Communications Security

Communications Security

Communications security (COMSEC) means preventing unauthorized access to communications infrastructure and communicated messages, while still providing the communications service between intended parties

Once upon a time this subject was only of interest for military communications
but it is now crucial to businesses and residential customers as well

In the early days of the Internet, the model was *trust everyone*

In the 1990s the model changed to *soft on the inside, hard on the outside*
which led to development of access policies, firewalls, etc.

Today the only reasonable policy is

- trust no-one
- constantly monitor everything
- pro-actively search for vulnerabilities

Threats

Before adopting a security measure, one must understand the *threat*

Potential threats include :

- denial of service (DoS) to all or some parties
- theft of service
- access to confidential information
- modification of information
- control of restricted resources
- physical damage to resources

Security experts build *threat models*

- what are the risks ?
- who are the potential attackers
- what at vulnerabilities and attack vectors

before putting *countermeasures* into effect

Countermeasures

What can we do to combat threats ?

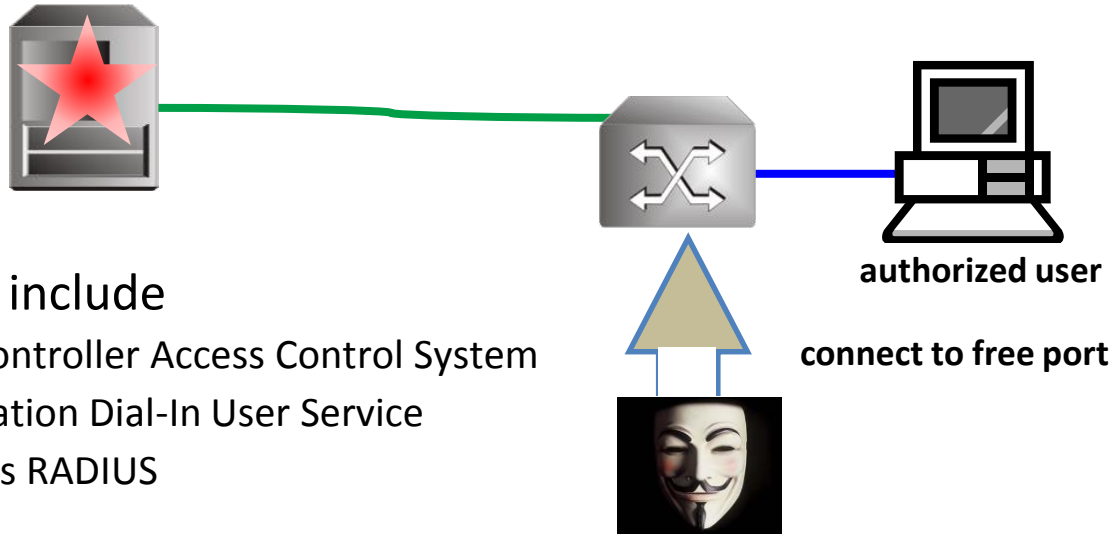
- Physical security – preventing access to communications devices and links
- Emission security – preventing interception and jamming
- Authorization – preventing unauthorized access to resources
- Source authentication – confirming the source of a message
- Integrity – preventing tampering with messages
- Confidentiality – preventing eavesdropping
- DoS blocking – preventing Denial of Service
- Topology hiding – thwarting traffic analysis
- Anti-hacking – preventing injection of computer malware
- Privacy – protecting user's personal data from mining and directed collection

We are going to deal only with these

Authorization

The first concern is who is given access to resources

AAA (user) Authentication, Authorization, and Accounting
means any mechanism for controlling access to resources



Well-known AAA systems include

- TACACS : Terminal Access Controller Access Control System
- RADIUS : Remote Authentication Dial-In User Service
- DIAMETER : twice as good as RADIUS

Such systems require

- a *supplicant* (party requesting service)
- to prove his identity (e.g., via a password)
- to an *authenticator* (which is often hierarchical)

Passwords

A password is a string provided by the supplicant and verified by the server
The server never stores passwords *in the open* (threat – theft of the password file)
rather stores a crypto-hash (1-way function) of the passwords

Threats to use of passwords:

- Guessing of password
- Theft of password
- Brute-force (exhaustive search) discovery

Countermeasures

- Using complex passwords (*at least one capital, one numeral, one punctuation*)
- Using long passwords (*thisisareasonablygoodpassword,* x!A0 isn't)
- Place delay after wrong guess
- CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart)
- Three-factor authentication
  - something you are
  - something you have
  - something you know

password

Crypto-hashes

In computer science a *hash* is a function that

1. maps strings (vectors) of arbitrary length to strings (vectors) of fixed length
2. ensures that small change in input map to large changes in output

The output length may be much smaller than the input length,
meaning that many inputs map to the same output (collisions!)

A crypto-hash has a further property of being a 1-way function

3. calculating the hash function is computationally easy
   finding an input that produces a given output is computationally hard

You may already know about check-sum and CRC hashes

these are good for random error correction, but are not crypto-hashes

Well-known crypto-hashes include :

- MD5 (no longer considered secure)
- Secure Hash Algorithm SHA1 (widely used, no longer considered secure)
- Secure Hash Algorithm SHA2 (actually 6 hashes SHA-224,256,384,512,512/224,512/256)
- Secure Hash Algorithm SHA3 (new)

CHAP

How can we use a hash to store passwords ?

- when registering the password is crypto-hashed and its hash value stored
- the password file/database only contains hash values
  from which passwords can not be recovered
- upon logging on the password is hashed and the hashes compared

This is great for logging on to a local computer
for communications it would require sending the password in the clear !

How about computing the hash before transmission and transmitting it ?
No improvement, since the eavesdropper would observe the correct hash and send it (replay attack) !

The solution is a **C**hallenge **H**andshake **A**uthentication **P**rotocol

- *authenticator* sends a *challenge* message to the *supplicant*
- supplicant responds with crypto-hash of challenge+password
- authenticator compares response with expected hash value
- if match supplicant admitted, else rejected

EAP

Extensible **A**uthentication **P**rotocol is a authentication *framework*
and runs over links layers (PPP, Ethernet, WiFi)
without needing IP

Originally developed for PPP (extends PPP's original CHAP)

Dozens of specific methods (EAP-PSK, EAP-MD5, EAP-TLS, EAP-IKEv2, EAP-SIM, …)

Used as a link layer authentication for WiFi (WPA, WPA2), IEEE 802.1X, …

EAP provides 1-sided authentication (supplicant by authenticator)
it must be run in both directions for mutual authentication

EAP operation

- optionally authenticator sends *Identity Request* to supplicant
- optionally supplicant sends *Identity response*
- authenticator sends *EAP request* with *authentication method* and data (challenge)
- supplicant sends *EAP response* (or NAK if doesn't support *authentication method*)
- authenticator sends *success* or *failure*

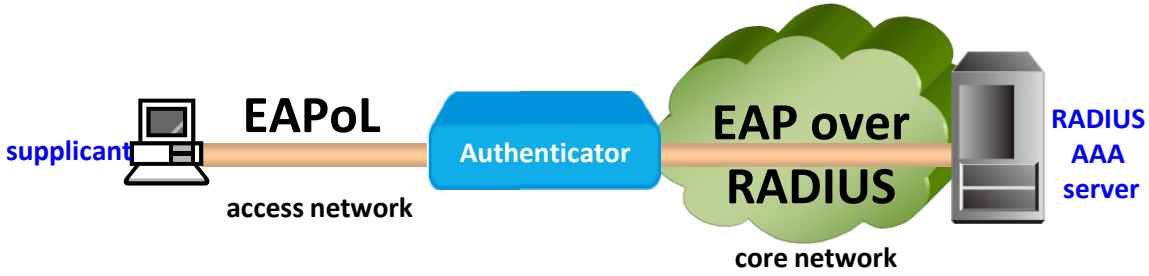
802.1X

802.1X is an IEEE standard for authenticating users of a LAN or WLAN

1X defines

- 3 parties: supplicant, authenticator, and authentication server
- an encapsulation of EAP called EAPoL (EAP over LAN)
- a Port Access Entity (PAE)
  - before authentication only EAPOL traffic can pass through the port
  - after authentication all traffic can pass through the port

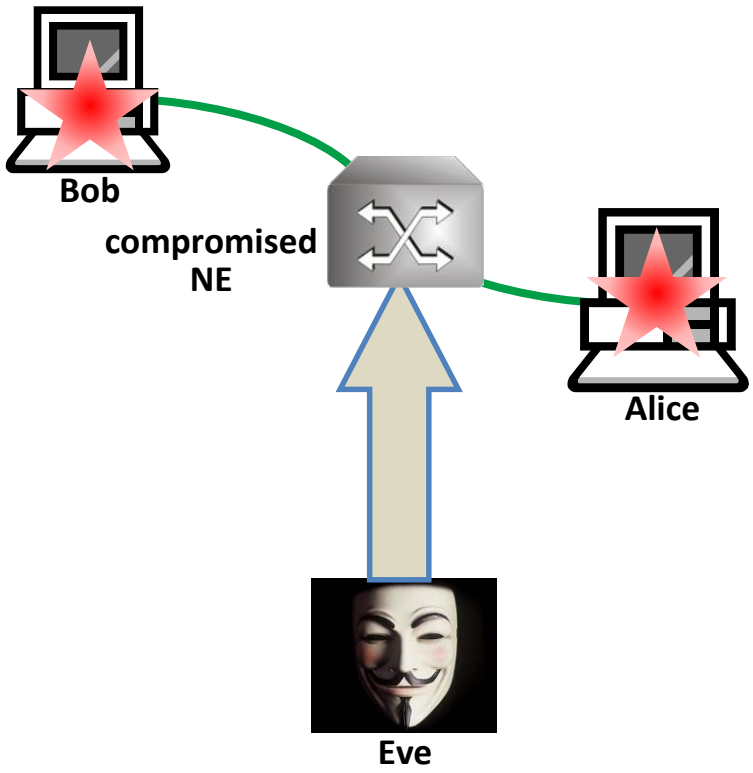
802.1X was extended in 2010 to authorize security associations and services in order to support MACsec (802.1AE) and Secure Device Identity (802.1AR)



Integrity (anti-tampering)

The next attack we need to consider is a *Man in the Middle* (MiM) attack

Here someone gains access to a NE and can tamper with packets on-the-fly



MACs and HMACs

We can provide integrity using a Message Authentication Code (MAC)

A MAC is a short block of information that
- is uniquely determined by the message
- verifies that the message has not been modified
  i.e., it is highly unlikely that a modified packet has the same MAC (collision)
- is difficult/impossible to forge

The MAC is inserted into packet headers
and is verified upon receipt

A Hash-based Message Authentication Code (HMAC)
uses a crypto-hash as a MAC (but block ciphers and other mechanisms can be used)

The simplest way to prevent forging MACs is by using a *shared key*
- Alice calculates MAC from message + key, and inserts into packet
- Bob calculates MAC using message + key, and compares to MAC in packet
- Eve, not knowing the key, can not forge the MAC

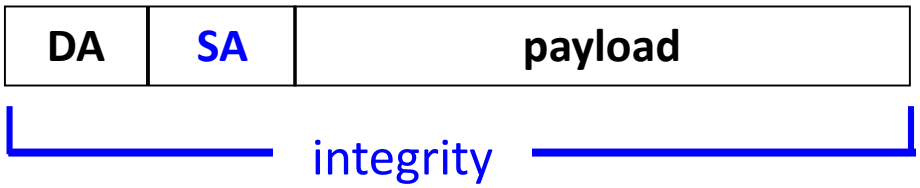
Source Authentication

Ethernet and IP packets contain Source Addresses (SA)
but these can be readily forged

MACs can also be used to authenticate a packet's source
that is, to prove that the SA correctly indicates the packet's source

We can reuse integrity mechanisms (e.g., MACs) to solve this problem !

All that is needed is to have the (H)MAC protect the SA
if the MAC is correct, then the SA indeed belongs to the claimed sender



Replay Attacks

Another type of attack is the replay attack

Here the MiM intercepts a packet and resends it multiple times
(transfer$100 → transfer$100, transfer$100, transfer$100, transfer$100)

Integrity and source authentication mechanisms do not detect replay attacks since the MACs calculate correctly

We can reuse integrity mechanisms (e.g., MACs) to solve this problem !

To defend against replay attacks

- add or utilize a packet sequence number field
- have the MAC protect the sequence number field
- if the same SN is received again, discard packet

Confidentiality

An important attack vector is interception
the packet content is observed by unintended parties

The standard countermeasure is *encryption*

There are two very different types of encryption – *symmetric key* and *public key*

Symmetric key encryption is based on the sides having a shared secret

For example, if Alice and Bob have an identical copies of a *one-time pad,*
i.e., a list of random bits at least as long as the message,

Alice can xor her message with the one-time pad
creating a message unreadable to Eve (who does not have the one-time pad)

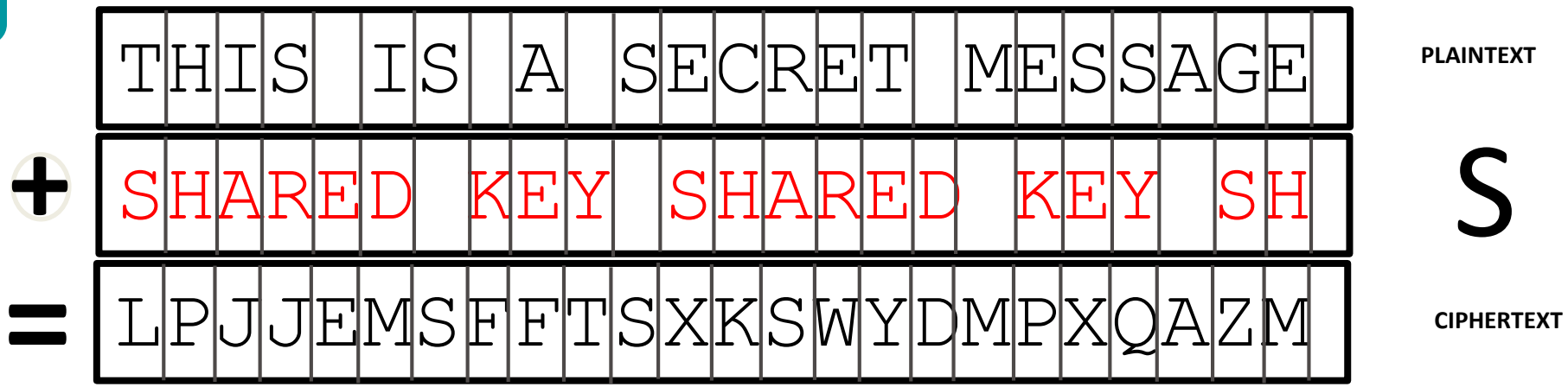
Bob can xor the encrypted message with the one-time pad
recovering the original message

Note that if the one-time pad bits are truly random
this is a provably secure system

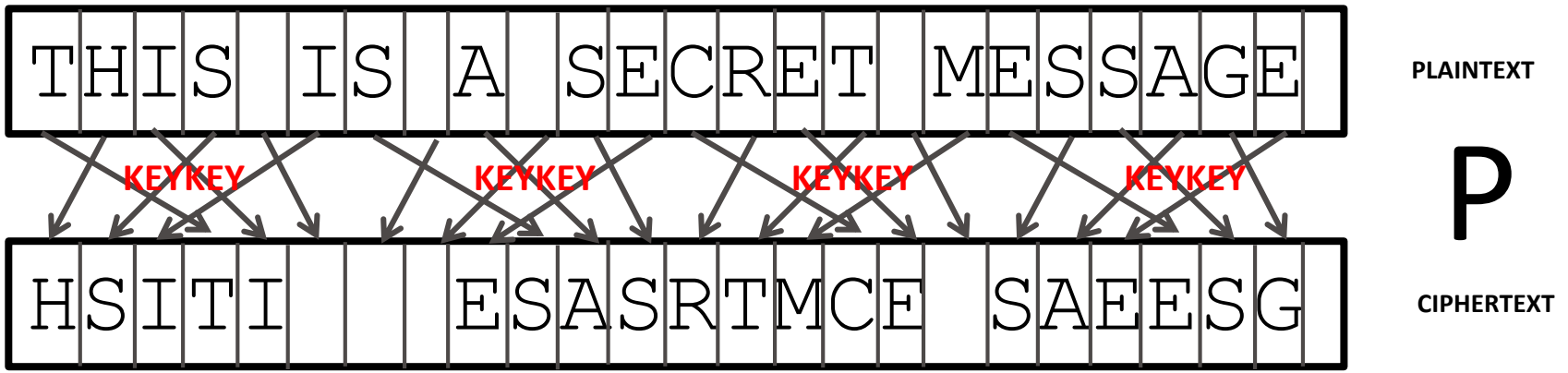
However, one-time pads need to be very long and are thus hard to use
instead most symmetric encryption algorithms use shared *keys*

Symmetric Encryption Using a Shared Key

Simple encryption algorithms are either substitution ciphers



Or permutation ciphers



DES

Modern symmetric key algorithms use Shannon's idea of alternating between S and P stages

In 1977 NIST published the **D**ata **E**ncryption **S**tandard based on the Feistel algorithm

- Inputs
  - 64 bits of plaintext
  - a 56 bit key
- Performs 16 rounds of S-boxes and P-boxes
- Outputs 64 bits of ciphertext

Due to using a short 56-bit key since the late 1990s DES is no longer secure against brute-force attacks (and possibly never was …)

In order to save DES, a method called triple-DES (3DES) was proposed
3DES uses a key of length 3*56=168 bits, but its effective length is 112 bits
NIST allows 3DES use until 2030, but there is something better …

AES

In 2001 NIST adopted an algorithm called Rijndael
(after its Belgian creators Vincent Rijmen and Joan Daemen)

as the Advanced Encryption Standard

This choice was the result of a 5-year process
in which 15 candidate algorithms were compared

AES

- Inputs
  - 128 bits of plaintext
  - a 128/192/256 bit key
- Performs 10/12/14 rounds of substitutions and permutations
- Outputs 128 bits of ciphertext

AES is approved by the NSA for protection of ***top secret*** data (w/ 256 bit key)

Block Cipher Modes (1)

The encryption mechanisms we have discussed so far are *block ciphers*
that is, they operate on blocks of N bits

But how do we use a block cipher to encrypt an arbitrary sized message ?

The simplest method is Electronic codebook (ECB)
which performs the block cipher independently on each N bits
(if needed, the last block is padded with zeros)

If the input repeats itself, ECB encrypts in the same way
revealing patterns in the plaintext (*leakage* - aiding the cipher breaker)

ECB is also susceptible to replay attacks

Thus, ECB should never be used

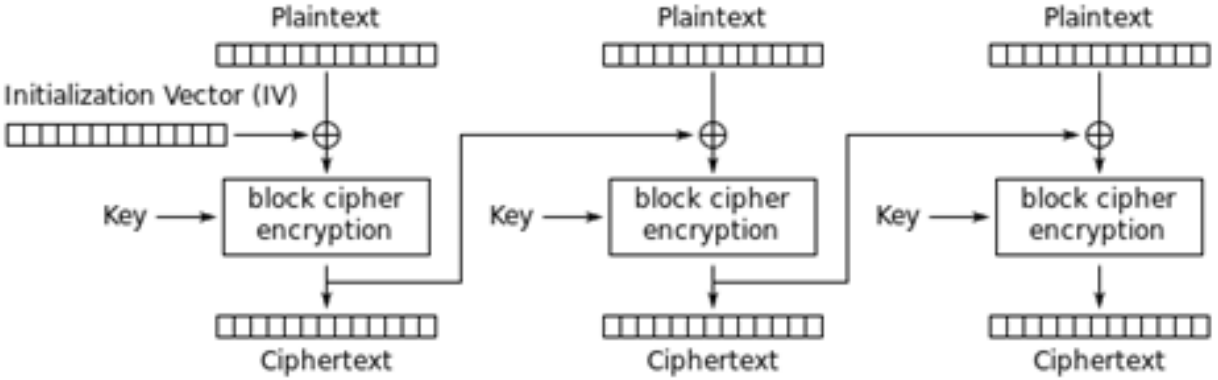
What can be done ?

Instead of encrypting the plaintext,

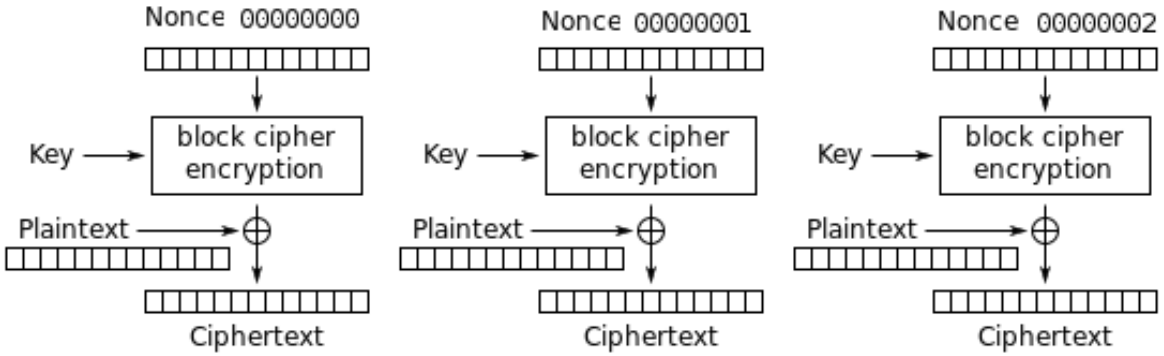
- we can xor the plaintext with the previously encrypt block before encrypting
- encrypt the previous ciphertext and then xor with the plaintext
- or even encrypt something else entirely (e.g., a counter) and xor with the plaintext

Block Cipher Modes (2)

Cipher block chaining CBC



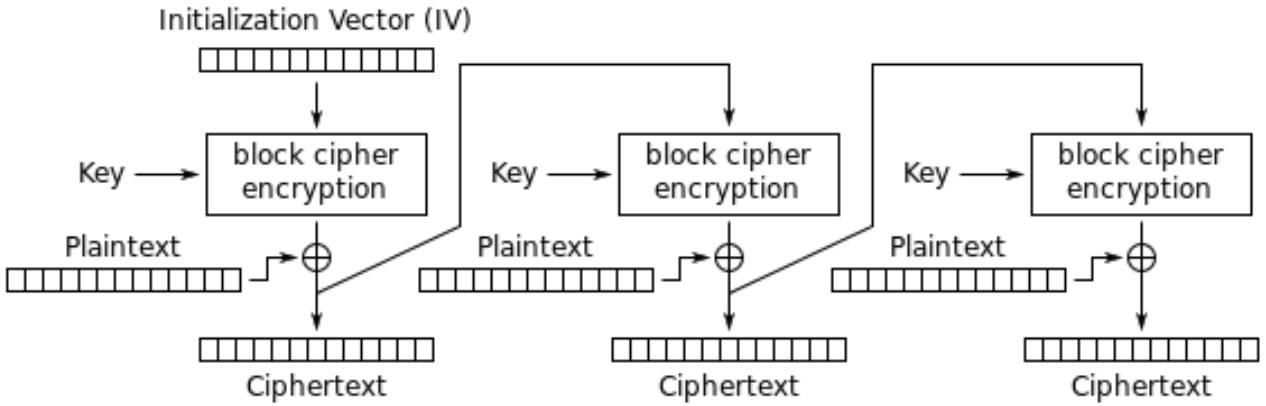
Counter mode CTR



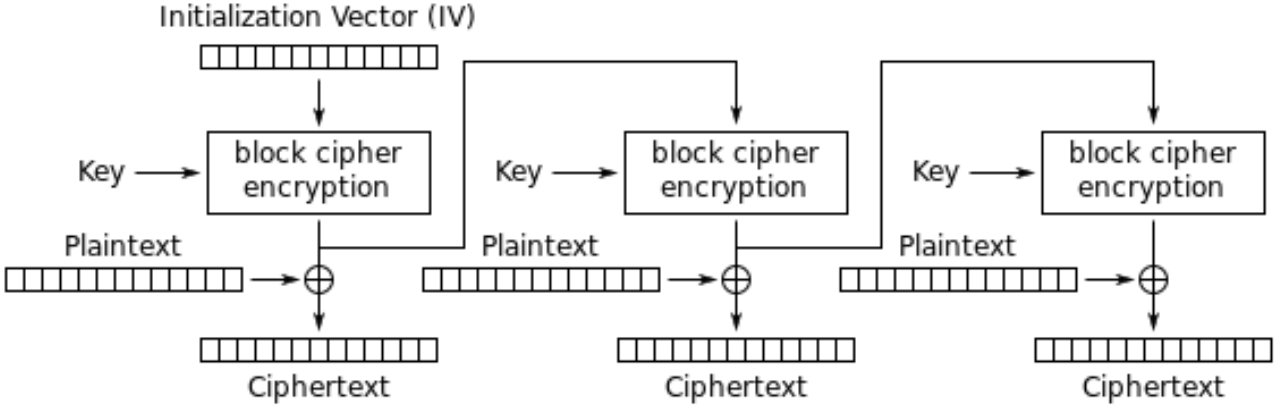
Galois Counter Mode GCM is CTR using Galois field operations

Block Cipher Modes (3)

Cipher feedback CFB

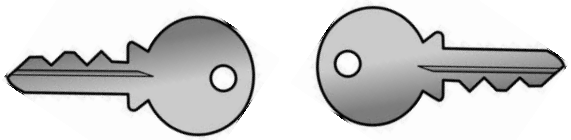


Output feedback OFB



Key Exchange / Distribution

The problem with symmetric key encryption
is that both parties must possess the *shared key*



This requires:

- face-face meetings
- trusted couriers
- synchronized key generators
- using an existing (but perhaps compromised) secure channel

This makes symmetric encryption logistics a nightmare

For many years mathematicians searched for a solution to this problem

it was first solved in principle in GCHQ by James Ellis (idea) and Chris Cocks (1-way function)
but remained unknown to the public as it was classified
key exchange was solved by Martin Hellman
public key was solved in theory by Whitfield Diffie (idea) and Ron Rivest (1-way function)

My classification

We can explain 4 types of encryption by the following analogies

Shared-key (symmetric) encryption (e.g., DES, AES)

- Alice and Bob both have identical copies of a the key to a strong-box
- Alice puts her message in the box, locks it with her key, and sends to Bob
- Bob uses his identical key to open the box and recovers the message

One private key encryption (e.g., RSA or ECC)

- Bob has the key to open a box that locks automatically upon closing (1-way function)
- Bob sends open box to Alice, Alice puts message in box, closes, sends back to Bob
- Bob opens the box with his key

Two private key encryption (e.g., RSA or ECC with signature)

- Alice has a *locking key* and Bob has an *unlocking key* to a strong-box
- Alice puts her message in the box, locks it with her locking key, and sends to Bob
- Bob unlocks the box with his unlocking key, and recovers *Alice's* message

Multi-exchange (e.g., DH)

- Alice puts her message in the box, locks with her padlock, and sends to Bob
- Bob places his padlock alongside Alice's, locks, and sends back to Alice
- Alice removes her padlock, and sends back to Bob
- Bob removes his padlock, and recovers the message

First attempts to encrypt without keys

Can we use one-time pads to communicate without shared secrets ?

ATTEMPT 1 – substitution code

- Alice xors message M with a one-time pad A and sends $D_1 = M + A$ to Bob
- Bob xors D_1 with his one-time pad B and sends $D_2 = M + A + B$ to Alice
- Alice xors D_2 with her old one-time pad A and sends $D_3 = M + A + B + A = M + B$
- Bob xors D_3 with his old one-time pad B and recovers $D_3 + B = M + B + B = M$

While the data of each transmission D_n is safe
if Eve observes all the transmissions, she can recover the message

- Eve observes $D_1 = M + A$, $D_2 = M + A + B$, and $D_3 = M + B$
- Eve xors $D_1 + D_2 = B$, and then xors $D_3 + B = M$

ATTEMPT 2 – permutation code

- Alice permutes M with a one-time permutation A and sends $D_1 = A*M$ to Bob
- Bob permutes D_1 with his one-time permutation B and sends $D_2 = B*A*M$ B to Alice
- permutations don't commute
  so if Alice permutes D_2 with the inverse of her old one-time permutation
  se obtains $D_3 = A^{-1} * B * A * M \neq B*M$

Public key cryptography

Two things are missing to make this idea work:

1. one-way functions
2. greater mathematical sophistication

The breakthrough is *public key cryptography* (AKA asymmetric cryptography)

With this method each party has a *public key* and a *private key*

Public keys may be advertised
but private keys are kept private

Operation

- Alice encrypts the message using Bob's public key (and optionally her private key*)
- Bob decrypts it using his private key (and optionally Alice's public key)
- Eve, not knowing the private key, can not decrypt the message

So no key distribution is needed
although you still have to ensure that Alice's public key is authentic

* if authentication is desired

Digital Signatures

Another application of public key cryptography is digital signatures

If a message is digitally signed by Alice
then Bob can verify that Alice really signed it, and not Eve

The idea behind the use of public key methods is
the signature proves that the signer had access to the private key
the private key is not divulged

Operation

- Alice signs the message with her private key and sends to Bob
- Bob can verify the signature using Alice's public key
- as a side effect, the verification also ensures integrity

Note that this is a kind of source authentication
but different from a MAC
since a MAC relies on a shared secret

Public key algorithms

Public-key cryptography (asymmetric cryptography)
relies on mathematical calculations that are hard to perform, such as

- **factoring large integers**
  it is easy to multiple 2 large primes p and q to find $n = pq$
  but it is hard to factor n to find p and q
- **finding discrete logarithms** (related : computational DH, decisional DH)
  given a finite group G
  it is easy to multiply an element b with itself n times to find $g = b^n$
  but it is hard to find given b and g to find n such that $b^n = g$
- **elliptic curve logarithms**
  given an elliptical curve $y^2 = x^3 + ax + b$ over the field F_p
  (elements 0 … p-1 with all operations modulo p)
  it is easy to perform the EC multiplication operation
  of element b with itself n times to obtain $b^n = b \cdot b \cdot b \ldots = g$
  but it is hard given b and g to find n such that $b^n = g$

RSA

RSA was one of the first public key algorithms to be discovered
Named after Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman who published it in 1977 although discovered earlier in GCHQ but classified

RSA is based on the facts that
- it is easy to multiple 2 large prime numbers
- it is hard to factor the product to recover the prime numbers

It is an open problem in mathematics whether factoring is indeed hard

To use:
- find two large prime numbers and multiply them
- based on number theory create public and private keys
- sign messages using private key or
- encrypt messages using recipient's public key

RSA *was* patented (by RSA Security, founded by R, S, and A) but expired in 2000
- there are rumors that RSA placed backdoors in their products for the NSA
- RSA Security was acquired by EMC in 2006

RSA is still often used instead of stronger elliptical curve methods due to the latter having patents in force

RSA algorithm

Preparation

- Bob selects two large primes p and q (there are many methods to find primes)
- Bob calculates $n = pq$ and $\Phi=(p-1)(q-1)$
- Bob selects e between 2 and Φ-1 which is co-prime to Φ
- Bob computes d such that $ed = 1 \bmod \Phi$ (using Euclid's algorithm)
- Bob publishes $\{n, e\}$ as his public key, and keeps d as his private key

Mathematical background

- Raising M to the e power modulo n : $M^e \bmod n$
  - is a 1:1 transform
  - is a one-way function
- if $C = M^e \bmod n$ then $M = C^d \bmod n$ (follows from *Fermat's little theorem* and *CRT*)

Operation

- Alice encrypts her message M into ciphertext thus: $C = M^e \bmod n$ and sends to Bob
- Bob decrypts thus : $C^d \bmod n = M$
- Eve, not knowing p and q and thus not d, can not recover M

El Gamal

Discrete logarithm cryptosystem published by Taher Elgamal in 1985 and used in NIST's **D**igital **S**ignature **S**tandard

Preparation

- Bob chooses a cyclic group G of order q with generator g
- Bob chooses a random number x between 1 and q-1
- Bob computes $h = g^x$
- Bob publishes $\{G, q, g, h\}$ as his public key, and keeps x as his private key

Operation

- Alice randomly chooses an ephemeral key y between 1 and q-1
- Alice calculates $c_1 = g^y$
- Alice calculates the shared secret $s = h^y$
- Alice encodes her message as an element m of G and calculates $c_2 = ms$
- Alice sends the ciphertext (c_1, c_2) to Bob
- Bob calculates the shared secret $s = c_1^x = g^{yx} = g^{xy} = h^y$
- Bob calculates the inverse of s : $s^{-1} = c_1^{q-x} = g^{(q-x)y}$
- Bob recovers the message $m = c_2 s^{-1} = ms\ s^{-1}$
- Eve, not knowing x, can not find s or s^{-1}

Certificates

I can be sure of Alice's public key if she personally hands it to me
if I am just given it – it may be forged by Eve !

How can we be sure of someone else's public key ?

The idea is to have someone trusted (i.e., for whom we already have a public key) vouch for it

How can that trusted party be sure ? Someone he trusts vouches for it!

There are two methods to implement this

- **P**ublic **K**ey **I**nfrastructure (e.g., X.509)
- web of trust (e.g., PGP)

X.509 is an ITU-T standard for PKI

It specifies the format of certificates
and a strict hierarchical system of Certificate Authorities that issue them

With the web of trust model anyone (not just CAs) may sign certificates

Secure key exchange

Using public key cryptography no key distribution is required

However, public key encryption is computationally much more expensive than symmetric encryption

One solution to this problem is to re-use public key cryptography

We only use public key encryption to encrypt a (symmetric) key
all the messages are sent using inexpensive symmetric cryptography

Operation

- Alice encrypts a random key using her private key and Bob's public key
- Bob decrypts the key using his private key and Alice's public key
- Alice and Bob now communicate using symmetric encryption
- after some amount of key use, the process is repeated

Diffie–Hellman key exchange

We do not need full public key cryptography to distribute symmetric keys

DH is a mechanism published by Whitfield Diffie and Martin Hellman in 1976
although discovered earlier in GCHQ but classified

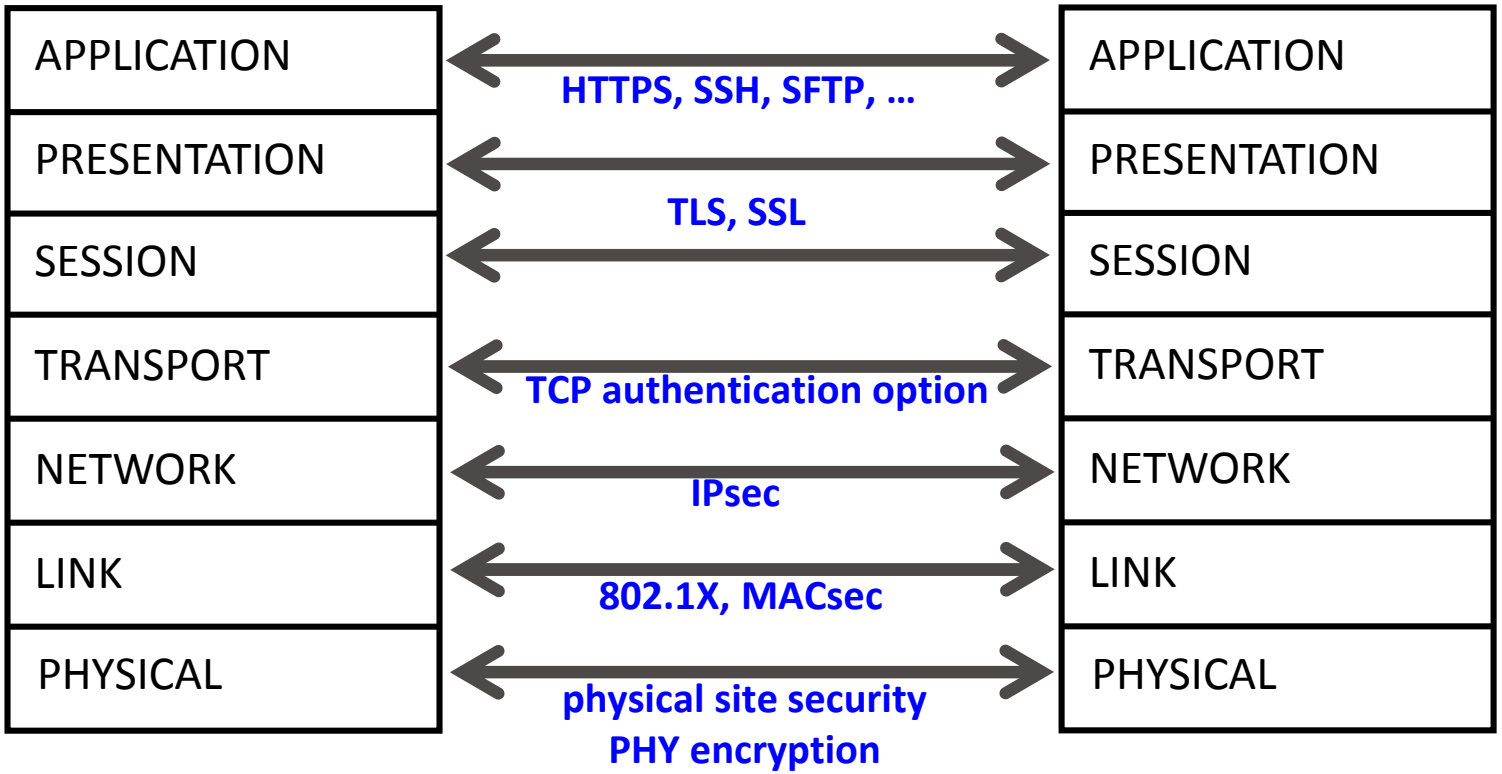
DH is based on the field F_p (numbers 0 … p-1, with operations modulo p)

Operation

- Alice and Bob agree to use a *prime number* ***p*** and a *primitive root* ***g***
  (***g*** is a primitive root, if *every* number is congruent to g^n for some n)
- Alice chooses a secret integer ***a*** and sends to Bob $\boldsymbol{A} = g^{\boldsymbol{a}} \bmod p$
- Bob chooses a secret integer ***b*** and sends to Alice $\boldsymbol{B} = g^{\boldsymbol{b}} \bmod p$
- Alice now computes $\boldsymbol{s} = B^{\boldsymbol{a}} \bmod p = g^{ba} \bmod p$
- Bob now computes $A^{\boldsymbol{b}} \bmod p = g^{ab} \bmod p = \boldsymbol{s}$
- Alice and Bob now share the secret **s**
- Alice and Bob now use ***s*** as a symmetric key !
- Eve, not knowing ***a*** or ***b***, can not deduce ***s***

Where to apply security ?

Every layer of the user plane can benefit from security mechanisms



IPsec

IP was designed when security was not an issue

IPsec is a set of open standards that rectify many of IP's deficiencies

IPsec is transparent to applications (apps needn't know that it will be used)

IPsec is based on the concept of a *security association* (SA)

SA is a relationship between two or more entities

- describes how the entities will securely communicate
- specifies which security features (authentication, integrity, encryption) will be used
- specifies algorithms (DES, AES, SHA-1, RSA, …) and options
- takes care of key exchange (ISAKMP Internet Security Association and Key Management Protocol)
  - pre-shared keys
  - Internet Key Exchange (IKE, IKEv2)
  - IPSECKEY DNS records

Internet **K**ey **E**xchange (UDP port 500) is used for establishing IPsec sessions (performing mutual authentication, establishing and maintaining SAs, key exchange)

IPsec modes

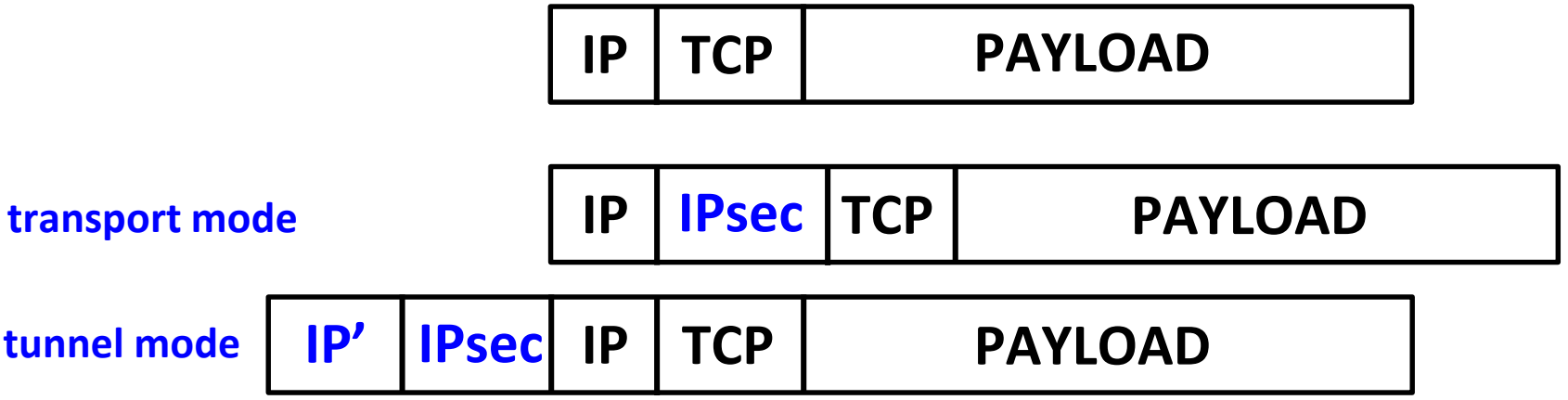
IPsec has two different modes of operation

Transport Mode

- IPsec header is inserted into an existing packet (between IP and TCP headers)
- packet routing is unaffected, but NAT traversal may be (new protocol numbers)
- adds security functionalities to existing flow
- if encrypting, TCP (or UDP) header is encrypted

Tunnel Mode

- entire IP packet is encapsulated (and optionally encrypted)
- can be used to create *IPsec VPN*



AH and ESP

IPsec has two different formats

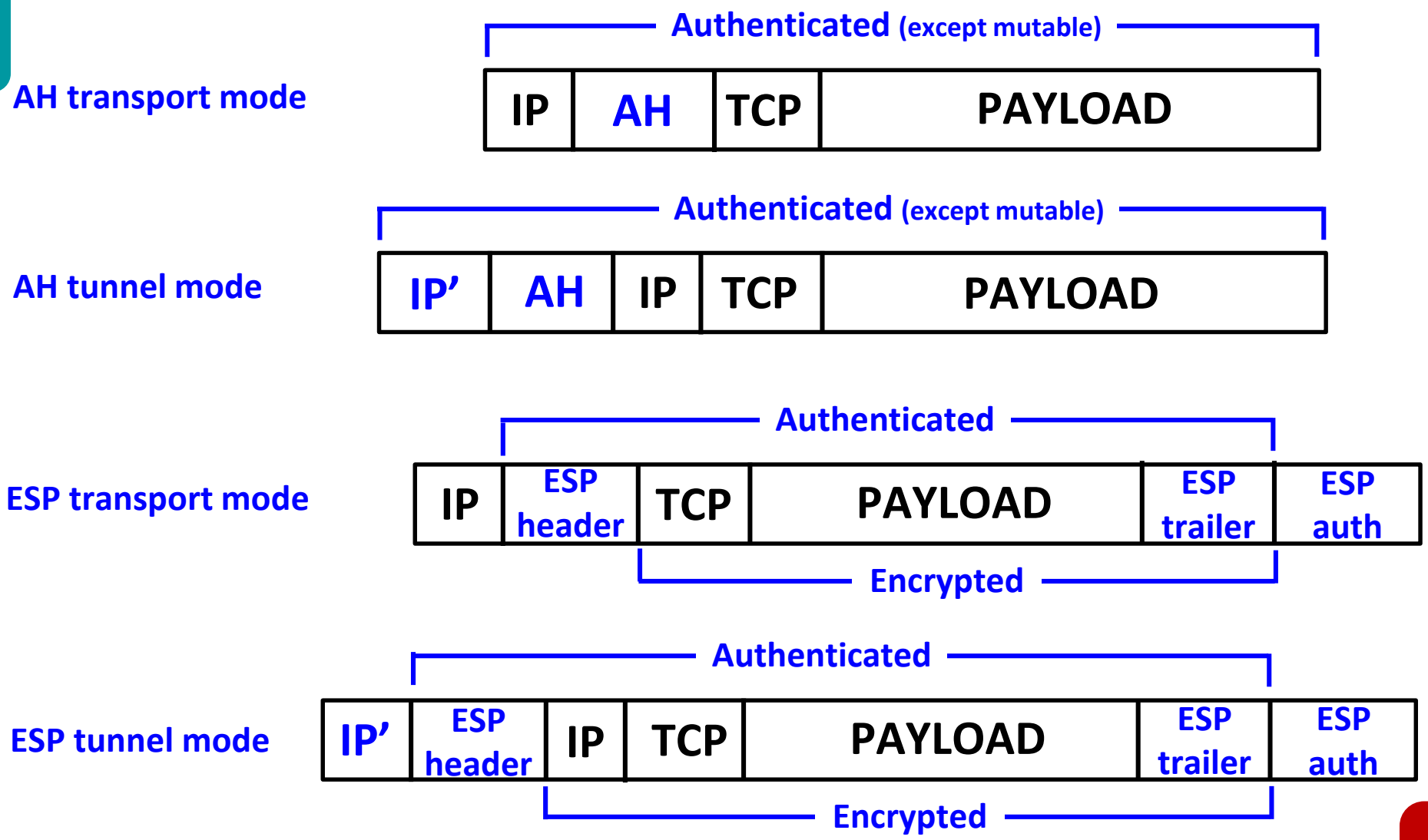
Authentication Header (AH)

- supports source authentication and data integrity only
- protection
  - in transport mode protects payload and header fields except mutable (e.g., TTL, DSCP, header checksum, offset, flags)
  - in tunnel mode protects entire packet
- uses protocol number 51

Encapsulating Security Payload (ESP)

- supports source authentication, data integrity, and encryption
  - similar to AH if null encryption is used
- adds *trailer(s)* in addition to IPsec *header*
- protection
  - in transport mode doesn't protect IP header (but does TCP header)
  - in tunnel mode entire packet is protected
- uses IP protocol 50

IPsec packet formats



IPsec algorithms

RFC 4835 is the most recent specification of supported algorithms for IPsec

Authentication

MUST	HMAC-SHA1-96
SHOULD+	AES-XCBC-MAC-96
MAY	HMAC-MD5-96

Encryption

MUST	NULL
MUST	AES-CBC with 128-bit keys
MUST-	3DES-CBC
SHOULD	AES-CTR
SHOULD NOT	DES-CBC
MAY	AES-CCM and AES-GCM

MACsec

802.1AE MACsec was approved in June 2006

based on well known AES-128 encryption

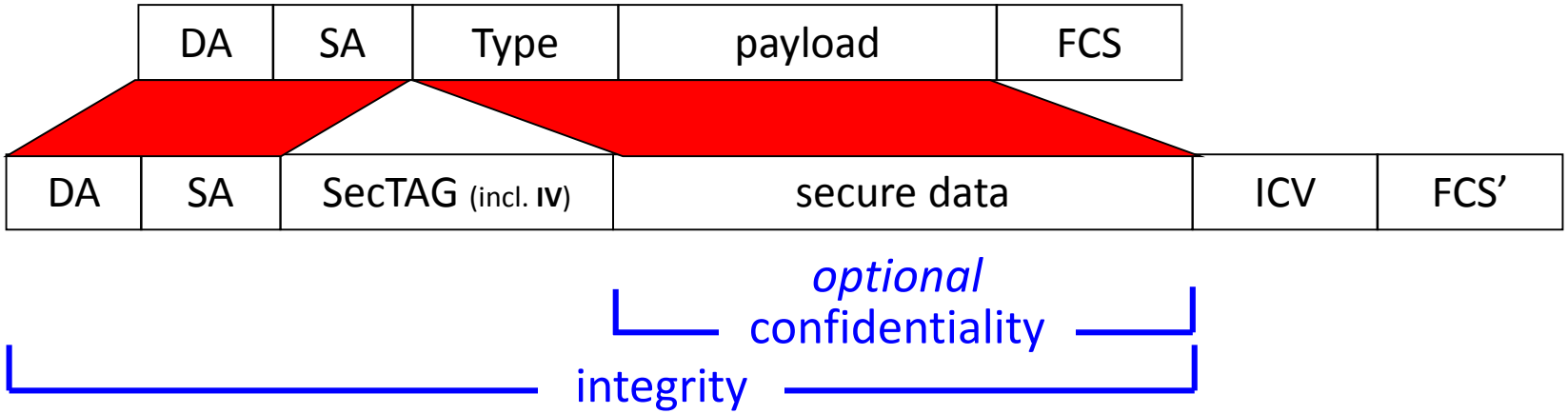
but with a new mode - **G**alois **C**ounter **M**ode

AES/GCM uses a single algorithm for integrity and encryption

MACsec

- works over Connectionless network by forming secure associations
- integrated into Ethernet frame format
- provides
  - source authentication
  - confidentiality
  - connectionless data integrity
  - replay protection
  - limited blocking of DoS attacks
- may degrade some QoS attributes (e.g. introduces bounded delay)

MACsec format



SecTAG contains

- MACsec Ethertype (88E5)
- 4B Packet Number (sequence number)
- 8B Secure Channel Identifier

} 12 B **I**nitialization **V**ector

ICV is a 16B **I**ntegrity **C**heck **V**alue

TLS (and SSL)

Sometimes it is better to provide security at a layer higher than L3

For these cases there is **T**ransport **L**ayer **S**ecurity (and previously versions of SSL)

Heartbleed bug in OpenSSL scared the whole world in April 2014

TLS

- adds integrity and encryption to client/server applications
  - https, smtp/pop/imap, sips, mysql, EAP-TLS…
- is based on SSL (developed by Netscape)
- is widely used to secure web browsing (https), email, ecommerce, etc.
- client picks a random number, encrypts with server's public key and sends to server client and server can now communicate using symmetric encryption and MACs

Compared to IPsec, TLS

- can be implemented in a browser (e.g., for online banking or commerce) and doesn't require/trust OS kernel support like IPsec VPNs
- only runs over connection oriented TCP (but there is DTLS is for UDP)
- can authenticate individual users, rather than IP source addresses
- is better than IPsec at NAT traversal