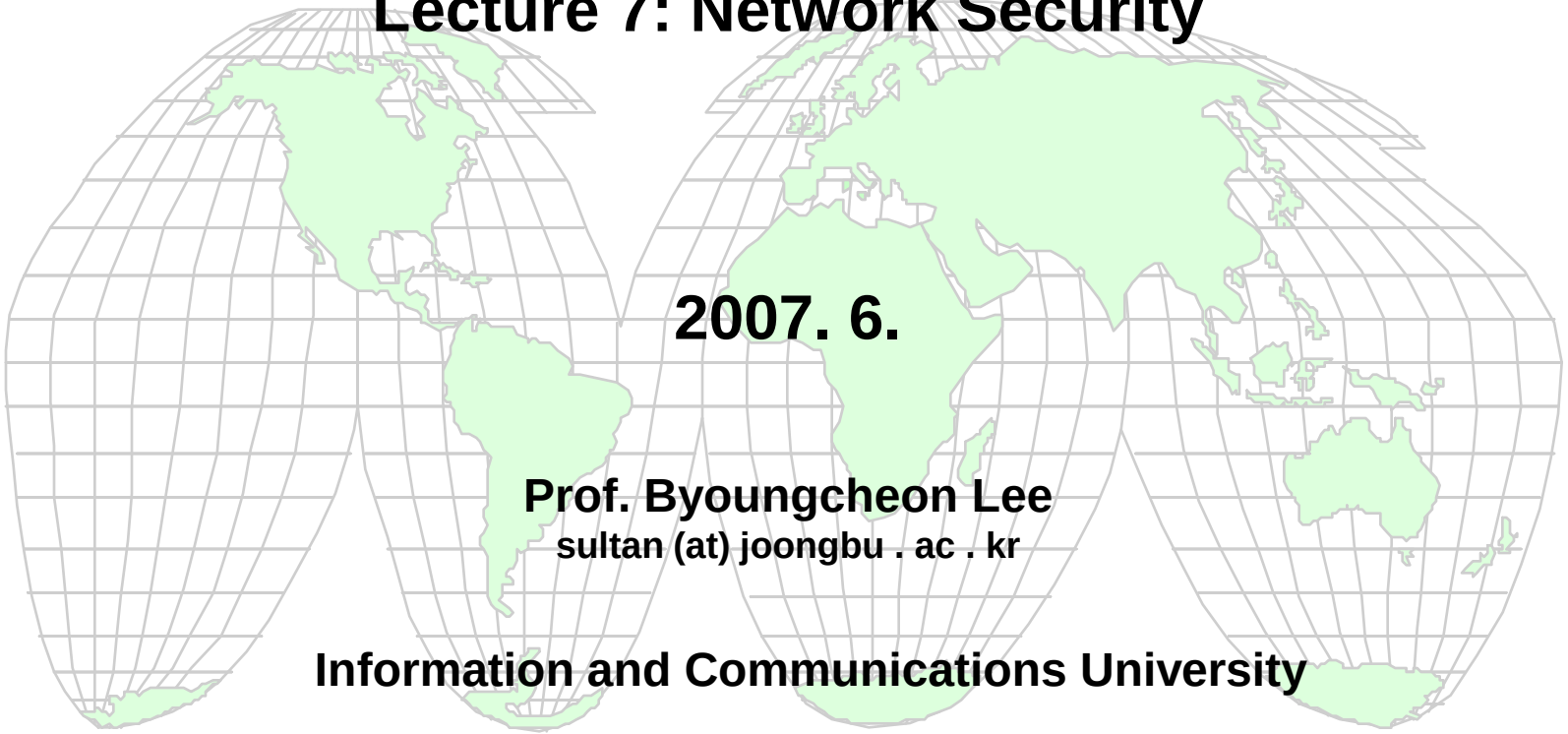
Introduction to Information Security

Lecture 7: Network Security



2007. 6.

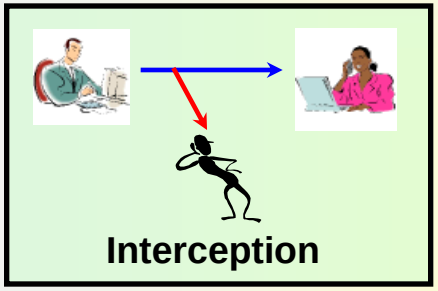
Prof. Byoungcheon Lee
sultan (at) joongbu . ac . kr

Information and Communications University

Contents

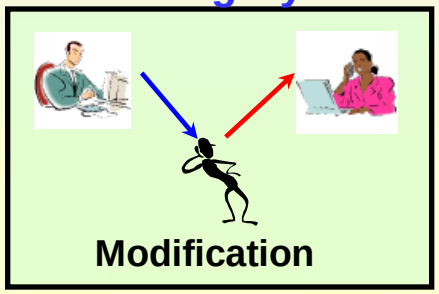
Security Needs for Network Communications

Confidentiality



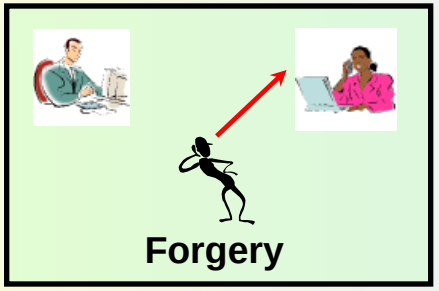
Is Private?

Integrity



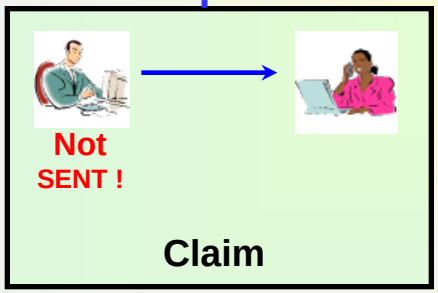
Has been altered?

Authentication



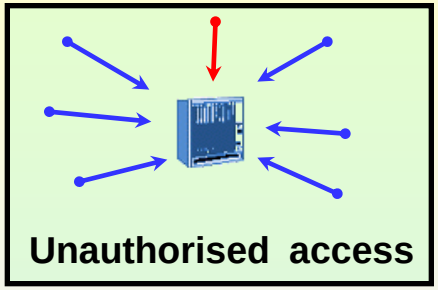
Who am I dealing with?

Non-Repudiation



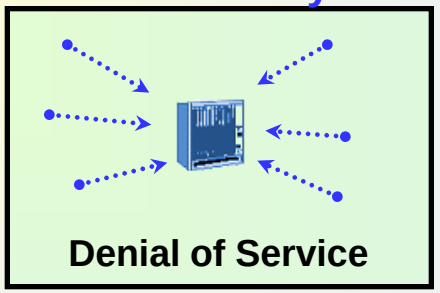
Who sent/received it?

Access Control



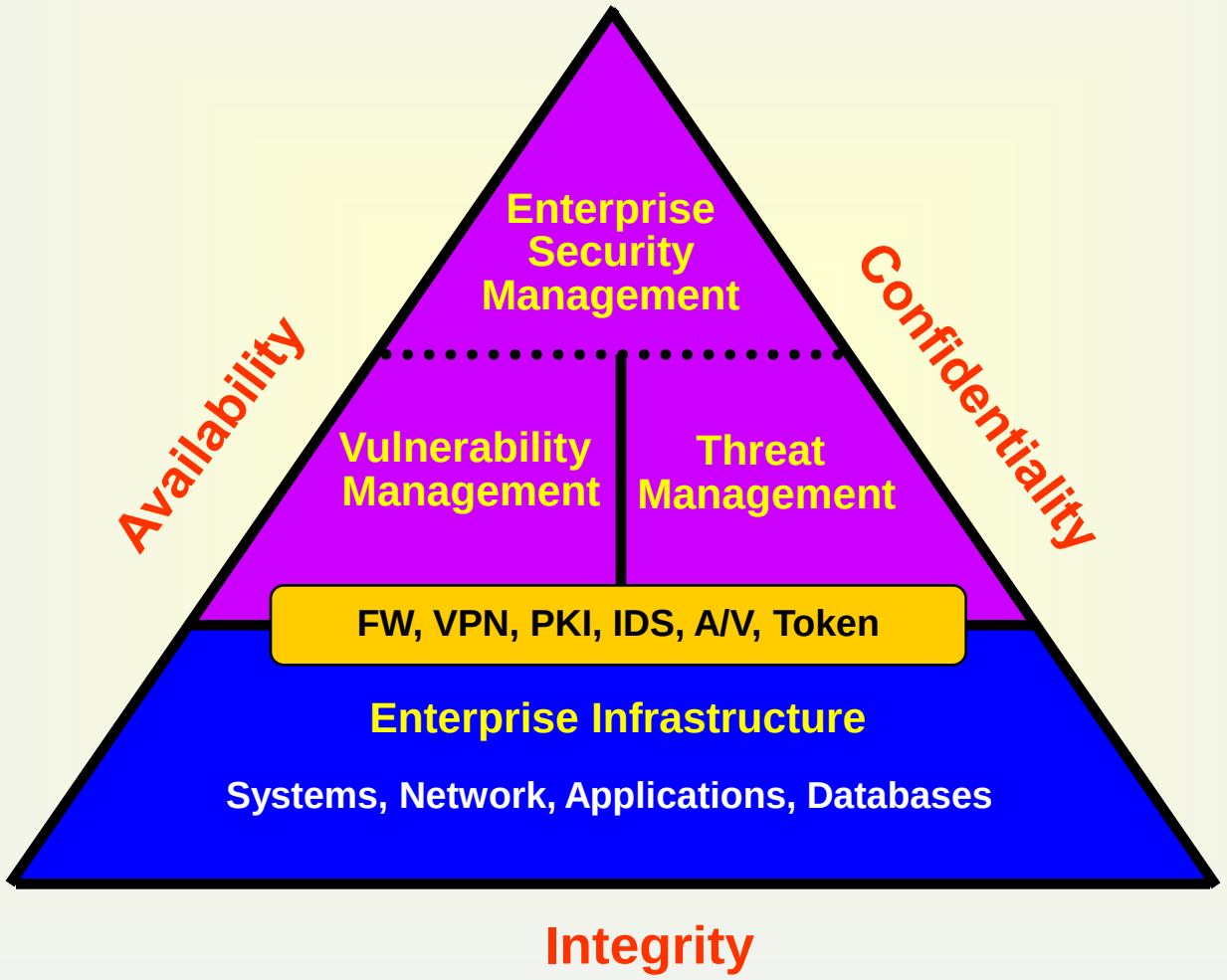
Have you privilege?

Availability

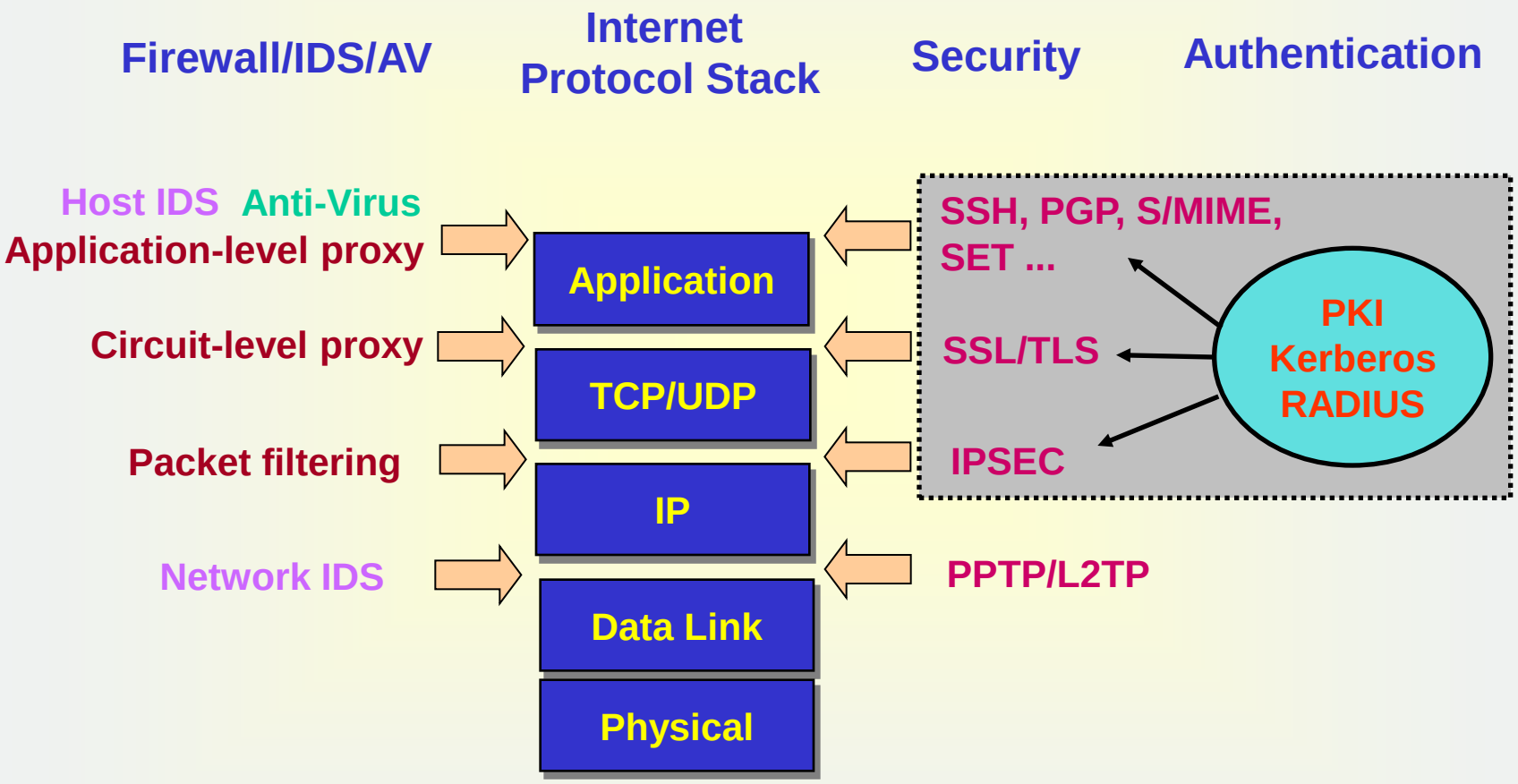


Wish to access!!

Enterprise Security Management



Major Internet Security Technologies



Cryptographic Primitives

- **Block / Stream Cipher (Symmetric Cryptosystem)**
  - 3DES, AES, SEED, RC5 … / RC4, SEAL …
- **Hash Function**
  - MD5, SHA1/SHA2, HAS160, RMD160, Tiger …
- **Message Authentication Code (MAC)**
  - HMAC, CBC-MAC, UMAC
- **Public Key (Asymmetric) Cryptosystem**
  - RSA, ElGamal; Hybrid systems (Asymmetric key agreement + symmetric encryption)
- **Digital Signature**
  - RSA, DSA/ECDSA, KCDSA/EC-KCDSA …
- **Key Exchange**
  - Diffie-Hellman, ECDH, RSA key transport
- **(Pseudo) Random Number Generators**
  - HW pure RNG; SW pseudo RNG

1. Introduction

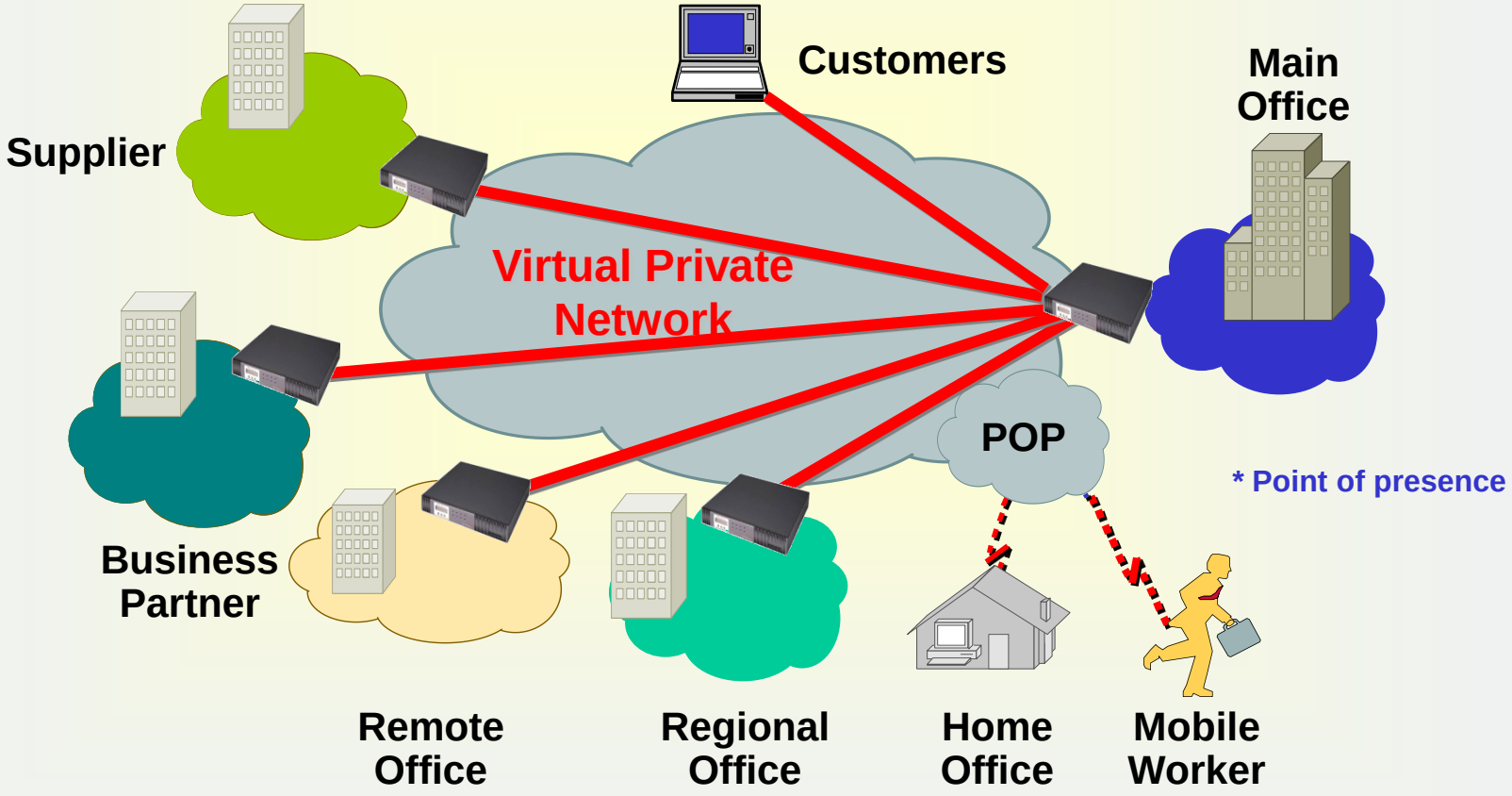
2. Virtual Private Networks(VPNs)

3. IP Security (IPSEC) Protocol

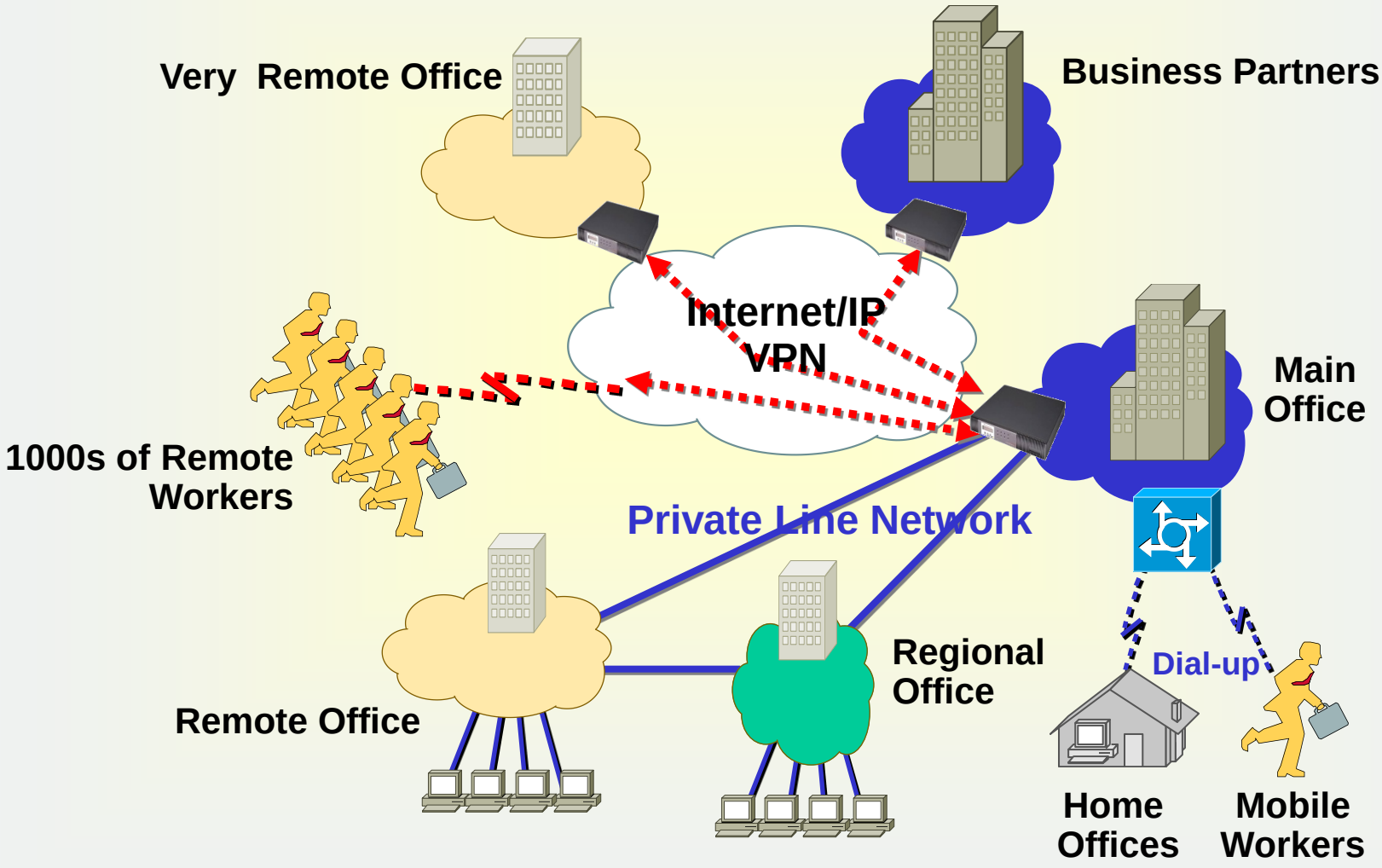
4. Transport Layer Security(TLS) Protocol

What Is a VPN?

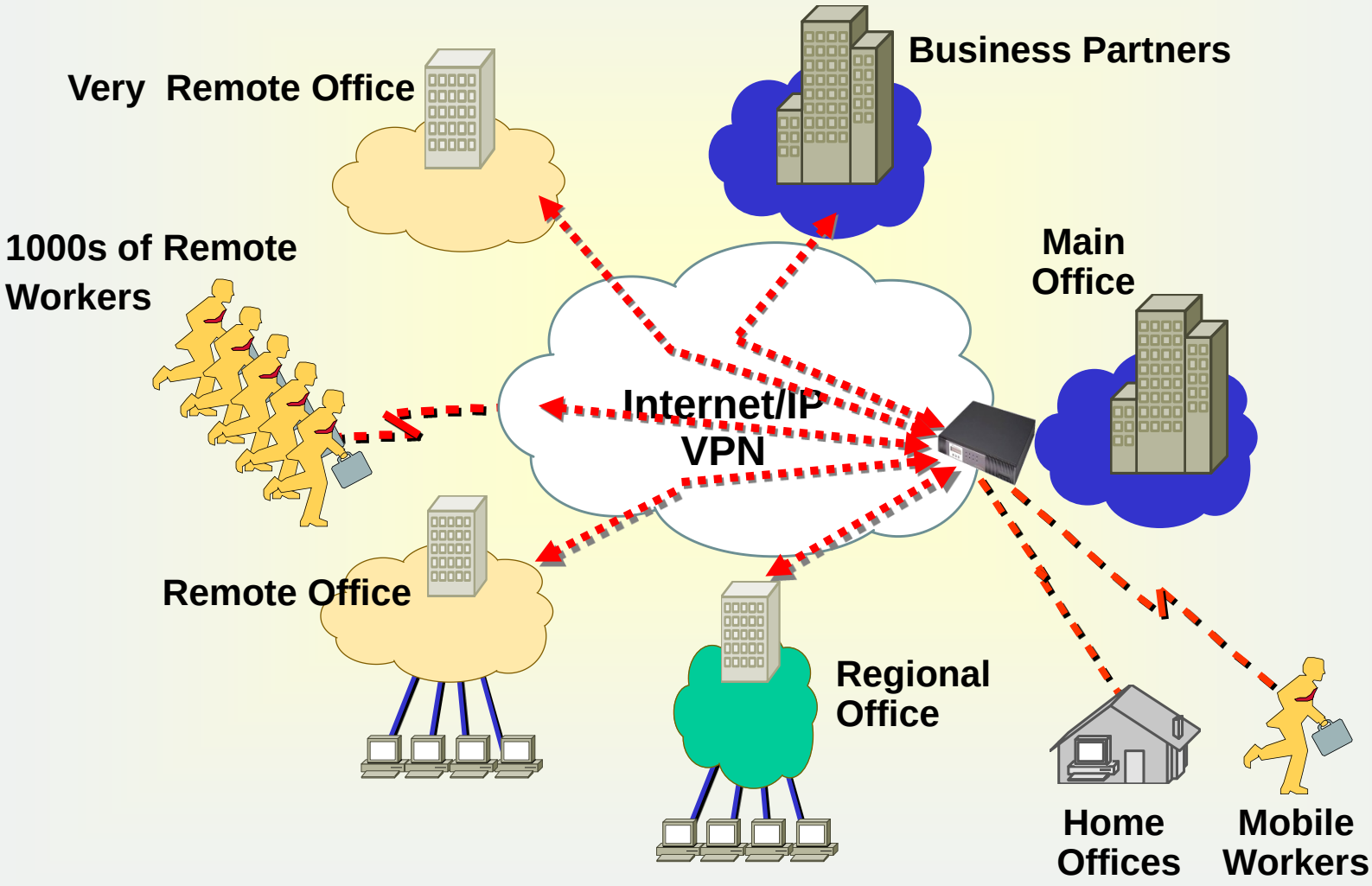
Secure connectivity deployed on a shared communication infrastructure with the same security policies and performance as a private network



How to Construct Private Networks?



VPNs Will Replace Existing Private Networks

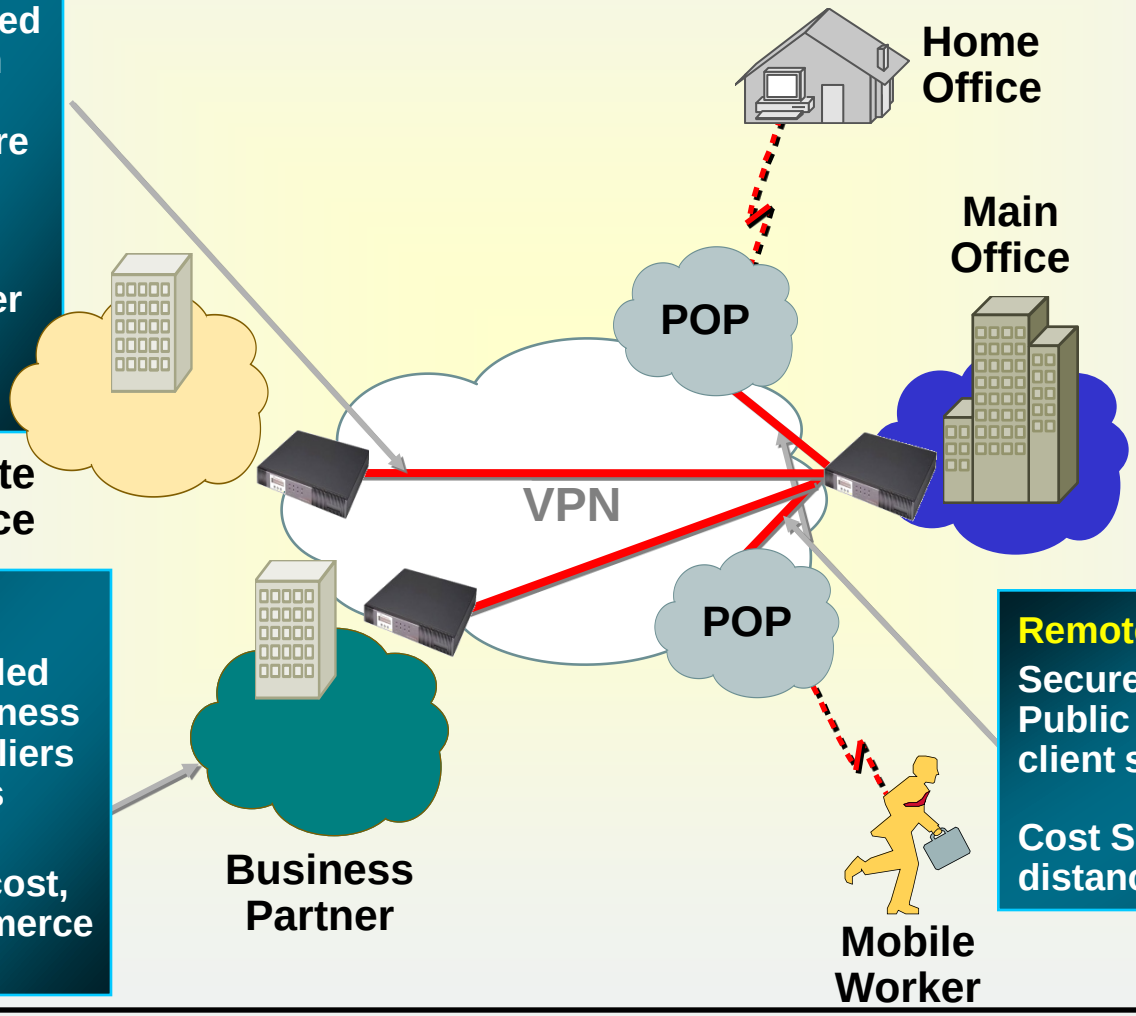


VPN Business Applications

Intranet VPN

Low Cost, tunneled connections with IPSec encryption and QoS to ensure security and reliability

Cost Savings over Frame Relay and Leased Lines



Extranet VPN

Allows controlled access to business partners, suppliers and customers

Provides low-cost, secure E-commerce infrastructure

Remote Access VPN

Secure tunnels across a Public Network with VPN client software

Cost Savings over long distance calls

VPN Benefits

- Build secure business infrastructure
  - Integrate dispersed business environments using secure, controlled connectivity over shared networks
  - Implement once for multiple applications
  - Centrally-controlled access policy
  - Enable multi-level, layered approach to security
- Use internet for remote access
  - Mobile users use internet accounts to gain access and tunnel to offices
- Create internal security
  - Protect sensitive internal traffic/systems from others
- Can also make private networks more private
- Can be used to back-up existing private networks

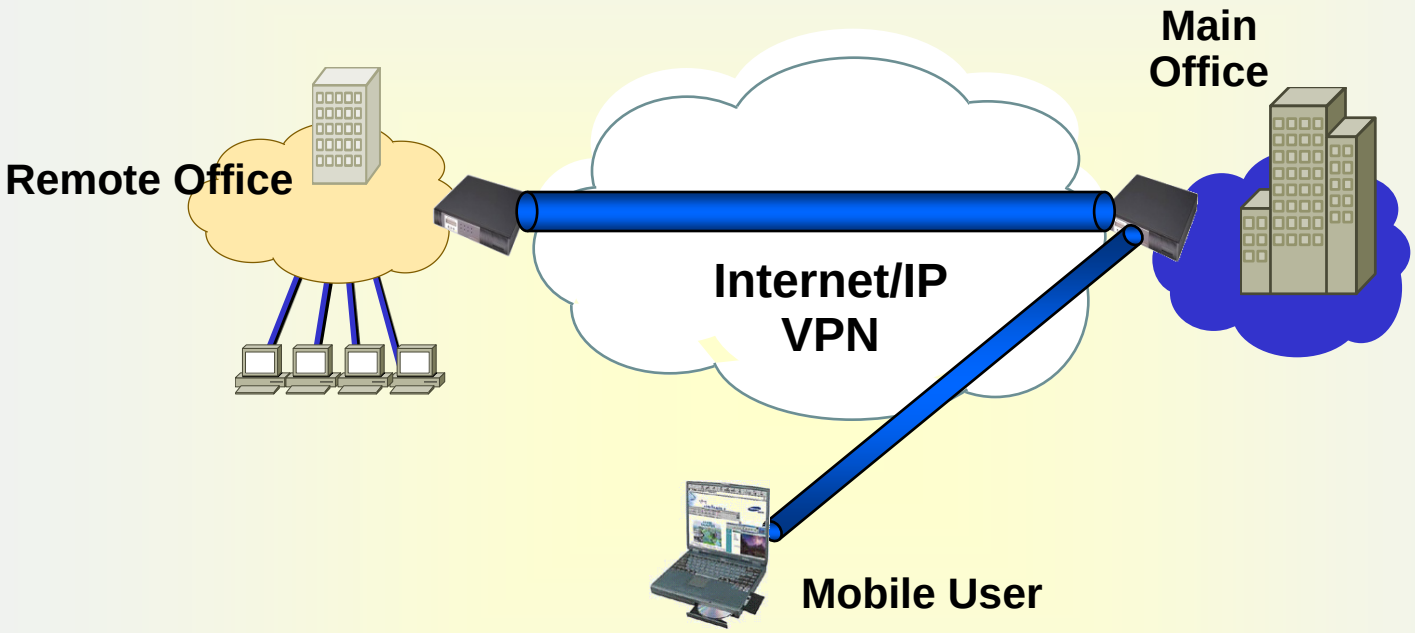
- VPN issues
  - Security
  - Quality of Service
  - Scalability / Reliability
  - Manageability

VPN Key Components

- **Tunneling**
  - PPTP, L2TP; MPLS; IPSEC, GRE, IP-in-IP; SSL/TLS
- **Security**
  - IPSEC vs. Virtual path(VC, PVC, LSP, etc.)
  - Encrypted tunnel vs. traffic separation
- **Access control**
  - Remote user authentication
  - Membership management
- **Policy Management**
  - Centralized policy control
  - Policy configuration, distribution & update
- **Quality of Service(QoS)**
  - Traffic classification, marking, policing & shaping
  - SLA: Latency, throughput, jitter, packet loss…
- **High Availability**
  - Transparent session fail-over
  - Load balancing, IP clustering

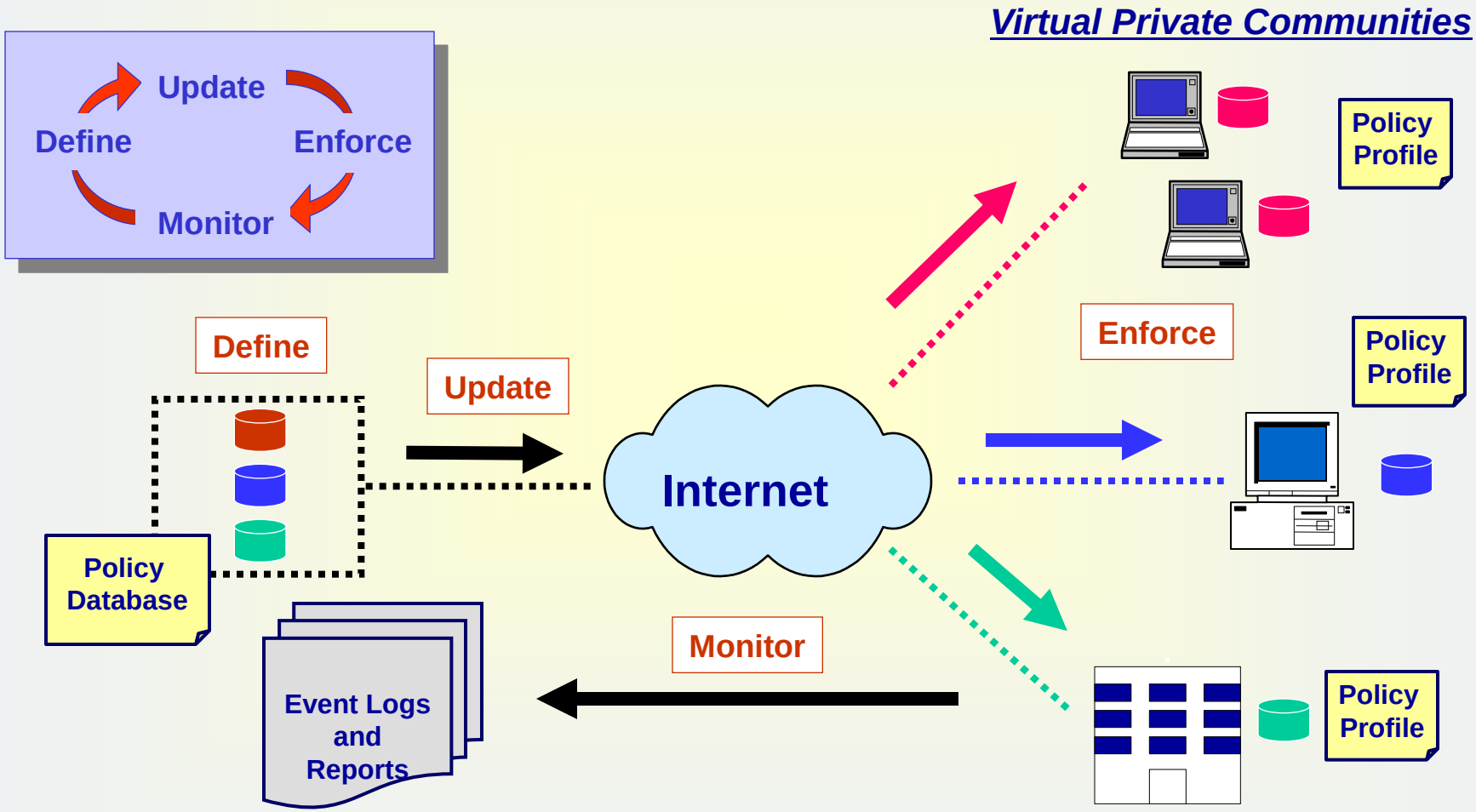
Generic Routing Encapsulation (GRE)
Multi-Protocol Label Switching (MPLS)
Quality of Service (QoS)
Service Level Agreements (SLA)
Point-to-Point Tunneling Protocol (PPTP)
Layer 2 Tunneling Protocol (L2TP)
Secure Socket Layer (SSL)
Internet Protocol Security (IPSEC)
Virtual Circuit (VC)
Permanent Virtual Circuit (PVC)
Label Switched Path (LSP)

Secure Tunneling: Virtual Presence



- **Tunneling** makes geographically dispersed offices/mobiles appear to be present in the same local network.
- **Security** makes VPN traffic separated from other traffic.

Policy Based VPN Management



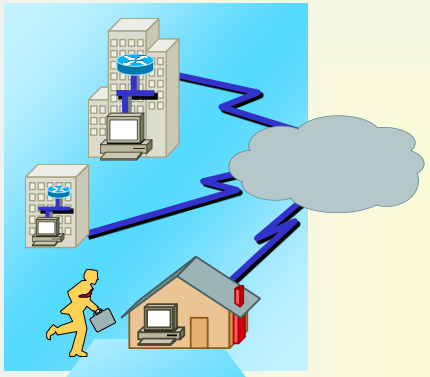
User Authentication Model

RSA SecurID

Two-factor authentication is based on something you know (a password or PIN) and something you have (an authenticator)

- Authentication Alternatives
  - Internal (Tunnel Server)
  - RADIUS (Remote Access Dial In User Service)
  - 2 Factor (e.g. Axent Defender , RSA SecurID)
  - NDS (NetWare Directory Services)
  - NT Domains, DEN (Directory Enabled Networks)
  - LDAP (Lightweight Directory Access Protocol)

Quality of Service in a VPN



QoS Benefits for VPNs

ISP

- Make optimum use of VPN WAN link(s)
- Provide bandwidth and priority to mission-critical applications
- Control non-mission-critical applications
- Exploit differentiated services offered by Service Provider

CPE Functions

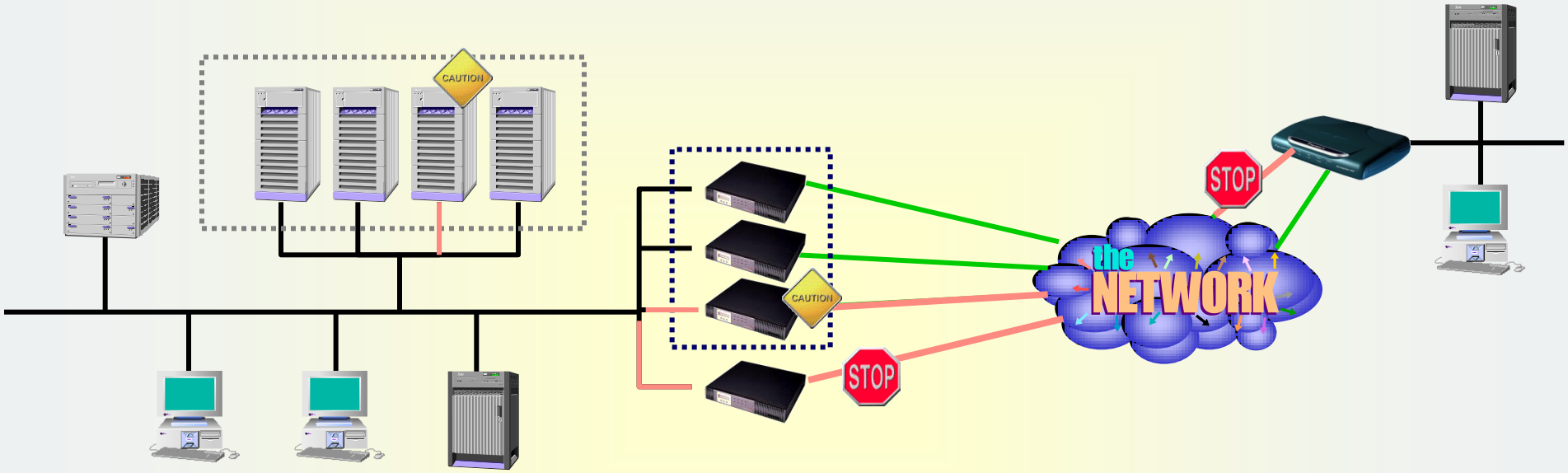
- Packet Classification
- Packet Marking
- WAN-Link Bandwidth Management
- Measurement/Reporting

SP Functions

- Adhere to SLA
  - Throughput
  - Latency
  - Availability
  - Control Congestion

* customer premises equipment

High Availability



- Failover: Active/standby
- Load Balancing: Active/active
- IP Clustering

Internet VPN Standards

VPN : Tunneling + Separation Technology

Protocol	PPTP	L2TP	MPLS	IPSEC	SOCKS5
Standard	MS	RFC	RFC	RFC	RFC
Layer	Link(2)	Link(2)	Layer 2.5	Network(3)	Session(5)
Security	PPP	Recommend IPSEC	None / IPSEC	IPSEC	SSL
WAN	IP only	Multi-protocol	Multi-protocol	IP only	IP only
Best for	Remote access	Remote access	Network-based VPN	Intra/extranet	Extranet

- PPTP/L2TP: Layer 2 tunneling by encapsulating PPP
- IPSEC : a set of IP layer end-to-end security protocols (AH, ESP, ISAKMP/IKE)
- MPLS : Multi-Protocol Label Switching
- SOCKS V5 : Session (circuit) level proxy with security features

1. Introduction

2. Virtual Private Networks(VPNs)

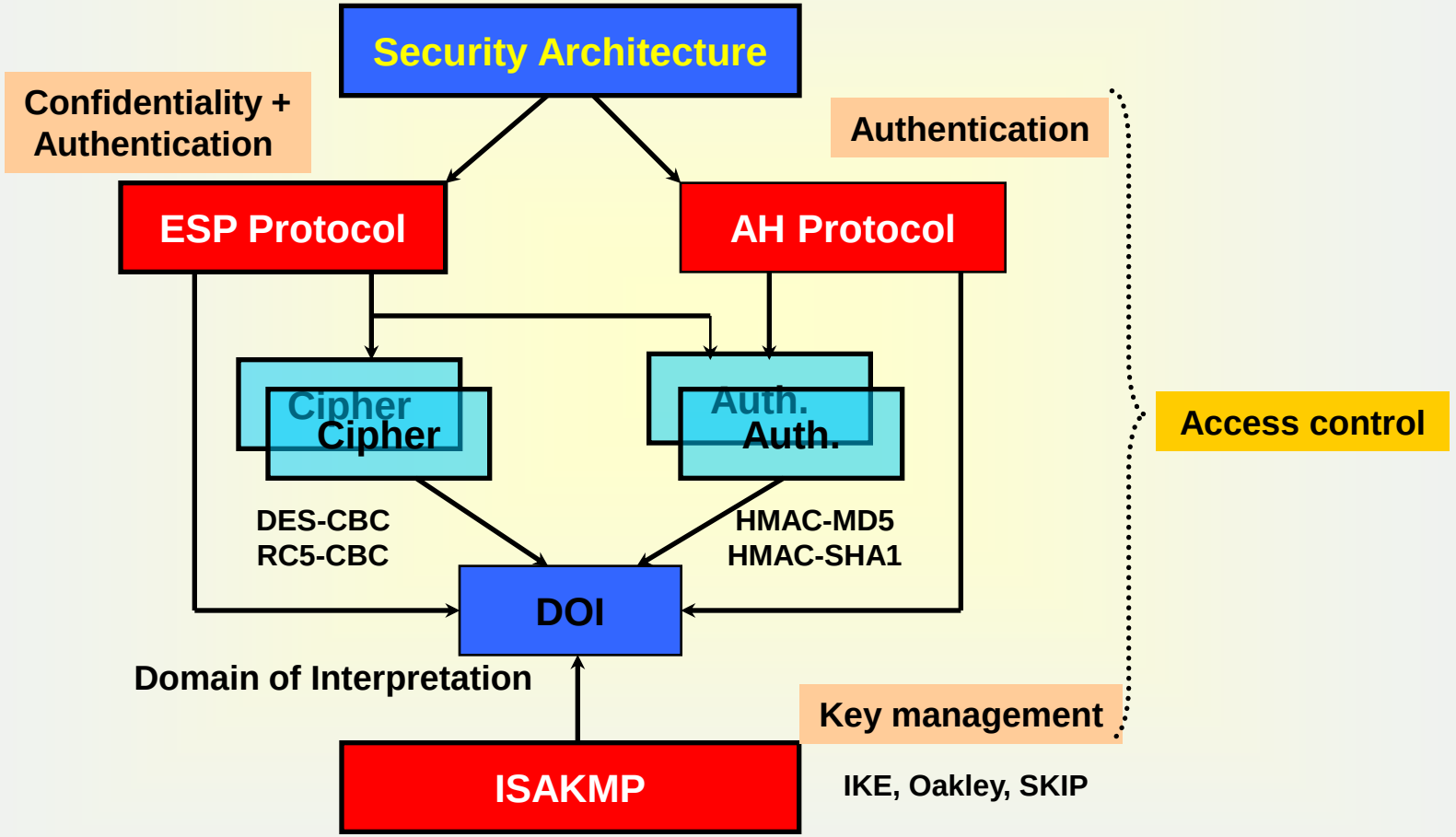
3. IP Security (IPSEC) Protocol

4. Transport Layer Security(TLS) Protocol

IPSEC Design Objectives

- **To Provide Interoperable, High quality, Crypto-based Security for IPv4 & IPv6**
- **Modularity**
  - **Designed to be algorithm-independent**
  - **Allows selection of different sets of algorithms without affecting the other part of the implementation**
- **Interoperability**
  - **Specify a standard set of default algorithms to facilitate interoperability in the global Internet**
- **Caveats (limitations)**
  - **Security offered by IPSEC ultimately depends on the implementation quality**
  - **Security offered by IPSEC critically depends on many aspects of the operating environment, e.g., OS security, Random number generators, system management protocols, etc.**

IPSEC Standard Document Roadmap



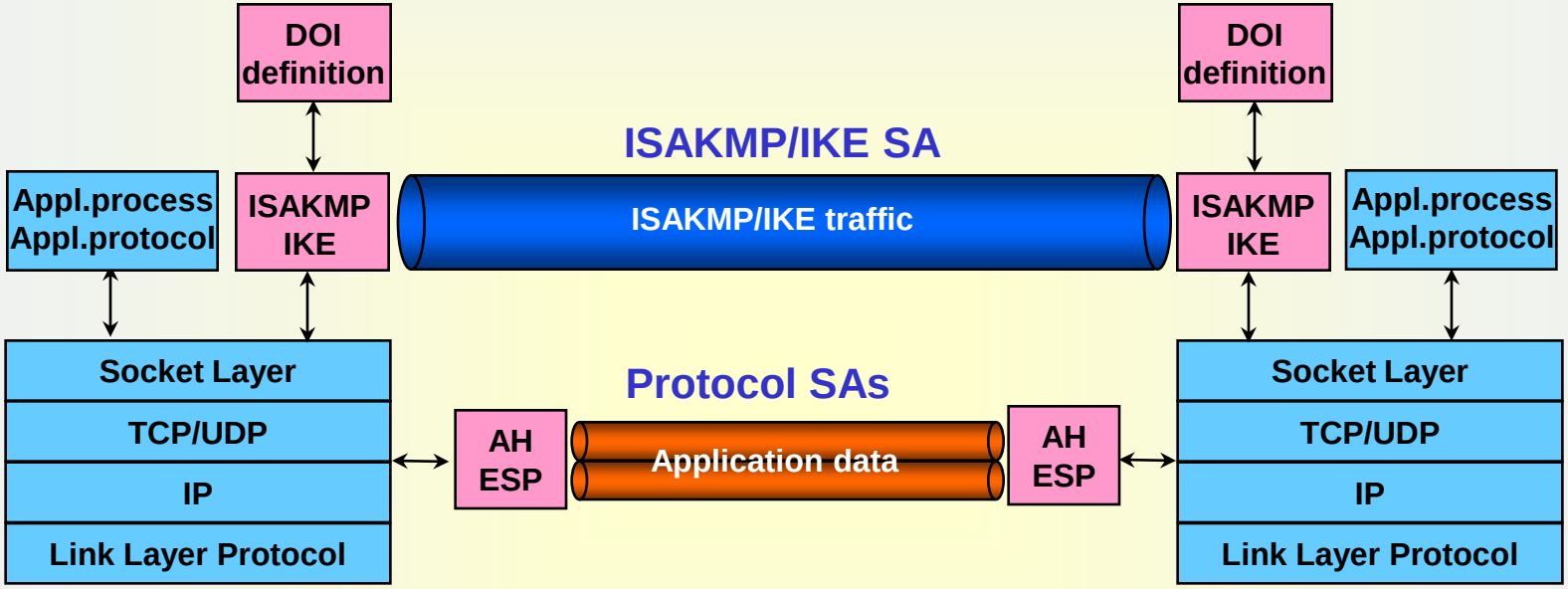
IPSEC System Overview

- **Two Security Protocols**
  - **AH primarily for authentication and optional anti-replay service**
    - ✓ **Mandatory-to-implement algorithms: HMAC-MD5, HMAC-SHA1**
  - **ESP primarily for confidentiality and optionally AH functionality (with limited protection range)**
    - ✓ **Mandatory-to-implement algorithms:**
      - – **DES-CBC (de facto: 3DES-CBC), NULL Encryption algorithm**
      - – **HMAC-MD5, HMAC-SHA1, NULL Authentication algorithm**
  - **AH & ESP are vehicles for access control**

- **Key Management**
  - **ISAKMP defines procedures and payload formats for security association (SA) / key management**
  - **Default automated SA/key management protocol for IPSEC:**
    - – **IKE (Internet Key Exchange) under IPSEC DOI**

- **Two Modes of Operations**
  - **Transport mode protects primarily upper layer protocols**
  - **Tunnel mode protects primarily tunneled IP packets**

Operations of IPSEC



Phase I (ISAKMP SA) : SA negotiation between two ISAKMP servers

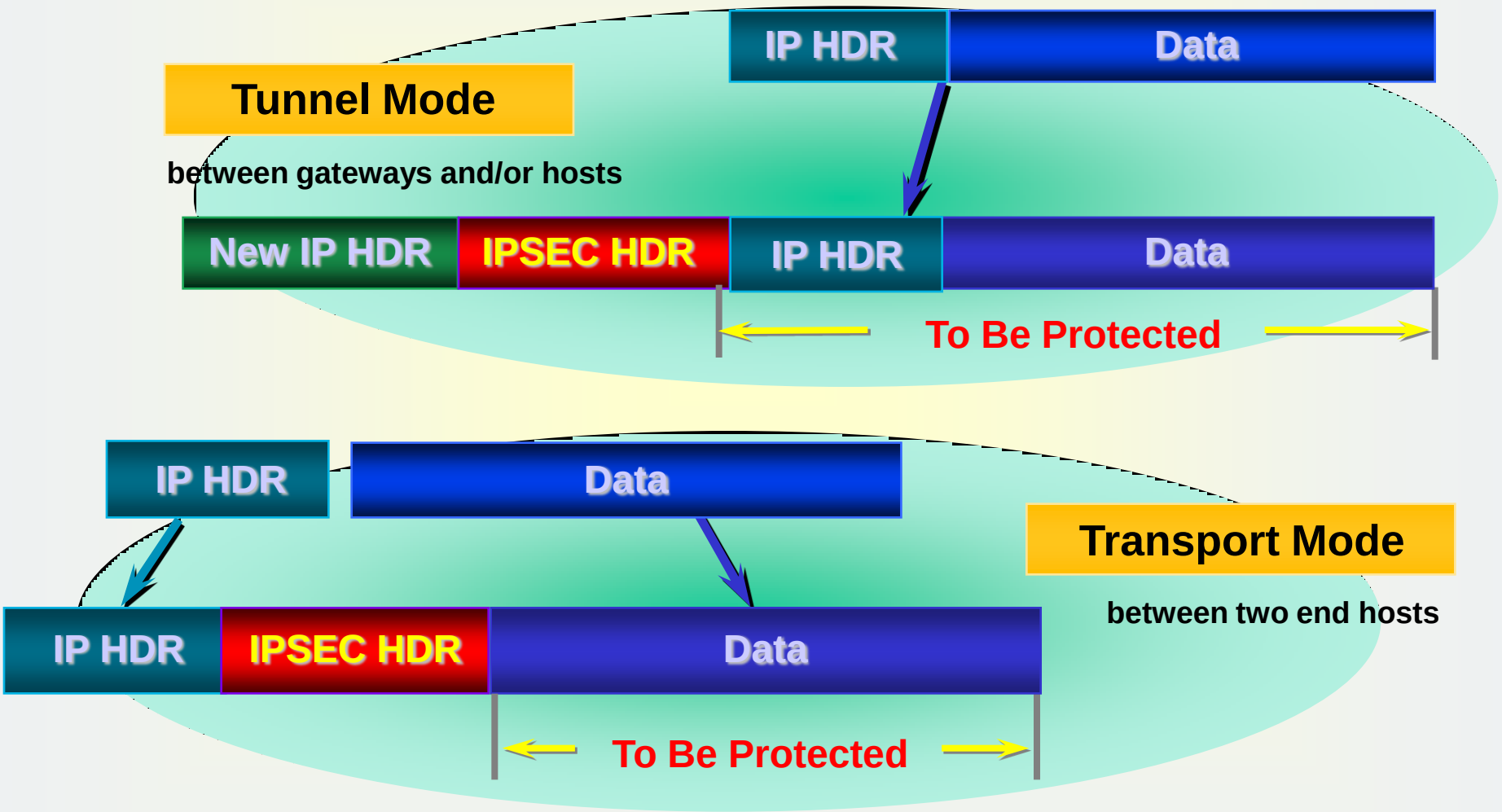
Phase II (Protocol SA) : SA negotiation for other security protocols (e.g., IPSEC AH) under the protection of ISAKMP SA

Security Association (SA) & SPI

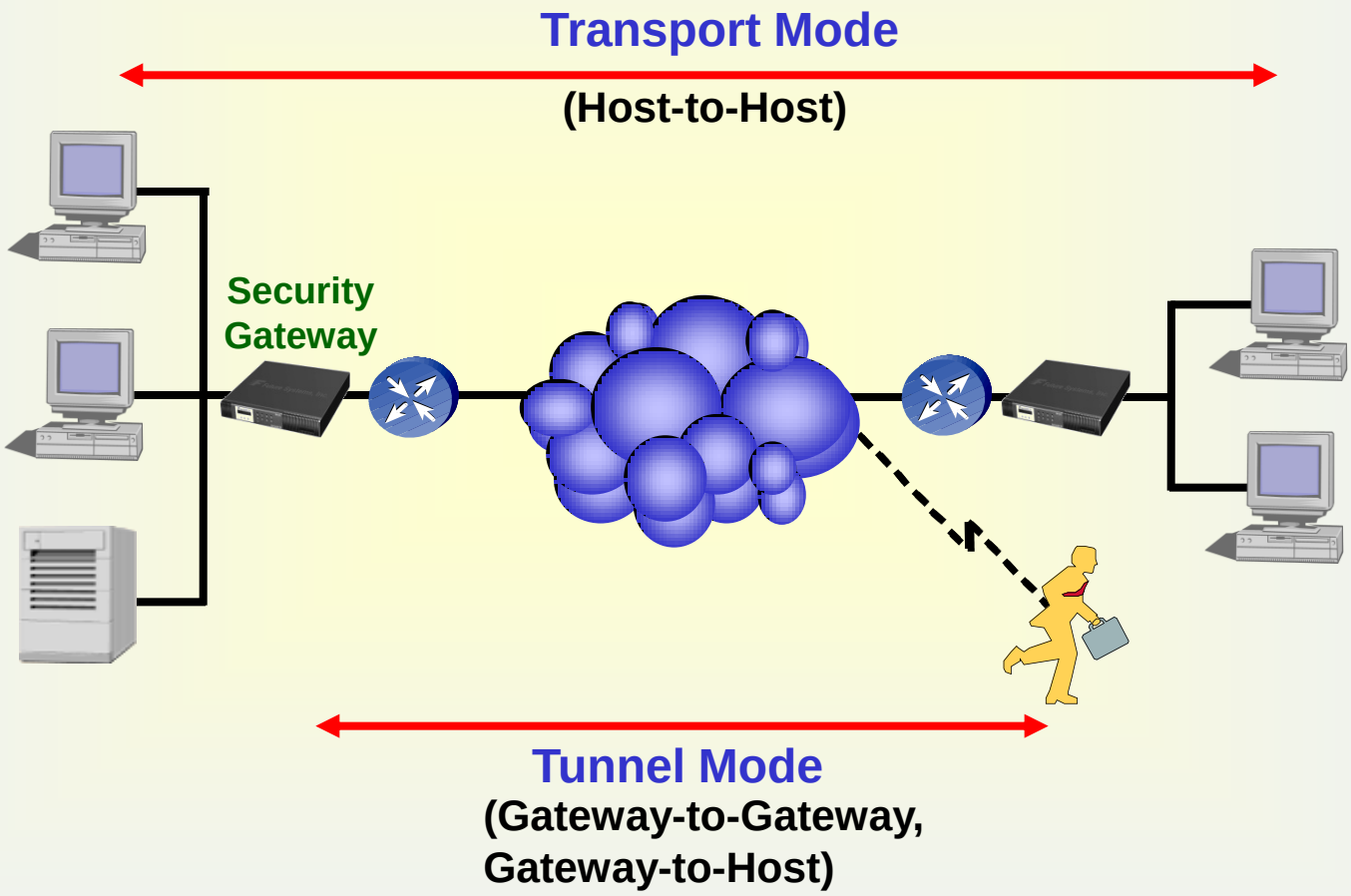
- **Security Association (SA)**
  - A set of security parameters that completely defines the security services and mechanisms to be provided by the security protocol (IKE, AH or ESP).
  - E.g., authentication/encryption algorithm, algorithm mode and secret keys, etc.
  - uniquely identified by a triple (SPI, Destination IP addr., Security protocol).
  - receiver-oriented: the SPI is selected by the destination.
  - ISAKMP/IKE SA : bidirectional (identified by a pair of (I-Cookie, R-Cookie))
  - Protocol SA : unidirectional - one for inbound and one for outbound.

- **Security Parameters Index (SPI)**
  - An identifier for a SA relative to some security protocol (IPSEC: 32 bits)
  - Each security protocol has its own “SPI-space”, and Initiator and Responder each select and exchange their own SPI during the security protocol negotiation.

IPSEC Mode of Operations



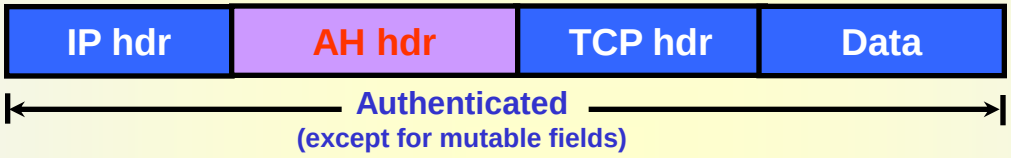
Transport Mode vs. Tunnel Mode



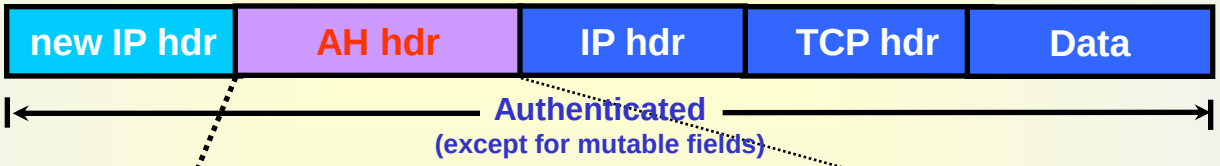
Authentication Header(AH)

Original IP Packet

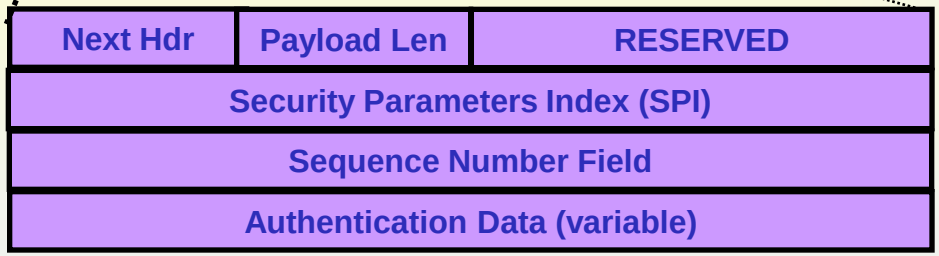
AH Transport Mode Protected Packet



AH Tunnel Mode Protected Packet

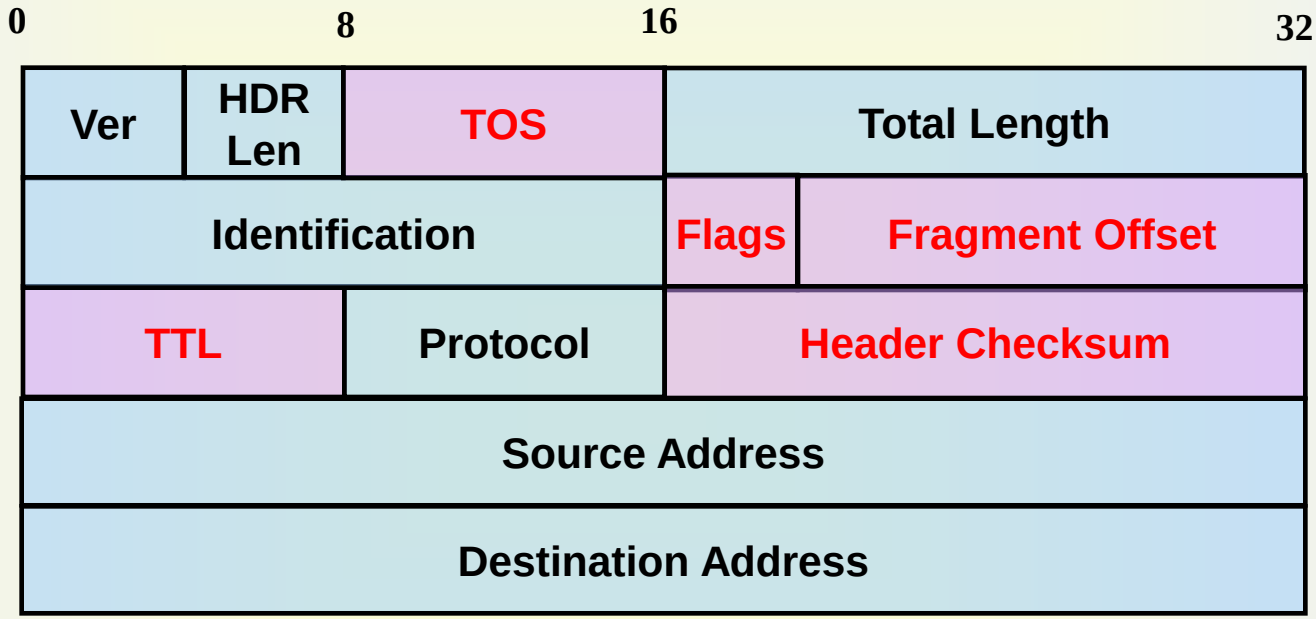


AH Header



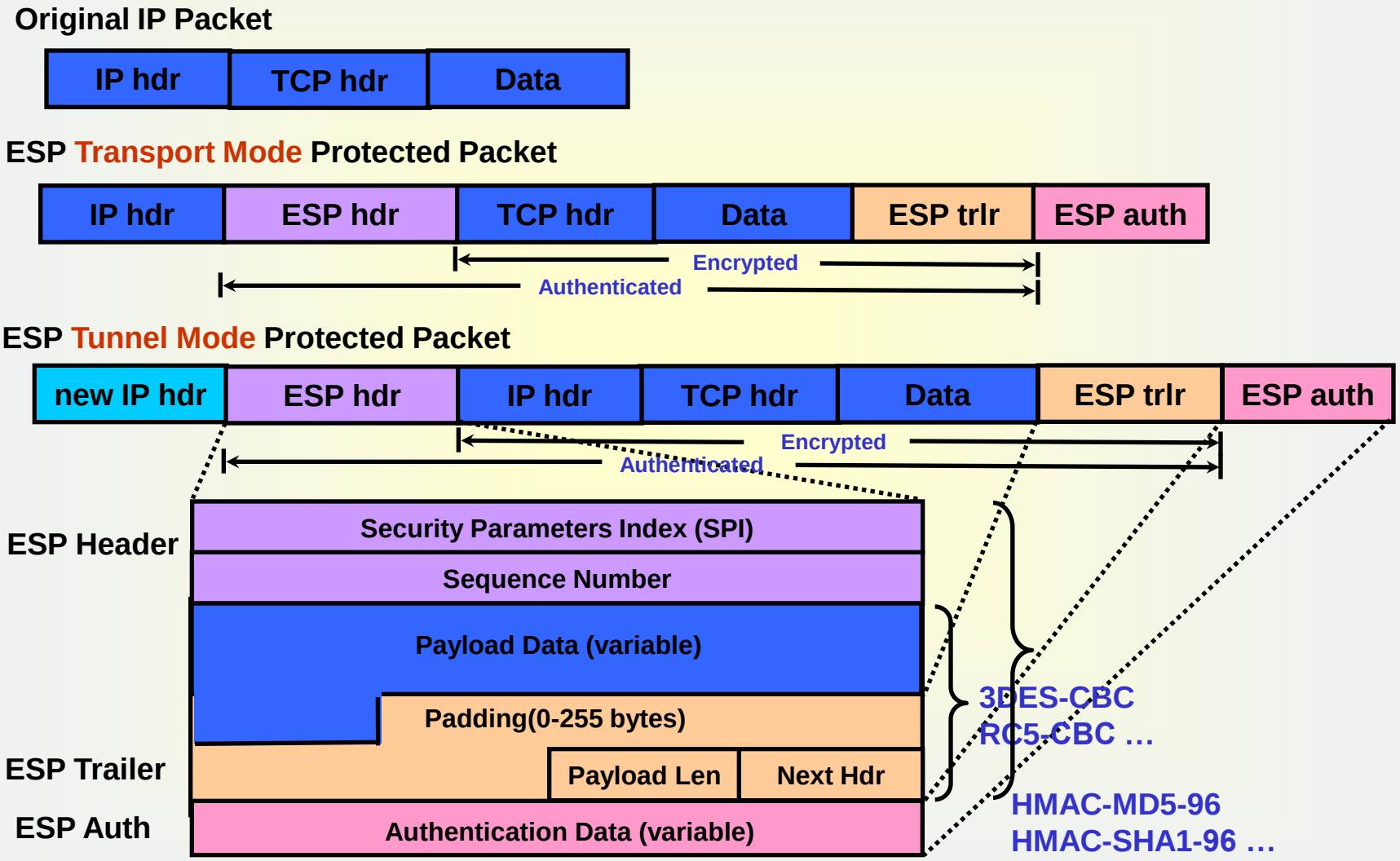
HMAC-MD5-96
HMAC-SHA1-96

Mutable Fields in Outer IP Header



These fields may be modified during transit, therefore they are not authenticated by AH

Encapsulating Security Payload(ESP)



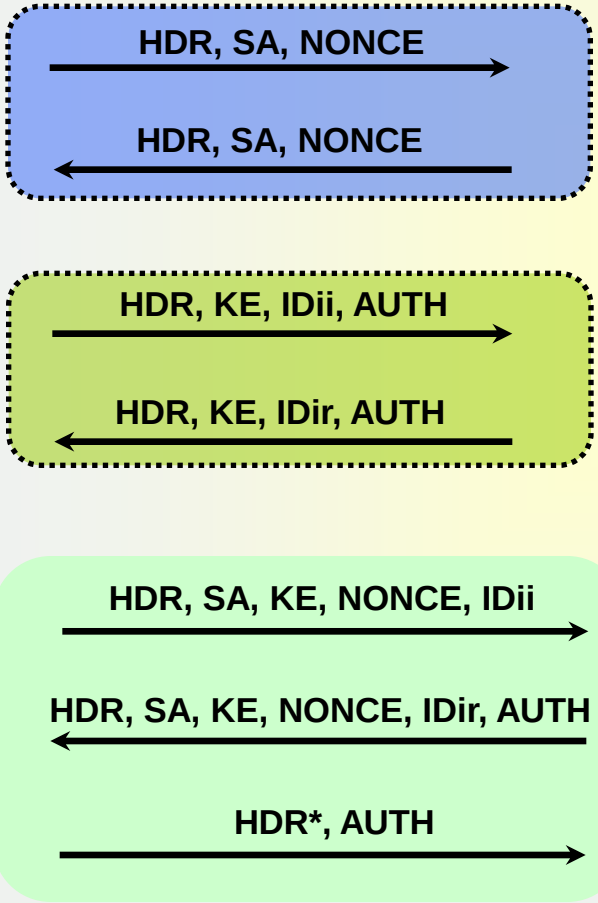
ISAKMP & IKE

- **ISAKMP:**
  - A general framework for establishing and managing SAs and keys
  - Header and payload definitions
  - Exchanges types for payload exchanges
  - General processing guidelines

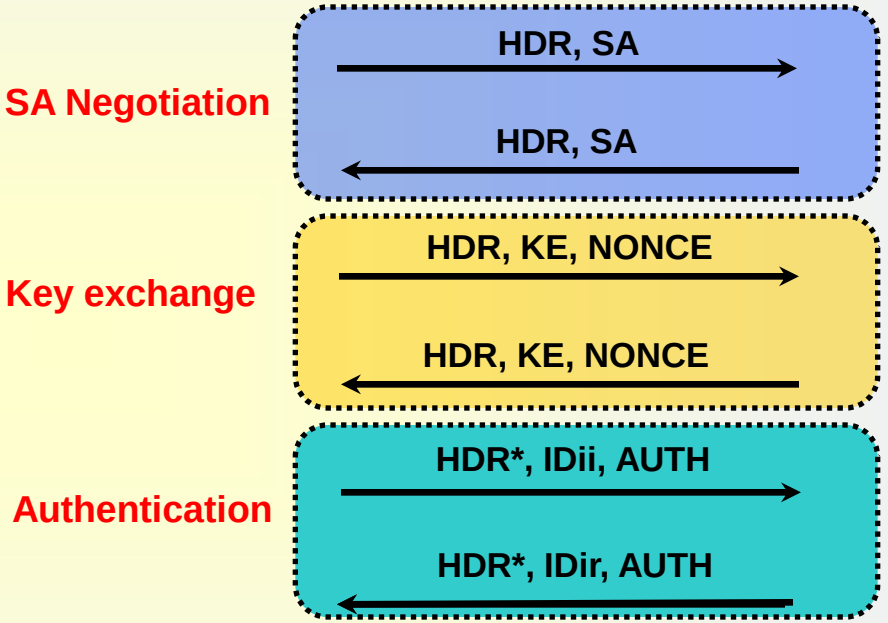
- **IKE:**
  - A hybrid protocol to negotiate keys and SAs in an authenticated and protected manner
  - An authenticated key exchange algorithm based on Diffie-Hellman, with added authentication and security features from Oakley and SKEME techniques
  - Authentication methods supported
    - ✓ Pre-shared secret
    - ✓ Digital signature
    - ✓ Public key encryption
  - Exchange types defined
    - ✓ Aggressive mode
    - ✓ Main mode

ISAKMP Exchanges

Base Exchange



Identity Protection Exchange

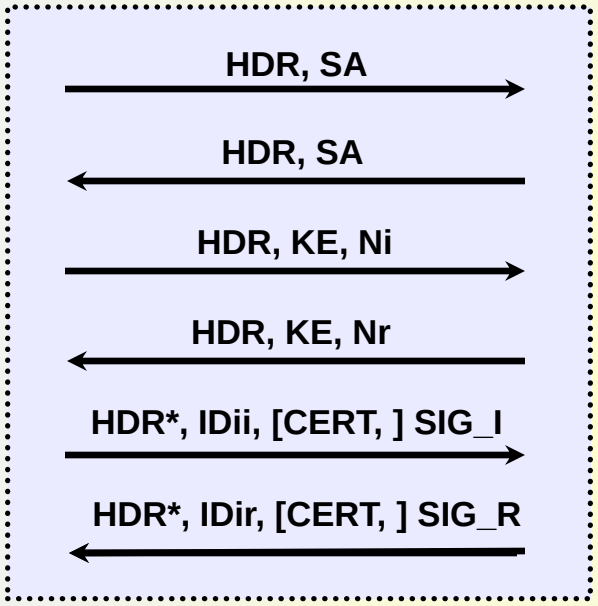


Aggressive Exchange

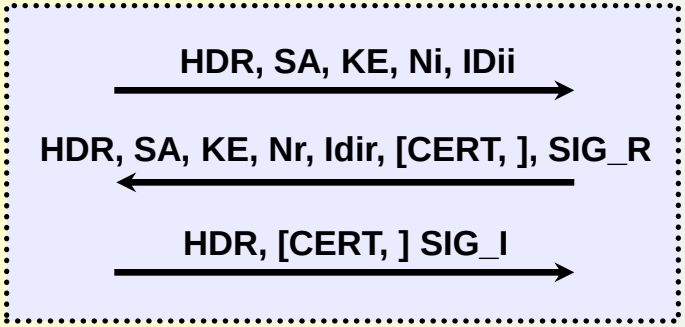
http://www.ietf.org/rfc/rfc2408.txt

IKE Phase 1 Authenticated With Signatures

Main Mode



Aggressive Mode



- **Derived SKEYIDs**

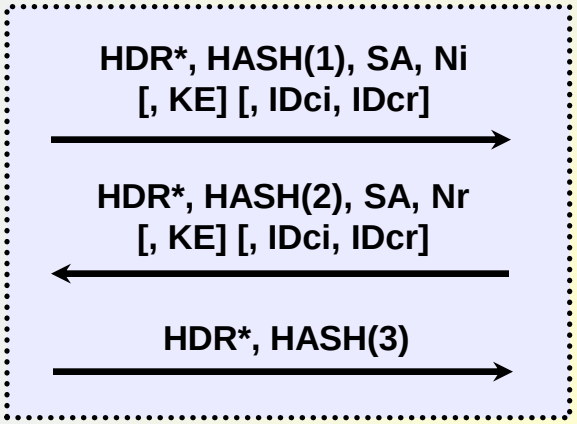
  SKEYID = prf(Ni_b | Nr_b, g^xy)

  SKEYID_d = prf(SKEYID, g^xy | CKY-I | CKY-R | 0) **for key derivation**

  SKEYID_a = prf(SKEYID, SKEYID_d | g^xy | CKY-I | CKY-R | 1) **for authentication**

  SKEYID_e = prf(SKEYID, SKEYID_a | g^xy | CKY-I | CKY-R | 2) **for encryptiosn**

IKE Phase 2 Quick Mode



HASH(1) = prf(SKEYID_a, M-ID | SA | Ni [| KE] [| IDci | IDcr])
HASH(2) = prf(SKEYID_a, M-ID | Ni_b | SA | Nr [| KE] [| IDci | IDcr])
HASH(3) = prf(SKEYID_a, 0 | M-ID | Ni_b | Nr_b)

- **KEYMAT = prf(SKEYID_d, [g(qm)^xy |] protocol | SPI | Ni_b | Nr_b)**

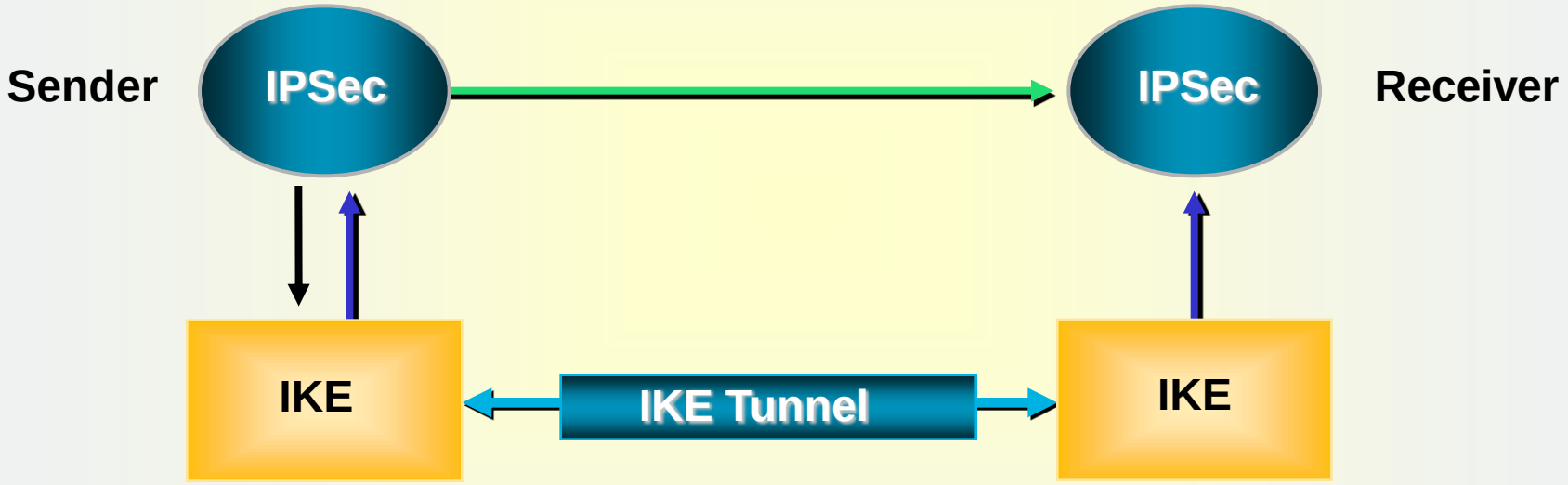
- **The identities of SAs are implicitly assumed to be the IP address of the peers.**
- **If ISAKMP is acting a client negotiator on behalf of another party, the identities of the parties MUST be passed as IDci and IDcr.**

- **A single SA negotiation results in two SAs - one inbound and one outbound: The SPI chosen by destination of SA is used to derive KEYMAT for that SA**

How IPSec Uses IKE

1. Outbound packet from Sender to Rcvr*—No IPSec SA

4. Packet Is Sent from Sender to Rcvr Protected by IPSec SA



2. Sender's IKE Begins Negotiation with Rcvrs

3. Negotiation Complete—Sender and Rcvr Now Have Complete Set of SAs in Place

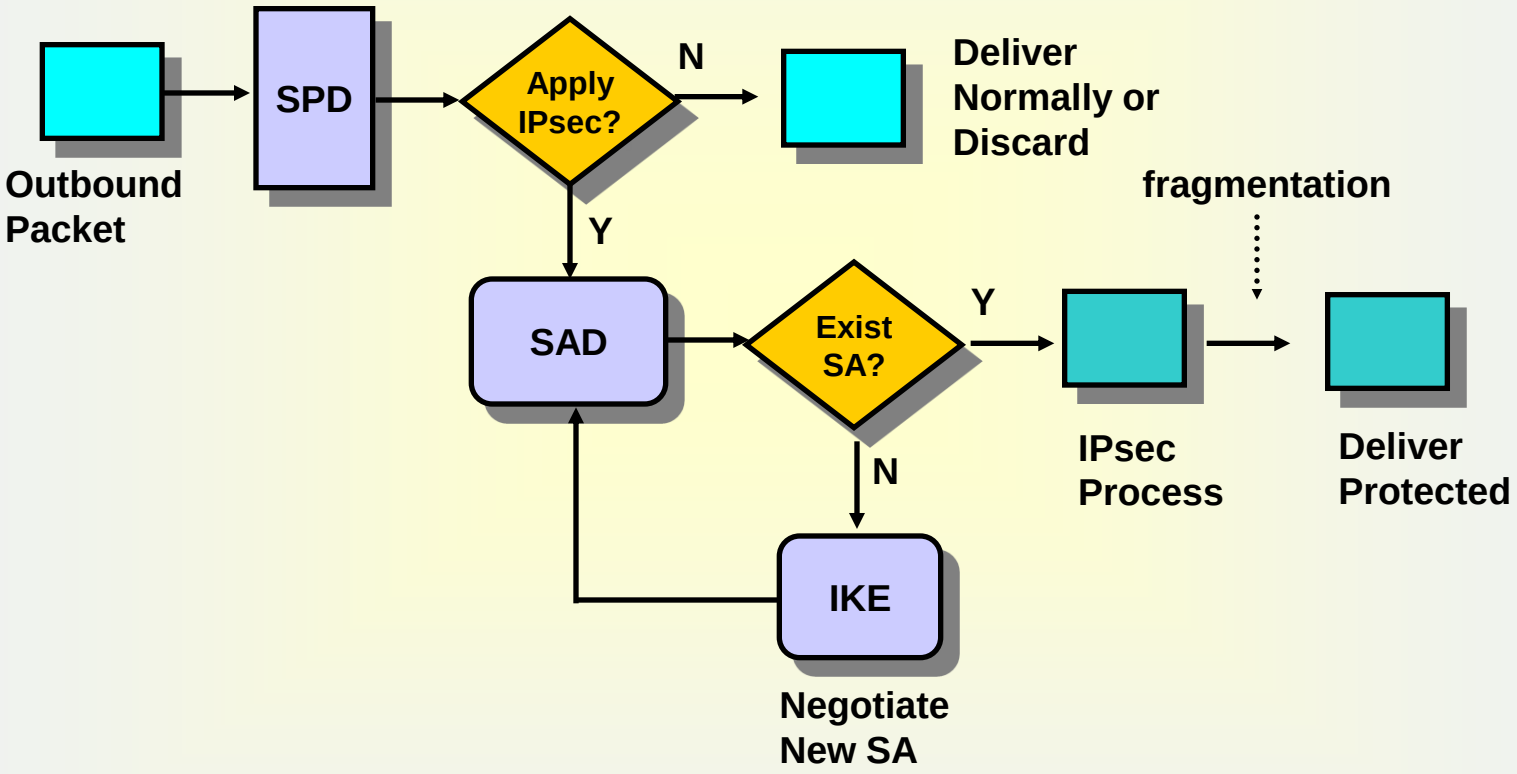
Security Policy Database (SPD)

- **Specifies what services are to be offered to datagrams and in what fashion**
  - **Three processing choices discard, bypass IPsec, apply IPsec**
  - **Used to map traffic to specific SAs or SA bundles**
  - **Must be consulted during of all traffic (inbound & outbound), including non-IPSEC traffic**
  - **SPD entries MUST be ordered and the SPD MUST always be searched in the same order**
- **Selectors**
  - **A set of IP and upper layer protocol field values that is used by SPD to map traffic to SA**
  - **Selector parameters**
    - ✓ Source/Destination IP Address
    - ✓ Name
    - ✓ Data sensitivity level
    - ✓ Transport Layer Protocol
    - ✓ Source and Destination Ports

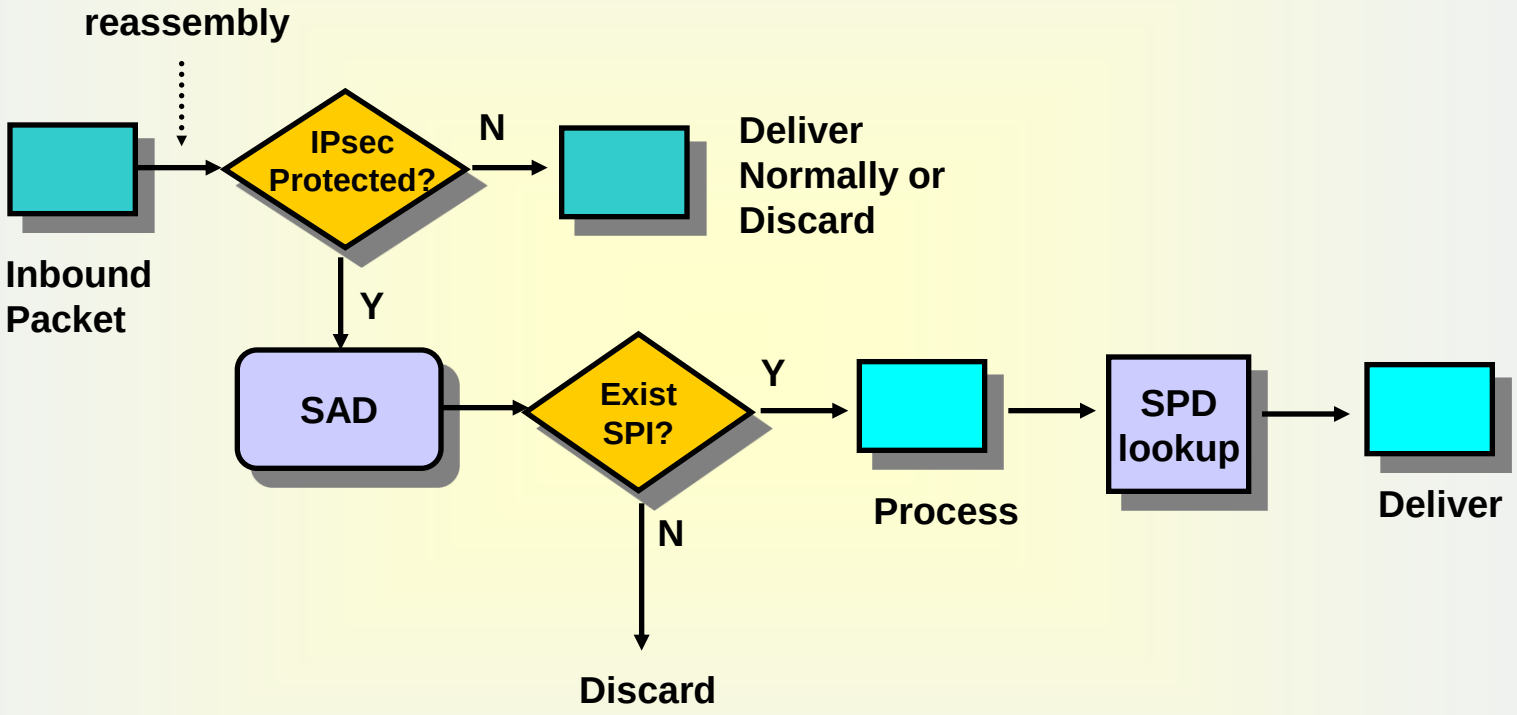
Security Association Database (SAD)

- SAD entry defines the parameters associated with one SA
  - For outbound processing, entries in SPD points to SAD entries
  - For inbound processing, triplet (SPI, IPSEC protocol, outer header's destination IP address) uniquely determines the SA
- If SPD entry does not currently point to an SA that is appropriate for the packet, the implementation creates an appropriates SA, and links SPD entry to SAD entry
- SAD fields
  - Sequence number counter / overflow, Anti-replay window
  - AH authentication algorithm, keys
  - ESP encryption algorithm, keys & IV
  - ESP authentication algorithm, keys
  - SA lifetime
  - IPSEC mode
  - Path MTU

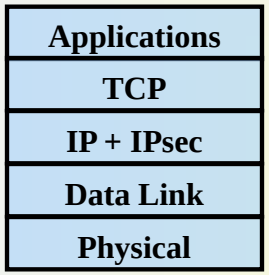
Outbound IPsec Processing

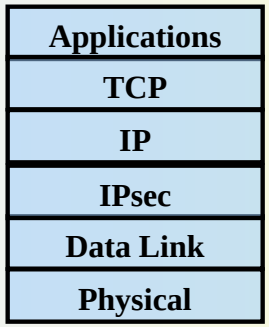


Inbound IPsec Processing



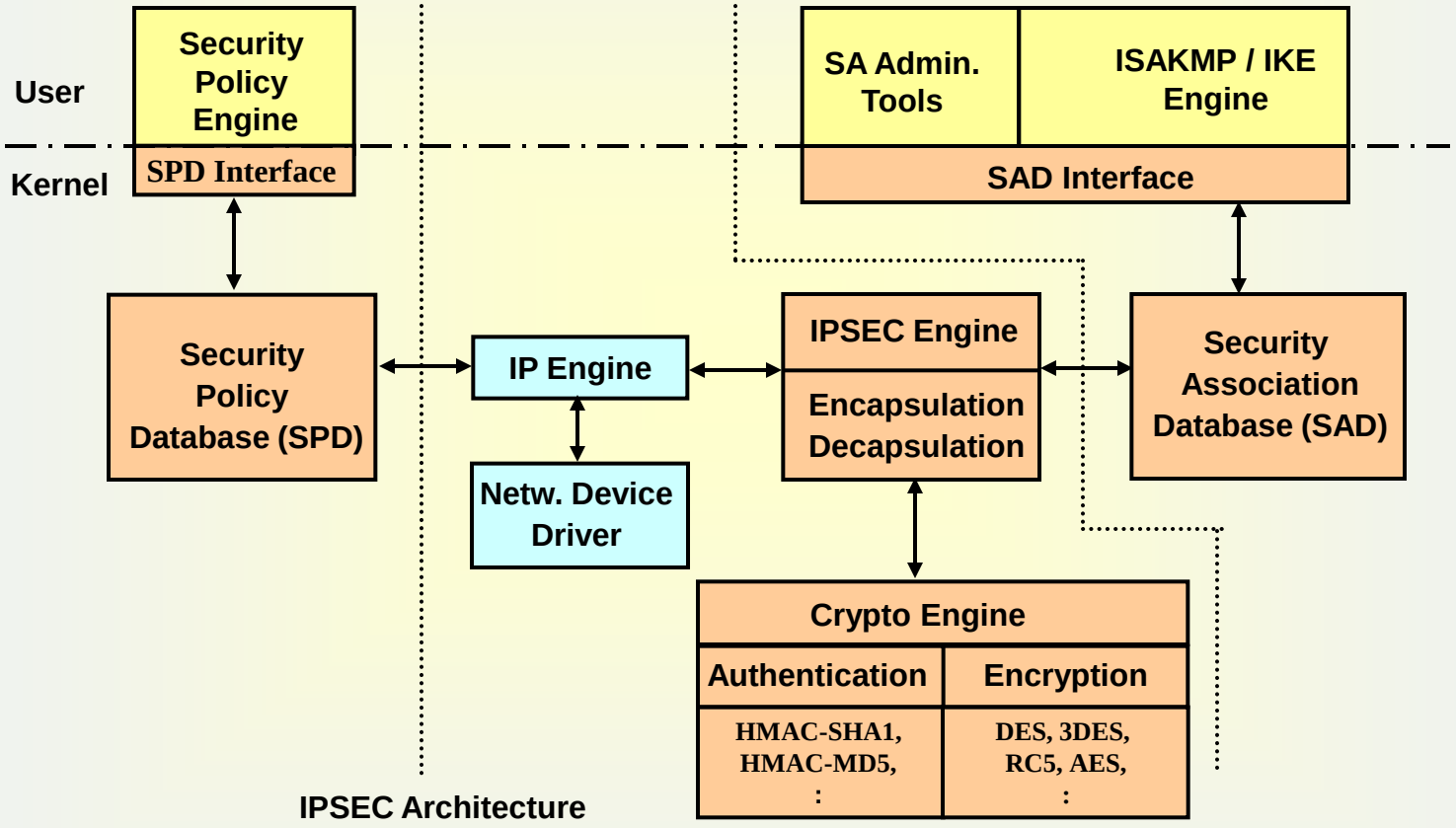
IPSEC Implementation



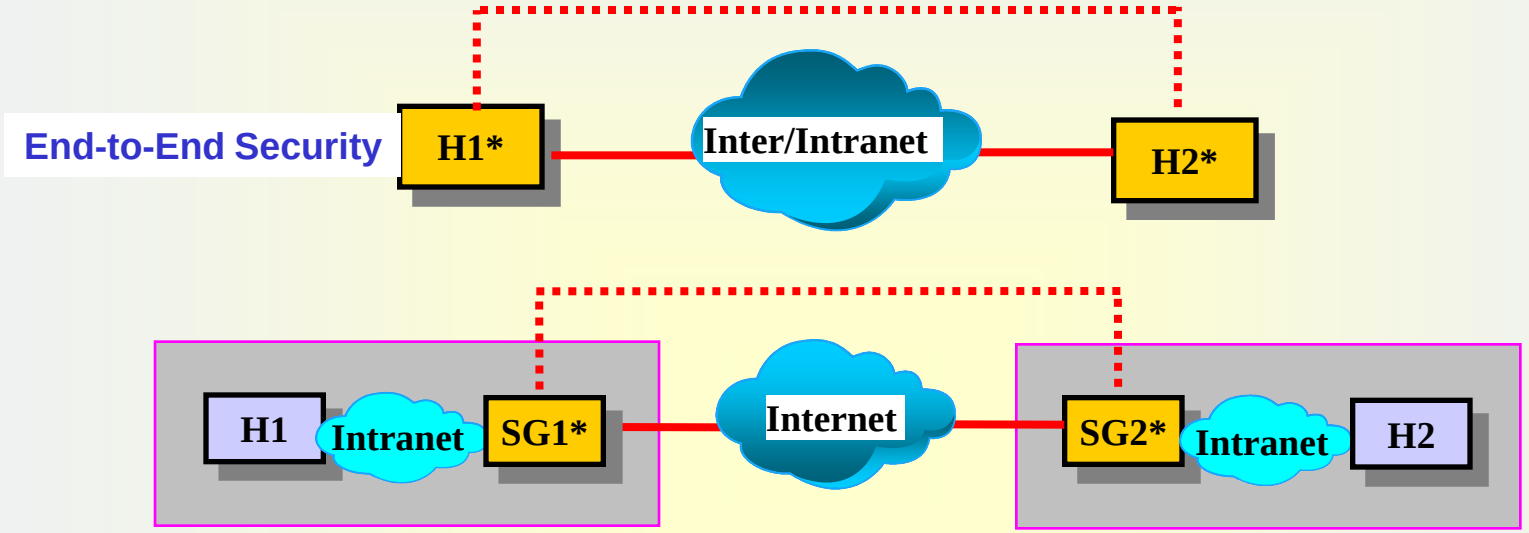


- **Integration into native IP implementation**
  - ✓ Requires access to IP source code
  - ✓ implemented in host or gateway
- **Bump-in-the-stack (BITS)**
  - ✓Between IP and local network driver
  - ✓source code access for the IP stack not required
  - ✓usually implemented in host
- **Bump-in-the-wire (BITW)**
  - ✓outboard crypto processor, serve host or gateway
  - ✓usually IP addressable
  - ✓analogous to BITS in supporting host
  - ✓like a security gateway in supporting router or firewall

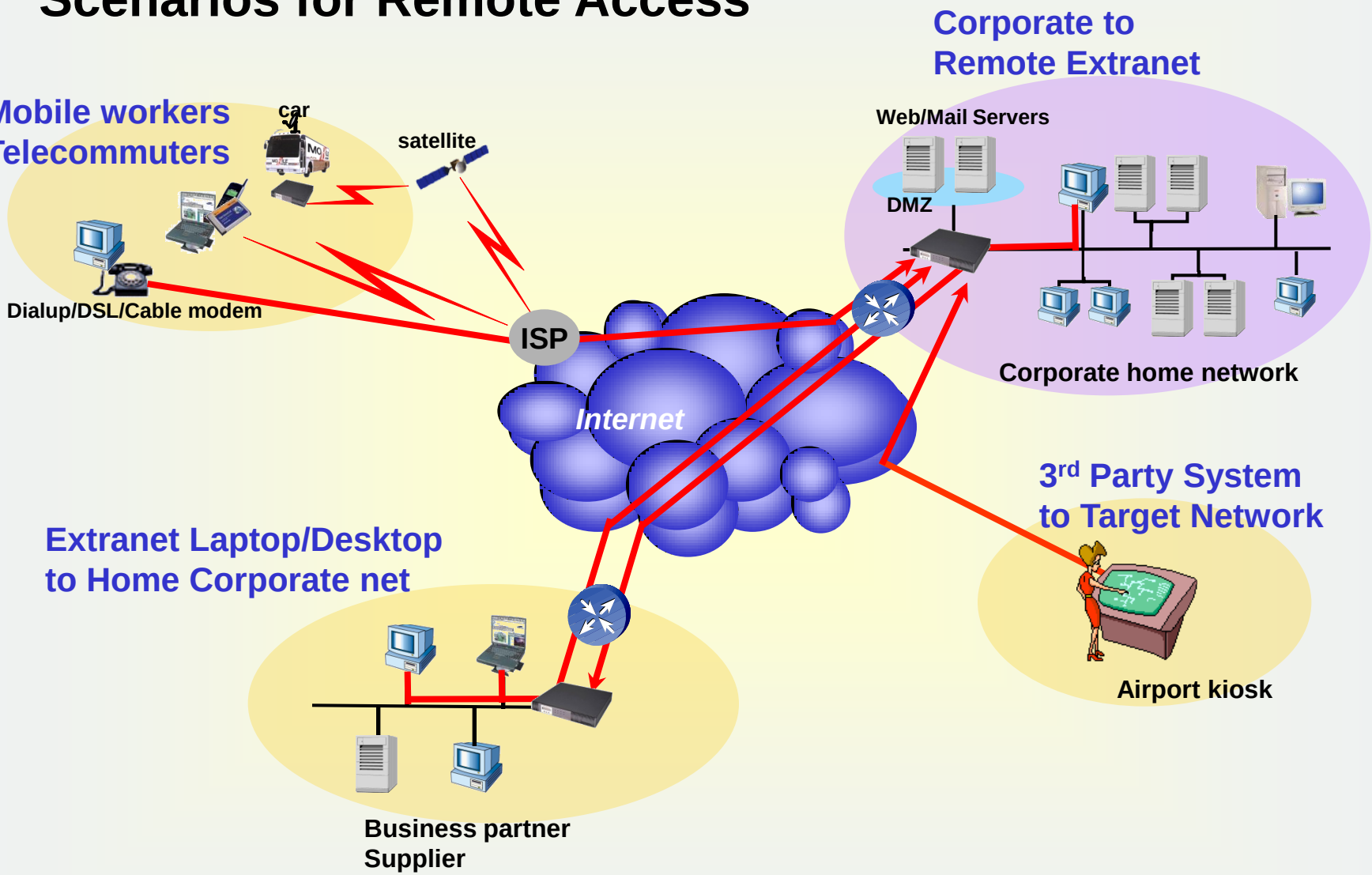
IPSEC Implementation Model : An Example



Typical IPSEC Application Scenarios



Scenarios for Remote Access



IPSEC VPNs, Firewalls & NATs

- **VPNs simplify firewall policy**
  - **Open holes for IPSEC traffic based on authenticated users/hosts**
  - **Open UDP port 500 for IKE**
  - **Open IP protocols 50 (for ESP) and 51 (for AH)**
- **VPNs conflict firewall policy**
  - **Encrypted/tunneled contents invisible to firewalls for filtering and inspection**
  - **May need additional firewall(s) outside tunnels to handle tunneled traffic**
- **IPSEC VPN cannot pass through NAT/NAPT**
  - **Problem raised in true end-to-end VPN**
  - **Standardization of NAT traversal under way**

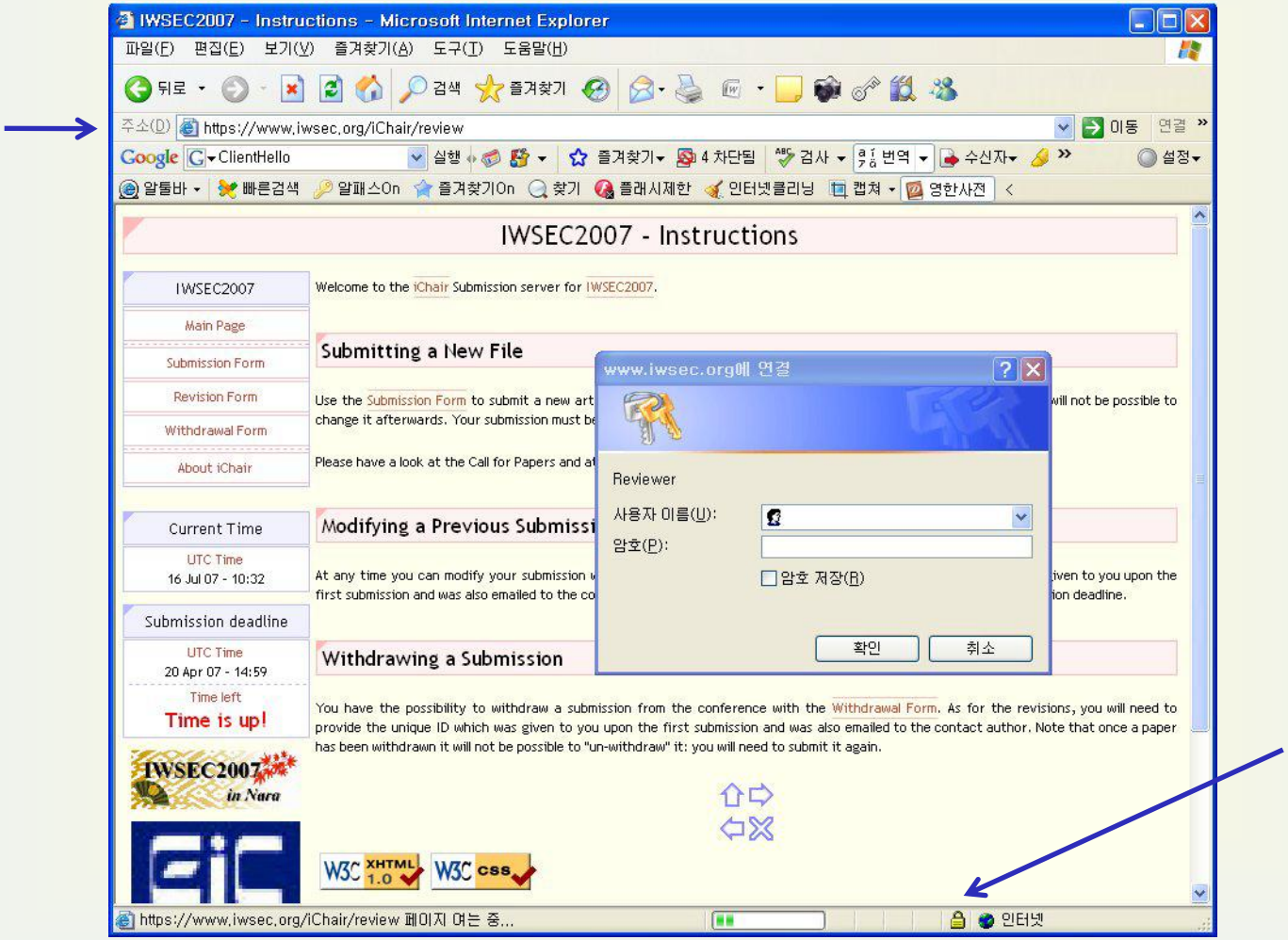
1. Introduction

2. Virtual Private Networks(VPNs)

3. IP Security (IPSEC) Protocol

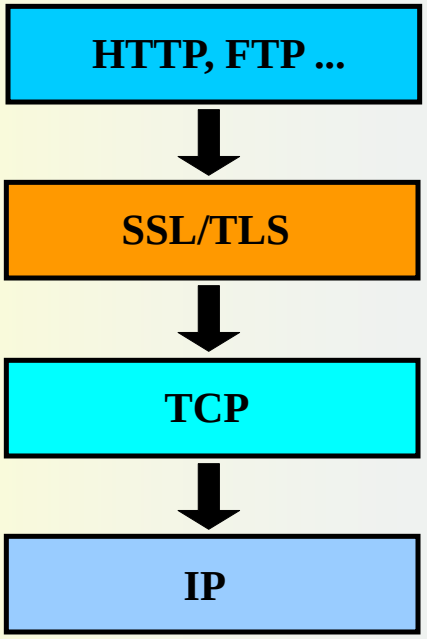
4. Transport Layer Security(TLS) Protocol

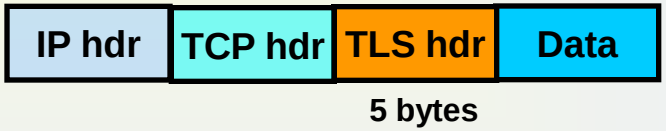
Transport Layer Security (TLS) Protocol



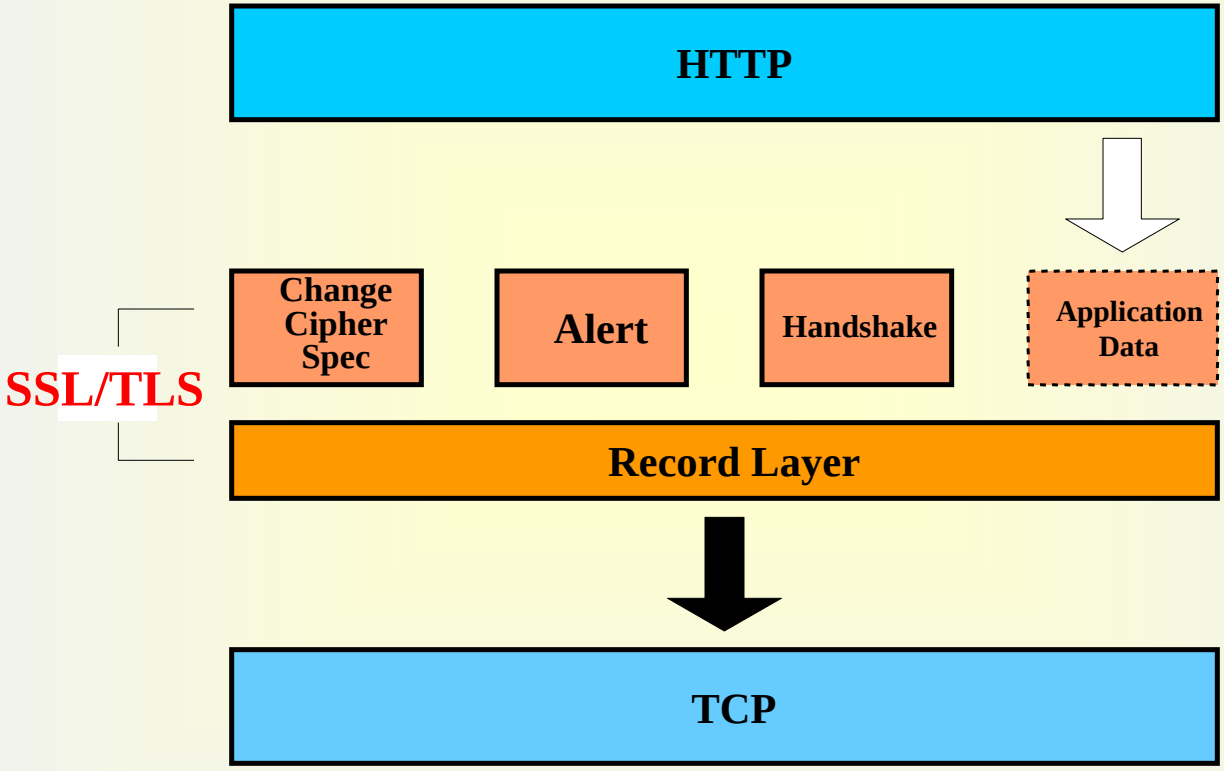
Transport Layer Security (TLS) Protocol

- **SSL/TLS**
  - Layered on top of reliable transport protocols, e.g., TCP
  - Application protocol independent
  - Record Protocol & Handshake Protocol
- **Record Protocol**
  - Encapsulation of higher level protocols
  - Data encryption using CBC block ciphers or stream ciphers
  - Data integrity using HMAC
- **Handshake Protocol**
  - Security parameter negotiation: keys & algorithms
  - Entity authentication using public key cryptography (RSA, DSS; static DH)
  - Key exchange & verification (RSA key transport, DH key exchange)

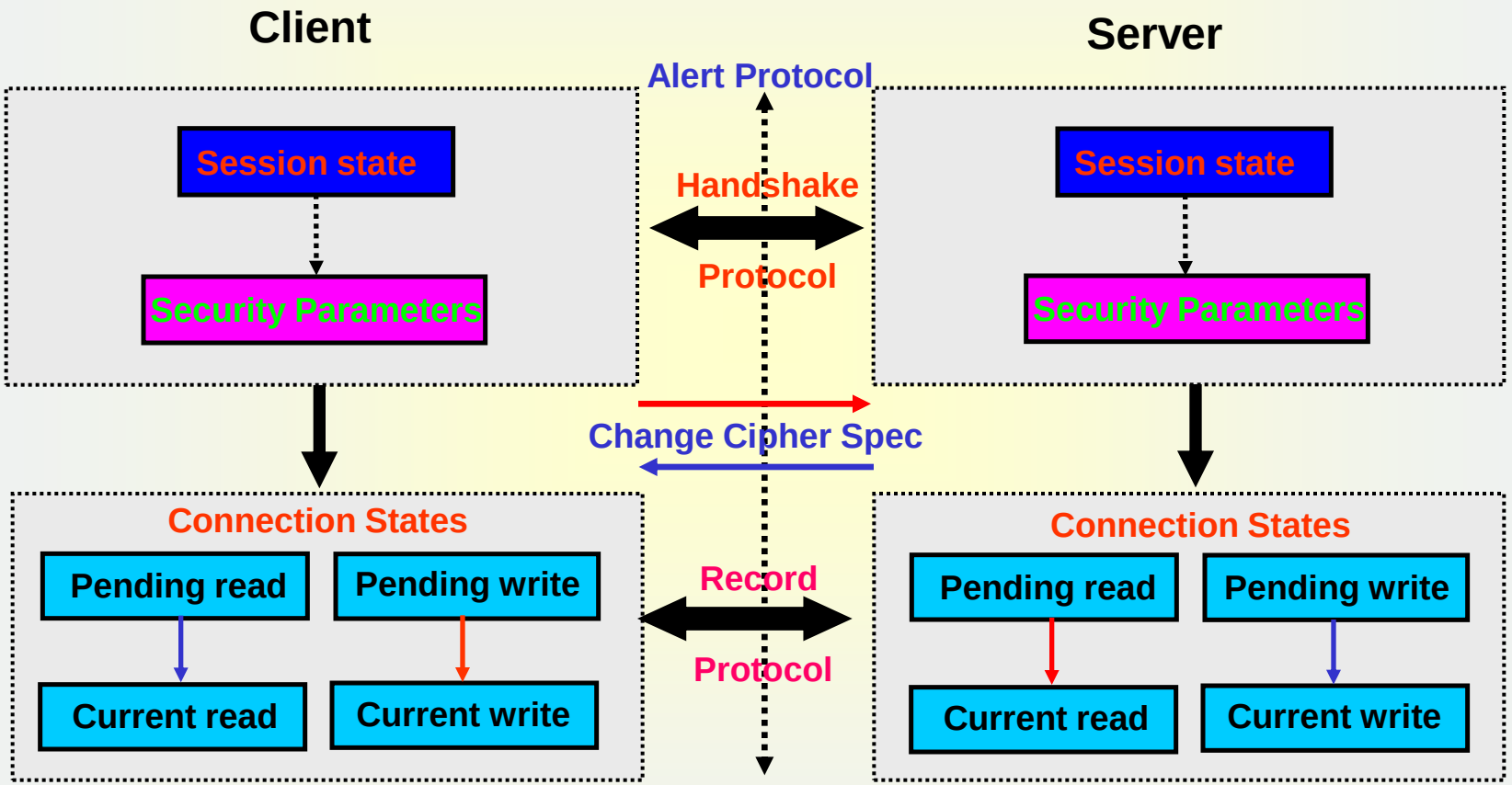




SSL/TLS Layering



SSL/TLS Operations Overview



TLS Session State

- used to create security parameters for use by the Record Layer (TLS)

- Session state consists of
  - session identifier : An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
  - peer certificate: X509v3 [X509] certificate of the peer. may be null.
  - compression method : The algorithm used to compress data prior to encryption.
  - cipher spec : Specifies the bulk data encryption algorithm (such as null, DES, etc.) and a MAC algorithm (such as MD5 or SHA).
  - master secret : 48-byte secret shared between the client and server.
  - is resumable : A flag indicating whether the session can be used to initiate new connections.

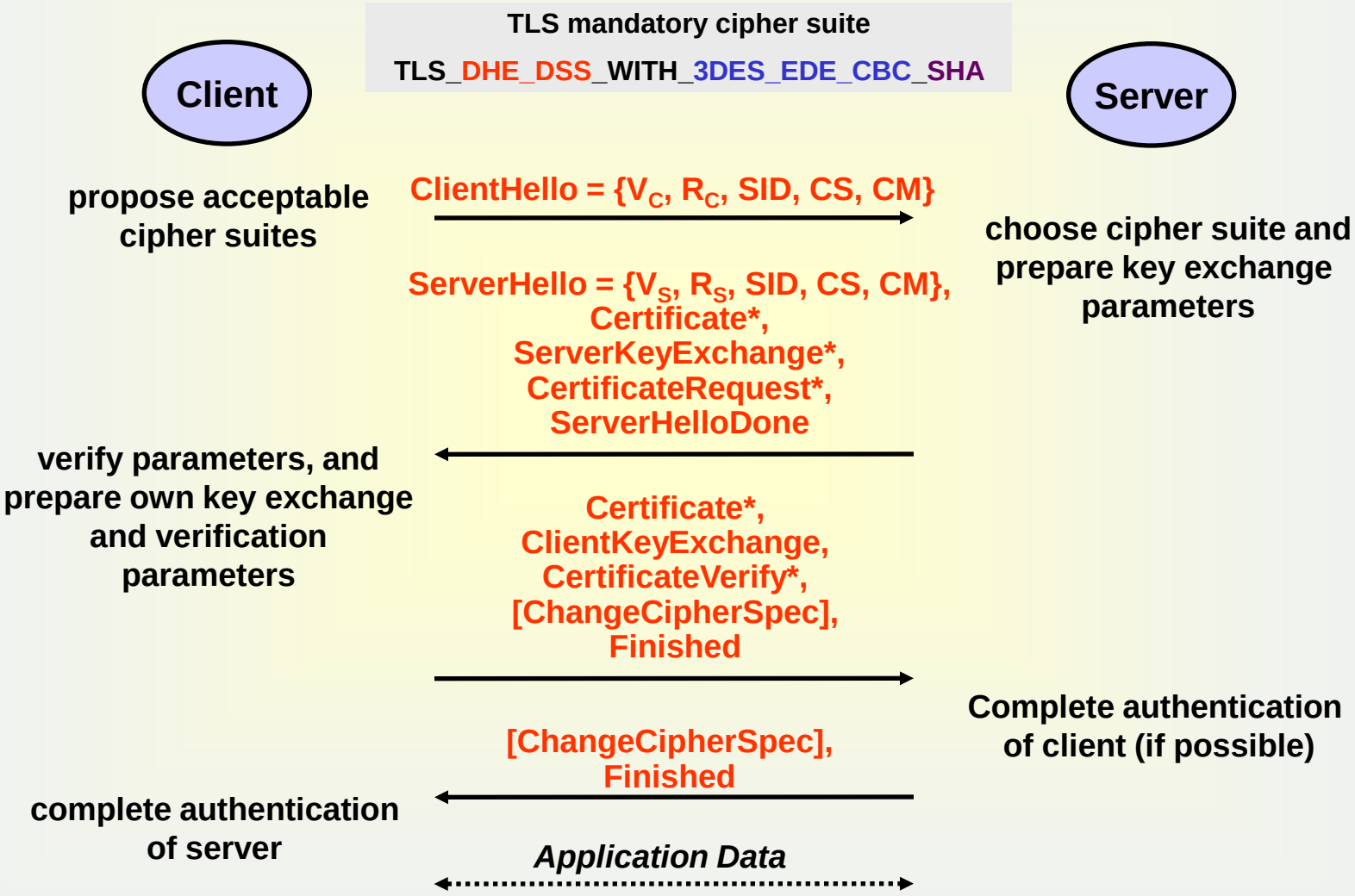
TLS Security Parameters

```
struct {
        ConnectionEnd              entity;                  /* server, client */
        BulkCipherAlgorithm        bulk_cipher_algorithm;   /* NULL, rc4, rc2, des, 3des, des40 */
        CipherType                 cipher_type;             /* stream, block */
        uint8                      IV_size;                 /* IV size for block ciphers */
CipherSpec
        uint8                      key_material_length;     /* write key size */
        IsExportable               is_exportable;           /* true, false */
        MACAlgorithm               mac_algorithm;           /* NULL, md5, sha */
        uint8                      hash_size;               /* MAC secret size */
        CompressionMethod          compression_algorithm;   /* NULL */
        opaque                     master_secret[48];       /* 48 byte master secret */
        opaque                     client_random[32];       /* Random from ClientHello */
        opaque                     server_random[32];       /* Random from ServerHello */
} SecurityParameters;
```

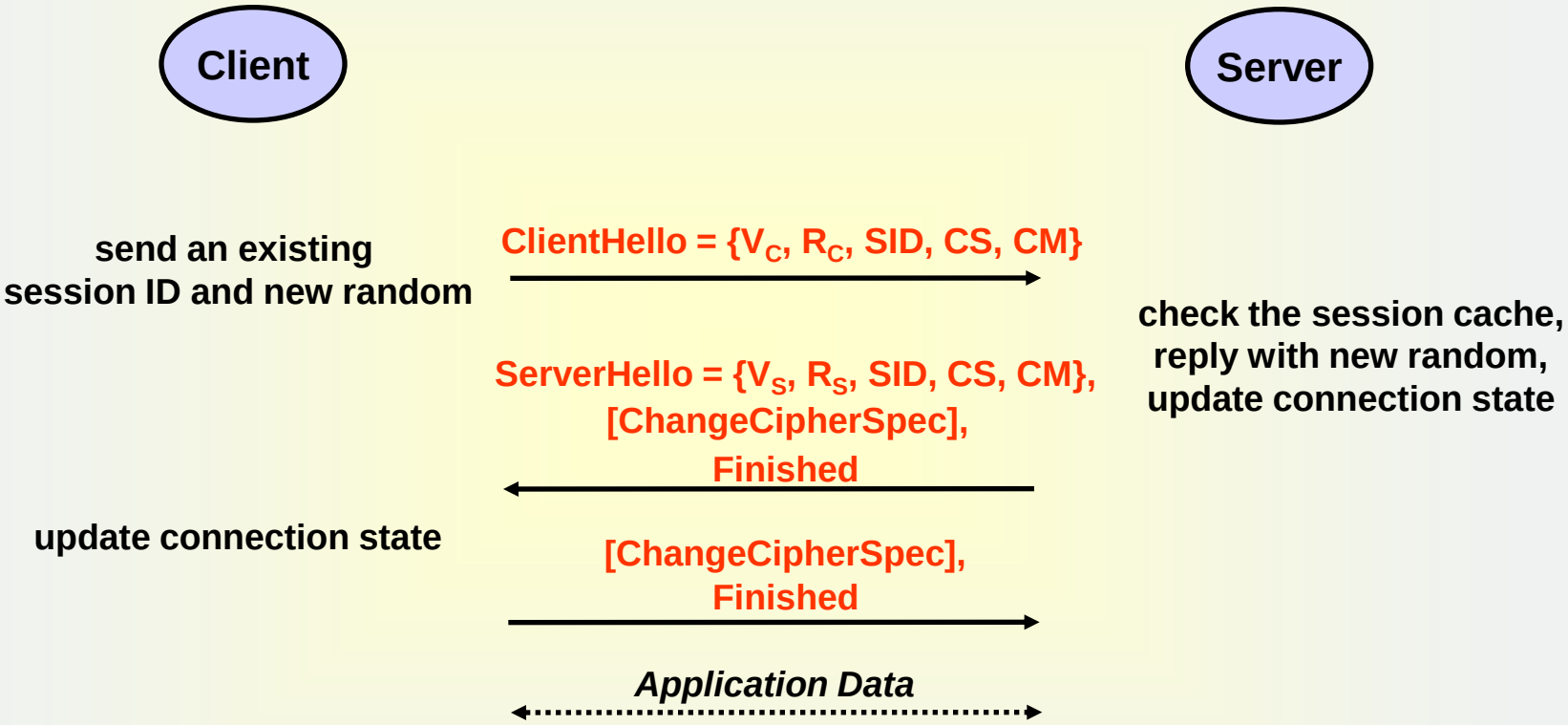
TLS Connection States

- Direction-dependent operating environment of TLS Record Protocol
  - pending (read/write) states: derived from SecurityParameters set by TLS Handshake Protocol
  - current (read/write) states: operating states at TLS Record Protocol
  - ChangeCipherSpec : pending states → current states (pending states initialized with empty state)
  - initial current states: NULL (no encryption, compression or MAC)
- TLS Connection State
  - compression state: The current state of the compression algorithm.
  - cipher state : The current state of the encryption algorithm.
    - Scheduled key for that connection + CBC initialization vectors or Stream state info
  - MAC secret : The MAC secret for this connection.
  - sequence number : of type uint64
    - maintained separately for read and write states.
    - set to zero whenever a connection state is made active and incremented after each record.

TLS Full Handshake



TLS Abbreviated Handshake



TLS Handshake Protocol Messages

```
enum {
        hello_request(0), client_hello(1), server_hello(2), certificate(11), server_key_exchange (12),
        certificate_request(13), server_hello_done(14), certificate_verify(15),
        client_key_exchange(16), finished(20), (255)
      } HandshakeType;

struct {
        HandshakeType msg_type;          /* handshake type */
        uint24 length;                   /* bytes in message */
        select (HandshakeType) {
            case hello_request:          HelloRequest;
            case client_hello:           ClientHello;
            case server_hello:           ServerHello;
            case certificate:            Certificate;
            case server_key_exchange:    ServerKeyExchange;
            case certificate_request:    CertificateRequest;
            case server_hello_done:      ServerHelloDone;
            case certificate_verify:     CertificateVerify;
            case client_key_exchange:    ClientKeyExchange;
            case finished:               Finished;
           } body;
      } Handshake;
```

Client Hello / Server Hello

```
struct {
        uint8       major, minor;
      } ProtocolVersion;              /* SSL: {3,0}, TLS: {3, 1} */

struct {
         uint32    gmt_unix_time;
         opaque    random_bytes[28];
       } Random;
opaque            SessionID<0..32>;

 uint8            CipherSuite[2];       /* Cryptographic suite selector */

enum { null(0), (255) } CompressionMethod;
```

TLS mandatory cipher suite

TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA

```
struct {
        ProtocolVersion          client_version;
        Random                   random;
        SessionID                session_id;
        CipherSuite              cipher_suites<2..2^16-1>;
        CompressionMethod        compression_methods<1..2^8-1>;
      } ClientHello;
```

TLS Cipher Suites - RSA Key Exchange

CipherSuite	Key Exchange	Cipher	Hash
TLS_NULL_WITH_NULL_NULL	* NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	* RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	* RSA	NULL	SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5	* RSA_EXPORT	RC4_40	MD5
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	* RSA_EXPORT	RC2_CBC_40	MD5
TLS_RSA_WITH_IDEA_CBC_SHA	RSA	IDEA_CBC	SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA	* RSA_EXPORT	DES40_CBC	SHA
TLS_RSA_WITH_DES_CBC_SHA	RSA	DES_CBC	SHA

TLS Cipher Suites - DH Key Exchange

CipherSuite	Key Exchange	Cipher	Hash
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA	* DH_DSS_EXPORT	DES40_CBC	SHA
TLS_DH_DSS_WITH_DES_CBC_SHA	DH_DSS	DES_CBC	SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	DH_DSS	3DES_EDE_CBC	SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA	* DH_RSA_EXPORT	DES40_CBC	SHA
TLS_DH_RSA_WITH_DES_CBC_SHA	DH_RSA	DES_CBC	SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA	DH_RSA	3DES_EDE_CBC	SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	* DHE_DSS_EXPORT	DES40_CBC	SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA	DHE_DSS	DES_CBC	SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DHE_DSS	3DES_EDE_CBC	SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA	* DHE_RSA_EXPORT	DES40_CBC	SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES_CBC	SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	DHE_RSA	3DES_EDE_CBC	SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5	* DH_anon_EXPORT	RC4_40	MD5
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH_anon	DES40_CBC	SHA
TLS_DH_anon_WITH_DES_CBC_SHA	DH_anon	DES_CBC	SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA

Certificate

```
opaque    ASN.1Cert<1..2^24-1>;
```

```
struct {
        ASN.1Cert    certificate_list<0..2^24-1>;
    } Certificate;
```

Key Exchange Algorithm	Certificate Key Type
RSA	RSA public key for encryption
RSA_EXPORT	RSA public key (≥ 512bits for signing, ≤ 512 bits for either encryption or signing)
DHE_DSS	DSS public key.
DHE_DSS_EXPORT	DSS public key.
DHE_RSA	RSA public key for signing
DHE_RSA_EXPORT	RSA public key for signing
DH_DSS	Diffie-Hellman key signed by CA using DSS.
DH_RSA	Diffie-Hellman key signed by CA using RSA

Server Key Exchange

- sent only when the server certificate is not enough for exchange of a pre-master secret:
  - RSA_EXPORT (RSA signing key > 512bits)
  - DHE_DSS, DHE_DSS_EXPORT, DHE_RSA, DHE_RSA_EXPORT, DH_anon

```
enum { rsa, diffie_hellman } KeyExchangeAlgorithm;

struct {
      opaque rsa_modulus<1..2^16-1>;
      opaque rsa_exponent<1..2^16-1>;
    } ServerRSAParams;

struct {
      opaque dh_p<1..2^16-1>;
      opaque dh_g<1..2^16-1>;
      opaque dh_Ys<1..2^16-1>;
    } ServerDHParams;  /* Ephemeral DH parameters */

struct {
      select (KeyExchangeAlgorithm) {
        case diffie_hellman:
          ServerDHParams params;
          Signature signed_params;
        case rsa:
          ServerRSAParams params;
          Signature signed_params;
      };
    } ServerKeyExchange;
```

Signature

md5_hash MD5(ClientHello.random + ServerHello.random + ServerParams);

sha_hash SHA(ClientHello.random + ServerHello.random + ServerParams);

enum { anonymous, rsa, dsa } SignatureAlgorithm;

```
select (SignatureAlgorithm)
    {   case anonymous: struct { };
        case rsa:
            digitally-signed struct {
                opaque md5_hash[16];
                opaque sha_hash[20];
            };
        case dsa:
            digitally-signed struct {
                opaque sha_hash[20];
            };
    } Signature;
```

Certificate Request

```
enum {
      rsa_sign(1), dss_sign(2), rsa_fixed_dh(3), dss_fixed_dh(4), (255)
   } ClientCertificateType;

opaque    DistinguishedName<1..2^16-1>;
```

```
struct {
      ClientCertificateType certificate_types<1..2^8-1>;
      DistinguishedName certificate_authorities<3..2^16-1>;
   } CertificateRequest;
```

Server Hello Done

```
Struct { } ServerHelloDone
```

Client Key Exchange

```
struct {
        ProtocolVersion  client_version;   /* the newest version supported; counter version rollback attacks */
        opaque           random[46];
    } PreMasterSecret;

struct {
        public-key-encrypted     PreMasterSecret pre_master_secret;
    } EncryptedPreMasterSecret;

enum { implicit, explicit } PublicValueEncoding;

struct {
        select (PublicValueEncoding) {
            case implicit:   struct { };
            case explicit:   opaque dh_Yc<1..2^16-1>;
        } dh_public;
    } ClientDiffieHellmanPublic;

struct {
        select (KeyExchangeAlgorithm) {
            case rsa:                EncryptedPreMasterSecret;
            case diffie_hellman:     ClientDiffieHellmanPublic;
        } exchange_keys;
    } ClientKeyExchange;
```

Certificate Verify

```
struct {
        Signature signature;
    } CertificateVerify;
```

CertificateVerify.signature.md5_hash MD5(handshake_messages);

CertificateVerify.signature.sha_hash SHA(handshake_messages);

- **handshake_messages refers to the concatenation of all handshake messages exchanged so far, including the type and length fields of the handshake messages.**

Finished

- to verify that the key exchange and authentication processes were successful
- Always sent immediately after a change cipher spec message
- the first message protected with the just-negotiated security parameters

```
struct {
      opaque     verify_data[12];
  } Finished;
```

verify_data

PRF(master_secret, finished_label, MD5(handshake_messages) + SHA-1(handshake_messages)) [0..11];

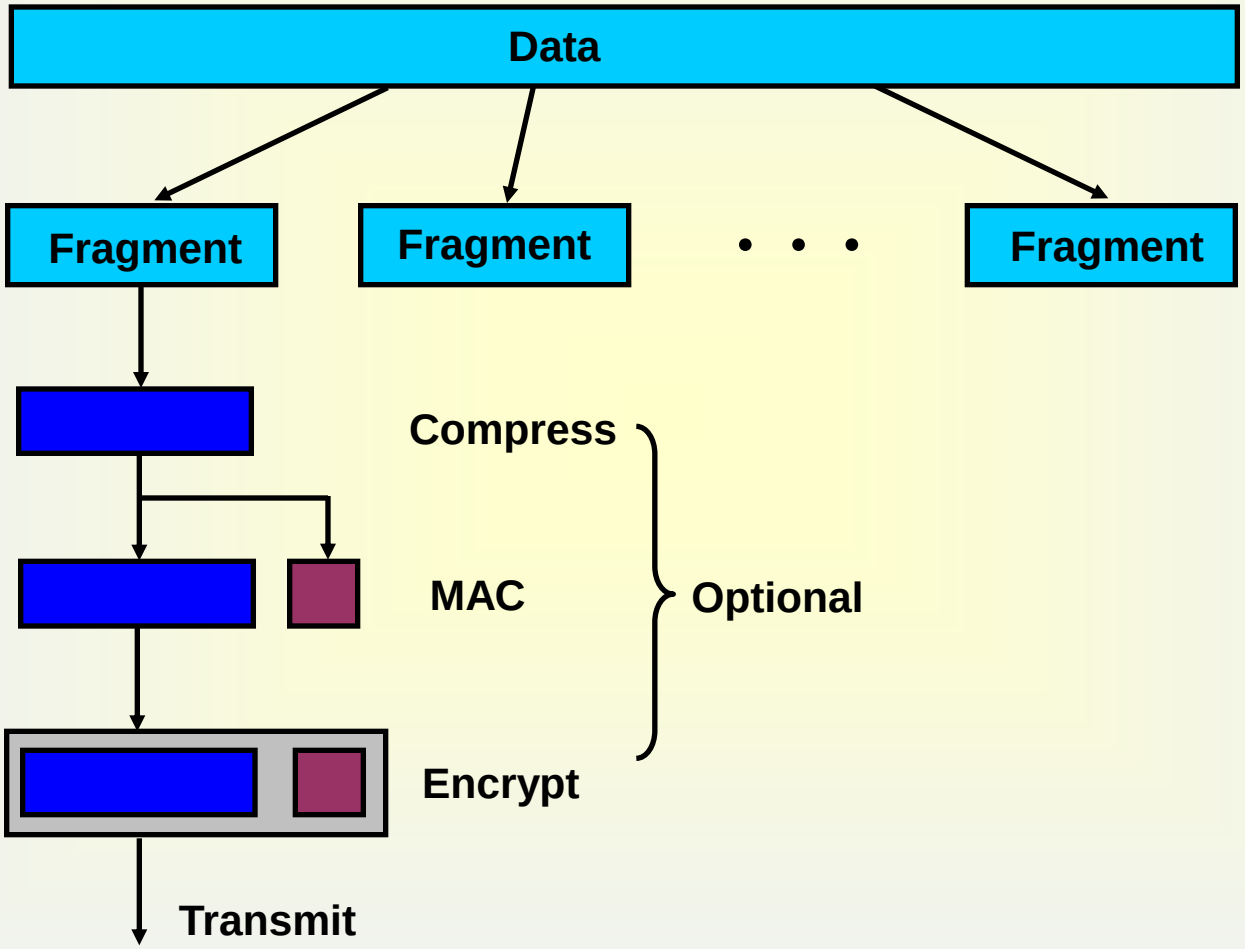
finished_label	"client finished" or "server finished"
handshake_messages	the concatenation of all handshake messages exchanged so far, including the type and length fields of the handshake messages.

* Note that the handshake messages for Client are different from those for Server.

SSL/TLS Key Exchange Algorithms

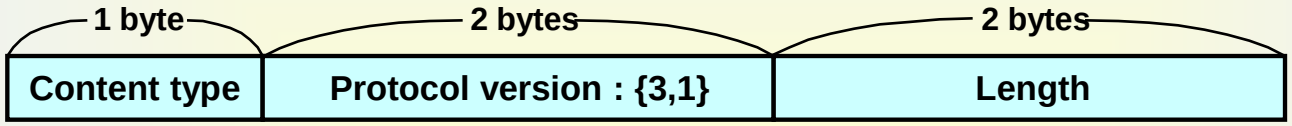
Key Exchange Alg.	DH_DSS, DH_DSS_EXPORT / DH_RSA, DH_RSA_EXPORT		
Certificate(S)	DH (signed by CA's DSS/RSA key)		
ServerKeyExchange	None		
CertificateRequest	if dss_sign/rsa_sign,	if rsa_fixed_dh/dss_fixed_dh	if None
Certificate(C)	DSS/RSA	DH	None
ClientKeyExchange	Ephemeral DH key	empty	Ephemeral DH key
CertificateVerify	DSS/RSA signature	None	None
Key Exchange Alg.	**DHE_DSS, DHE_DSS_EXPORT / DHE_RSA, DHE_RSA_EXPORT**		
Certificate(S)	DSS / RSA (signed by CA's DSS/RSA key)		
ServerKeyExchange	DSS/RSA-signed ephemeral DH key		
CertificateRequest	if dss_sign/rsa_sign,	if None	
Certificate(C)	DSS/RSA	None	
ClientKeyExchange	Ephemeral DH key	Ephemeral DH key	
CertificateVerify	DSS/RSA signature	None	
Key Exchange Alg.	**RSA, RSA_EXPORT (for encryption)**		
Certificate(S)	RSA(e) (RSA(s) for RSA_EXPORT)		
ServerKeyExchange	None (RSA-signed temporary short RSA key)		
CertificateRequest	if dss_sign/rsa_sign,	if None	
Certificate(C)	DSS/RSA(s)	None	
ClientKeyExchange	RSA-encrypted pre-master secret	RSA-encrypted pre-master secret	
CertificateVerify	DSS/RSA signature	None	

TLS Record Protocol



TLS Record Protocol (1)

TLS header



- **Fragmentation**: information block → TLSPlaintext

```
struct {
      ContentType type;                    /* 1byte */
      ProtocolVersion version;     /* 2bytes */
      uint16 length;                       /* must not exceed 2^14 */
      opaque fragment[TLSPlaintext.length];
  } TLSPlaintext;

    enum{ change_cipher_spec(20), alert(21), handshake(22), application_data(23), (255) } ContentType;
    struct{ unit8 major, minor; } ProtovolVersion;
```

- **Compression**: TLSPlaintext = TLSCompressed (no compression used)

TLS Record Protocol (2)

- **Record Payload Protection: TLSCompressed → TLSCiphertext**

```
struct {
        ContentType type;
        ProtocolVersion version;
        uint16 length;                                  /* should not exceed 2^14+2048 */
        select (SecurityParameters.cipher_type) {
           case stream: GenericStreamCipher;
           case block: GenericBlockCipher;
        } fragment;
    } TLSCiphertext;
```

```
stream-ciphered struct {
      opaque content[TLSCompressed.length];
      opaque MAC[SecurityParameters.hash_size];
    } GenericStreamCipher;
```

```
block-ciphered struct {
      opaque content[TLSCompressed.length];
      opaque MAC[SecurityParameters.hash_size];
      uint8 padding[GenericBlockCipher.padding_length];
      uint8 padding_length;
    } GenericBlockCipher;
```

- **MAC Generation**

  HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type || TLSCompressed.version || TLSCompressed.length || TLSCompressed.fragment))

TLS Key Derivation

- **PRF (Pseudo-Random Function):**

  P_hash(secret, seed) = HMAC_hash(secret, A(1) || seed) ||
  HMAC_hash(secret, A(2) || seed) ||
  HMAC_hash(secret, A(3) || seed) || …
  (A(0) = seed, A(i) = HMAC_hash(secret, A(i-1))

  PRF(secret, label, seed) = P_MD5(S1, label || seed) ⊕ P_SHA-1(S2, label || seed)
  (S1, S2 : 1st and 2nd half of secret)

- **Master Secret (48 bytes) -- in Handshake protocol**

  master_secret = PRF(pre_master_secret, "master secret",
  ClientHello.random || ServerHello.random) [0..47];

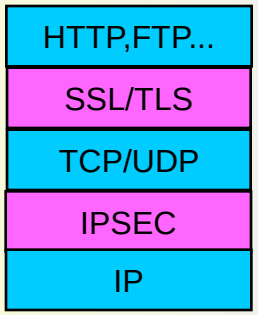
- **Key Block -- in Record protocol**

  key_block = PRF(SecurityParameters.master_secret, "key expansion",
  SecurityParameters.server_random || SecurityParameters.client_random)
  = client_write_MAC_secret[SecurityParameters.hash_size] ||
  server_write_MAC_secret[SecurityParameters.hash_size] ||
  client_write_key[SecurityParameters.key_material_length] ||
  server_write_key[SecurityParameters.key_material_length] ||
  client_write_IV[SecurityParameters.IV_size] ||
  server_write_IV[SecurityParameters.IV_size]

IPSEC vs. SSL/TLS

IPSEC

- Network layer security protocol
- Confidentiality, Integrity, Authentication, Access control, Auditing
- Transport protocol independent
- No change to applications (application/user transparency)
- Peer-to-Peer model: Host-to-Server, Host-to-Subnet, Subnet-to-Subnet
- More secure; too complex, special client SW
- IPv4 (optional), IPv6 (mandatory)



SSL/TLS

- Transport layer security protocol
- Confidentiality, Integrity, Authentication (usually client-to-server only)
- Works only with TCP (not UDP): HTTP, SMTP, POP3, NNTP, FTP, LDAP…
- Minimal changes to applications
- Client-Server model: Host-to-Server (secure Web transactions)
- Free : built in to nearly all browsers and Web servers

Homework #7

❖ Security Protocol Analysis

Ethereal is one of the most popular network protocol analyzer. Using Ethereal we try to analyze network security protocols which were introduced in this lecture.

1. **Install Ethereal on your PC (http://www.ethereal.com/) and get yourself accustomed to it how to use the software. Ssldump is another program (http://www.rtfm.com/ssldump/) to analyze SSL protocol.**
2. **Construct or prepare experimental environment to test SSL/TLS protocols. For example, you install Apache web server with SSL enabled. You can use any existing available environment.**
3. **Experiment on the SSL/TLS protocol and reconstruct one of the successful protocol execution example on the following processes.**
    1. **Handshake protocol**
    2. **Record protocol**

Describe your experimental results.

Alternatively you can experiment on IPSEC protocol.