COMP 3355 Web Security – SSL/TLS

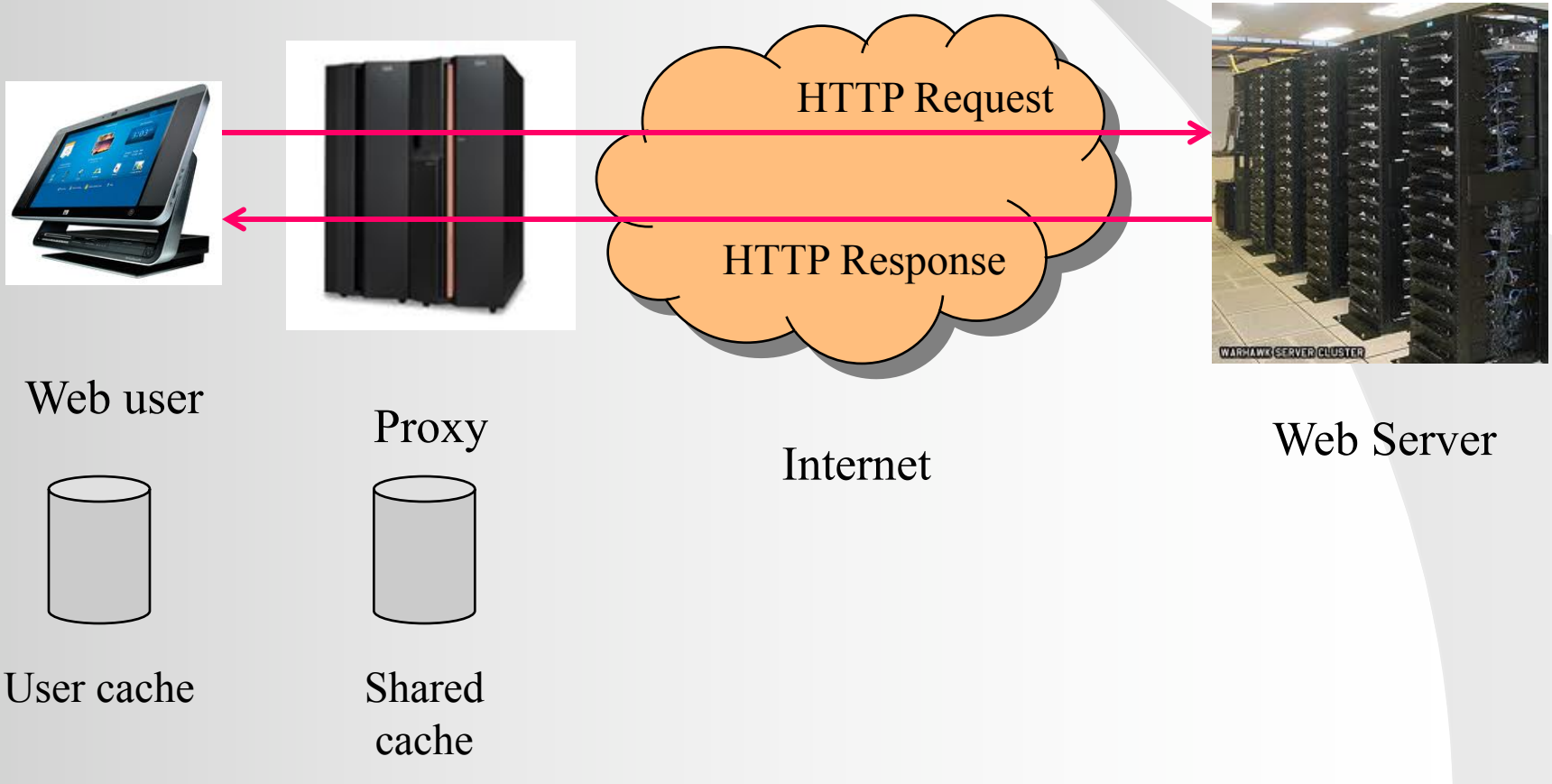
K.P. Chow

University of Hong Kong

HTTP BASICS

HTTP Protocol

- HTTP is used for client-server communication in the web



HTTP Messages (1)

- HTTP messages are requests sent from clients to servers, and responses from servers to clients
- HTTP message structure:
  - Start line
  - 0 or more headers (e.g. “From”, “Date”), each header contains
    - field name, colon, field value
    - E.g. From: chow@cs.hku.hk
    - E.g. Date: 28/03/2012
  - Optional message body

HTTP Messages (2)

- HTTP Request
  - Start line: request method
    - GET, HEAD: information retrieval
    - POST: request an action to be performed, e.g. perform a payment
    - PUT, DELETE, …
- HTTP Response
  - Start line: status line, contains
    - HTTP version used by the server
    - Status code:
    - Reason phrase
  - E.g. “200 OK”, “404 Not Found”, “401 Unauthorized”, “403 Forbidden”

HTTP Cache Security Issues

- A cache is a local store which may be created and maintained by a client, a proxy or a gateway
- Cache control can be implemented by setting in header "Cache-Control"
- Sensitive information may be cached:
  - The response may contain "payment receipt", which should not be cached
  - If a client is authorized to retrieve a document, its Authorization header carries user's authentication information (e.g. password), which should not be cached
- Dishonest proxy may cache "everything"

HTTP AUTHENTICATION

HTTP Authentication

- HTTP basic authentication
- Form-based authentication
- HTTPS mutual authentication
- HTTP digest authentication

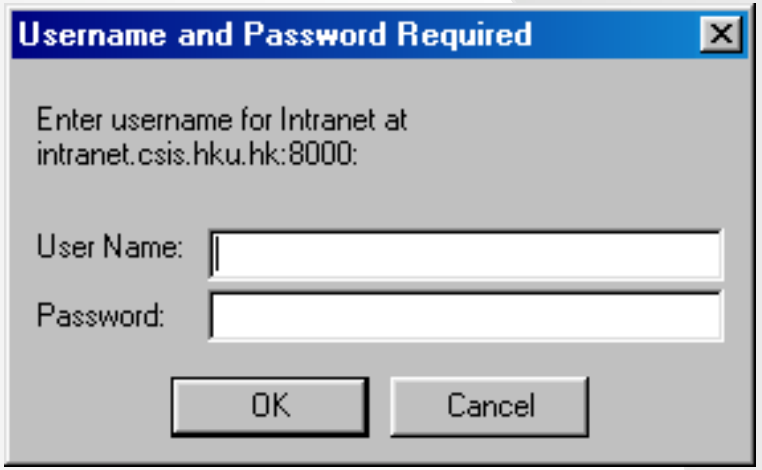
HTTP Basic Authentication

- Deliver documents from within certain directories
- Restrict certain documents to specified users
- Use HTTP standard basic authentication: challenge-response procedure:
  - Client clicks on link to restricted page and send
    `Request: GET http://server/restricted.html`
  - Server checks permissions and rejects request and send
    `Response: Status 401 Realm "User"`
  - Client generates pop-up menu and asks for ID and password
  - Client resends request with ID and password in header
    `Request: GET http://server/restricted.html`

HTTP Basic Authentication

- Simplest form of authentication that a servlet might employ
- Server asks the browser for a username and password
- User supply a username and the password
- Each request from the browser to the server will include the username the password, BASE64-encoded

- Problems: weak security because username and password are BASE64-encoded and sent with every request, which can be intercept and read



Basic Authentication

- Client to server

  GET http://www.hku.hk/pub/WWW/Project.html HTTP/1.1

  User-Agent: Mozilla/4.0

- Server to client

  HTTP/1.1 401 Unauthorized

  WWW-Authenticate: Basic realm="Users"

- Client to server

  GET http://www.hku.hk/pub/WWW/Project.html HTTP/1.1

  User-Agent: Mozilla/4.0

  Authorization: Basic QWxhZGRrbjpvcGVuIHNlc2FtZQ=

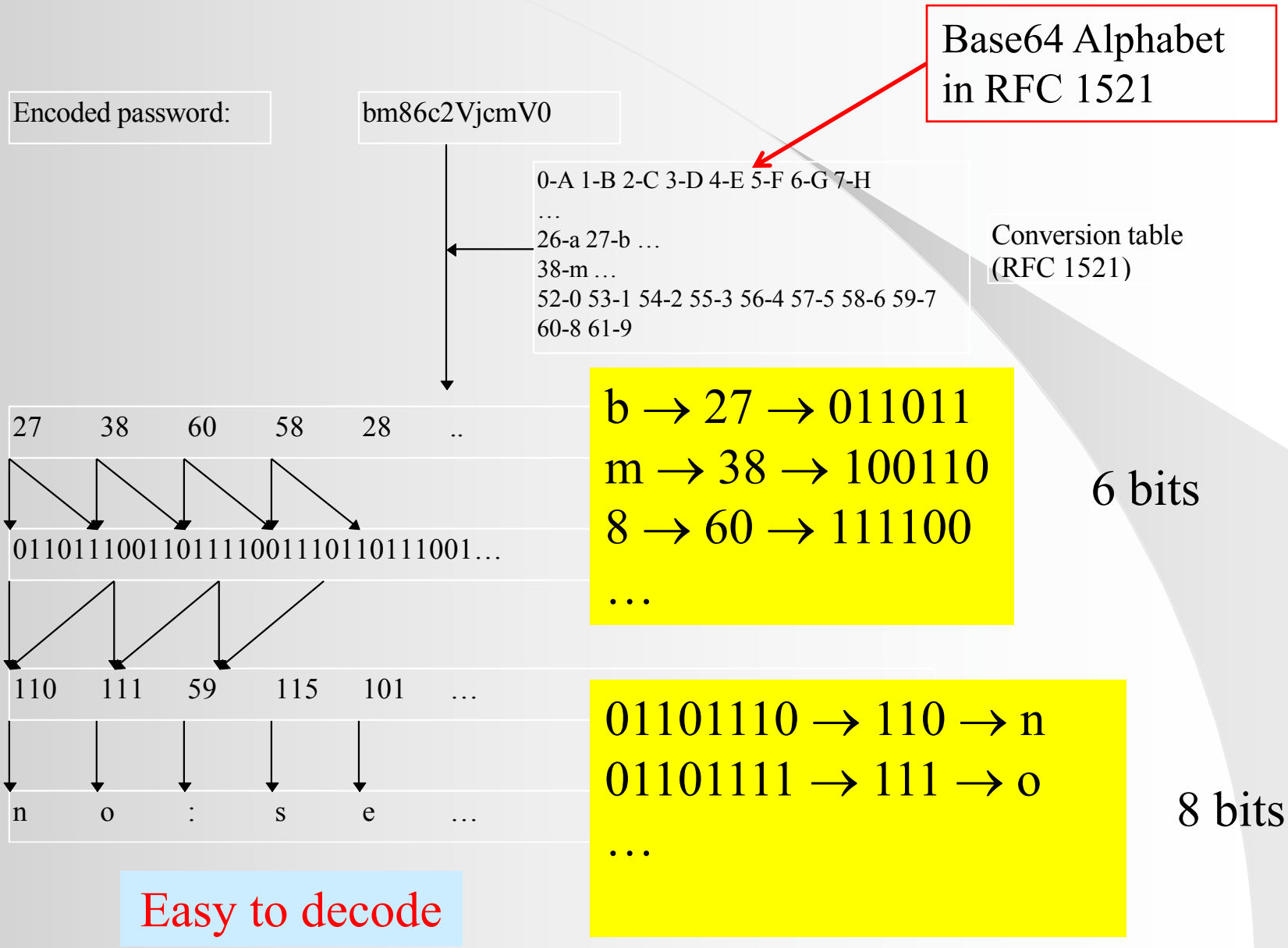
Base64-encoded "Aladdin:open sesame"

How secure is basic HTTP authentication scheme?

How sure can you be that the user really is who he claims to be?

- The user has given the ID to another person
- The user writes down the ID and someone picks it up
- Someone guesses the password
- Someone intercepts the user ID and password between the client and server
- HTTP protocol uses base64 encoding to encode the user ID and password (use to translate 8 bit data to 7 bit data, no encryption), documented in MIME standard (RFC 1521)

How to reverse base64 encoding?



Form-Based Authentication

- Use a standard HTML page to request the username and password (instead of the pop-up dialog box in HTTP Basic Authentication)
- The HTML page includes a form with the action j_security_check and 2 fields: j_username and j_password, e.g.

```
<HTML><HEAD><TITLE>Log in</TITLE></HEAD>
<BODY><FORM METHOD="POST" ACTION="j_security_check"
    Username: <INPUT TYPE="TEXT" NAME="j_username"><BR>
    Password:  <INPUT TYPE="PASSWORD"
      NAME="j_password"><BR>
    <INPUT TYPE="SUBMIT" VALUE="Submit">
</FORM></BODY></HTML>
```

Form-Based Authentication

- The password is sent in plain text from the client to the server
- After a password is sent, a cookie is typically passed back-and-forth between the client and the server to indicate that the client is already authenticated: the cookie can be intercepted

HTTPS Mutual Authentication

- Strongest form of authentication
- Client is required to have a private key and a corresponding certificate to be authenticated
- The server only allows clients that have a certificate that has been authorized to access the given resource
- In order for each user to have a certificate, he should obtain one from the CA (e.g. Verisign), or establish your own CA to issue certificates

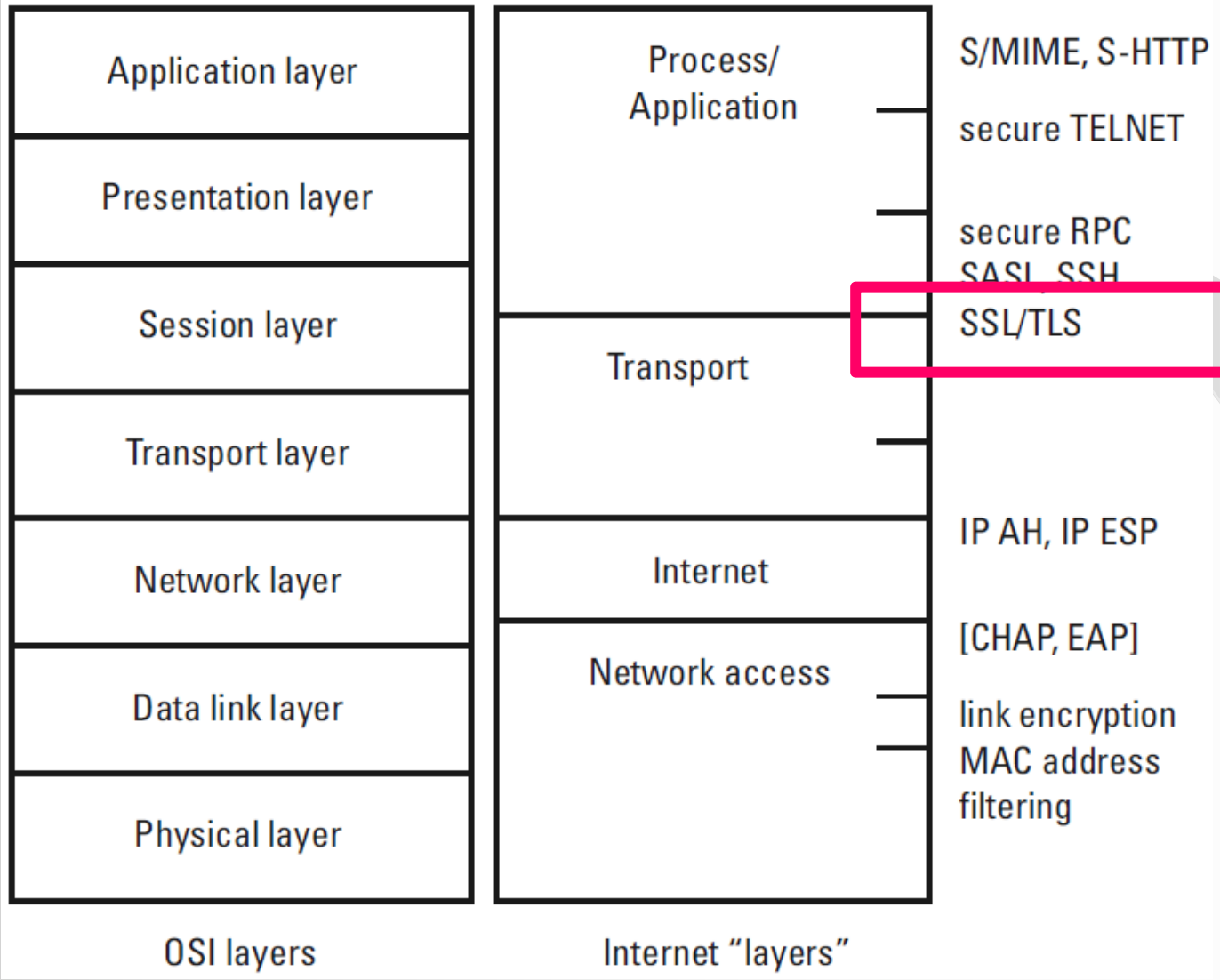
Not a popular approach

HTTP Digest Authentication

- Use message digest to exchange information proving that the user knows their password without sending the password itself
  1. The server creates a **nonce**, a random value that is unique, e.g. the client IP address + timestamp + random data "147.0.0.1:958579743033:upAD3=aHi6wX"
  2. The server sends the nonce to the client
  3. The client hashes the nonce together with its username and password
  4. The client sends the hash back to the server
  5. The server then computes the hash with the username, password and nonce
  6. The server compares the two hashes:
     - If they match, the client is given access to the protected resource
     - If not match, access is denied
- Why use timestamp with a random value?
  - To avoid replay attack

SSL/TLS

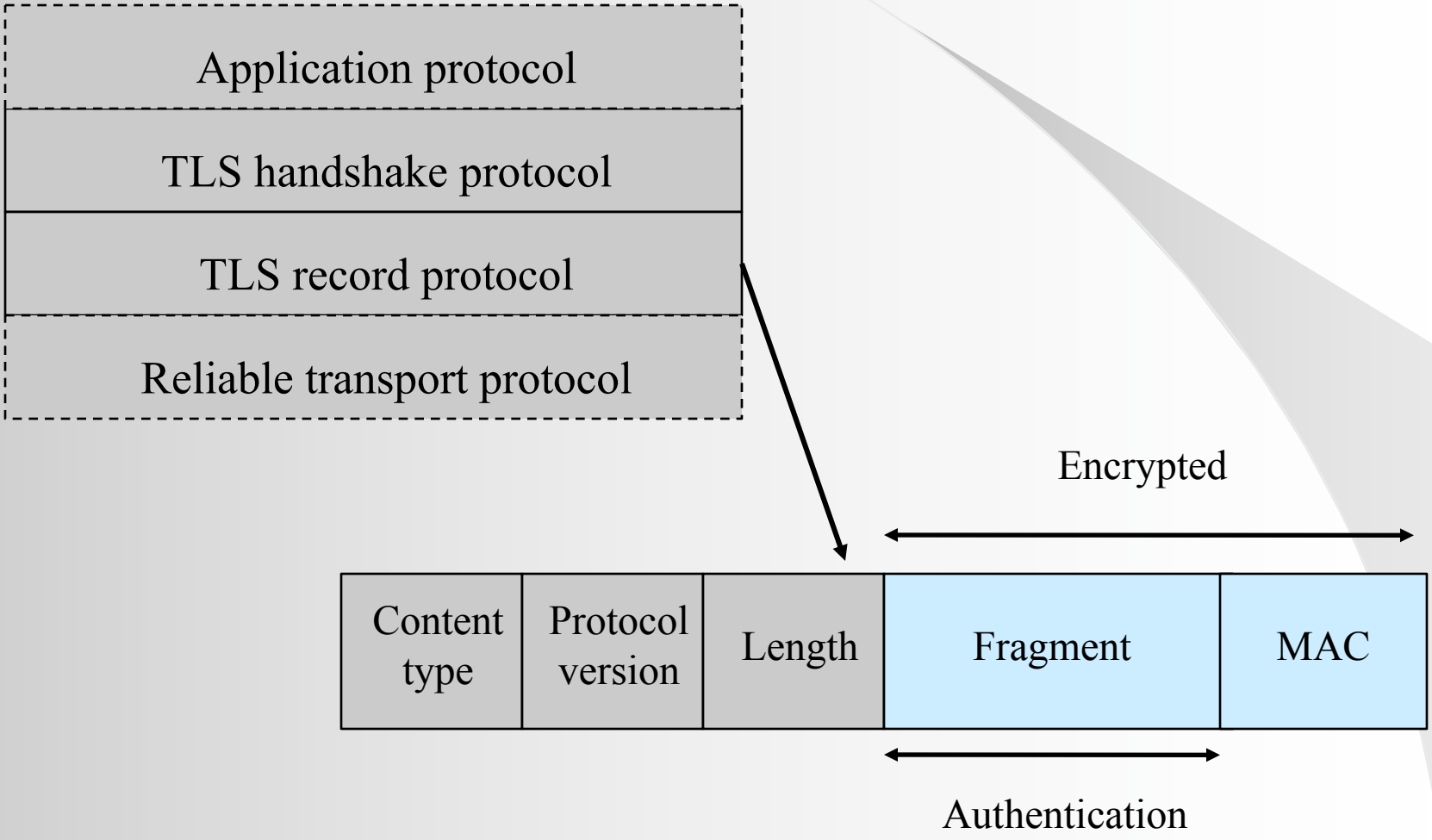
Where is SSL/TLS?



Transport Layer Security (TLS)

- A security protocol on top of TCP that replaces the widely used SSL
- Applications use TLS should be extended with the corresponding functions calls (TLS programming library)
- TLS v1.0 developed from SSLv3
- TLS and SSLv3 are very similar, but different enough to ensure NOT inoperable, e.g. SSL uses MD5 and TLS uses HMAC, but TLS can “back down” to SSLv3
- TLS provides data integrity, data confidentiality, and peer entity authentication

TLS Encrypted Packet



TLS Record Protocol Packet

TLS Record Protocol

- Uses for encapsulation of higher level protocols
- Takes messages to be transmitted, fragments data into manageable blocks, applies a MAC, encrypts and transmits

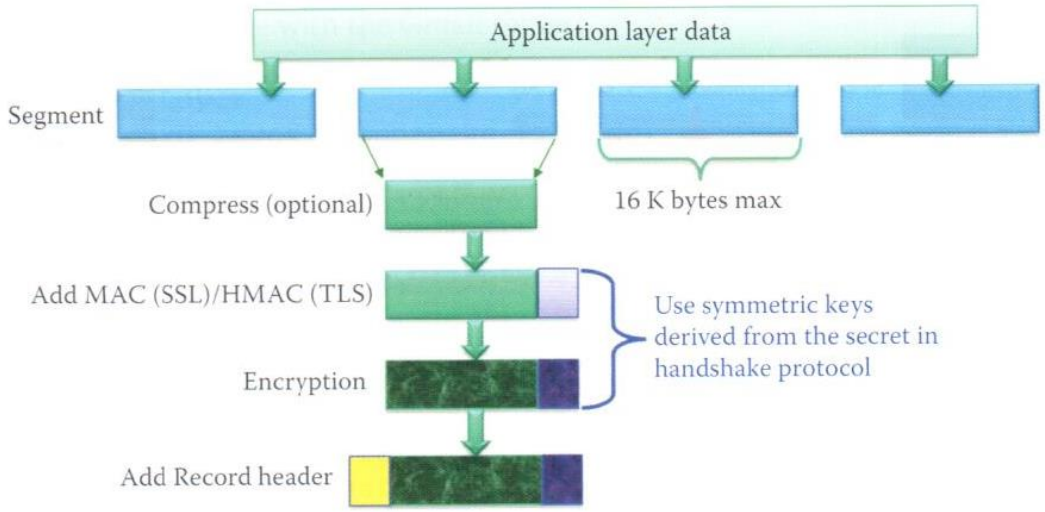


FIGURE 23.7 The SSL/TLS record protocol packet format.

TLS Handshake Protocol

- For the client and the server to authenticate each other and negotiate an encryption algorithm and cryptographic keys before the application-level data is transmitted
- Consists of the following 3 protocols:
  - Handshake protocol: the client and the server agree on a protocol version, select cryptographic algorithm, optionally authenticate each other, and generate shared secret key
  - Change cipher protocol: a single message sent by both the client and the server to notify the receiver that the subsequent message will be protected with the agreed security parameters
  - Alert protocol: various alert messages for notifying the receiver that the connection will be closed, or that an error condition has occurred
- 3 authentication modes: server authentication only, authentication of both parties, and total anonymity

TLS Handshake with Server Authentication

TLS_RSA_EXPORT_WITH_RC4_40_MD5

ClientHello (random_c, version, CipherSuite)

ServerHello (random_s, version, CipherSuite)

Certificate

ServerHelloDone

ClientKeyExchange (E_s(secret))

ChangeCipherSpec

Finished

ChangeCipherSpec

Finished

What are the purposes of random_c and random_s?

TLS Handshake with Client/Server Authentication

ClientHello

ServerHello
Certificate
CertficateRequest
ServerHelloDone

Certificate
ClientKeyExchange
CertificateVerify
[ChangeCipherSpec]
Finished

[ChangeCipherSpec]
Finished

TLS Handshake with Total Anonymity

ClientHello

ServerHello
ServerKeyExchange
ServerHelloDone

ClientKeyExchange
[ChangeCipherSpec]
Finished

[ChangeCipherSpec]
Finished

Diffie-Hellman Key exchange is used.

HTTP WITH SSL/TLS

HTTP Transaction

- For HTTP transaction over Internet, we usually require server being authenticated (How client is authenticated?)
- SSL/TLS suits HTTP since it can provide some protections if only one side of the communication is authenticated
- HTTPS is a URI scheme which has identical syntax with HTTP
- HTTPS signals the browser to use an added encryption layer of SSL/TLS to protect the traffic, which provides a secure channel over an insecure network

Trust in HTTPS

- The user trusts the browser software correctly implements HTTPS with correctly pre-installed certificates from recognized CAs
- The user trusts the CA to vouch only for legitimate websites
- The website provides a valid certificate
- The certificate correctly identifies the website, e.g. when the browser visits https://hku.hk, the received certificate is for HKU in HK
- The user trusts the SSL/TLS protocol

Standard HTTP

Socket API

Socket API

TCP/IP Connection

HTTP over TLS

TLS API

TLS API

TLS Record Protocol

Socket API

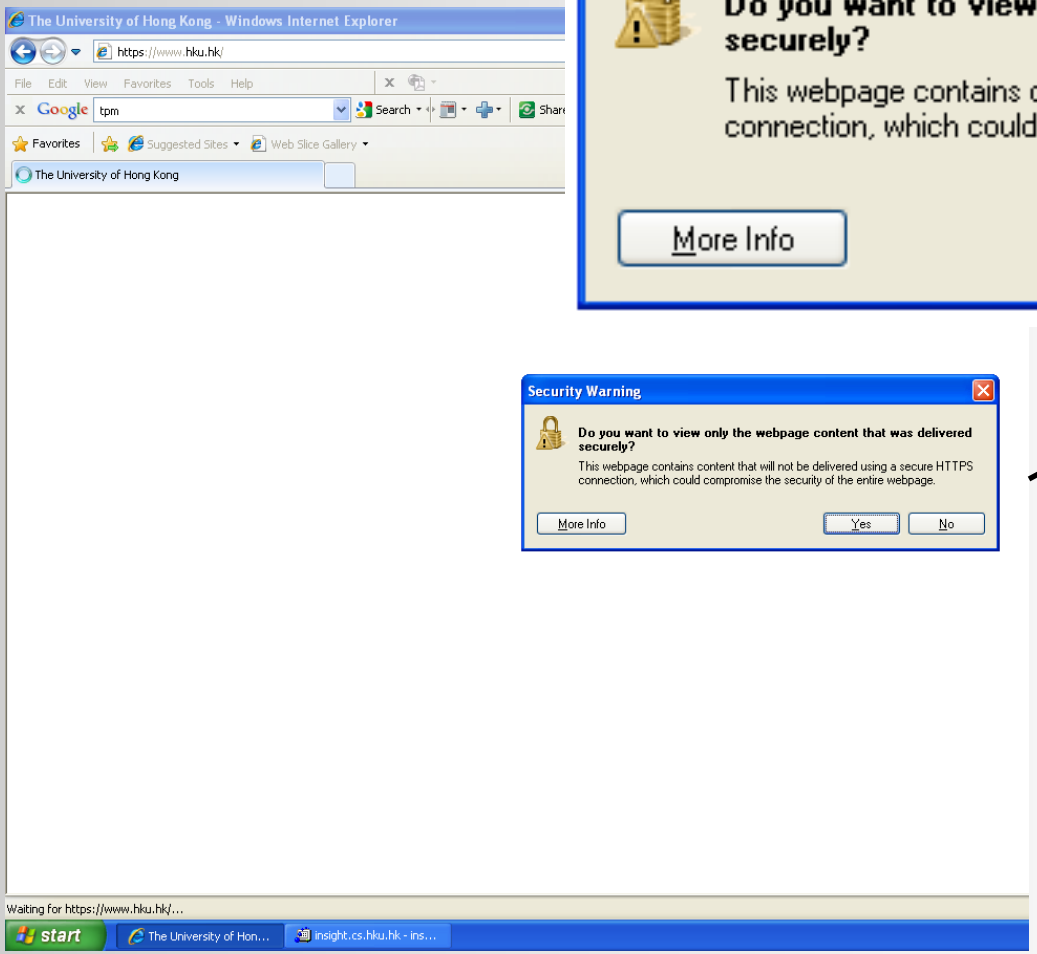
Socket API

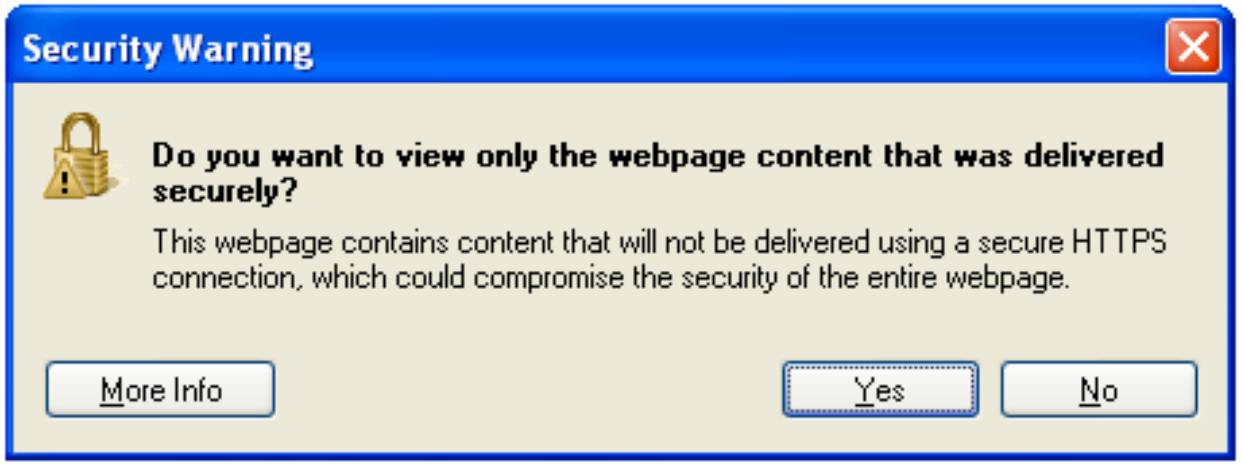
TCP/IP Connection

HTTPS

1. The browser requests a *document* with a special URL which commences https: instead of http:
2. The browser recognizes the TLS request, and establishes a connection through TCP port 443 to the TLS code on the server
3. The browser then initiates the TLS handshake phase, using the TLS Record Protocol as a carrier
4. Following the handshake, both browser and the server have a shared secret **master** key
5. From the **master** key, both browser and the server can generate other session keys, which are used to encrypt the session data
6. The requested document is then encrypted with the session key and then the sent from the server to the client

SSL Mixed Content problem





The risk: data unprotected by SSL *may* be seen by intermediate routers. In many cases this is still safe.
BUT: attack code in non-SSL data can be dangerous!!

SSL Protection

- SSL provides secure encryption in the two points (browser and server)
  - No intermediate routers or processes can see the content transmitted between the browser and the server
- Limitation: the two endpoints can still leak information

LET'S LOOK AT SSL HEARTBLEED PROBLEM

SSL 2.0 VULNERABILITIES

SSL version 2.0 [SSL2] deficiencies

- **Message authentication uses MD5**. Most security-aware users have already moved away from any use of MD5.
- **Handshake messages are not protected**. This permits a man-in-the-middle to trick the client into picking a weaker cipher suite than it would normally choose.
- **Message integrity and message encryption use the same key**, which is a problem if the client and server negotiate a weak encryption algorithm.
- **Sessions can be easily terminated**. A man-in-the-middle can easily insert a TCP FIN to close the session, and the peer is unable to determine whether or not it was a legitimate end of the session.

SSL Heartbleed Problem

What is OpenSSL?

- One of the open source development project
- Also one of the most popular implementation of Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
- With a set of cryptographic toolkits

Source : OpenSSL project website. [8]

Timeline – How Heartbleed was introduced

Dec-1998

Official start of the OpenSSL project

Dec-2011

- Robin Seggelmann, one of RFC's authors, implemented the Heartbeat Extension for OpenSSL.
- Stephen Henson, one of OpenSSL's core developers, reviewed the code and introduced it into OpenSSL's source code repository.

Feb-2012

Internet Engineering Task Force (IETF) released RFC 6520: Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension

14- Mar-2012

The vulnerable code was adopted into OpenSSL v1.0.1 and widely spread.

Source: Heartbleed from Wikipedia. [3]

Timeline
– Discovery of Heartbleed (2014)

- Affected versions of OpenSSL software: versions 1.0.1 through 1.0.1f
- It took **2 years** for the bug to be discovered after version 1.0.1 was released in 2012.

Neel Me
Security
Heartble
vulnerab

p

Marc

the flaw, which later becomes known as "Heartbleed".

National Cyber Security Centre Finland (NCSC-FI).

April 7

SSL publishes
rtbleed
ty advisory
on its website.
Versions 1.0.1g and later have implemented a fix.

Source: Grubb, B., "Heartbleed disclosure timeline: who knew what and when" at The Sydney Morning Herald, 15 Apr 2014. [2]

Impact of Heartbleed

Well-known websites: Facebook, Google, Twitter, Instagram, YouTube, Gmail, Yahoo!Mail, Amazon, DropBox, etc.

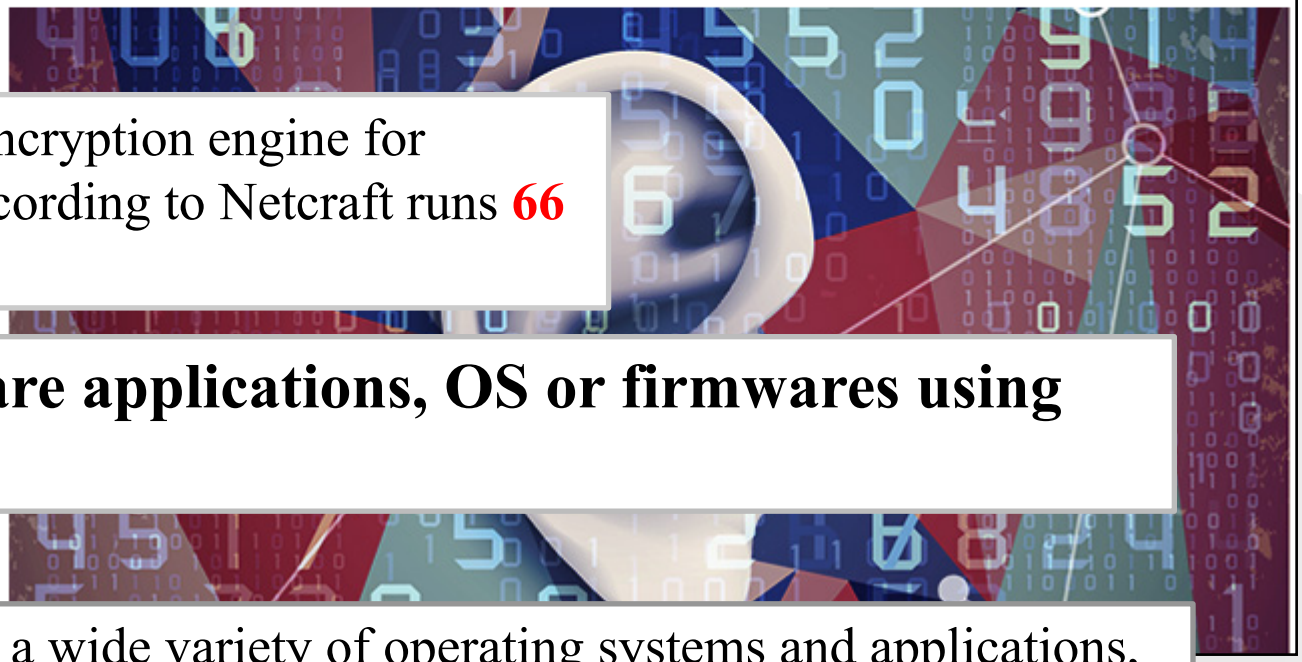
Source : http://arstechnica.com/security/2014/04/critical-crypto-bug-in-openssl-opens-two-thirds-of-the-web-to-eavesdropping/

Critical crypto bug in OpenSSL opens two-thirds of the Web to eavesdropping

Exploits allow attackers to obtain private keys used to decrypt sensitive data.

by **Dan Goodin** - Apr 8 2014, 8:10am CST

HACKING OPEN SOURCE 215



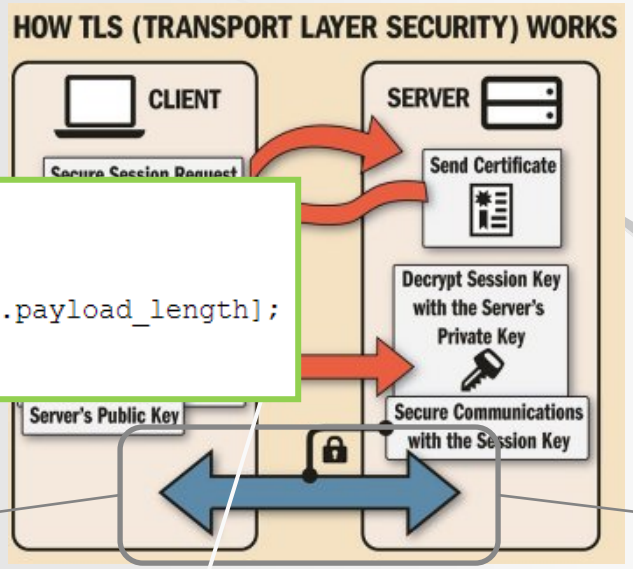
OpenSSL is the default encryption engine for Apache, nginx, which according to Netcraft runs **66 percent of websites**.

How about software applications, OS or firmwares using OpenSSL?

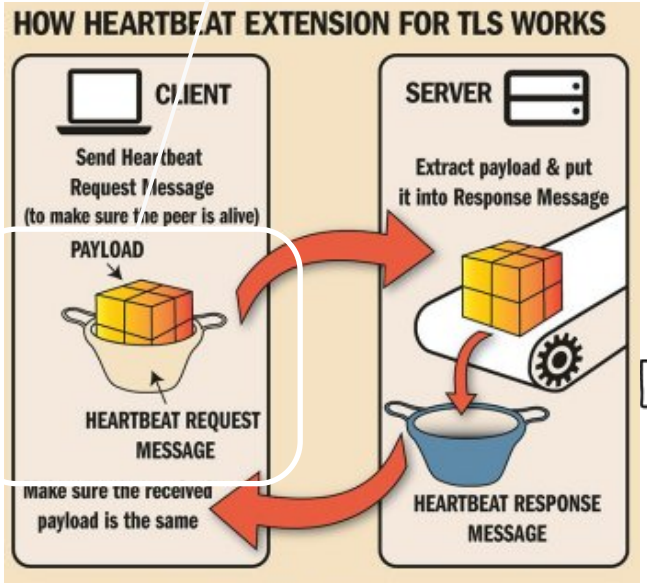
OpenSSL also ships in a wide variety of operating systems and applications, including the Debian Wheezy, Ubuntu, CENTOS, Fedora, OpenBSD, FreeBSD, and OpenSUSE distributions of Linux.

Source : Heartbleed from Wikipedia. [3]

What is Heartbleed?




```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```



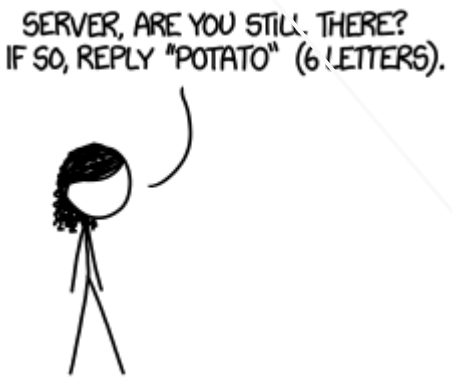
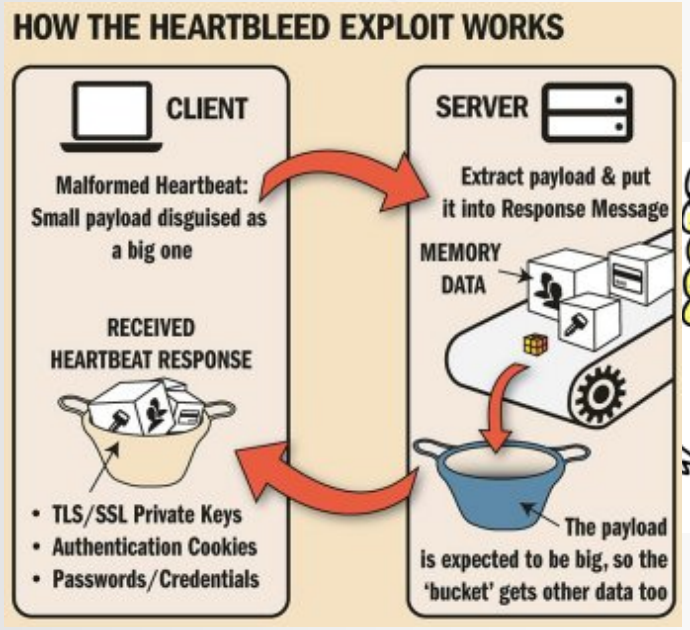
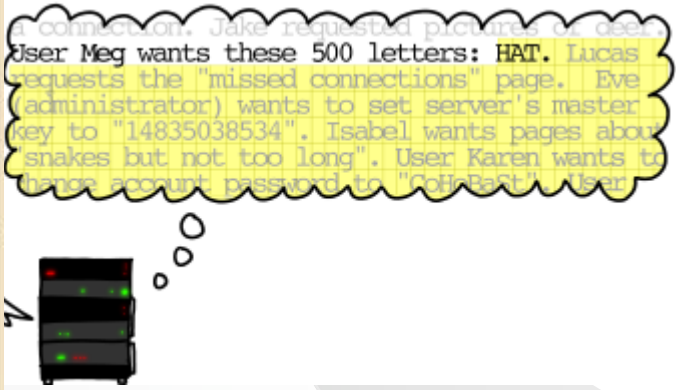


Image Source: http://cdn.grahamcluley.com/wp-content/uploads/2014/04/heartbleed-explained.jpeg

Source: Matthew, G., "Attack of the week: OpenSSL Heartbleed" at A Few Thoughts on Cryptographic Engineering, 8 Apr 2014. [13]

What is Heartbleed?





- An attacker can request that a running TLS server hand over a relatively large slice (up to 64KB) of its private memory space.

- This private memory space can *potentially* store:
  - *(a)* long-term server private key
  - *(b)* TLS session keys
  - *(c)* confidential data like passwords
  - *(d)* session ticket keys

Image Source: http://cdn.grahamcluley.com/wp-content/uploads/2014/04/heartbleed-explained.jpeg
Source: Matthew, G., "Attack of the week: OpenSSL Heartbleed" at A Few Thoughts on Cryptographic Engineering, 8 Apr 2014. [13]

WHAT HAPPENS BEHIND THE HEARTBLEED BUG?

Heartbleed
- Is it an implementation problem?

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

```
diff
inde
---
+++
@@ -2588,16 +2588,20 @@ tls1_process_heartbeat(SSL *s)
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;
-
        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       hbtype = *p++;
+       n2s(p, payload);
+       if (1 + 2 + payload + 16 > s->s3->rrec.length)
+               return 0; /* silently discard per RFC 6520 sec. 4 */
+       pl = p;
+
        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
```

OpenSSL source code

- Heartbleed implementation problems
  - Did not check for invalid user input!
  - Did not check for buffer over-read!

stops zero-length heartbeats

checks to make sure that the actual record length is sufficiently long

Source : Sean, “Diagnosis of the OpenSSL Heartbleed Bug” at Sean’s thoughts, 7 Apr 2014. [12]

Reference

[1] Cantrell, T., “Heartbleed and its impact on embedded security” at www.embedded.com, 30 Apr 2014. Retrieved at http://www.embedded.com/design/safety-and-security/4430090/Heartbleed-and-its-impact-on-embedded-security

[2] Grubb, B., “Heartbleed disclosure timeline: who knew what and when” at The Sydney Morning Herald, 15 Apr 2014. Retrieved at http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html

[3] Heartbleed from Wikipedia. http://en.wikipedia.org/wiki/Heartbleed

[4] Heartbleed official website. http://heartbleed.com/

[5] IETF, “RFC 1122: Requirements for Internet Hosts – Communication Layers”, Oct 1989. Retrieved at http://tools.ietf.org/html/rfc1122.

[6] IETF, “RFC 4821: Packetization Layer Path MTU Discovery”, Mar 2007. Retrieved at http://tools.ietf.org/html/rfc4821.

[7] IETF, “RFC 6520: TLS Heartbeat Extension”, Feb 2012. Retrieved at http://tools.ietf.org/html/rfc6520

[8] OpenSSL project website. http://www.openssl.org/

[9] Openssl from Wikipedia. http://en.wikipedia.org/wiki/OpenSSL

[10] Mesander, B., “Heartbleed wasn’t a single bug” at www.cardinalpeak.com , 14 Apr 2014. Retrieved at http://www.cardinalpeak.com/blog/heartbleed-wasnt-a-single-bug/

[11] Saltzer, J. H. & Schroeder, M. D. "The Protection of Information in Computer Systems." 1278-1308. *Proceedings of the IEEE 63,* 9, September 1975.

[12] Sean, “Diagnosis of the OpenSSL Heartbleed Bug” at Sean’s thoughts, 7 Apr 2014. Retrieved at http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html

[13] Matthew, G., “Attack of the week: OpenSSL Heartbleed” at A Few Thoughts on Cryptographic Engineering, 8 Apr 2014. Retrieved at http://blog.cryptographyengineering.com/2014/04/attack-of-week-openssl-heartbleed.html

STORY IS NOT COMPLETE WITHOUT IPSEC

IPsec Implementation

- IPsec provides transport security for all users of IP, without changing the interface to IP, i.e. upper layer protocols need not be modified to invoke security and need not be aware that their traffic is protected at the IP layer
- IPsec provides host-to-host security, but not user-to-user or application-to-application security

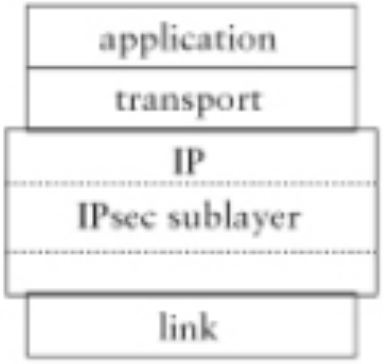


Figure 16.10: IP Security

IP Security Protocol (IPsec)

- A suite of "complex" protocols that provides security at the network layer, e.g. RFC 4301: overall IP security architecture
- Network layer security:
  - Provides secrecy by encrypting all IP datagrams (e.g. TCP/UDP segments, ICMP messages)
  - Transparent to applications above the IP layer
- Source authentication: able to authenticate the IP source addresses, to prevent spoofing IP addresses
- 2 principle protocols:
  - Authentication Header (**AH**) protocol: provides authentication and integrity for packet sources
  - Encapsulation Security Payload (**ESP**) protocol: provides authentication and integrity for packet sources, and confidentiality using AES

With ESP, why you need AH?

- Authentication Header (**AH**) protocol: provides authentication and integrity for packet sources
- Encapsulation Security Payload (**ESP**) protocol: provides authentication and integrity for packet sources, and confidentiality using AES

- In the 1990s, export restrictions on encryption algorithms required "exported" products to provide an authentication-only mechanism
- With no export restrictions today, we can use ESP only now

IPsec

- Services:
  - Indirect access control support
  - Connectionless integrity
  - Data origin authentication
  - Protection against replaying/reordering of IP packets
  - Confidentiality
  - Limited traffic flow confidentiality
- Key components:
  - Security protocols (AH, ESP)
  - Security association (SA)
  - Algorithm for authentication and encryption
  - Key management (IKE)

IPsec Modes

- Transport mode: protection is afforded from host-to-host and host-to-gateway

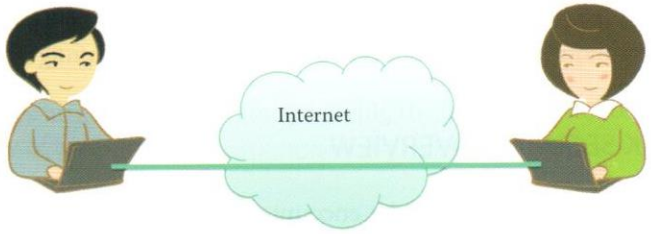


FIGURE 24.2 IPsec in the transport mode for a host to host VPN.

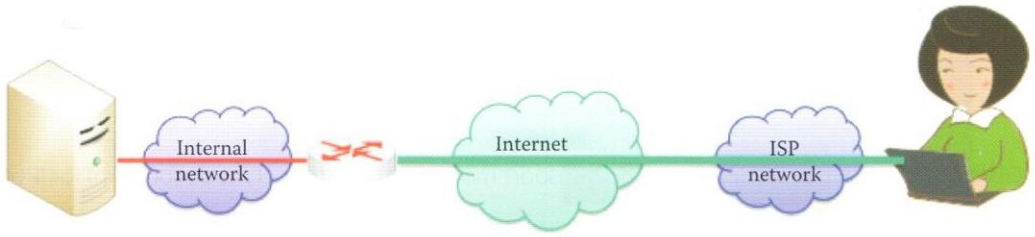


FIGURE 24.4 Host to Gateway VPN.

IPsec Modes

- Tunnel mode: supports gateway-to-gateway and host-to-gateway connections, rarely used for host-to-host connection

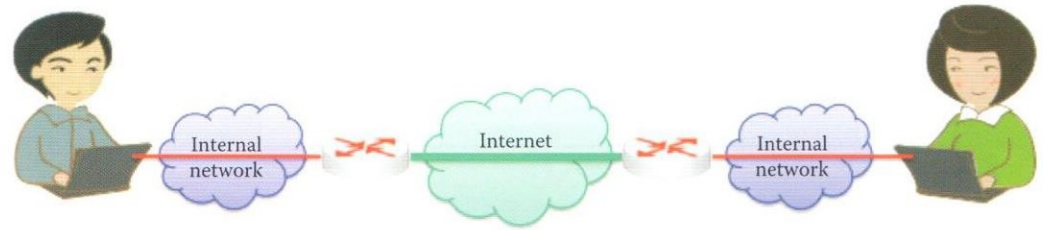


FIGURE 24.3 IPsec in the tunnel mode for a *gateway-to-gateway* VPN.

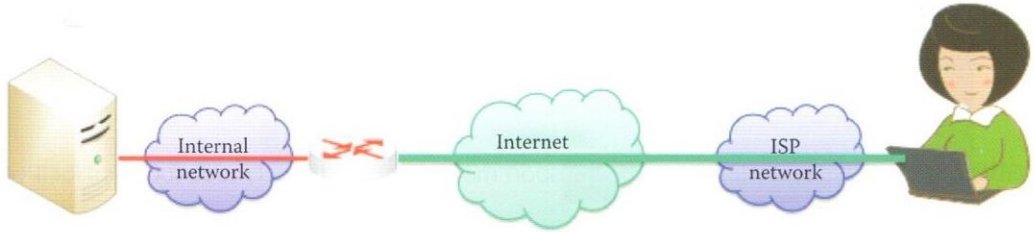


FIGURE 24.4 Host to Gateway VPN.

Transport Mode vs. Tunnel Mode

- Transport mode uses the original IP header: protects the packet payload
  - An upper-layer protocol frame, e.g. TCP or UDP frame, is encapsulated within the ESP
  - The IP header is not encrypted
  - Provides end-to-end protection of packets between 2 hosts, both hosts have to IPsec aware
- Tunnel mode uses IPsec header: protects both the original IP header and the payload, more difficult for attackers to identify targets
  - The whole IP packet is treated as a new payload of an outer IP packet, the original inner IP packet is encapsulated within the outer IP packet
  - End hosts need not be IPsec aware
  - Provides gateway-to-gateway security rather than end-to-end security

Transport Mode vs. Tunnel Mode

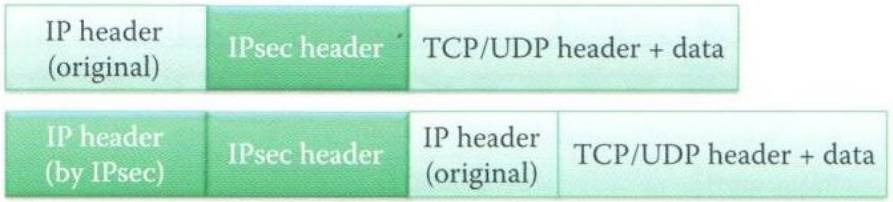


FIGURE 24.5 Transport Mode (top) vs. Tunnel Mode (bottom).

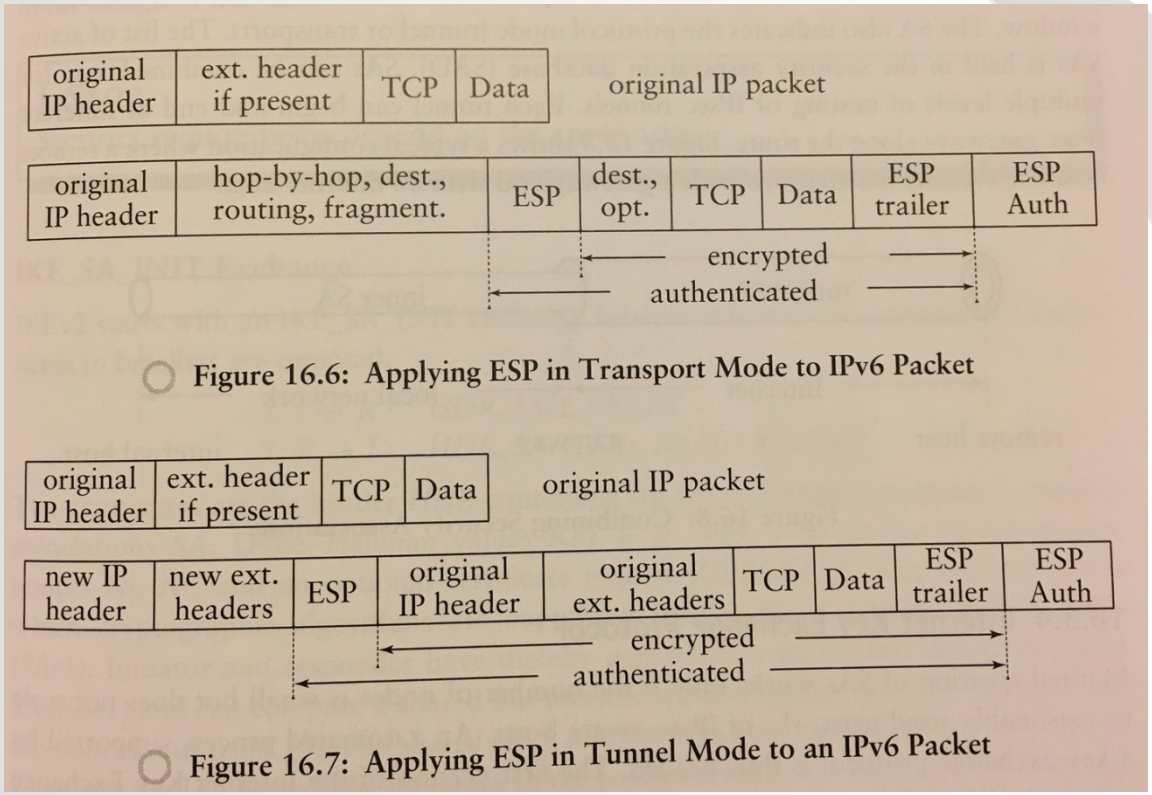


Figure 16.6: Applying ESP in Transport Mode to IPv6 Packet

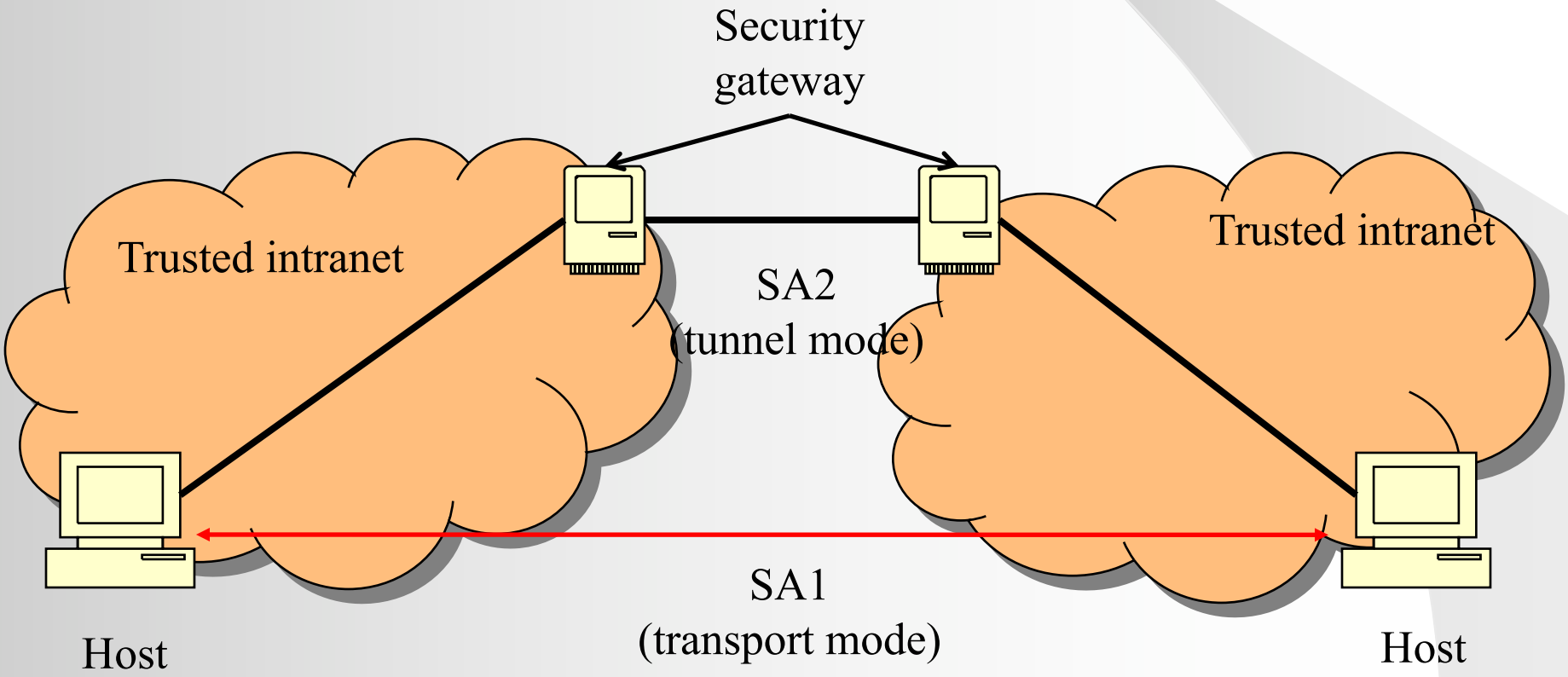
Figure 16.7: Applying ESP in Tunnel Mode to an IPv6 Packet

Security Association (SA)

- Specifies the manners in which packets are protected, includes cryptographic algorithms, keys, IV, lifetimes, sequence numbers and the mode
- Before sending secured data packets from the source to the destination, the source and network hosts handshake and create a network-layer logical connection, called security association (SA)
- For two-way communication between a sender and a receiver, 2 SAs are required
- An SA is a 3-tuple:
  - A security protocol (AH or ESP) identifier
  - The source IP address
  - A 32-bit connection identifier called the Security Parameter Index (SPI): all datagrams in the SA will have the same SPI

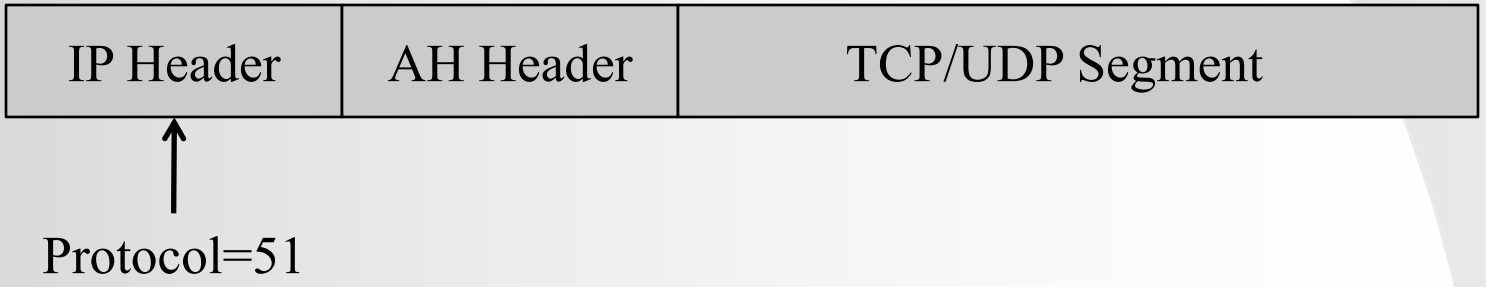
IPsec Modes SA

- Transport mode vs. tunnel mode SA



AH Protocol

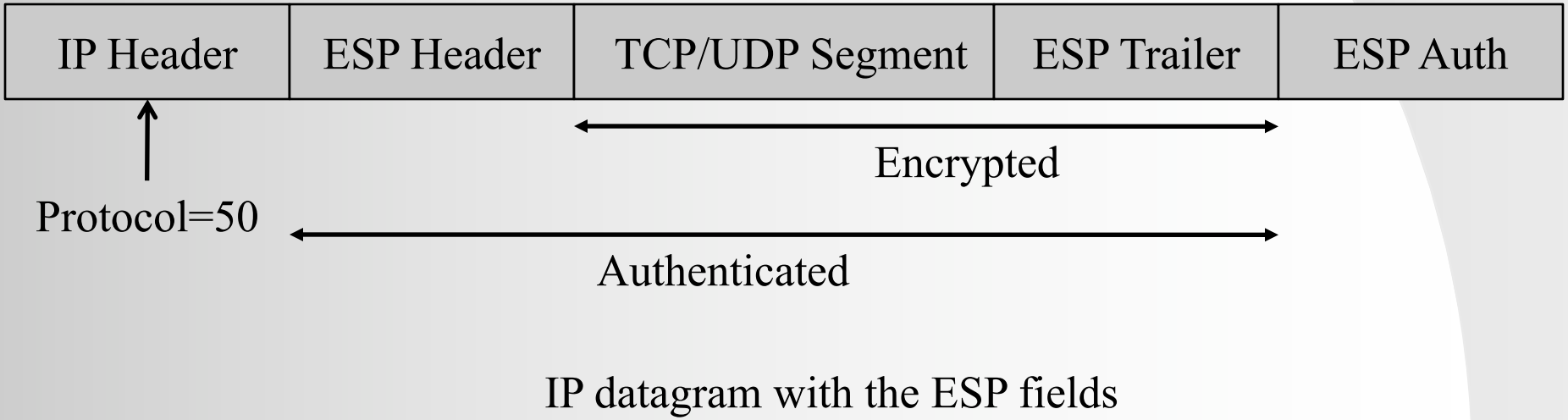
- Supports source host identification and data integrity but not secrecy
- When a source host wants to send datagrams to a particular destination
  - It first establishes an SA with the destination
  - It then send secure datagrams to the destination



IP datagram with the AH header

ESP Protocol

- Supports network-layer secrecy as well as source host authentication
- When a source host wants to send datagrams to a particular destination
  - It first establishes an SA with the destination
  - It then send secure datagrams to the destination



IP datagram with the ESP fields