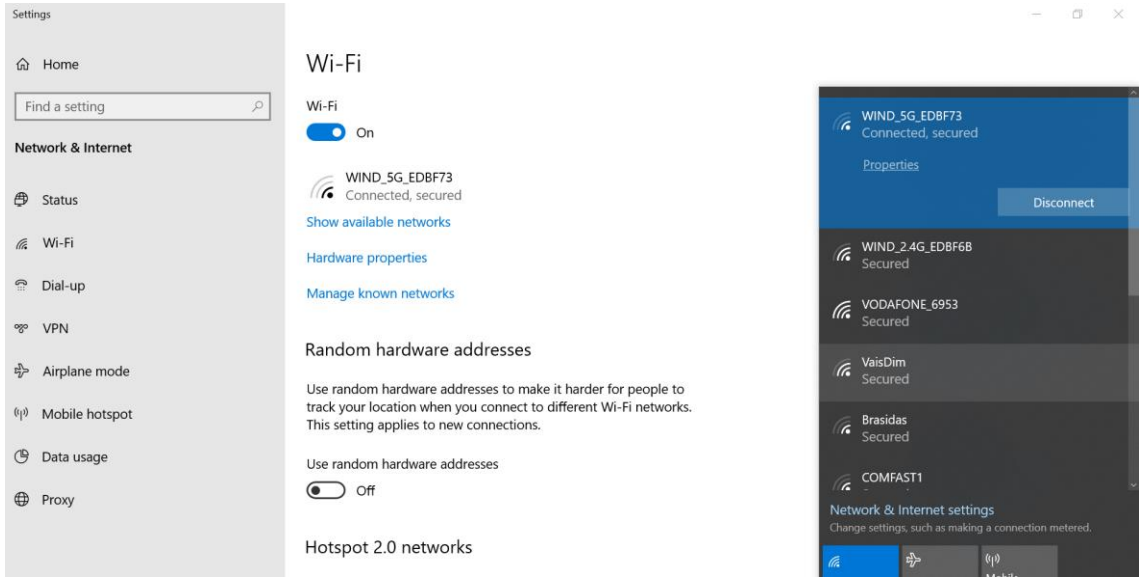
- PSK = Preshared key (or password)
- PMK = pairwise master key
- MIC = Message Integrity Code (known as MAC)
- KDF = Key derivation Function
- TK = Temporal Key for encryption and MIC for data (**session key**)
- KEK=Key encryption key
- KCK = Key confirmation key

# Authentication using WPA2



# Where is Wi-Fi encryption performed?

- The operating system runs the handshake 802.1X and generates the keys



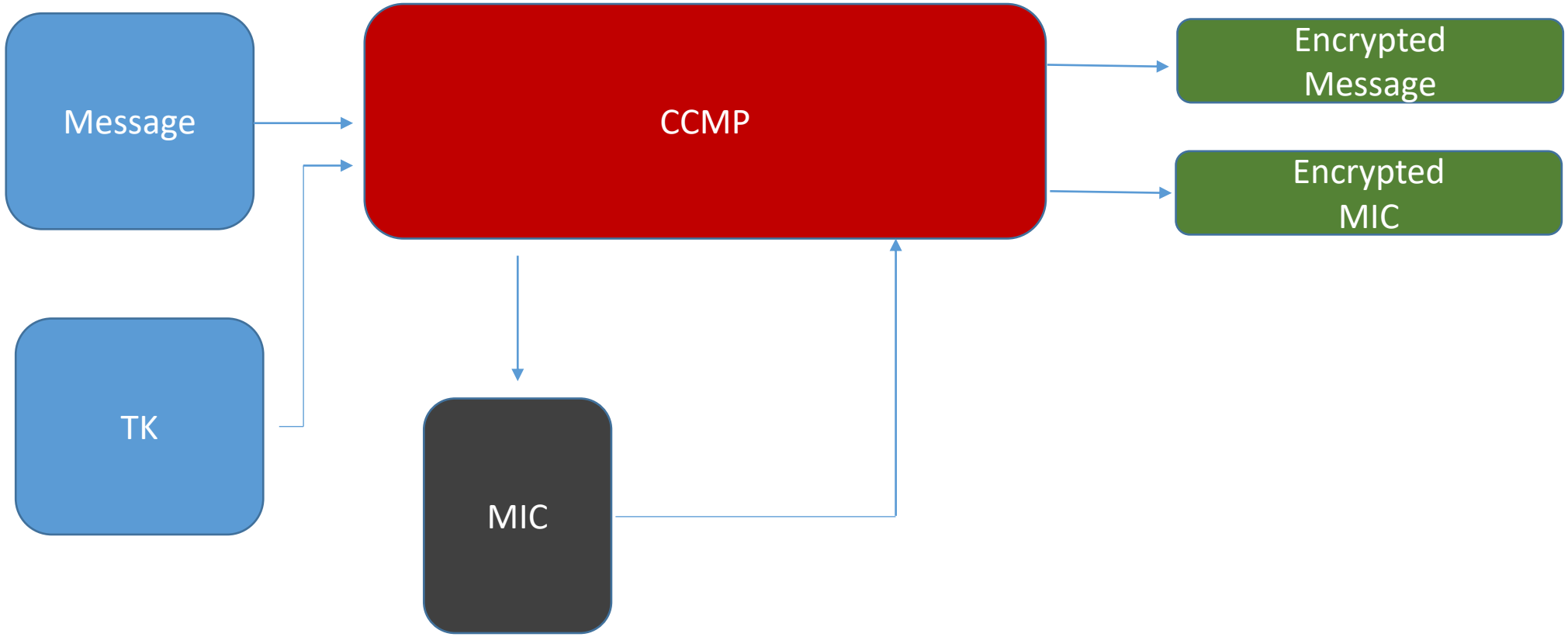

- The wireless NIC installs the TK key and encrypts with WPA/WPA2



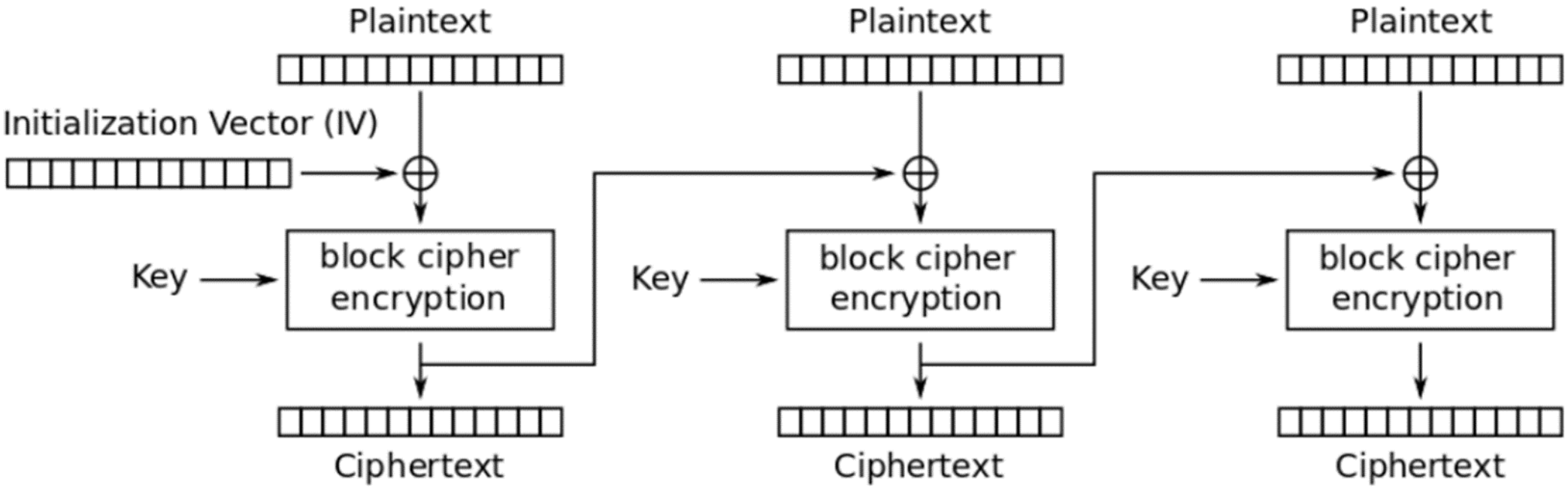
# WPA2

- WPA2 uses AES using the CCMP mode
- CCMP = AES Counter Mode for encryption (confidentiality)+CBC-MAC for MIC (integrity)
- AES key length of 128 bits
- AES in CCMP uses the same key for **both message encryption and message integrity/authentication**.

# 2 to 1 in CCMP

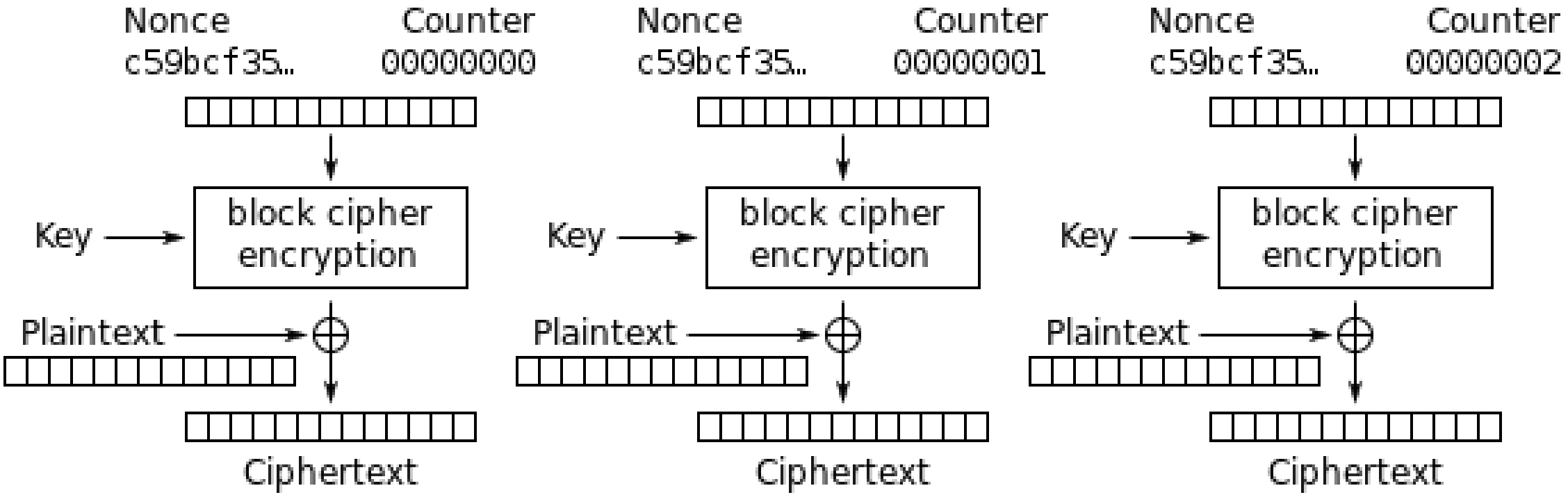


# AES CBC mode




Cipher Block Chaining (CBC) mode encryption

# AES Counter Mode




Counter (CTR) mode encryption

# Implementing MAC (or MIC)

- There are two ways to implement a MAC...

1. HMAC (uses hash functions)
2. CBC-MAC (uses encryption!)

# MAC generation using a HMAC

- HMAC is a generic construction that build a MAC **using a hash function and a secret key**




- HMAC-MD5, HMAC-SHA1, HMAC-SHA256
- What about the key size? The key can be any length. However, the recommended size is **64 bytes**

# Hash Message Authentication Codes (HMAC)

MAC key $K$, plaintext $P$, ciphertext $C=E(P)$.

$MAC=H(K \mid\mid P)$? $MAC=H(P \mid\mid K)$?

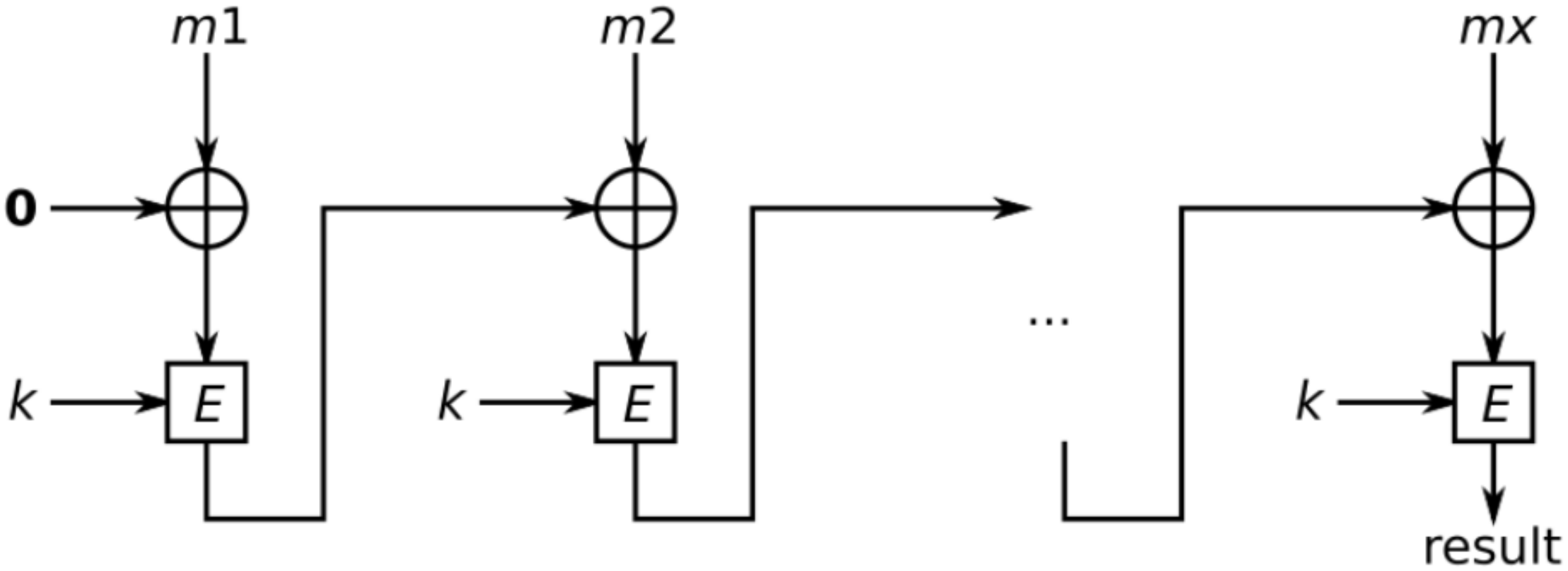
$MAC=H(K \mid\mid C)$? $MAC=H(C \mid\mid K)$?

There are weaknesses with *all* of the above (**length extension attacks**).

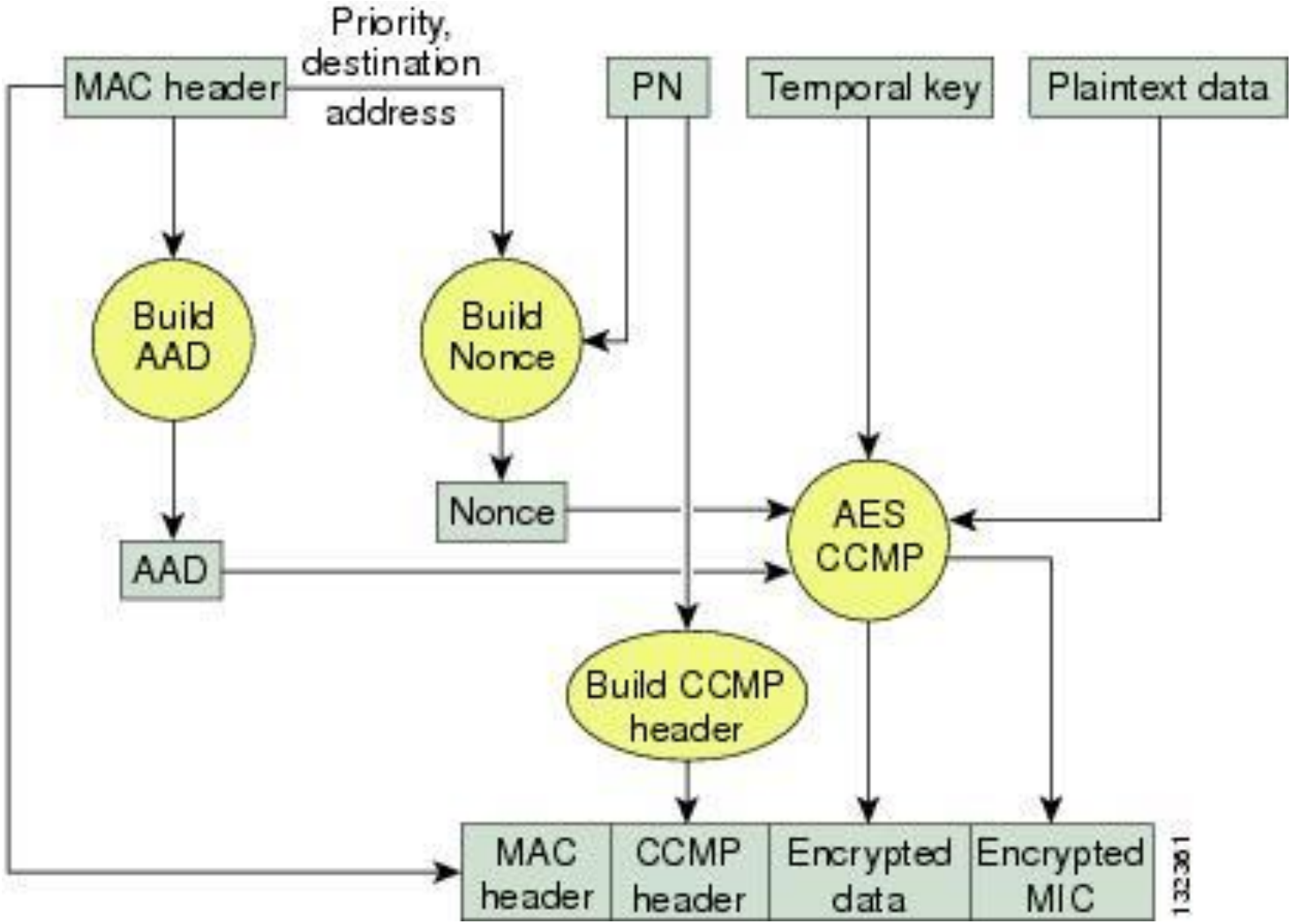
$HMAC = H(K \mid\mid H(K \mid\mid P))$

# CBC-MAC

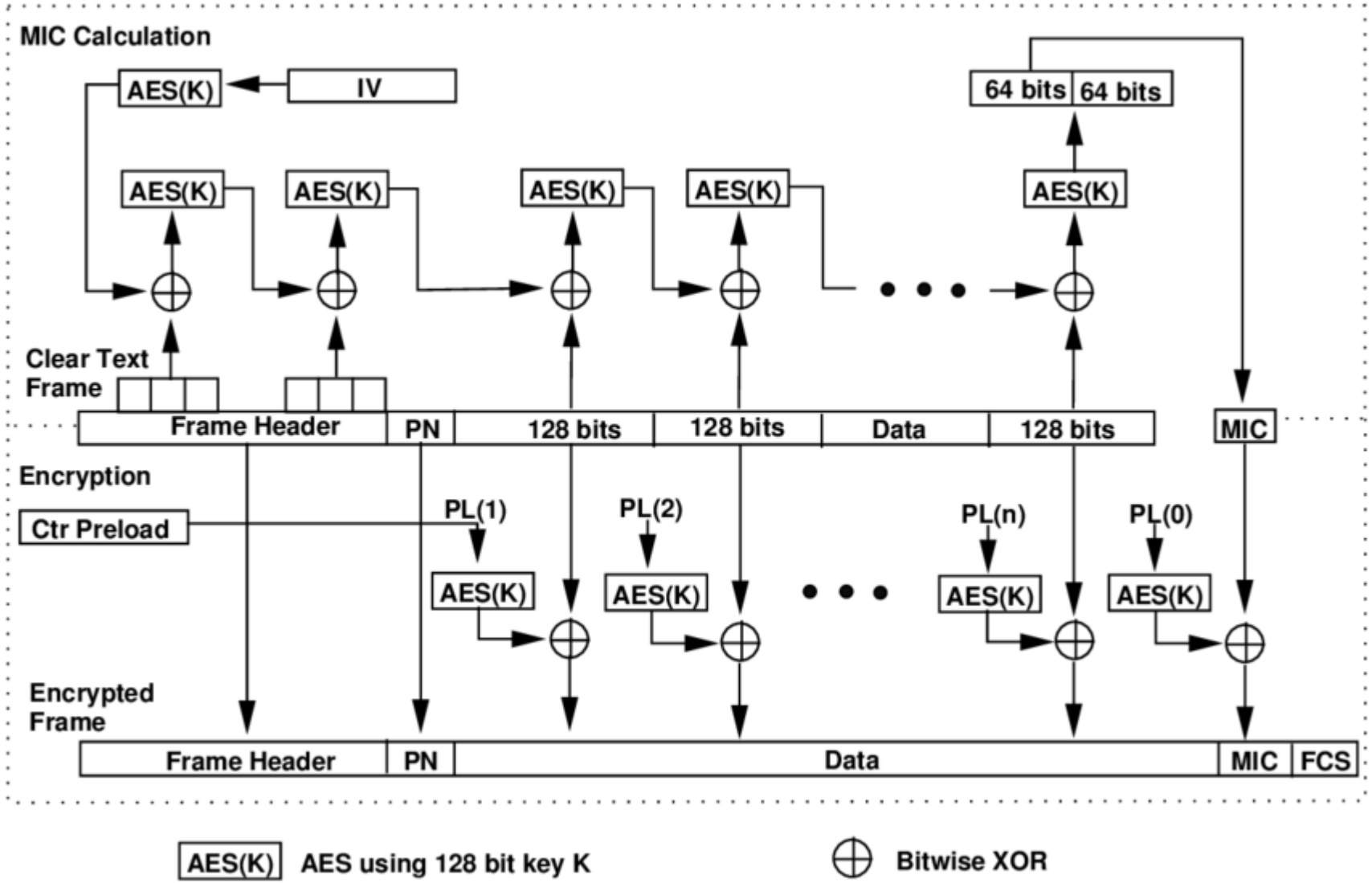
Question: How to construct a MAC using encryption?



# CCMP


Priority, destination address
MAC header
PN
Temporal key
Plaintext data
Build AAD
Build Nonce
Nonce
AAD
AES CCMP
Build CCMP header
MAC header
CCMP header
Encrypted data
Encrypted MIC
132361

# CCMP !!!!


MIC Calculation
AES(K)
IV
64 bits
64 bits
AES(K)
AES(K)
AES(K)
AES(K)
AES(K)
Clear Text
Frame
Frame Header
PN
128 bits
128 bits
Data
128 bits
MIC
Encryption
Ctr Preload
PL(1)
PL(2)
PL(n)
PL(0)
AES(K)
AES(K)
AES(K)
AES(K)
Encrypted
Frame
Frame Header
PN
Data
MIC
FCS
AES(K)
AES using 128 bit key K
Bitwise XOR

# Crypto Hygiene

Do I really need to use different keys for encryption and integrity?

It's always a good idea to use separate keys for separate functions, **but the keys can be derived from the same master**.

$K_1$=H("Key1",K)  $K_2$=H("Key2",K)

# KDF to generate keys

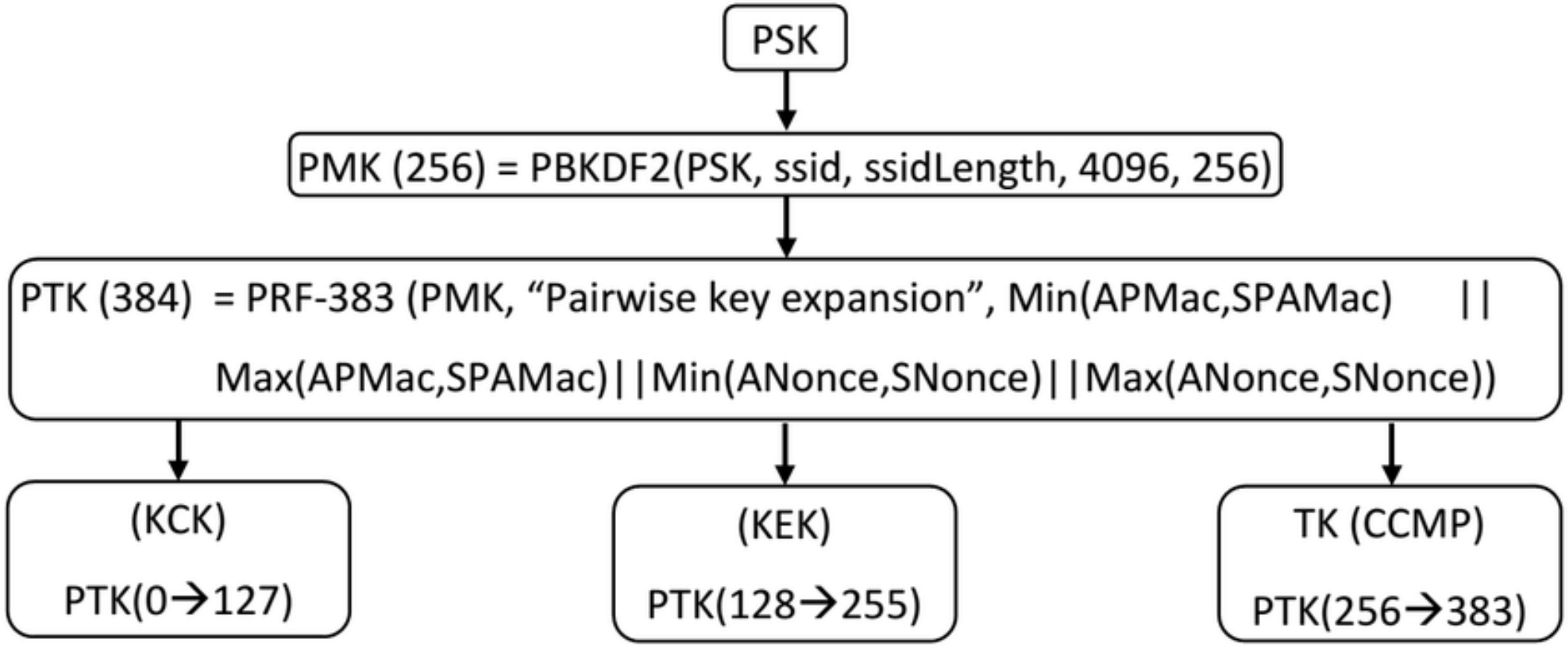
- How do we calculate 3 different keys from one key which is PSK?

- It is the same as TLS → Use of a Key Derivation Function (KDF)

- All kdfs follow two steps:

1. Generate entropy from the initial key (PSK in WPA2)
2. Expand the key to the required size

# Key derivation function (KDF) for WPA2



# PTK is not used directly but is broken down to a set of keys




- Key Confirmation Key (KCK) → Used for integrity protection of handshake
- Key Encryption Key (KEK) → Used for encrypting group key (GTK) during handshake
- Temporal Key (TK) → Used for encryption of data packets

# MIC of exchanged data is different than MIC of handshake

- MIC = HMAC-SHA1 (KCK, handshake message)

- Do not confuse it with MIC from CCMP which is calculated from CCMP!!!

# Breaking WPA2



- No way to break it directly !
- Only brute force the PMK to calculate KCK and compute the same MIC in a handshake.
- Trigger a de-authenticate message to force handshake
- Capture it with a Wi-Fi card in **monitor** mode (Kali Linux)

# Comparison table

| | WEP | 802.11i Methods | |
|---|---|---|---|
| | | WPA | WPA2 |
| Security Protocols | RC4 | TKIP | CCMP |
| Cipher | RC4 | RC4 | AES |
| Key Length | 40 or 104 bits | 128 bits encryption 64 bits authentication | 128 bits |
| Key Life | 24 bit IV | 48 bit IV | |
| Key Generation | Concatenation | Two phase mixing function | Not needed |
| Data Integrity | CRC-32 | Michael | CBC-MAC |
| Header Integrity | None | Michael | CBC-MAC |
| Replay Protection | None | Packet Number | |
| Key Management | None | EAS-based | |
| Authentication | Open or Shared key | 802.11x or Pre-Shared Key (PSK) | |