



# CS670: Network security

IPsec. TLS.

# Sources

---

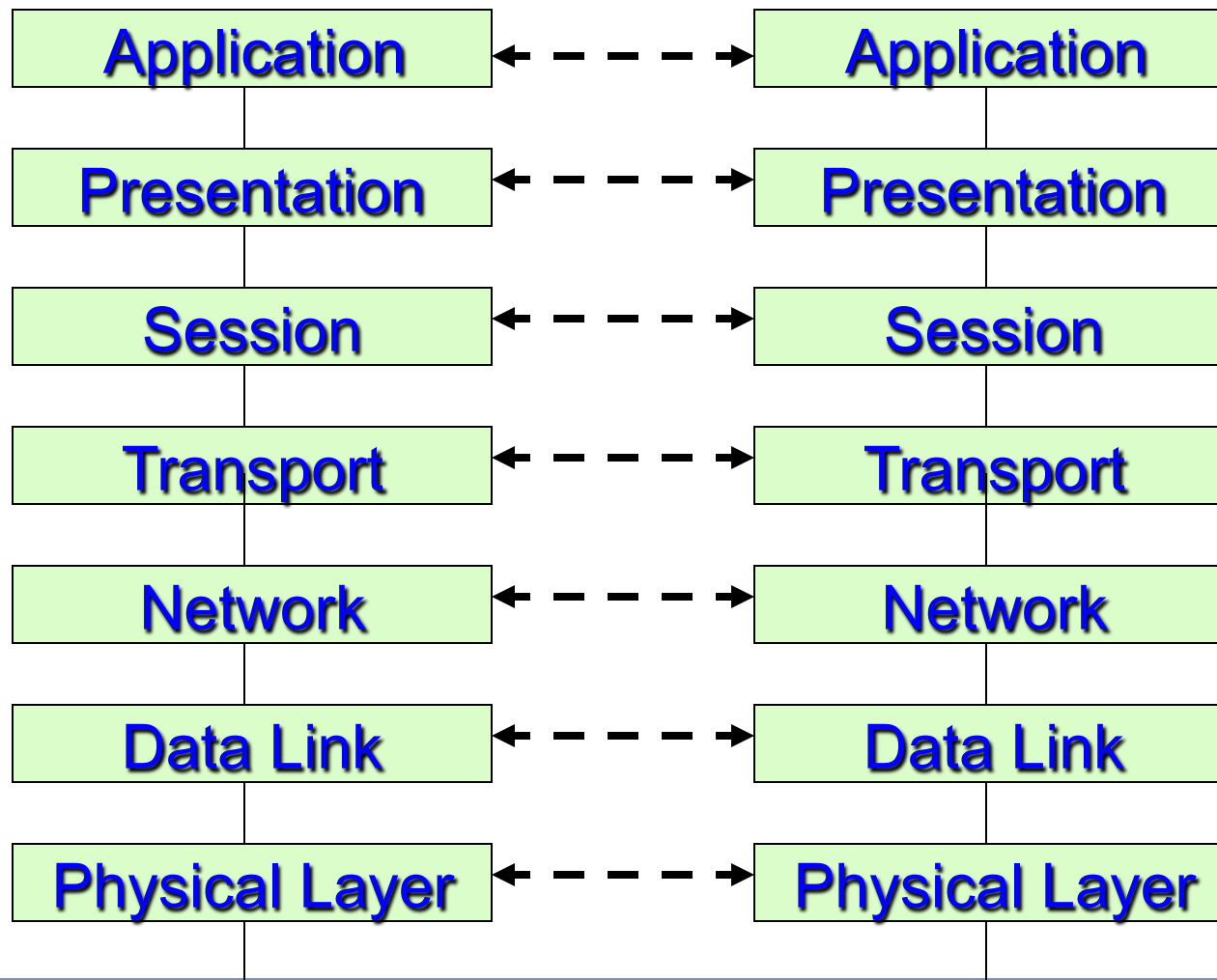
1. Many slides courtesy of Christo Wilson and Wil Robertson
2. IPSec Key management based on slides from B. LaMacchia
3. Analysis of the HTTPS Certificate Ecosystem, IMC 2013:  
<https://jhalderm.com/pub/papers/https-imc13.pdf>
4. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed, IMC 2014:  
<http://www.ccs.neu.edu/home/cbw/pdf/imc254-zhang.pdf>



## 1: Protecting IP: IPsec

# OSI/ISO Model

---



# IPsec design goals

---

- ▶ **Design philosophy:** applications cannot be trusted to implement end-to-end security properly and security should be built into the network itself
  - ▶ Transparent to applications (below transport layer )
- ▶ **Goal:** Facilitate direct IP connectivity between sensitive hosts through untrusted networks
- ▶ It is designed to be extremely flexible
  - ▶ Different crypto algorithms and key exchange supported
  - ▶ Different security services compositions
  - ▶ Different granularities of protection

# Security goals

---

- ▶ IPsec provides:
  - ▶ access control
  - ▶ integrity
  - ▶ data origin authentication
  - ▶ rejection of replayed packets
  - ▶ confidentiality
- ▶ IETF IPSEC Working Group
- ▶ Documented in RFCs and Internet drafts

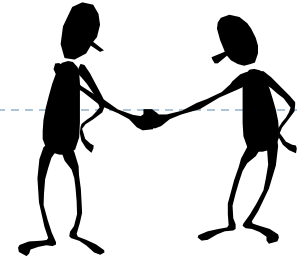
# Security and deployment mechanisms

---

- ▶ Operates based on security associations
- ▶ Deployment:
  - ▶ Transport-mode: encapsulates an upper-layer protocol (e.g. TCP or UDP) and prepends an IP header in clear
  - ▶ Tunnel-mode: encapsulates an entire IP datagram into a new packet adding a new IP header
- ▶ Security:
  - ▶ Authentication Header (AH): provides integrity and authentication without confidentiality
  - ▶ Encapsulating Security Payload (ESP): provides confidentiality and can also provide integrity and authentication

# Security associations (SA)

---



- ▶ A relationship between a sender and a receiver
- ▶ Identified by three parameters:
  - ▶ Security Parameter Index (SPI)
  - ▶ IP Destination address (IP of the destination SA, can be a host, a firewall or a router)
  - ▶ Security Protocol Identifier (ESP or AH)
- ▶ SPI + IP destination address uniquely identifies a particular Security Association
- ▶ SAs are unidirectional, sender supplies SPI to receiver



# Parameters of a security association

---

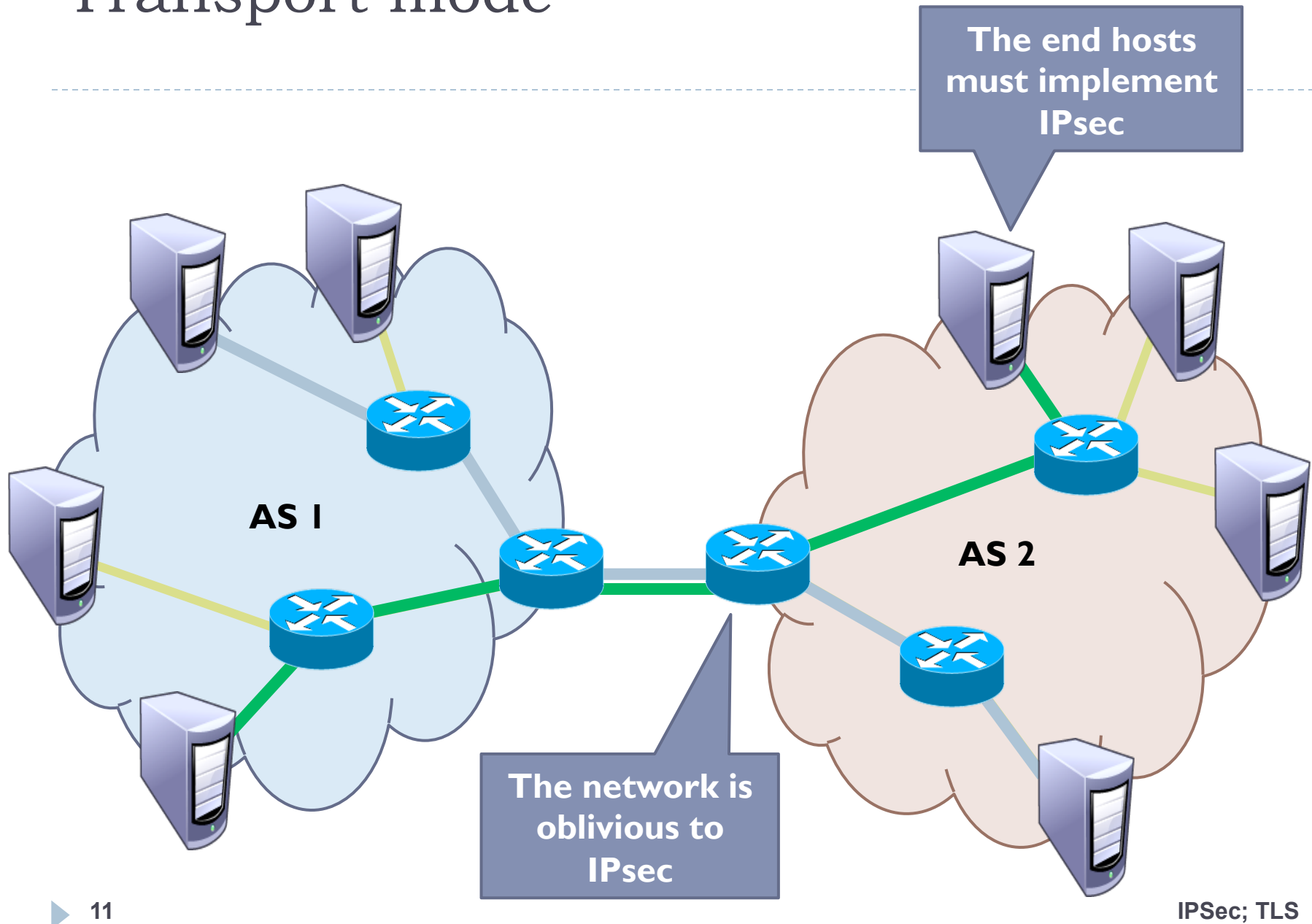
- ▶ Sequence number counter: used to generate a sequence number in headers
- ▶ Sequence counter overflow: if sequence counter overflow should generate an auditable event and prevent further transmission of packets on this SA
- ▶ Anti-replay window: used to determine if an inbound packet is a replay
- ▶ AH information: auth. keys, key lifetime
- ▶ ESP information: encryption, auth., key, key lifetime, initial values
- ▶ Lifetime of the Security Association
- ▶ Protocol Mode: tunnel, transport

# Transport mode

---

- ▶ ESP in Transport Mode: encrypts and optionally authenticates the IP payload (data), but not the IP header.
- ▶ AH in Transport Mode: authenticates the IP payload and selected portions of the IP header.

# Transport mode

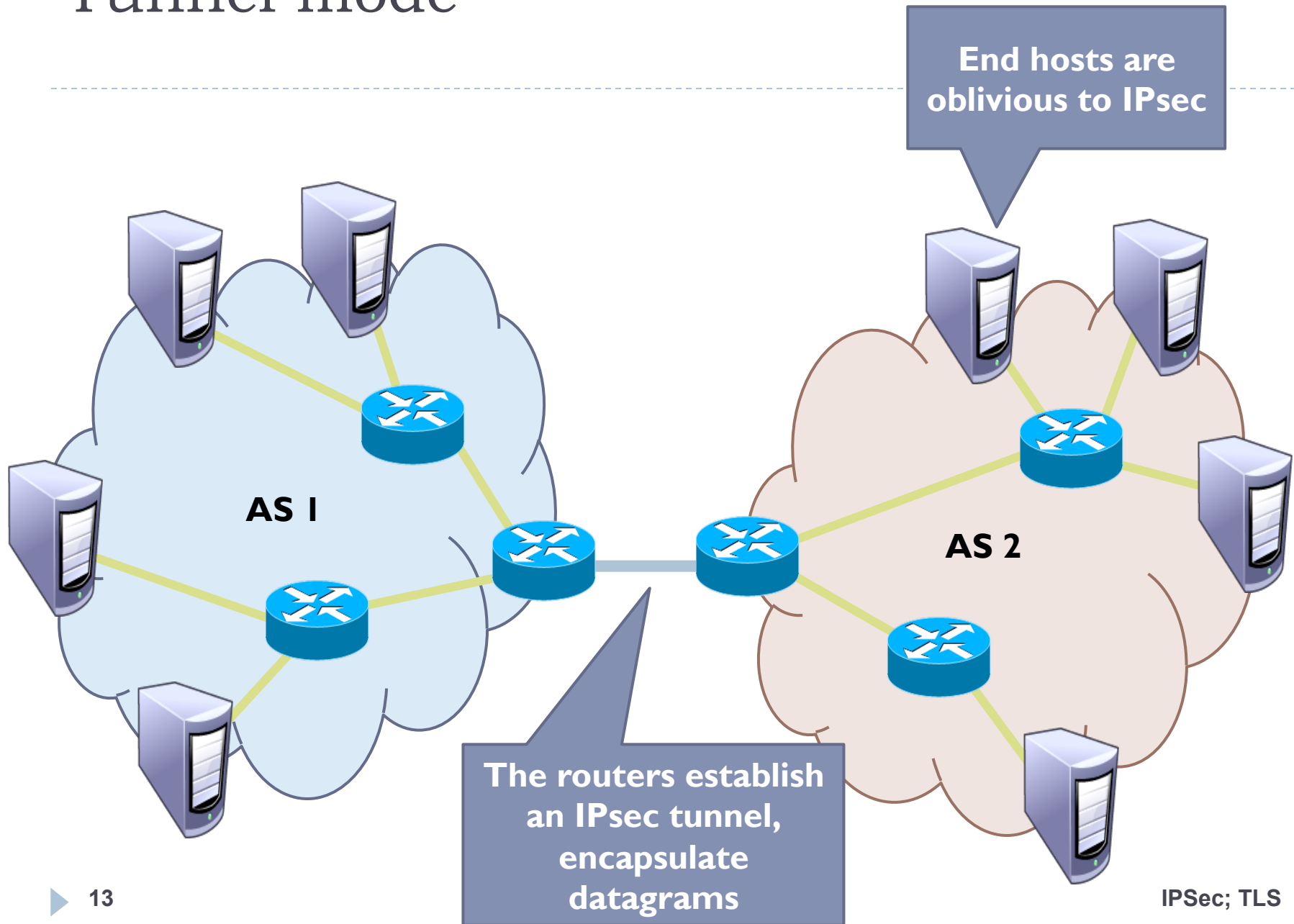


# Tunnel mode

---

- ▶ ESP in Tunnel Mode: encrypts and optionally authenticates the entire inner IP packet, including the inner IP header.
- ▶ AH in Tunnel Mode: authenticates the entire inner IP packet and selected portions of the outer IP header.

# Tunnel mode



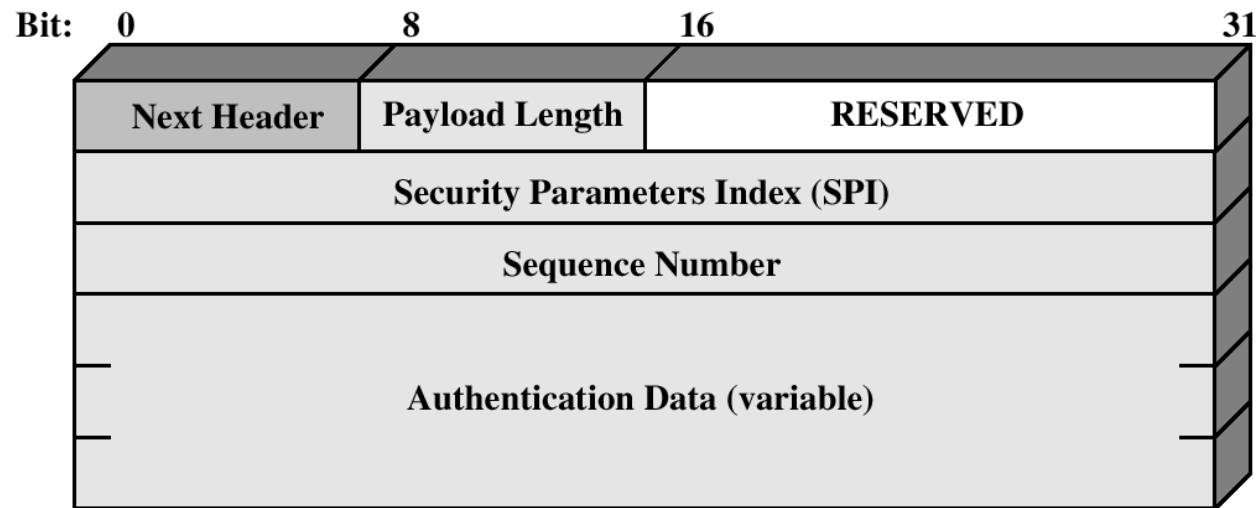
# IPsec headers

---

- ▶ **Authentication Header (AH) – IP protocol #51**
  - ▶ Protects integrity of the IP header and payload data
  - ▶ Provides data origin authentication
  - ▶ Protection against replay attacks
- ▶ **Encapsulating Security Payloads (ESP) – IP protocol #50**
  - ▶ Can be used in encryption+authentication or just authentication mode
  - ▶ Protects integrity (and confidentiality) of payload data (but not the IP header)
  - ▶ Provides data origin authentication
  - ▶ Protection against replay attacks

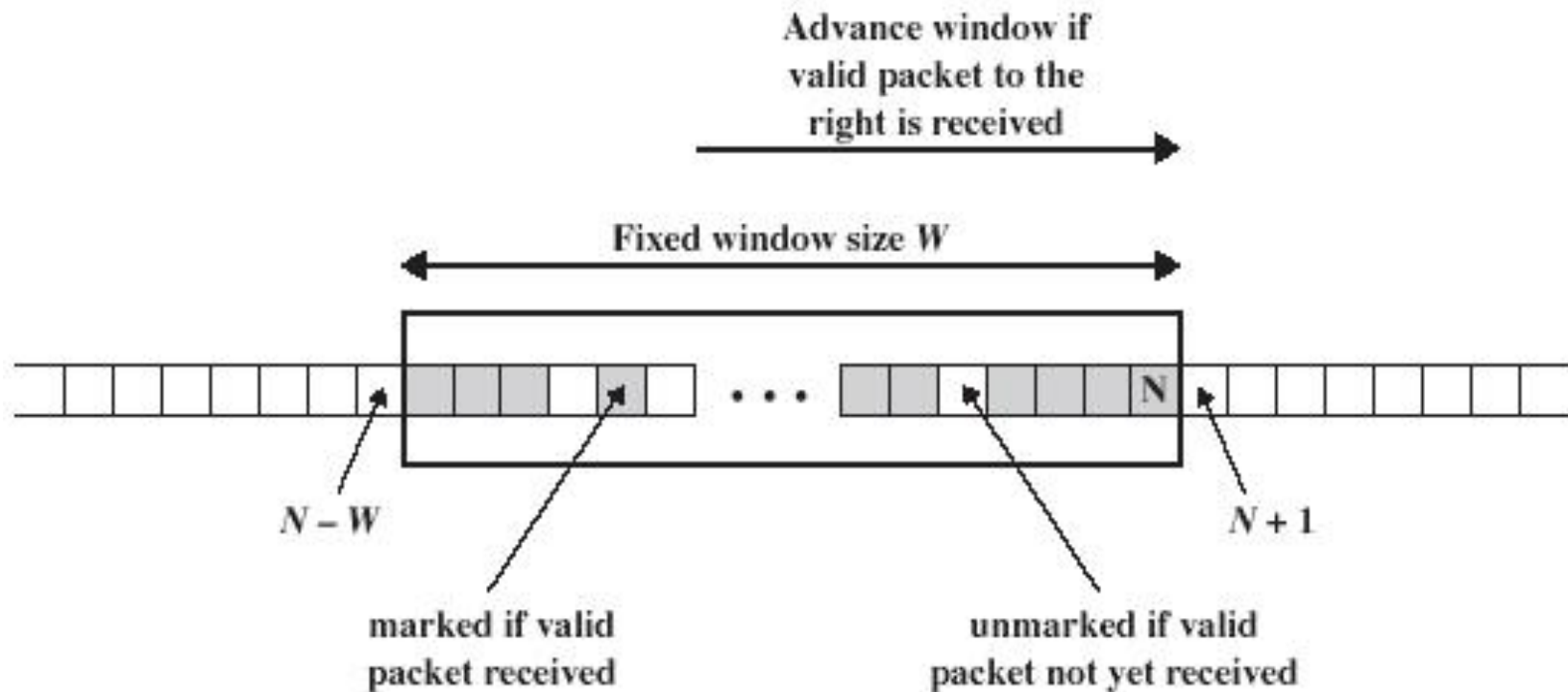
# Authentication header

- ▶ Provides support for data integrity and authentication (MAC) of IP packets, using HMAC based on MD5 or SHA1.
- ▶ Defends against replay attacks (sequence number).



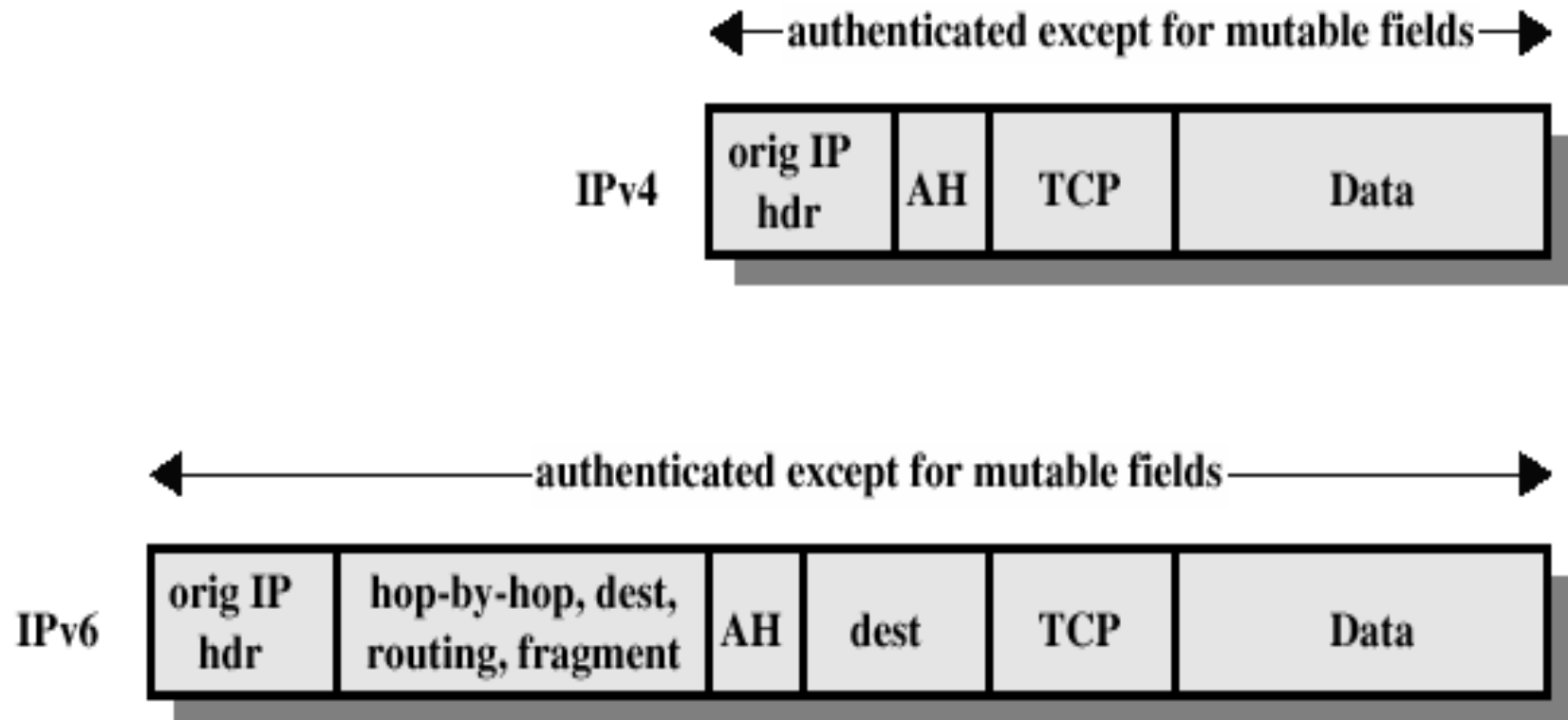
# AH: Preventing replay

- ▶ When a SA is established, sender initializes sequence counter to 0. Window size min 32 packets, preferred 64..
- ▶ Every time a packet is sent the counter is incremented and is set in the sequence number in the AH header.

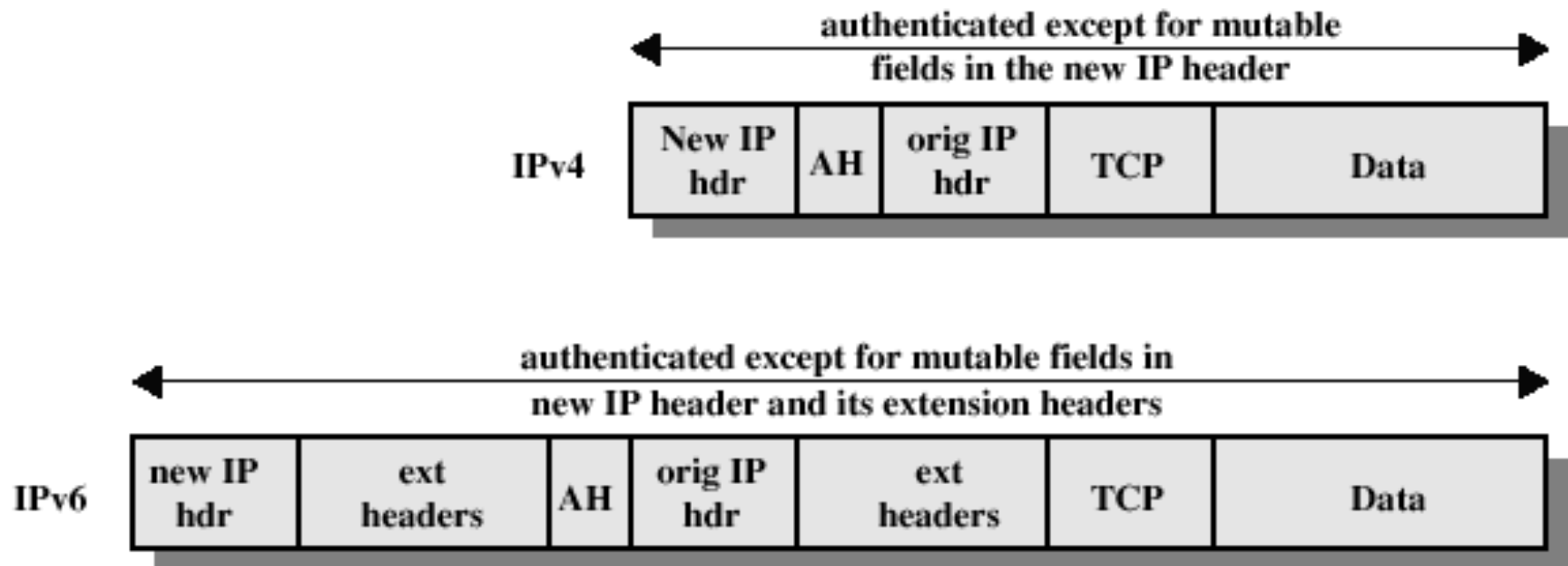




# AH Authentication: Transport mode



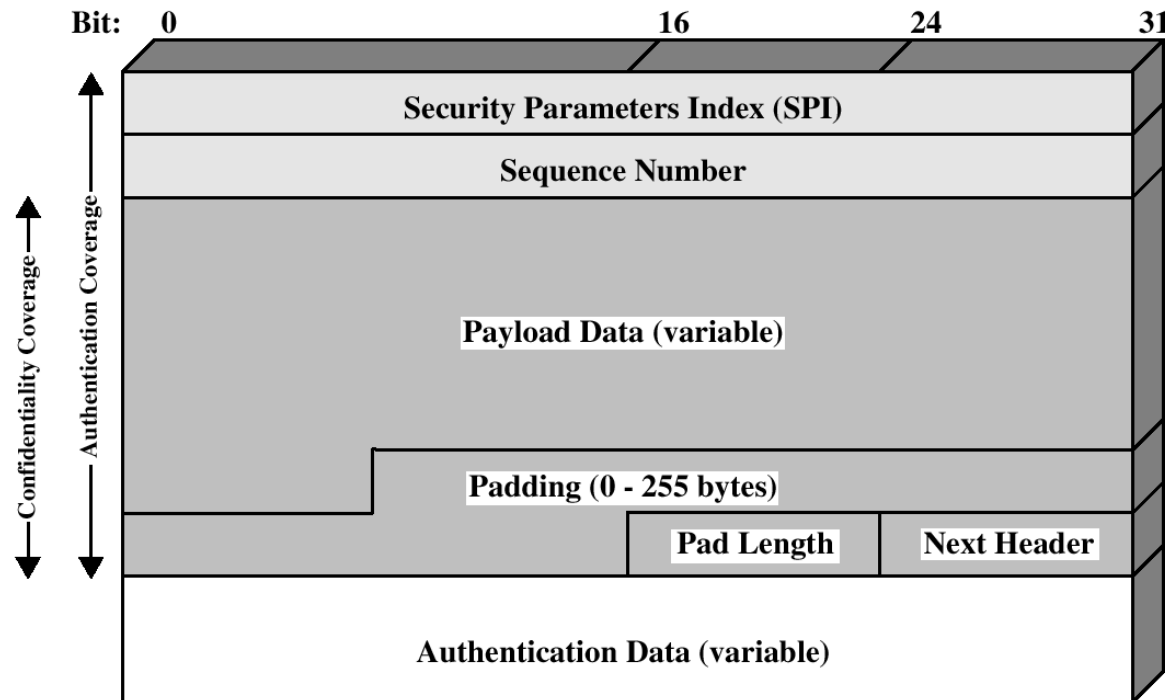
# AH authentication: Tunnel mode



The new IP header contains different IP addresses than the ultimate destination and source

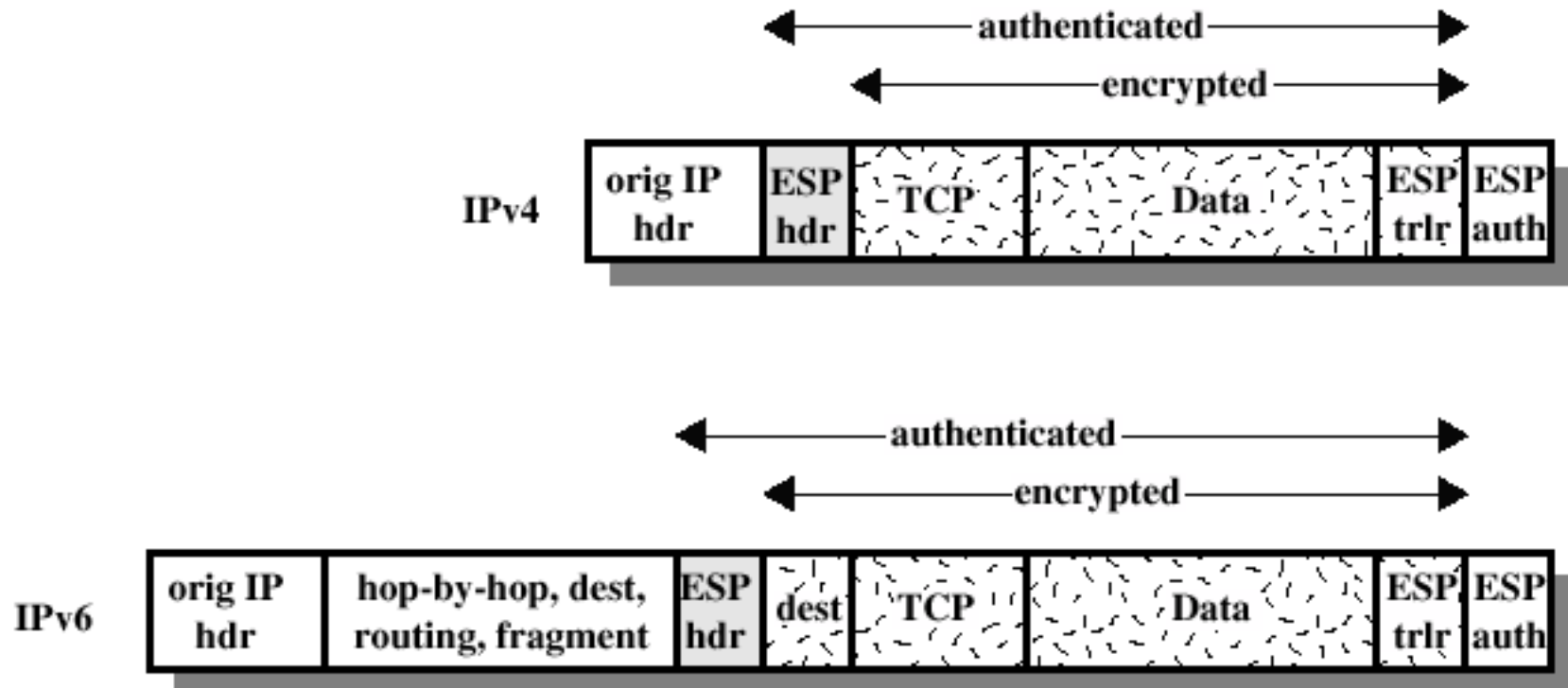
# Encapsulating Security Payload

- ▶ ESP provides confidentiality services, optionally can provide the same services as AH
- ▶ Encryption: 3DES, Blowfish, CAST, IDEA, 3IDEA



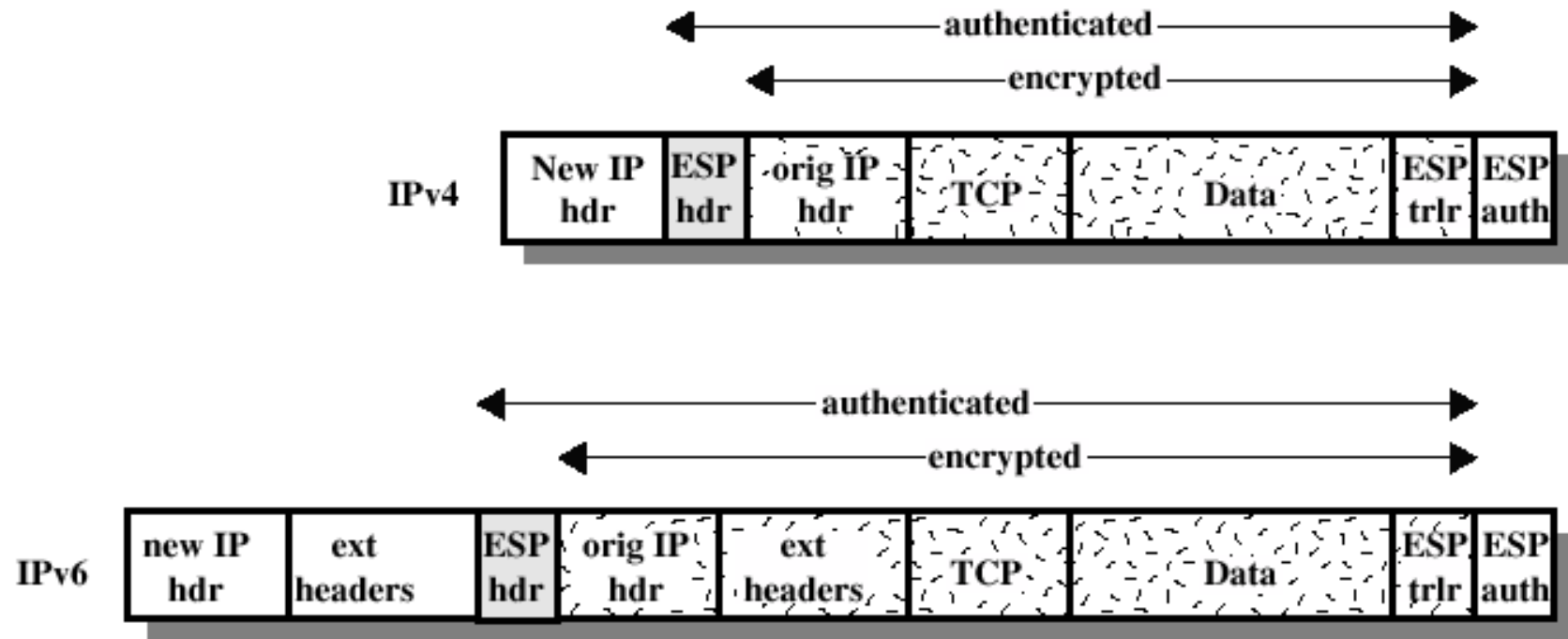
# ESP encryption and authentication: Transport mode

---



# ESP encryption and authentication: Tunnel mode

---



# Cryptographic algorithms

---

- ▶ **Encryption:**
  - ▶ TripleDES-CBC
  - ▶ AES-CBC
  - ▶ AES-GCM: authentication and confidentiality
- ▶ **Authentication:**  
HMAC-SHA1

<http://tools.ietf.org/html/rfc7321>, August 2014

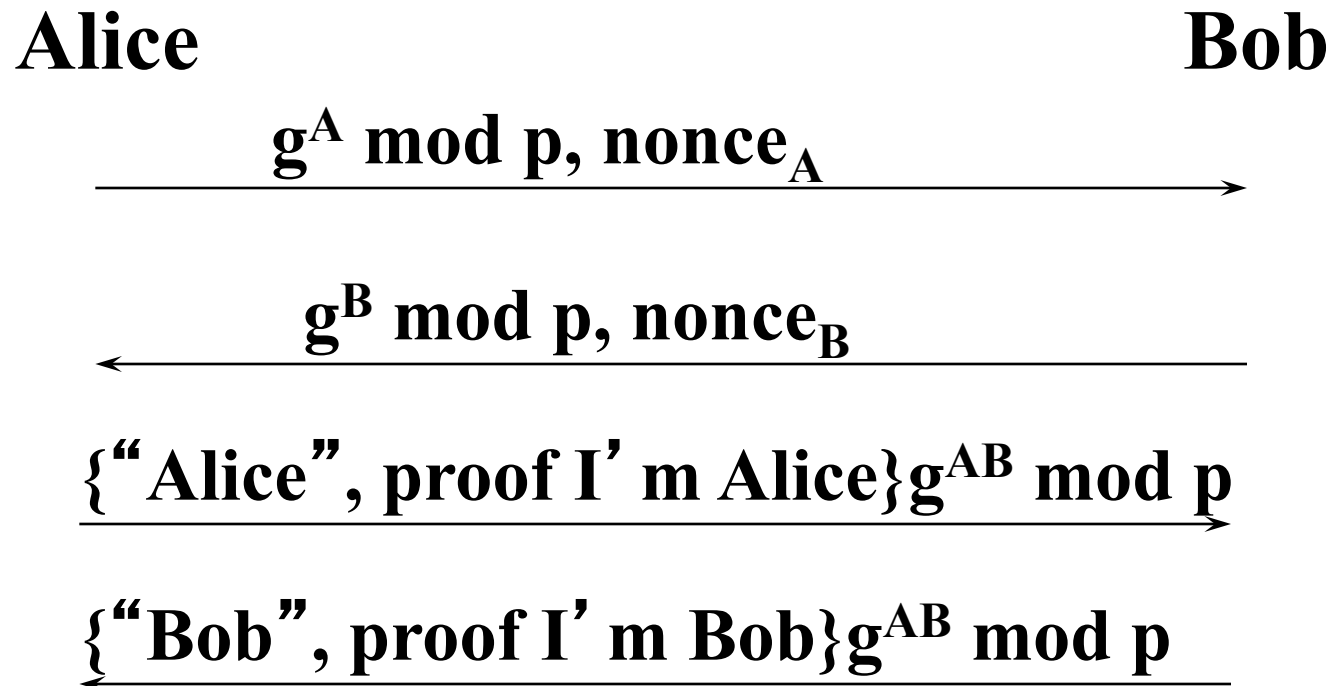
# IPsec key management

---

- ▶ Goal: setting up SA and keys between pairs of communicating hosts
- ▶ Manual: system administrator configures the keys for hosts
- ▶ Automated: on-demand creation of keys
  - ▶ Oakley Key Determination Protocol (based on Diffie-Hellman): authenticated, prevents replays, negotiates global parameters
  - ▶ Internet Security Association and Key Management Protocol (ISAKMP): Internet key management and negotiation, defines procedures and packet formats to establish, negotiate, update, and destroy SAs
  - ▶ IKE: Resynchronize two ends of an IPsec SA: Choose cryptographic keys; Reset sequence numbers to zero; Authenticate endpoints

# General idea of IKEv2

---





# IKEv2, RFC 4306

---

$A \rightarrow B : (g^a \bmod p, N_a)$

$B \rightarrow A : (g^b \bmod p, N_b)$

$$K = f(g^{ab} \bmod p, N_a, N_b)$$

3.  $A \rightarrow B : \{\text{Sign}_K(A, \text{Sign}_A(M1, M2), g^c \bmod p, N_{a2})\}_K$

4.  $B \rightarrow A : \{\text{Sign}_K(B, \text{Sign}_B(M1, M2), g^d \bmod p, N_{b2})\}_K$

$$\text{First session key} = f(g^{cd} \bmod p, N_{a2}, N_{b2})$$

# IKE contenders

---

- ▶ Photuris: Signed Diffie-Hellman, stateless cookies, optional hiding endpoint IDs
- ▶ SKIP: Diffie-Hellman public keys, so if you know someone's public key  $g^B$ , you automatically know a shared secret  $g^{AB}$ . Each msg starts with per-msg key  $S$  encrypted with  $g^{AB}$
- ▶ ISAKMP: “framework”, not a protocol. Complex encodings. Flexible yet constraining.
  - ▶ Phase 1 expensive, establishes a session key with which to negotiate multiple phase 2 sessions

# IKE

---

- ▶ IKE session runs in user space, i.e. over the UDP protocol
- ▶ UDP is best effort, no reliability or ordering
- ▶ The standard specifies:
  - ▶ retransmissions
  - ▶ timeouts
  - ▶ concurrent exchanges
  - ▶ synchronization
  - ▶ protection of the messages pertaining to the key establishment
  - ▶ use of digital certificates

# Why isn't IPsec deployed?

---

- ▶ **Extremely complex to set up**
  - ▶ ISAKMP, IKEv2, IPsec... Too many protocols!
  - ▶ >34 different RFCs!
- ▶ **Bad interactions between NAT and IPsec ESP**
  - ▶ How can the NAT route packets if the TCP header is encrypted?
- ▶ **Application-level security is easier to setup and deploy incrementally**
  - ▶ No need for OS or network-level support
  - ▶ Easy usually wins vs. better :(



## 2: VPNs

# What's a VPN

---

- ▶ A VPN is a virtual network built on top of existing physical networks that can provide a secure communications:
  - ▶ data protection, including confidentiality, integrity
  - ▶ data origin authentication
  - ▶ replay protection
  - ▶ access control.
- ▶ VPNs are used most often to protect communications carried over public networks such as the Internet
- ▶ VPNs can reduce the risks of networking, but they cannot totally eliminate them
- ▶ The most common way to achieve a VPN is using IPsec

# Models for VPN architectures

---

- ▶ **Gateway-to-gateway:** protects communications between two specific networks
  - ▶ Example: main office network and a branch office network
- ▶ **Host-to-gateway:** protects communications between one or more individual hosts and a specific network belonging to an organization.
  - ▶ Example: traveling employees and telecommuters, to gain access to internal organizational services, email, web server, etc
- ▶ **Host-to-host:** protects communication between two specific computers.
  - ▶ Example: small number of users need to use or administer a remote system that requires the use of inherently insecure protocols.

# NIST recommendations for VPN IPSec

---

- ▶ If any of the information that will traverse a VPN should not be seen by non-VPN users, then the VPN must provide confidentiality protection (encryption) for that information.
- ▶ A VPN must use a FIPS-approved encryption algorithm. AES-CBC (AES in Cipher Block Chaining mode) with a 128-bit key is highly recommended; Triple DES (3DES-CBC) is also acceptable. The Data Encryption Standard (DES) is also an encryption algorithm; since it has been successfully attacked, it should not be used.



# NIST recommendations for VPN IPSec

---

- ▶ A VPN must always provide integrity protection.
- ▶ A VPN must use a FIPS-approved integrity protection algorithm. HMAC-SHA-1 is highly recommended. HMAC-MD5 also provides integrity protection, but it is not a FIPS-approved
- ▶ algorithm.
- ▶ A VPN should provide replay protection.

# NIST recommendations for VPN IPSec

---

- ▶ For IKEv1, IKE Security Associations (SAs) should have a lifetime no greater than 24 hours (86400 seconds) and IPsec SAs should have a lifetime no greater than 8 hours (28800 seconds).
- ▶ For IKEv2, IKE SAs should be re-keyed after at most 24 hours and child SAs should be re-keyed after at most 8 hours.

# NIST recommendations for VPN IPSec

---

- ▶ The Diffie-Hellman (DH) group used to establish the secret keying material for IKE and IPsec should be consistent with current security requirements. DH group 2 (1024-bit MODP) should be used for Triple DES and for AES with a 128-bit key. For greater security, DH group 5 (1536-bit MODP) or DH group 14 (2048-bit MODP) may be used for AES.
- ▶ The larger DH groups will result in increased processing time.

# What is NIST

---

- ▶ The National Institute of Standards and Technology (NIST), known between 1901 and 1988 as the National Bureau of Standards (NBS), is a measurement standards laboratory, also known as a National Metrological Institute (NMI), which is a non-regulatory agency of the United States Department of Commerce.
- ▶ Computer Security division: standards and recommendations.
  - ▶ Federal Information Processing Standards (FIPS) Publications are standards issued by NIST  
<http://csrc.nist.gov/publications/PubsFIPS.html>
  - ▶ SP - Special Publications,

# Implementations

---

- ▶ Lots of products
- ▶ VPN is the most common form
- ▶ Open source: OpenSwan, strong Swan

# Take home lessons

---

- ▶ IPSec does not trust applications to implement security
- ▶ IPSec provides: authentication, confidentiality, integrity, anti-replay at IP layer
- ▶ Based on concept of security associations, unidirectional index selected by sender
- ▶ Two type of services; AH and ESP
- ▶ Two deployment modes: tunnel and transport
- ▶ Flexible and complex





## 2: SSL/TLS

# What is Transport Layer Security (TLS)

---

- ▶ Protocol that allows to establish an end-to-end secure channel, providing: confidentiality, integrity and authentication
- ▶ Defines how the characteristics of the channel are negotiated: key establishment, encryption cipher, authentication mechanism
- ▶ Requires reliable end-to-end protocol, so it runs on top of TCP
- ▶ It can be used by other session protocols (such as HTTPS)
- ▶ Several implementations: for example SSLeay, open source implementation ([www.openssl.org](http://www.openssl.org))



# TLS vs. IPSEC

---

- ▶ Security goals are similar
- ▶ IPSec more flexible in services it provides, decouples authentication from encryption
- ▶ Different granularity: IPSec operates between hosts, TLS between processes
- ▶ Performance vs granularity

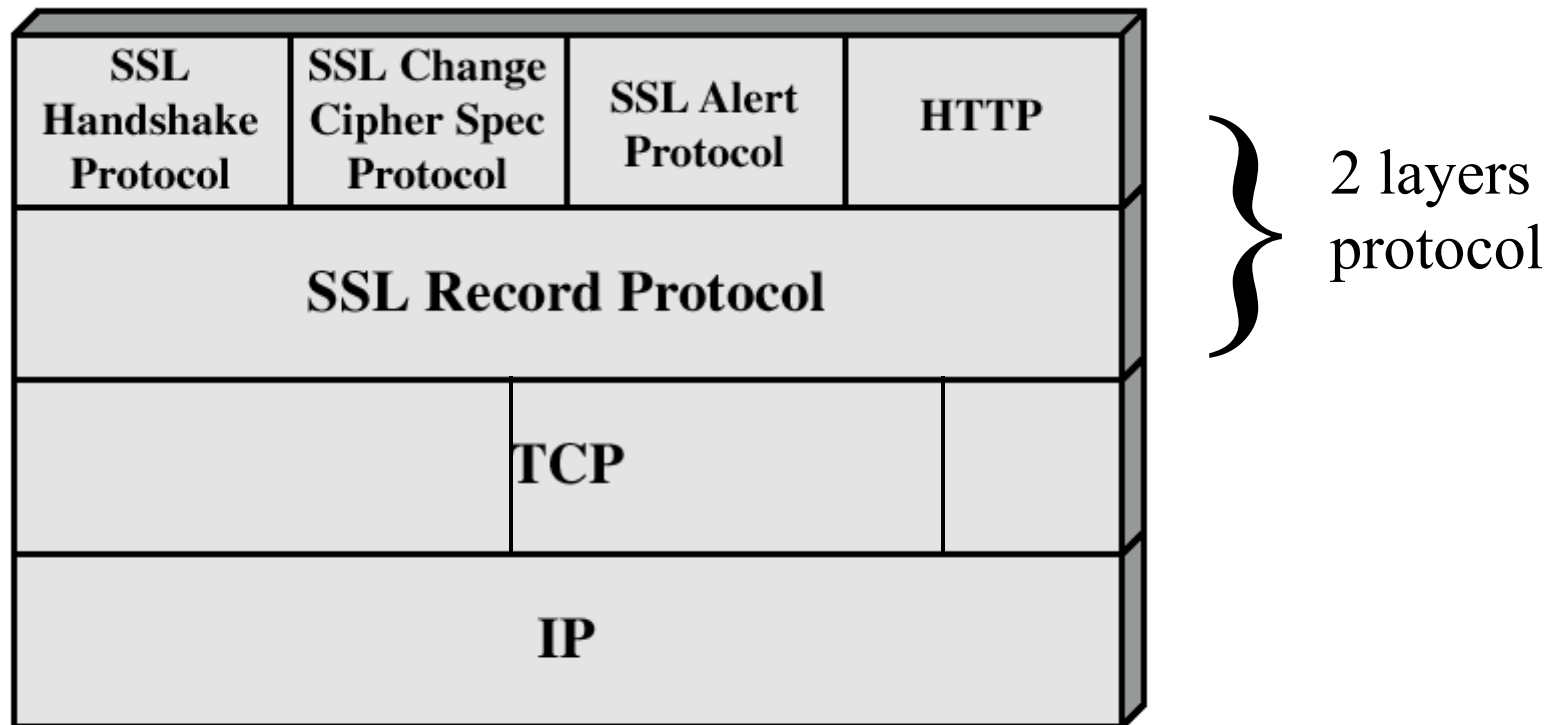
# TLS goals

---

- ▶ **Confidentiality**: Achieved by encryption
- ▶ **Integrity**: Achieved by computing a MAC and send it with the message;
- ▶ **Key exchange**: relies on public key encryption
- ▶ Several version algorithms changed with versions;
- ▶ TLS 1.2:
  - ▶ Replaced the use of MD5-SHA1 with SHA-256
  - ▶ AES, CCM and GCM modes
- ▶ TLS 1.3, draft
  - ▶ <https://tools.ietf.org/html/draft-ietf-tls-rfc5246-bis-00>

# TLS: Protocol architecture

---



# Session and connection

---

## ▶ **Session:**

- ▶ association between a client and a server;
- ▶ created by the Handshake Protocol;
- ▶ defines secure cryptographic parameters that can be shared by multiple connections.

## ▶ **Connection:**

- ▶ end-to-end reliable secure communication;
- ▶ every connection is associated with a session.

# Session

---

- ▶ **Session identifier**: generated by the server to identify an active or resumable session.
- ▶ **Peer certificate**: X 509v3 certificate.
- ▶ **Compression method**: algorithm used to compress the data before encryption.
- ▶ **Cipher spec**: encryption and hash algorithm, including hash size.
- ▶ **Master secret**: 48 byte secret shared between the client and server.
- ▶ **Is resumable**: indicates if the session can be used to initiate new connections.

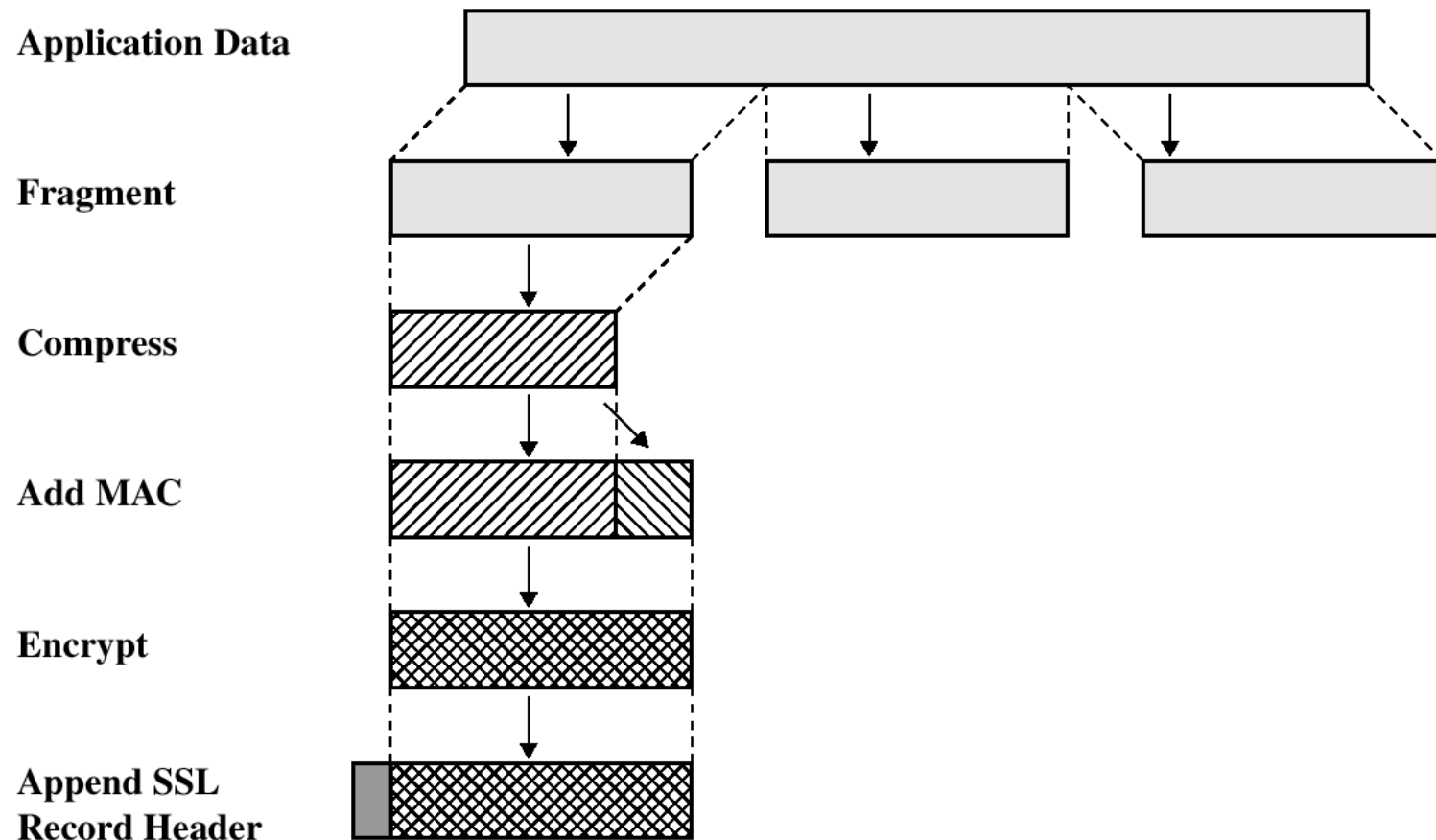
# Connection

---

- ▶ **Server and client random**: chosen for each connection.
- ▶ **Server write MAC secret**: shared key used to compute MAC on data sent by the server.
- ▶ **Client write MAC secret**: same as above for the client
- ▶ **Server write key**: shared key used by encryption when server sends data.
- ▶ **Client write key**: same as above for the client.
- ▶ **Initialization vector**: initialization vectors required by encryption.
- ▶ **Sequence numbers**: both server and client maintains such a counter to prevent replay, **cycle is  $2^{64} - 1$** .

# TLS: SSL Record Protocol

- Provides confidentiality and message integrity using shared keys established by the Handshake Protocol



# Alert Protocol

---

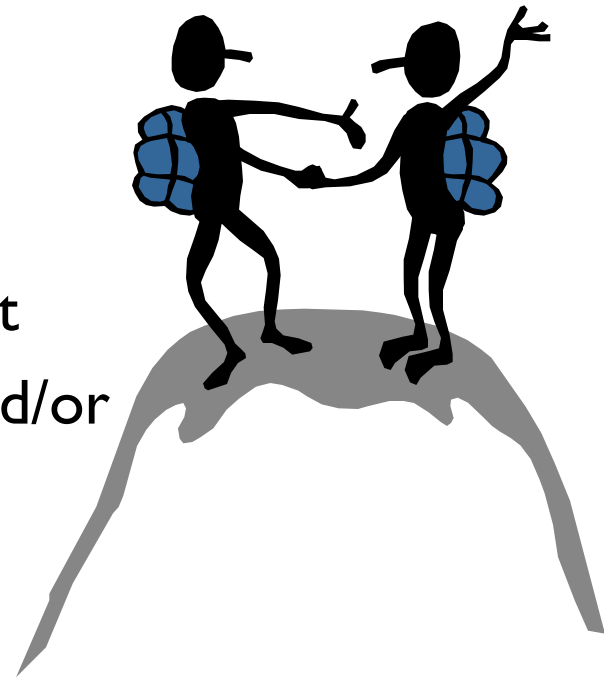
- ▶ Used to send TLS related alerts to peers
- ▶ **Alert messages are compressed and encrypted**
- ▶ Message: two bytes, one defines fatal/warnings, other defines the code of alert
- ▶ Fatal errors: decryption\_failed, record\_overflow, unknown\_ca, access\_denied, decode\_error, export\_restriction, protocol\_version, insufficient\_security, internal\_error
- ▶ Other errors: decrypt\_error, user\_cancelled, no\_renegotiation



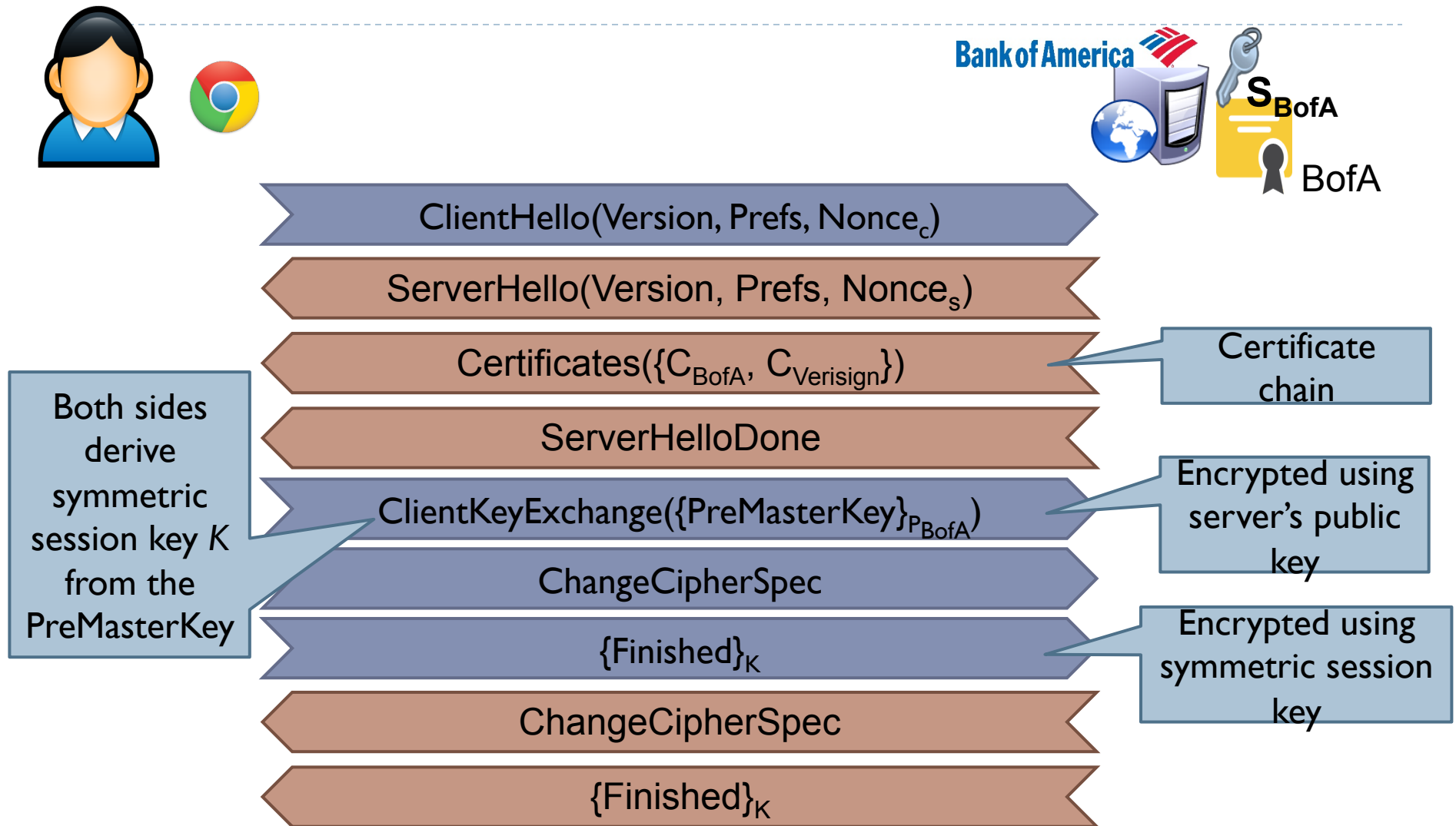
# TLS: Handshake Protocol

---

- ▶ Negotiate Cipher-Suite Algorithms
  - ▶ Symmetric cipher to use
  - ▶ Key exchange method
  - ▶ Message digest function
- ▶ Establish the shared master secret
- ▶ Optionally authenticate server and/or client



# TLS Handshake



# Handshake Protocol: Hello

---

- ▶ Client\_hello\_message has the following parameters:
  - ▶ Version
  - ▶ Random: timestamp + 28-bytes random
  - ▶ Session ID
  - ▶ CipherSuite: cipher algorithms supported by the client, first is key exchange
  - ▶ Compression method
- ▶ Server responds with the same
- ▶ Client may request use of cached session
  - ▶ Server chooses whether to accept or not

# Supported key exchange

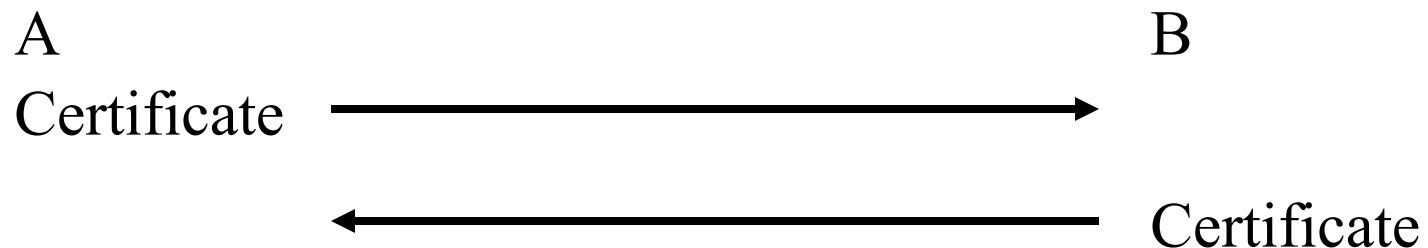
---

- ▶ **RSA:**
  - ▶ shared key encrypted with RSA public key
- ▶ **Fixed Diffie-Hellman:**
  - ▶ public parameters provided in a certificate
- ▶ **Ephemeral Diffie-Hellman:**
  - ▶ the best; Diffie-Hellman with temporary secret key, messages signed using RSA or DSS
- ▶ **Anonymous Diffie-Hellman:**
  - ▶ vulnerable to man-in-the-middle

# TLS: Authentication

---

- ▶ Verify identities of participants
- ▶ Client authentication is optional
- ▶ Certificate is used to associate identity with public key and other attributes, more about this later



# TLS: Change Cipher Spec/Finished

---

- ▶ Change Cipher Spec completes the setup of the connections.
- ▶ Announce switch to negotiated algorithms and values
- ▶ The client sends a message under the new algorithms, allows verification of that the handshake was successful.

# TLS requires digital certificates

---

- ▶ You need a certificate. How do you get one?
- ▶ Option 1: generate a certificate yourself
  - ▶ Use *openssl* to generate a new asymmetric keypair
  - ▶ Use *openssl* to generate a certificate that includes your new public key
  - ▶ Drawback:
    - ▶ Your new cert is self-signed, i.e. not signed by a trusted CA
    - ▶ Browsers cannot validate that the cert is trustworthy
- ▶ Option 2:
  - ▶ Pay a well-known CA to sign your certificate
  - ▶ Any browser that trusts the CA will also trust your new cert

# Certificate authorities (CA)

---

- ▶ CAs are the roots of trust in the TLS PKI
  - ▶ Symantec, Verisign, Thawte, Geotrust, Comodo, GlobalSign, Go Daddy, Digicert, Entrust, and hundreds of others
  - ▶ Issue signed certs on behalf of third-parties
- ▶ How do you become a CA?
  1. Create a self-signed root certificate
  2. Get all the major browser vendors to include your cert with their software
  3. Keep your private key secret at all costs
- ▶ What is the key responsibility of being a CA?
  - ▶ Verify that someone buying a cert for *example.com* actually controls *example.com*



# X.509 Certificate (Part 1)

## Certificate:

Data:

Version: 3 (0x2)

Serial Number:

0c:00:93:10:d2:06:db:e3:37:55:35:80:11:8d:dc:87

Signature Algorithm: sha256WithRSAEncryption

Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert SHA2  
Extended Validation Server CA

Validity

Not Before: Apr 8 00:00:00 2019

Not After : Apr 12 12:00:00 2019

Certificates  
expire

Used for  
revocation

Subject: businessCategory=Private Organization/1.3.6.1.4.1.311.60.2.1.3=US/  
1.3.6.1.4.1.311.60.2.1.2=Delaware/serialNumber=5157550/street=548 4th Street/  
postalCode=94107, C=US, ST=California, L=San Francisco, O=GitHub, Inc.,  
CN=github.com

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (2048 bit)

Modulus:

00:b1:d4:dc:3c:af:fd:f3:4e:ed:cd:67:ad:e6:c

Github's public  
key

- Subject: who owns this cert?
- This is Github's certificate
- Must be served from github.com

IPSec; TLS

# X.509 Certificate (Part 2)

## X509v3 extensions:

X509v3 Subject Alternative Name:

DNS:github.com, DNS:www.github.com

Additional DNS names  
that may serve this  
cert

X509v3 CRL Distribution Points:

Full Name:

URI:http://crl3.digicert.com/sha2-ev

Full Name:

URI:http://crl4.digicert.com/sha2-ev-server-gl.crl

If this cert is  
revoked, it's serial  
will be in the lists at  
these URLs

X509v3 Certificate Policies:

Policy: 2.16.840.1.1.14412.2.1

CPS: https://www.digicert.com/

Authority Information Access:

OCSP - URI:http://ocsp.digicert.com

Policy numbers are  
magic (more on this  
later)

This cert's revocation  
status may also be checked  
via OCSP

IPSec; TLS

# TLS Certificate Authentication

---

- ▶ During the TLS handshake, the client receives a **certificate chain**, i.e. the server's cert, as well as the certs of the signing CA(s)
- ▶ The client must **validate** the certificate chain to establish trust
  - ▶ Does the server's DNS name match the common name in the cert?
    - ▶ E.g. *example.com* cannot serve a cert with common name *google.com*
  - ▶ Are any certs in the chain expired?
  - ▶ Is the CA's signature cryptographically valid?
  - ▶ Is the cert of the root CA in the chain present in the client's trusted key store?
- ▶ 59 ▶ Is any cert in the chain **revoked**? (more on this later)

# Extended Validation Certificates

---

- ▶ What differs between a DV and an EV certs?
  - ▶ To get a DV cert, the CA verifies that you control the given common name
  - ▶ To get an EV cert, the CA does a background check on you and your company; EV certs cost a lot more than DV certs
  - ▶ Other than the background check, EV certs offer the same security as DV certs
- ▶ How does your browser tell the difference between DV and EV certs? Uses the *policy number* in the X.509 certificate?
  - ▶ Each CA designates certain **magic** policy numbers to indicate EV status
  - ▶ Your browser contains a hard-coded list of magic policy numbers to identify EV certs :(

# Let's encrypt

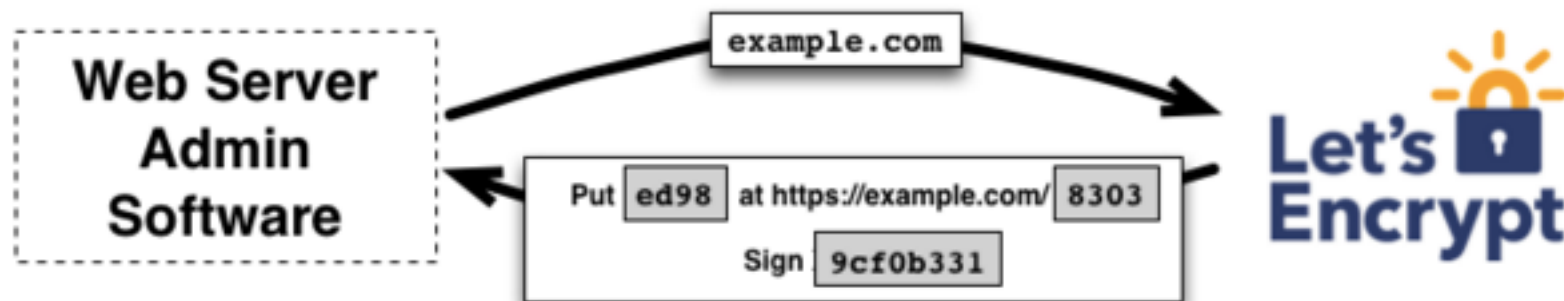
---

- ▶ Free certificate service from a non-profit organization that includes stakeholders from academia and industry
- ▶ Very new, first certificate was issued in Sept. 2015
- ▶ 90 days lifetime: to be proactive to certificate compromised and encourage automation of renewal
- ▶ Revocation integrated with CRL and OCSP

# Let's encrypt: validation

---

- ▶ Provisioning a DNS record under example.com,
- ▶ Provisioning an HTTP resource under a well-known URI on https://example.com/





## 3: Problems with TLS

# Problems with TLS

---

- ▶ TLS is a widely deployed and extremely successful protocol
- ▶ ... but its not perfect
- ▶ Problems with TLS:
  1. CA trustworthiness
  2. Weak cyphers and keys
  3. Protocol attacks
  4. Man-in-the-middle attacks
  5. Secret key compromise
  6. Implementation bugs



# CAs, Revisited

---

- ▶ A CA is essentially a trusted third party
  - ▶ Certificate signatures are attestations of authenticity for the server and (optionally) the client
  - ▶ Remember: **trust is bad and should be minimized!**
- ▶ If a CA mistakenly (or purposefully) signs a certificate for a domain and provides it to a malicious principal, TLS can be subverted
  - ▶ Recall: any CA can sign a cert for any domain
- ▶ Not only must we trust root CAs, but also **intermediate CAs** that have been delegated signing authority

# CA Trustworthiness

---

- ▶ Clearly, the CA secret key must be protected at all costs
  - ▶ Possession of the CA secret key grants adversaries the ability to sign any domain
  - ▶ Attractive target for adversaries
- ▶ Signatures should only be issued after verifying the identity of the requester
  - ▶ Basic verification = Domain Validation
  - ▶ Expensive verification = Extended Validation
  - ▶ Should be easy, right?

# CA Failures

---

Issued to: Microsoft Corporation  
Issued by: VeriSign Commercial Software Publishers CA  
Valid from 1/29/2001 to 1/30/2002  
Serial number is 1B51 90F7 3724 399C 9254 CD42 4637 996A

Issued to: Microsoft Corporation  
Issued by: VeriSign Commercial Software Publishers CA  
Valid from 1/30/2001 to 1/31/2002  
Serial number is 750E 40FF 97F0 47ED F556 C708 4EB1 ABFD

- ▶ In 2001, Verisign issued two executable signing certificates to someone **claiming** to be from Microsoft
  - ▶ Could be used to issue untrusted software updates

# Independent Iranian hacker claims responsibility for Comodo hack

Posts claiming to be from an Iranian hacker responsible for the Comodo hack ...

by Peter Bright - Mar 28 2011, 11:15am EDT

65

```
1. Hello
2.
3. I'm writing this to the world, so you'll know more about me..
4.
5. At first I want to give some points, so you'll be sure I'm the hacker:
6.
7. I hacked Comodo from InstantSSL.it, their CEO's e-mail address mfpenco@mfpenco.com
8. Their Comodo username/password was: user: gtadmin password: [trimmed]
9. Their DB name was: globaltrust and instantsslcms
```

The alleged hacker's claim of responsibility on pastebin.com

The hack that resulted in **Comodo creating certificates** for popular e-mail providers including Google Gmail, Yahoo Mail, and Microsoft Hotmail has been claimed as the work of an independent Iranian patriot. A **post** made to data sharing site pastebin.com by a person going by the handle "comodohacker" claimed responsibility for the hack and described details of the attack. A second **post** provided source code apparently reverse-engineered as one of the parts of the **hack**; TLS

## Another fraudulent certificate raises the same old questions about certificate authorities

For the second time this year, Iranian hackers have created a fraudulent ...

by Peter Bright - Aug 29 2011, 11:12pm EDT

42

Earlier this year, an [Iranian hacker](#) broke into servers belonging to a reseller for certificate authority Comodo and issued himself a range of certificates for sites including Gmail, Hotmail, and Yahoo! Mail. With these certificates, he could eavesdrop on users of those mail providers, even if they use SSL to protect their mail sessions.

It's happened again. This time, Dutch certificate authority DigiNotar has issued a fraudulent certificate for google.com and all subdomains. As before, Gmail appears to be the target. The perpetrator also appears to be Iranian, with [reports](#) that the certificate has been used in the wild for man-in-the-middle attacks in that country. The certificate was issued on July 10th, and so could have been in use for several weeks prior to its discovery.

DigiNotar has revoked the certificate, which provides some protection to users (though many applications do not bother checking for revocations). However, the company has so far not disclosed how the certificate was issued in the first place, making it unclear that its integrity has been restored. As a result, Google and Mozilla have both made patches to [Chrome](#) and [Firefox](#) respectively that blacklist the entire certificate authority.

IPSec; TLS

# TrustWave

## Trustwave issued a man-in-the-middle certificate



Certificate authority [Trustwave](#) issued a certificate to a company allowing it to issue valid certificates for any server. This enabled the company to listen in on encrypted traffic sent and received by its staff using services such as Google and Hotmail. Trustwave has since revoked the CA certificate and [vowed](#) to refrain from issuing such certificates in future.

According to Trustwave, the CA certificate was used in a data loss prevention (DLP) system,



All of the major browsers contain the Trustwave root certificate

# Weak cipher suites

---

- ▶ TLS allows the use of different cryptographic algorithms
- ▶ Known weaknesses in RC4 and MD5

Cipher Suite	Usage in Certs (as of 2013)
RC4-MD5	2.8%
RC4-SHA1	48.9%
AES128-SHA1	1.2%
AES256-SHA1	46.3%

# Weak keys

---

- ▶ The ZMap team constantly collects all TLS certificates visible in the IPv4 address space
  - ▶ <http://zmap.io/> (data at <https://scans.io/>)
  - ▶ Currently, around 8.3 million certs being served on the Internet
- ▶ Observed repeated keys in-the-wild due to low entropy
  - ▶ Some systems auto-generate TLS keys at boot
  - ▶ Low boot-time entropy results in duplicate keys
- ▶ Default TLS keys often shipped in network devices
  - ▶ Attackers can extract private keys from firmware!



# Protocol attacks (1)

---

- ▶ Renegotiation attacks

- ▶ Allows attacker to **renegotiate** a connection to the NULL algorithm and inject plaintext data
- ▶ Fixed by requiring cryptographic verification of previous TLS handshakes

- ▶ Version downgrade attacks

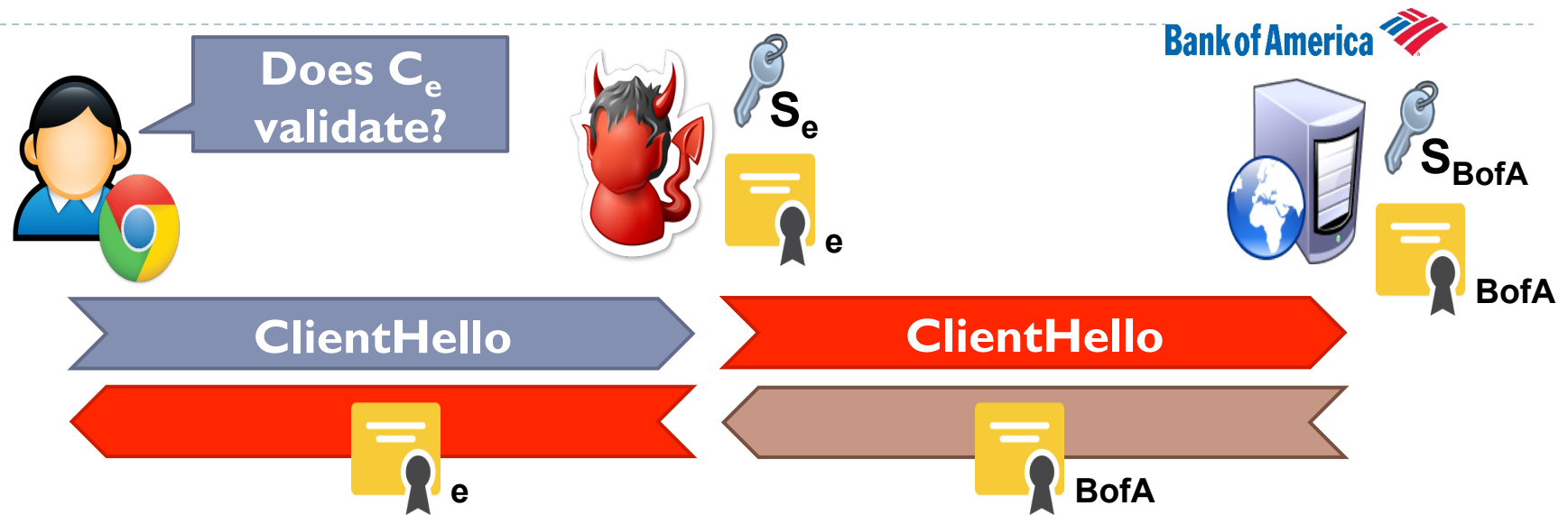
- ▶ False Start TLS extension allows attackers to modify the cipher suite list the client sends to server during handshake
- ▶ Can force the usage of a known insecure cipher

# Protocol attacks (2)

---

- ▶ **Padding Oracle On Downgraded Legacy Encryption (POODLE)**
  - ▶ Cryptographic attack against CBC-mode cyphers when used with SSL 3.0
  - ▶ Attacker can use a downgrade attack to force TLS connections into SSL 3.0
  - ▶ Allowing security degradation for the sake of interoperability is **dangerous**

# TLS Man-in-the-Middle attack



- ▶ If  $C_e$  is self-signed, the user will be shown a warning
- ▶ If the attacker steals  $C_{BofA}$  and  $S_{BofA}$ , then this attack will succeed unless:
  1. Bank of America **revokes** the stolen cert
  2. The client checks to see if the cert has been revoked
- ▶ If the attacker manages to buy a valid BofA cert from a CA, then the only defense against this attack is **certificate pinning**

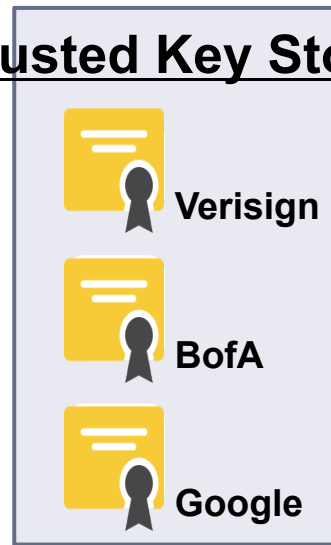
# Certificate pinning

---

- ▶ Certificate pinning is a technique for detecting sophisticated MitM attacks
  - ▶ Browser includes certs from well-known websites in the trusted key store by default
  - ▶ Usually, only certs from root CAs are included in the trusted key store
- ▶ Example: Chrome ships with pinned copies of the \*.google.com certificate
- ▶ Pinning isn't just for browsers
  - ▶ Many Android and iPhone apps now include pinned certificates
  - ▶ E.g. Facebook's apps include a pinned cert



## Trusted Key Store



IPSec; TLS

# Secret key compromise

---

- ▶ Secret key compromise leads to many devastating attacks
  - ▶ Attacker can successfully MitM TLS connections (i.e. future connections)
  - ▶ Attacker can decrypt historical TLS packets encrypted using the stolen key
- ▶ Changing to a new keypair/cert does not solve the problem!
  - ▶ The old, stolen key is still valid!
  - ▶ Attacker can still MitM connections!

# Certificate expiration

---

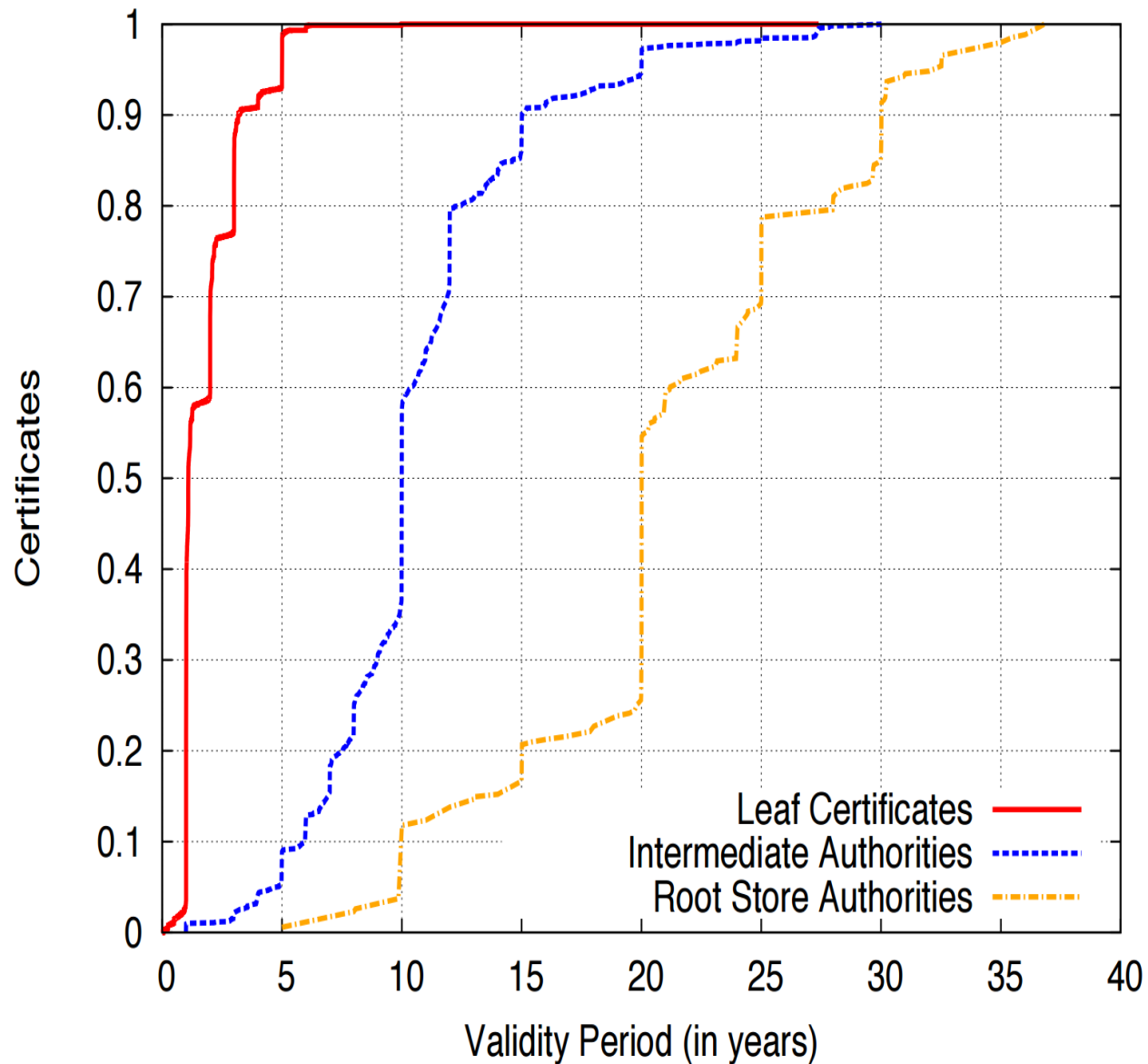
- ▶ Certificate expiration is the simplest, most fundamental defense against secret key compromise
  - ▶ All certificates have an expiration date
  - ▶ A stolen key is only useful before it expires
- ▶ All certs should have a short lifetime
  - ▶ Months, weeks, or even days
- ▶ In reality most certs have a one year lifetime
  - ▶ This gives an attacker plenty of time to abuse a stolen key

## **X.509 Certificate**

### **Validity**

**Not Before: Apr 8  
00:00:00 2014 GMT  
Not After : Apr 12  
12:00:00 2016 GMT**

# Certificate lifetimes



# Perfect Forward Secrecy

---

- ▶ Perfect Forward Secrecy (PFS) addresses the issue of an attacker decrypting past TLS sessions after a secret key compromise
- ▶ Uses Diffie-Hellman to compute the TLS session key
  - ▶ Session key is never sent over the wire, and is discarded after the session completes
  - ▶ Since the session key cannot be recovered, the attacker cannot decrypt historical TLS packets, even if they hold the secret key
- ▶ PFS does not prevent MiTM attacks; future TLS sessions are still in danger



<https://weakdh.org>

**Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice, D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann, Best Paper Award CCS 2015**

- ▶ Summary: Logjam, active MITM attack that downgrades TLS to 512-bit DHE export-grade cipher suites. They broke a 512 prime (many sites use the same one), estimate that an academic team can break a 768-bit prime and that a nation-state can break a 1024-bit prime.
- ▶ Impact: TLS with support for export cipher and any protocol using DH with 1024 or less and reusing the prime.
- ▶ What to do: Disable support for export cipher suites and use a 2048-bit Diffie-Hellman group

# Revocation

---

- ▶ Certificate **revocations** are another fundamental mechanism for mitigating secret key compromises
  - ▶ After a secret key has been compromised, the owner is supposed to **revoke** the certificate
- ▶ CA's are responsible for hosting databases of revoked certificates that they issued
- ▶ Clients are supposed to query the revocation status of all certificates they encounter during validation and not accept the revoked ones
- ▶ Two revocation protocols for TLS certificates
  1. Certificate Revocation Lists (CRLs)
  2. Online Certificate Status Protocol (OCSP)

# Certificate Revocation Lists (CRL)

---

- ▶ CRLs are the original mechanism for announcing and querying the revocation status of certificates
- ▶ CAs compile lists of serial numbers of revoked certificates
  - ▶ URL for the list is included in each cert issued by the CA
  - ▶ CRL is signed by the CA to protect integrity

# X.509 Certificates, Revisited

Certificate:

Data:

Subject: businessCategory=Private Organization/  
1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=Delaware/  
serialNumber=5157550/street=548 4th Street/  
postalCode=94107, C=US, ST=California, L=San Francisco,  
O=GitHub, Inc., CN=github.com

If the cert is revoked, this  
serial number will appear  
in the CRL

X509v3 extensions:

X509v3 Subject Alternative Name:

DNS:github.com, DNS:www.github.com

X509v3 CRL Distribution Points:

Full Name:

URI:http://crl3.digicert.com/sha2-ev-server-g1.crl

Full Name:

URI:http://crl4.digicert.com/sha2-ev-server-g1.crl

URLs where clients  
can find the CRLs  
for this cert

Authority Information Access:

OCSP - URI:http://ocsp.digicert.com

# Problems with CRLs

---

- ▶ Clients should check the revocation status of every cert they encounter, i.e, leaf, intermediate, and root certs
- ▶ Problems
  - ▶ Latency – additional RTTs of latency are needed to check CRLs before a page will load
  - ▶ Size – CRLs can grow to be quite large (~MBs), downloads may be slow
  - ▶ MitM attackers can block access to the CRL/OCSP URLs
    - ▶ Browsers default-accept certificates if the revocation status cannot be checked
- ▶ Does caching CRLs mitigate these performance problems?
  - ▶ Yes, somewhat
  - ▶ But caching CRLs for long periods is dangerous: they may be out of date

# Online Certificate Status Protocol

---

- ▶ **OCSP is the modern replacement for CRLs**
  - ▶ API-style protocol that allows clients to query the revocation status of one or more certs
  - ▶ No longer necessary to download the entire CRL
- ▶ **CA's host an OCSP server that clients may query**
  - ▶ OCSP URL included in OCSP-compliant certs
  - ▶ Responses are signed by the CA to maintain integrity
  - ▶ Responses also include an expiration date to prevent replay attacks

# X.509 Certificates, Revisited

Certificate:

Data:

Subject: businessCategory=Private Organization/  
1.3.6.1.4.1.311.60.2.1.3=US/1.3.6.1.4.1.311.60.2.1.2=Delaware/  
serialNumber=5157550/street=548 4th Street/  
postalCode=94107, C=US, ST=California, L=San Francisco,  
O=GitHub, Inc., CN=github.com

X509v3 extensions:

X509v3 Subject Alternative Name:

DNS:github.com, DNS:www.github.com

X509v3 CRL Distribution Points:

Full Name:

URI:http://crl3.digicert.com/sha2-ev-server-gl.crl

Full Name:

URI:http://crl4.digicert.com/sha2-ev-se

Authority Information Access:

OCSP - URI:http://ocsp.digicert.com

Query the serial number  
to see if this cert has  
been revoked

URLs where clients  
can find the OCSP  
server for this cert

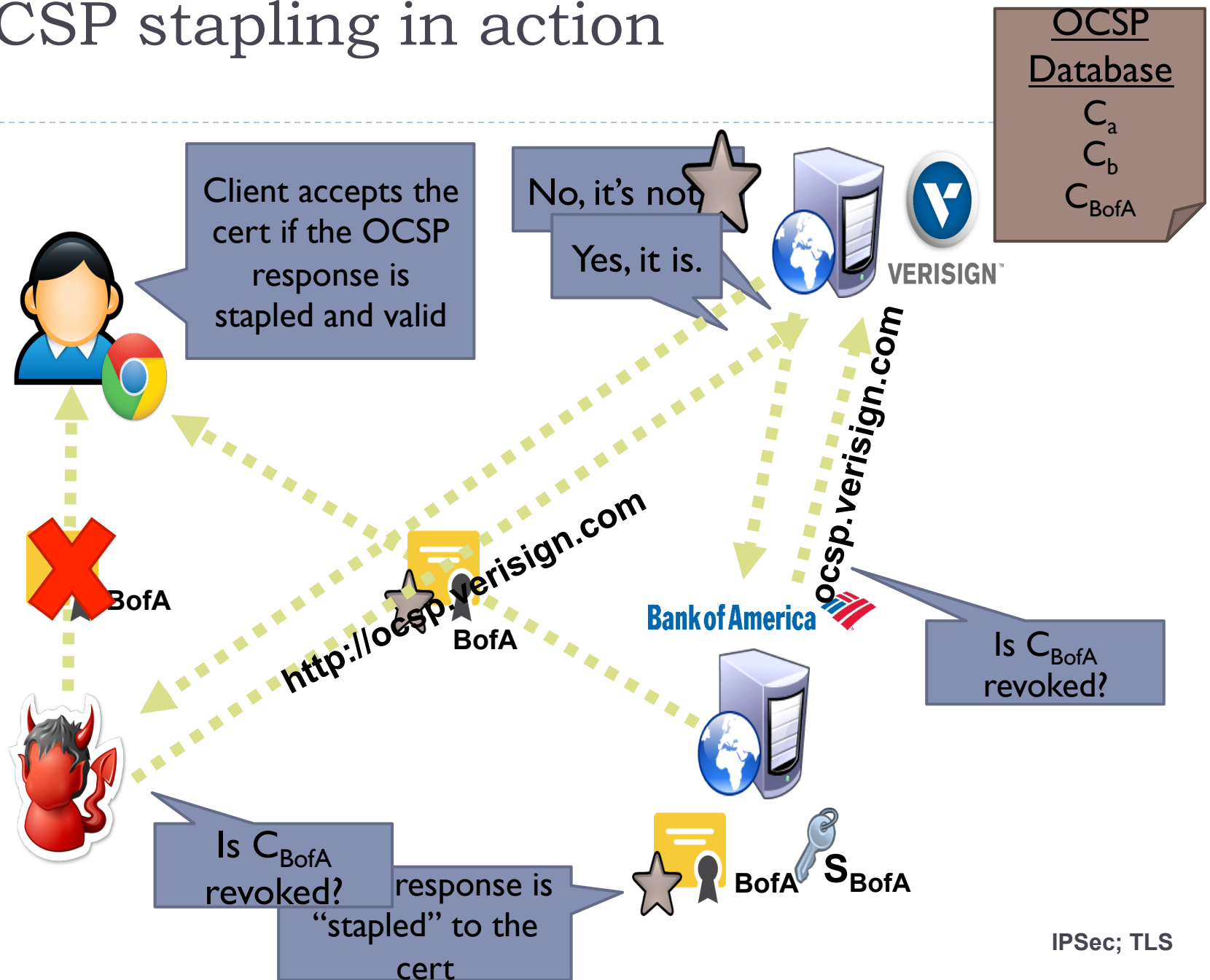
# OCSP Stapling

---

- ▶ It allows the presenter of a certificate to bear the resource cost involved in providing OCSP responses by appending ("stapling") a time-stamped OCSP response signed by the CA to the initial TLS Handshake, eliminating the need for clients to contact the CA.
- The good:
  - Clients don't need to query revocation status at all
  - Attacker cannot prevent clients from receiving revocation information
- The bad:
  - OCSP Must-Staple is very new, not supported by many browsers and certs



# OCSP stapling in action



# Revocation in practice (1)

---

- ▶ Revocation is one of the most broken parts of the TLS ecosystem
- ▶ Many administrators fail to revoke compromised certificates
- ▶ MitM attackers can block access to the CRL/OCSP URLs
  - ▶ Browsers default-accept certificates if the revocation status cannot be checked
  - ▶ Solved by OCSP Stapling, but this extension is not well deployed

# Revocation in practice (2)

---

- ▶ Many browsers do not perform proper revocation checks
- ▶ Chrome only does CRL/OCSP checks on EV certs, and only on some platforms
  - ▶ Windows – Yes,
  - ▶ Linux and Android – No
  - ▶ Chrome uses an alternative implementation called CRLset which is busted
- ▶ Firefox only supports OCSP
  - ▶ But fewer than 5% of certificates use OCSP
- ▶ Mobile browsers almost never check for revocations
  - ▶ Adds additional latency to HTTPS connections onto already slow mobile networks

# Implementation bugs

---

- ▶ Cryptography often assumed to be perfect
  - ▶ Usually the math is solid, but the implementation is found wanting
- ▶ Two major recent examples of security vulnerabilities due to TLS implementation bugs
  - ▶ Apple's Double Fail
  - ▶ Heartbleed

# Apple's Double Fail, a.k.a. Goto Fail

---

## ► What's wrong with this code?

```
// ...
if ((err = SSLHashSHA1.update(&hashCtx,
&serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx,
&signedParams)) != 0)
    goto fail;
goto fail;
if ((err = SSLHashSHA1.final(&hashCtx,
&hashOut)) != 0)
    goto fail;
// ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

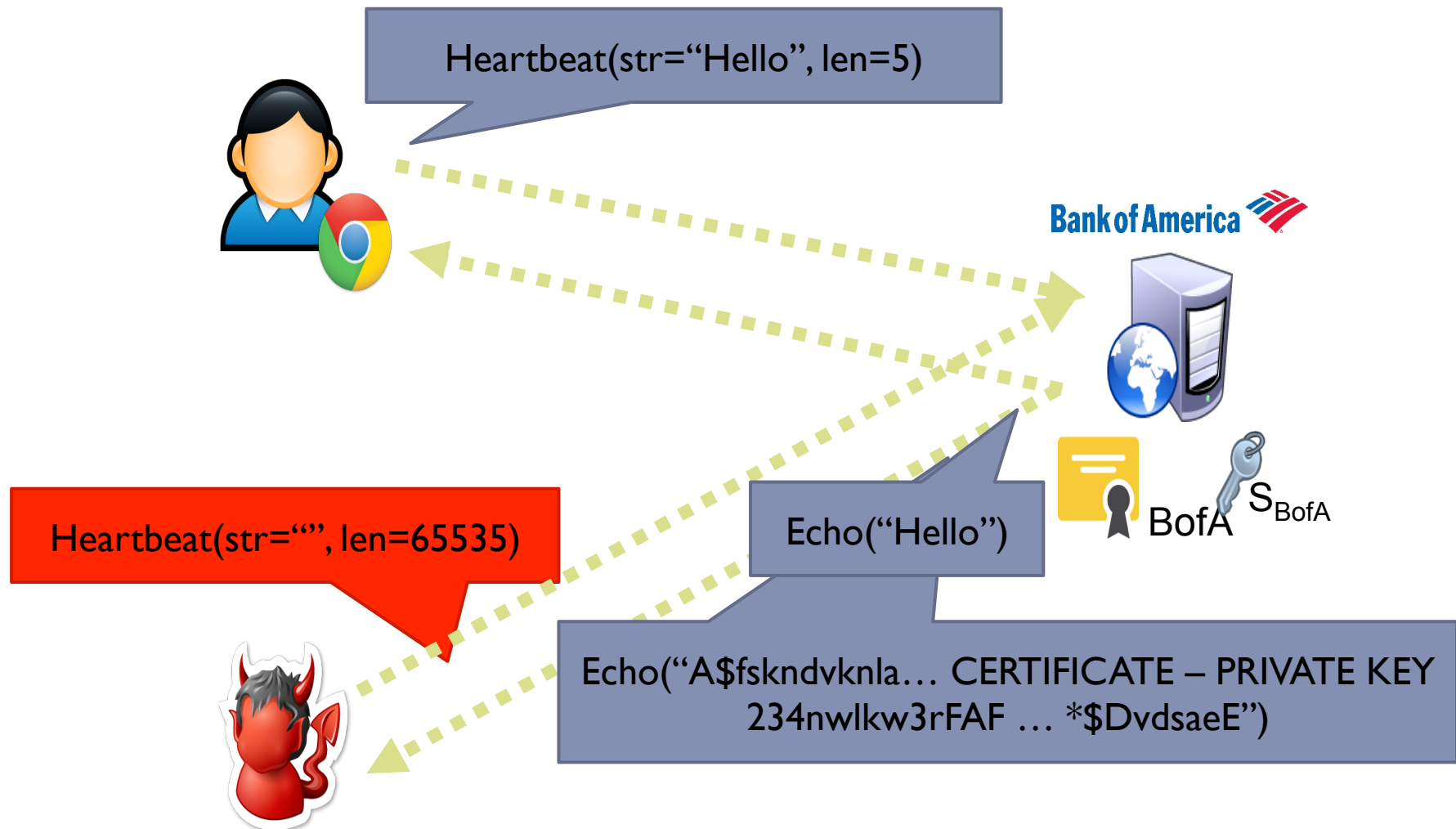
- Example of an implementation vulnerability in TLS signature verification
- Found in February 2014, present in iOS 6 and OS X

# HeartBleed

---

- ▶ **Serious vulnerability OpenSSL versions 1.0.1 – 1.0.1f**
  - ▶ Publicly revealed April 7, 2014
  - ▶ Exploits a bug in the TLS heartbeat extension
- ▶ **Allows adversaries to read memory of vulnerable services**
  - ▶ i.e., buffer over-read vulnerability
  - ▶ Discloses addresses, sensitive data, potentially TLS secret keys
- ▶ **Major impact**
  - ▶ OpenSSL is the de facto standard implementation of TLS, so used everywhere
  - ▶ Many exposed services, often on difficult-to-patch devices
  - ▶ Trivial to exploit

# Heartbleed Exploit Example



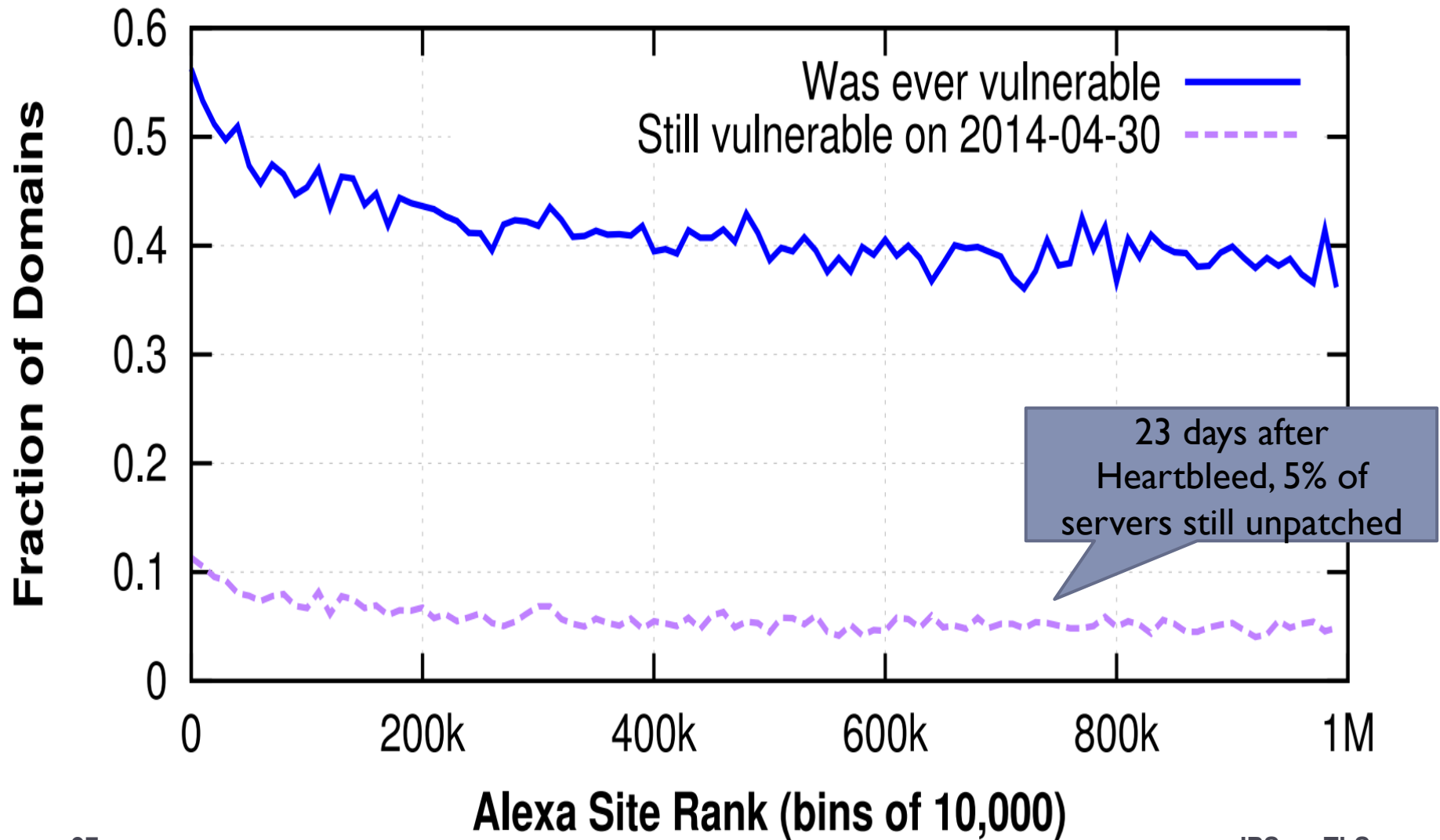
# Heartbleed as a natural experiment

---

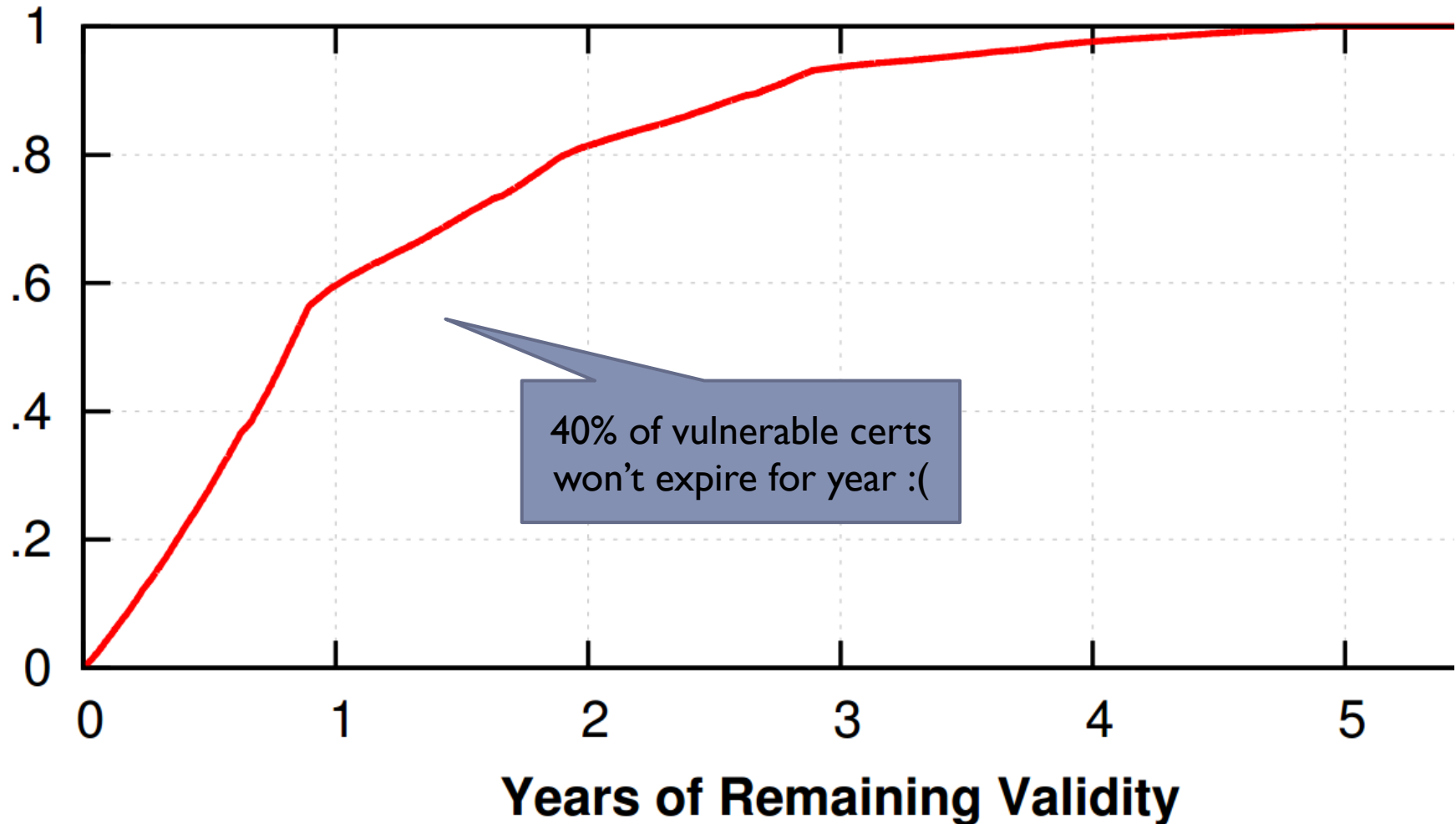
- ▶ Secret keys could have been stolen from all Heartbleed-vulnerable servers
- ▶ We know that administrators should have done three things on April 7, 2014:
  1. Patch their copy of OpenSSL
  2. Reissue their certificate with a new asymmetric keypair
  3. Revoke their old (potentially compromised) certificate
- ▶ Question: did administrators do these things?
  - ▶ If so, how quickly did they respond?



# Heartbleed-vulnerable Servers



# How Long Will We Be Dealing With Heartbleed?



# Other TLS Attacks

---

- ▶ Other interesting attacks on TLS
  - ▶ TLS stripping
  - ▶ BEAST
  - ▶ CRIME
  - ▶ BREACH
  - ▶ Lucky Thirteen
- ▶ We'll talk about these in the context of web security

# Take home lessons

---

- ▶ TLS is crucial for maintaining security and privacy on the Web
  - ▶ Mature, well supported protocol
  - ▶ Its security was proven as of 2014
- ▶ Unfortunately, TLS is plagued by many issues
  - ▶ Many different protocol-level issues that enable MitM attacks
  - ▶ TLS implementations are buggy
  - ▶ Human beings fail to reissue/revoke certificates properly
  - ▶ Browsers fail to perform revocation checks

