Protocols
Part I

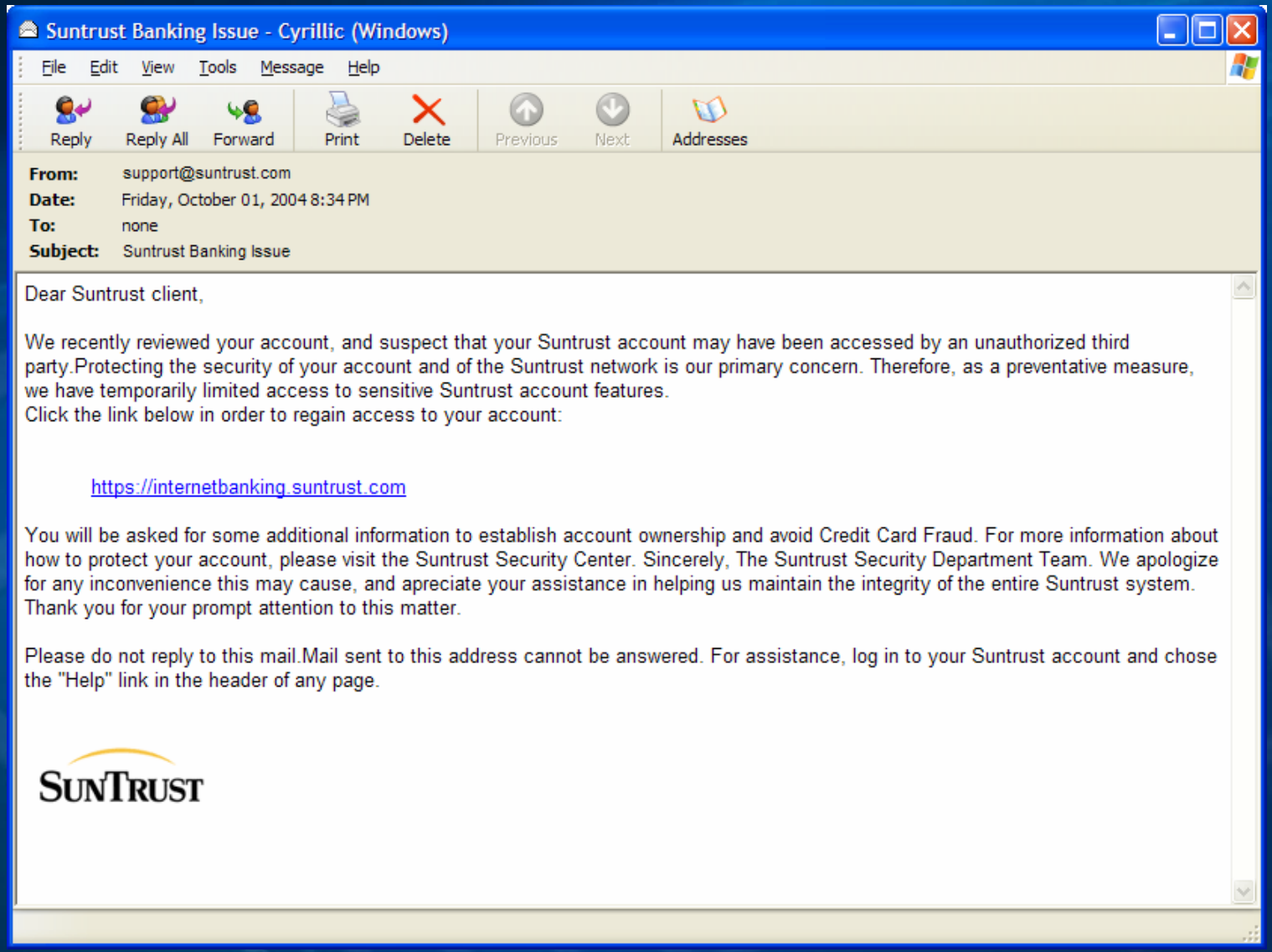
Brian A. LaMacchia
bal@cs.washington.edu
bal@microsoft.com

Agenda

- Introduction to protocols
- Session-based protocols
  - Kerberos
  - SSL/TLS
  - IPSEC
- Message-based protocols
  - S/MIME
  - XMLDSIG & XMLENC
- Advanced Key Exchange Algorithms

Introduction to Protocols

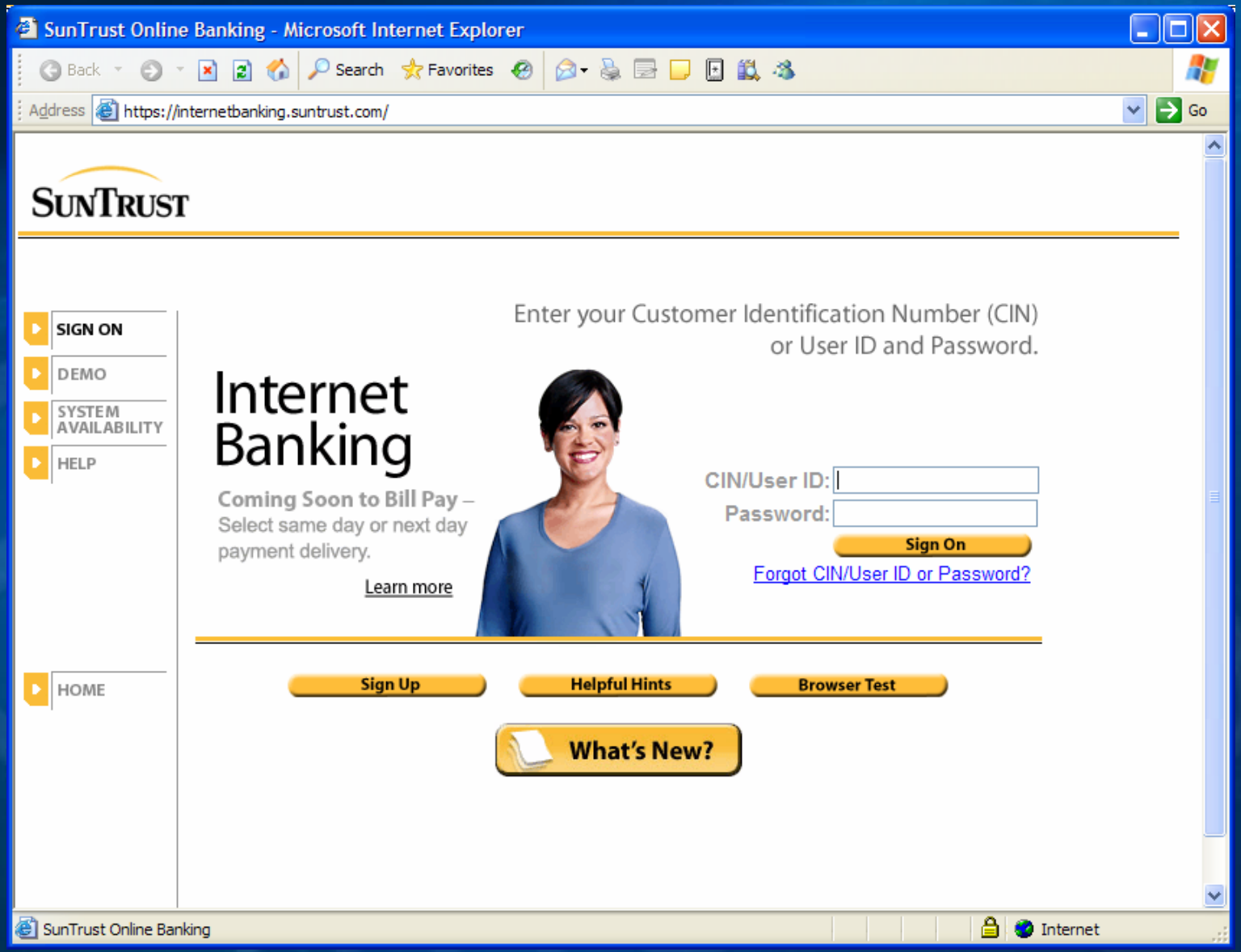
Motivation


Suntrust Banking Issue - Cyrillic (Windows)
File Edit View Tools Message Help
Reply Reply All Forward Print Delete Previous Next Addresses
From: support@suntrust.com
Date: Friday, October 01, 2004 8:34 PM
To: none
Subject: Suntrust Banking Issue
Dear Suntrust client,
We recently reviewed your account, and suspect that your Suntrust account may have been accessed by an unauthorized third party.Protecting the security of your account and of the Suntrust network is our primary concern. Therefore, as a preventative measure, we have temporarily limited access to sensitive Suntrust account features.
Click the link below in order to regain access to your account:
https://internetbanking.suntrust.com
You will be asked for some additional information to establish account ownership and avoid Credit Card Fraud. For more information about how to protect your account, please visit the Suntrust Security Center. Sincerely, The Suntrust Security Department Team. We apologize for any inconvenience this may cause, and apreciate your assistance in helping us maintain the integrity of the entire Suntrust system. Thank you for your prompt attention to this matter.
Please do not reply to this mail.Mail sent to this address cannot be answered. For assistance, log in to your Suntrust account and chose the "Help" link in the header of any page.
SUNTRUST

Motivation


WWW.SUNTRUSTBANKINGONLINE.COM - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.suntrustbankingonline.com/
Go Links
SUNTRUST
SIGN ON
DEMO
SYSTEM AVAILABILITY
HELP
HOME
Enter your Customer Identification Number (CIN) and Password.
Internet Banking
Now you can update your Password and User ID online.
Learn more
CIN:
Password:
Sign On
Sign Up
Helpful Hints
Browser Test
What's New?
Internet

Motivation


SunTrust Online Banking - Microsoft Internet Explorer
Back
Search
Favorites
Address
https://internetbanking.suntrust.com/
Go
SUNTRUST
SIGN ON
DEMO
SYSTEM AVAILABILITY
HELP
HOME
Enter your Customer Identification Number (CIN) or User ID and Password.
Internet Banking
Coming Soon to Bill Pay – Select same day or next day payment delivery.
Learn more
CIN/User ID:
Password:
Sign On
Forgot CIN/User ID or Password?
Sign Up
Helpful Hints
Browser Test
What's New?
SunTrust Online Banking
Internet

Motivation

- How do I know the web site I'm talking to is really who I think it is?
- Is it safe to view to give sensitive information over the Web?
  - What keeps my CC#, SSN, financial information or medical records out of the hands of the bad guys?
- How do I know that the information I'm looking at hasn't been malicious modified?
  - Has someone tampered with it?

Security Protocol Properties

- Confidentiality
  - Keeping message content secret, even if the information passes over a public channel
- Integrity
  - Keeping messages tamper-free from origin to destination
- Authentication
  - Determining the origin of messages (author and/or sender)

Kerberos

Kerberos History

- Designed as part of MIT's Project Athena in the 1980's
  - Kerberos v4 published in 1987
- Migration to the IETF
  - RFC 1510 (Kerberos v5, 1993)
- Used in a number of products
  - Example: part of Windows 2000
  - MS Passport is essentially Kerberos done w/ client-side cookies over HTTP

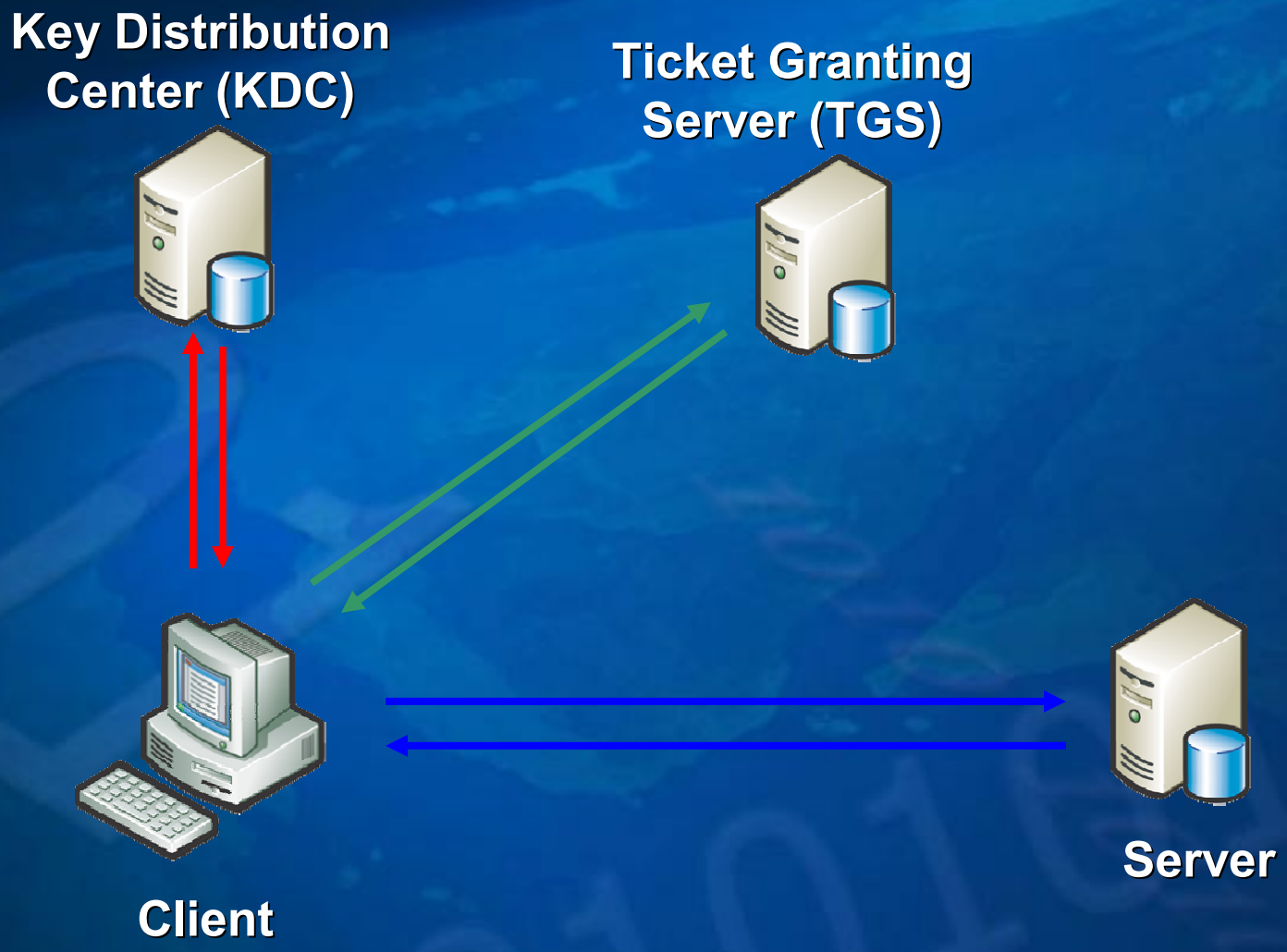
Kerberos

- Designed for single "administration domain" of machines & users: users, client machines, server machines, and the Key Distribution Center (KDC)
- No public key crypto
- Provides authentication & encryption services
- "Kerberized" servers provide authorization on top of the authenticated identities

The Kerberos Model

- Clients
- Servers
- The Key Distribution Center (KDC)
- Centralized trust model
  - KDC is trusted by all clients & servers
  - KDC shares a secret, symmetric key with each client and server
- A "realm" is single trust domain consisting of one or more clients, servers, KDCs

Picture of a Kerberos Realm



Joining a Kerberos Realm

- One-time setup
  - Each client, server that wishes to participate in the realm exchanges a secret key with the KDC
  - If the KDC is compromised, the entire system is cracked
- Because the KDC knows everyone's individual secret key, the KDC can issue credentials to each realm identity

Kerberos Credentials

- Two types of credentials in Kerberos
  - Tickets
  - Authenticators
- Tickets are credentials issued to a client for communication with a specific server
- Authenticators are additional credentials that prove a client knows a key at a point in time
  - Basic idea: encrypt a "nonce"

The Basic Kerberos Protocol

Assume client C wishes to authenticate to and communicate with server S

Phase 1: C gets a Ticket-Granting Ticket (TGT) from the KDC

Phase 2: C uses the TGT to get a Ticket for S

Phase 3: C communicates with S

Protocol Definitions

Following Schneier (Section 24.5):

- C = client, S = server
- TGS = ticket-granting service
- K_x = x's secret key
- $K_{x,y}$ = session key for x and y
- $\{m\}K_x$ = m encrypted in x's secret key
- $T_{x,y}$ = x's ticket to use y
- $A_{x,y}$ = authenticator from x to y
- N_x = a nonce generated by x

The Basic Kerberos Protocol (1)

Phase 1: C gets a Ticket-Granting Ticket

1. C sends a request to the KDC for a "ticket-granting ticket" (TGT)
   - A TGT is a ticket used to talk to the special ticket-granting service
   - A TGT is relatively long-lived (~8-24 hours typically)

$$C \rightarrow KDC: C, TGS, N_C$$

Sent in the clear!

The Basic Kerberos Protocol (2)

Phase 1: C gets a Ticket-Granting Ticket

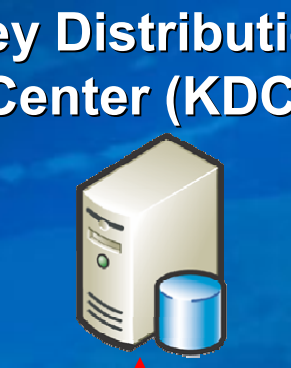
2. KDC responds with two items
   - The ticket-granting ticket
     - A ticket for C to talk to TGS
   - A copy of the session key to use to talk to TGS, encrypted in C's shared key

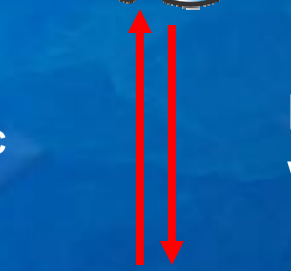
$$KDC \rightarrow C: \{T_{C,TGS}\}K_{TGS}, \{K_{C,TGS}\}K_C$$

where $T_{c,s} = s, \{c, c\text{-addr}, lifetime, K_{c,s}\}K_s$

   - Only the TGS can decrypt the ticket
   - C can unlock the second part to retrieve $K_{C,TGS}$

Picture of a Kerberos Realm





$C \rightarrow KDC: C, TGS, N_C$

$KDC \rightarrow C: \{T_{C,TGS}\}K_{TGS}, \{K_{C,TGS}\}K_C$
where $T_{c,s} = s, \{c, c\text{-addr}, lifetime, K_{c,s}\}K_s$

The Basic Kerberos Protocol (3)

Phase 2: C gets a Ticket for S

3. **C requests a ticket to communicate with S from the ticket-granting service (TGS)**
   - **C sends TGT to S along with an authenticator requesting a ticket from C to S**

$$C \rightarrow TGS: \{A_{C,S}\}K_{C,TGS}\ ,\ \{T_{C,TGS}\}K_{TGS}$$

where $A_{c,s}$ **= {c, timestamp, opt. subkey}**$K_{c,s}$

   - **First part proves to TGS that C knows the session key**
   - **Second part is the TGT C got from the KDC**

The Basic Kerberos Protocol (4)

Phase 2: C gets a Ticket for S

4. **TGS returns a ticket for C to talk to S**

   (Just like step 2 above…)

   TGS → C: $\{T_{C,S}\}K_S$, $\{K_{C,S}\}K_{C,TGS}$

   - **Only S can decrypt the ticket**
   - **C can unlock the second part to retrieve $K_{C,S}$**

Picture of a Kerberos Realm

Ticket Granting Server (TGS)

C ➔ TGS: $\{A_{C,S}\}K_{C,TGS}$, $\{T_{C,TGS}\}K_{TGS}$
where $A_{c,s}$ = {c, timestamp, opt. subkey}$K_{c,s}$

TGS ➔ C: $\{T_{C,S}\}K_S$, $\{K_{C,S}\}K_{C,TGS}$

Client

The Basic Kerberos Protocol (5)

Phase 3: C communicates with S

5. **C sends the ticket to S along with an authenticator to establish a shared secret**

$$C \rightarrow S: \{A_{C,S}\}K_{C,S}\ ,\ \{T_{C,S}\}K_S$$

where $A_{c,s}$ = {c, timestamp, opt. subkey}$K_{c,s}$

- **S decrypts the ticket $T_{C,S}$ to get the shared secret $K_{C,S}$ needed to communicate securely with C**

The Basic Kerberos Protocol (6)

Phase 3: C communicates with S

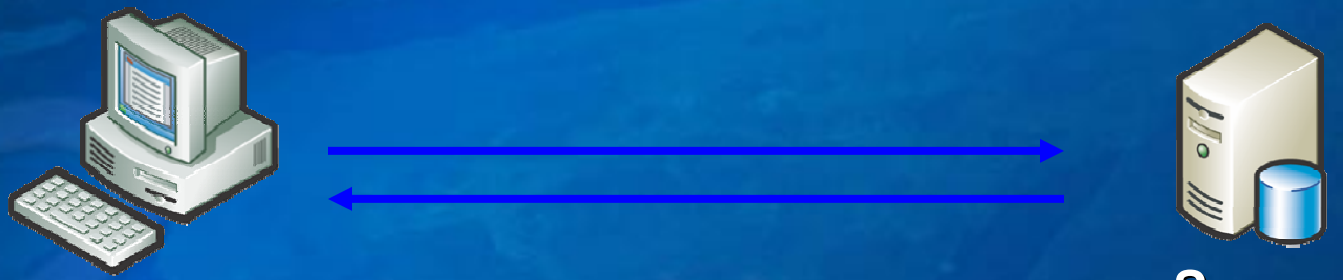
6. **S decrypts the ticket to obtain the $K_{C,S}$ and replies to C with proof of possession of the shared secret (optional step)**

   S → C: {timestamp, opt. subkey}$K_{c,s}$

   Notice that S had to decrypt the authenticator, extract the timestamp & opt. subkey, and re-encrypt those two components with $K_{c,s}$

Picture of a Kerberos Realm

C ➔ S: $\{A_{C,S}\}K_{C,S}$, $\{T_{C,S}\}K_S$
where $A_{c,s}$ = {c, timestamp, opt. subkey}$K_{c,s}$

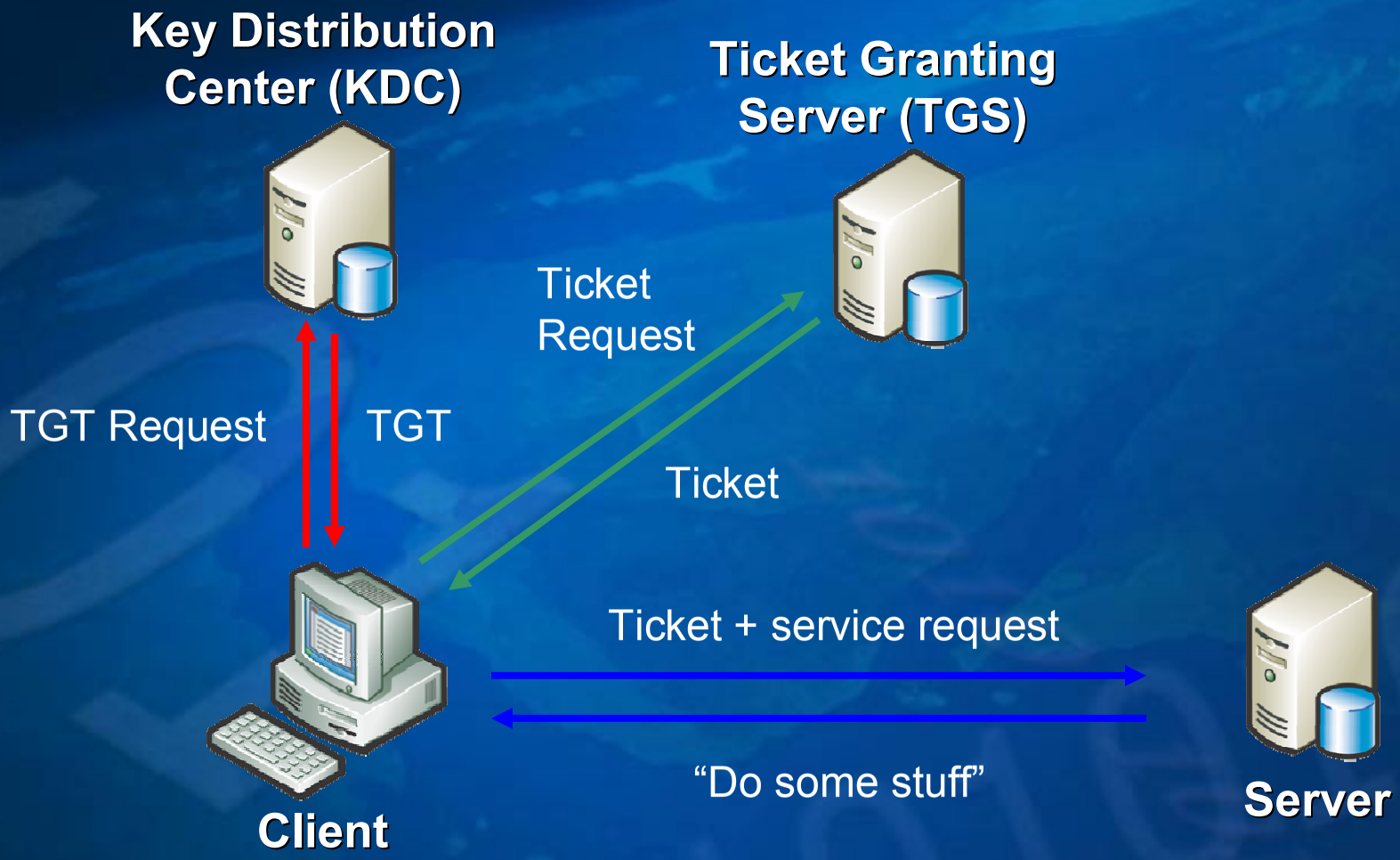


Client

Server

S ➔ C: {timestamp, opt. subkey}$K_{c,s}$

Picture of a Kerberos Realm



Thoughts on Kerberos…

- There's no public key crypto anywhere in the base Kerberos spec, but you can modify the base protocols to use PK…
  - Example: the initial "login" to the KDC could be done with public key for added security (e.g. PKINIT protocol)

PKINIT in Windows 2K/2K3

Key Distribution Center (KDC)

Verification and NT user account lookup

Active Directory

Logon request using Public Key

Kerberos Ticket Granting Ticket (TGT)

Client

Thoughts on Kerberos...(2)

- Only the KDC needs to know the user's password (used to generate the shared secret)
  - You can have multiple KDCs for redundancy, but they all need to have a copy of the username/password database
- Only the TGS needs to know the secret keys for the servers
  - You can split KDC from TGS, but it is common for those two services to reside on the same physical machine

Thoughts on Kerberos...(3)

- Cross-realm trust is possible
  - Just need to share a secret key between the KDCs for the two realms...
  - Once accomplished, a user in realm A can get a ticket for a service in realm B

Thoughts on Kerberos...(4)

- "Time" is very important in Kerberos
  - All participants in the realm need accurate clocks
  - Timestamps are used in authenticators to detect replay; if a host can be fooled about the current time, old authenticators could be replayed
  - Tickets tend to have lifetimes on the order of hours, and replays are possible during the lifetime of the ticket

Thoughts on Kerberos...(5)

- Password-guessing attacks are possible
  - Capture enough encrypted tickets and you can brute-force decrypt them to discover shared keys
  - (Another reason to use public key...)

Thoughts on Kerberos...(6)

- It's possible to screw up the implementation
  - In fact, Kerberos v4 had a colossal security breach due to bad implementations

RNGs in Kerberos v4

- Session keys were generated from a PRNG seeded with the XOR of the following:
  - Time-of-day in seconds since 1/1/1970
  - Process ID of the Kerberos server process
  - Cumulative count of session keys generated
  - Fractional part of time-of-day seconds
  - Hostid of the machine running the server

RNGs in Kerberos v4 (continued)

- The seed is a 32-bit value, so while the session key is used for DES (64 bits long, normally 56 bits of entropy), it has only 32 bits of entropy
- What's worse, the five values have predictable portions
  - Time is completely predictable
  - ProcessID is mostly predictable
  - Even hostID has 12 predictable bits (of 32 total)

RNGs in Kerberos v4 (continued)

- Of the 32 seed bits, only 20 bits really change with any frequency, so Kerberos v4 keys (in the MIT implementation) only have 20 bits of randomness
  - They could be brute-force discovered in seconds
- The hole was in the MIT Kerberos sources for *seven years!*

Securing Internet Traffic

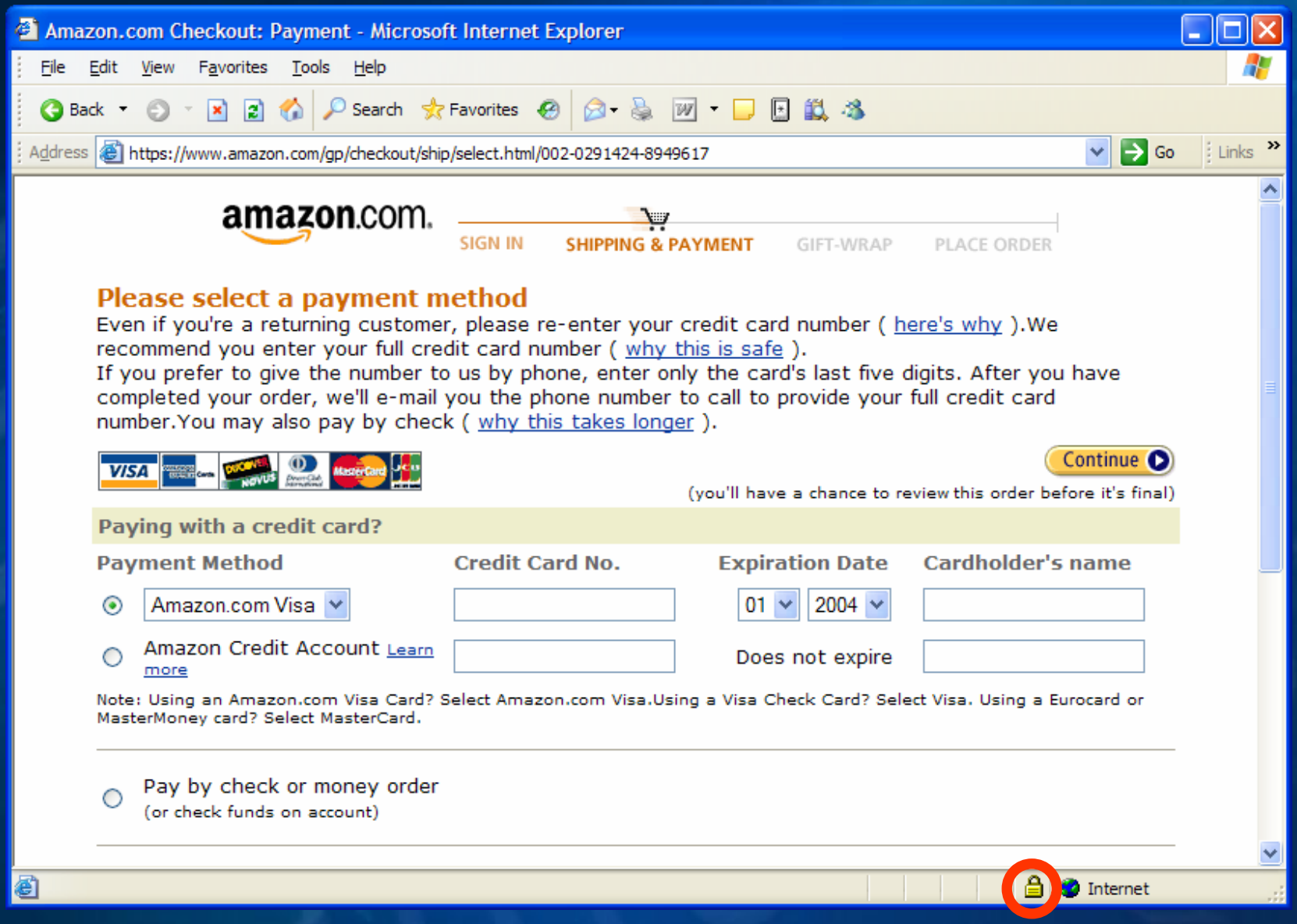
- Application-level security
  - Secure the traffic between two communicating applications
  - Application-specific protocols
  - Example: SSL/TLS for web traffic
- IP-level security
  - Secure traffic at the Internet Protocol layer (low-level wire format)
  - Applications don't have to know about security specifically, they "get it for free"
  - Example: IPSEC

Common Themes

- Three phases
  - Authentication
    - Verify the other party is someone you want to talk to
  - Key agreement
    - Agree on data encryption and integrity protection keys
  - Encrypted data exchange
    - Communicate over the encrypted channel

SSL/TLS

App-Level Security: SSL/TLS



SSL/PCT/TLS History

- 1994: Secure Sockets Layer (SSL) V2.0
- 1995: Private Communication Technology (PCT) V1.0
- 1996: Secure Sockets Layer (SSL) V3.0
- 1997: Private Communication Technology (PCT) V4.0
- 1999: Transport Layer Security (TLS) V1.0
- 2005/2006: TLS V1.1 (currently in the RFC Editor's Queue awaiting publication)

Typical Scenario

You (client) **Merchant (server)**

SSL/TLS

You (client) **Merchant (server)**

Let's talk securely.

Here is my RSA public key.

Here is a symmetric key, encrypted with your public key, that we can use to talk.

SSL/TLS

You (client) **Merchant (server)**

Let's talk securely.
Here are the protocols and ciphers I understand.

Here is my RSA public key.

Here is a symmetric key, encrypted with your public key, that we can use to talk.

SSL/TLS

You (client) **Merchant (server)**

Let's talk securely.
Here are the protocols and ciphers I understand.

I choose this protocol and ciphers.
Here is my public key and
some other stuff.

Here is a symmetric key, encrypted with your public key, that we can use to talk.

SSL/TLS

You (client) **Merchant (server)**

Let's talk securely.
Here are the protocols and ciphers I understand.

I choose this protocol and ciphers.
Here is my public key and
some other stuff.

Using your public key, I've encrypted
a random symmetric key to you.

SSL/TLS

All subsequent secure messages are sent using the symmetric key and a keyed hash for message authentication.

The five phases of SSL/TLS

1. Negotiate the ciphersuite to be used
2. Establish the shared session key
3. Client authenticates the server ("server auth")
   - Optional, but almost always done
4. Server authenticates the client ("client auth")
   - Optional, and almost never done
5. Authenticate previously exchanged data

Phase 1: Ciphersuite Negotiation

- Client hello (client→server)
  - "Hi! I speak these n ciphersuites, and here's a 28-byte random number (nonce) I just picked"
- Server hello (client←server)
  - "Hello. We're going to use this particular ciphersuite, and here's a 28-byte nonce I just picked."
- Other info can be passed along (we'll see why a little later…)

TLS V1.0 ciphersuites

```
TLS_NULL_WITH_NULL_NULL
TLS_RSA_WITH_NULL_MD5
TLS_RSA_WITH_NULL_SHA
TLS_RSA_EXPORT_WITH_RC4_40_MD5
TLS_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_RC4_128_SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
TLS_RSA_WITH_IDEA_CBC_SHA
TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_RSA_WITH_DES_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_DSS_WITH_DES_CBC_SHA
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DH_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_RSA_WITH_DES_CBC_SHA
TLS_DH_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_DSS_WITH_DES_CBC_SHA
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
TLS_DHE_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_DH_anon_EXPORT_WITH_RC4_40_MD5
TLS_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_EXPORT_WITH_DES40_CBC_SHA
TLS_DH_anon_WITH_DES_CBC_SHA
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA
```

More defined in other specs

TLS-With-AES ciphersuites (RFC 3268)

```
TLS_RSA_WITH_AES_128_CBC_SHA          RSA
TLS_DH_DSS_WITH_AES_128_CBC_SHA       DH_DSS
TLS_DH_RSA_WITH_AES_128_CBC_SHA       DH_RSA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA      DHE_DSS
TLS_DHE_RSA_WITH_AES_128_CBC_SHA      DHE_RSA
TLS_DH_anon_WITH_AES_128_CBC_SHA      DH_anon

TLS_RSA_WITH_AES_256_CBC_SHA          RSA
TLS_DH_DSS_WITH_AES_256_CBC_SHA       DH_DSS
TLS_DH_RSA_WITH_AES_256_CBC_SHA       DH_RSA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA      DHE_DSS
TLS_DHE_RSA_WITH_AES_256_CBC_SHA      DHE_RSA
TLS_DH_anon_WITH_AES_256_CBC_SHA      DH_anon
```

Phase 2: Establish the shared session key

- Client key exchange
  - Client chooses a 48-byte "pre-master secret"
  - Client encrypts the pre-master secret with the server's RSA public key
  - Client→server encrypted pre-master secret
- Client and server both compute
  - PRF (pre-master secret, "master secret", client nonce + server nonce)
  - PRF is a pseudo-random function
  - First 48 bytes output from PRF form master secret

TLS's PRF

- PRF(secret, label, seed) =
  P_MD5(S1, label + seed) XOR
  P_SHA-1(S2, label + seed);
  where S1, S2 are the two halves of the secret
- 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))

Phases 3 & 4: Authentication

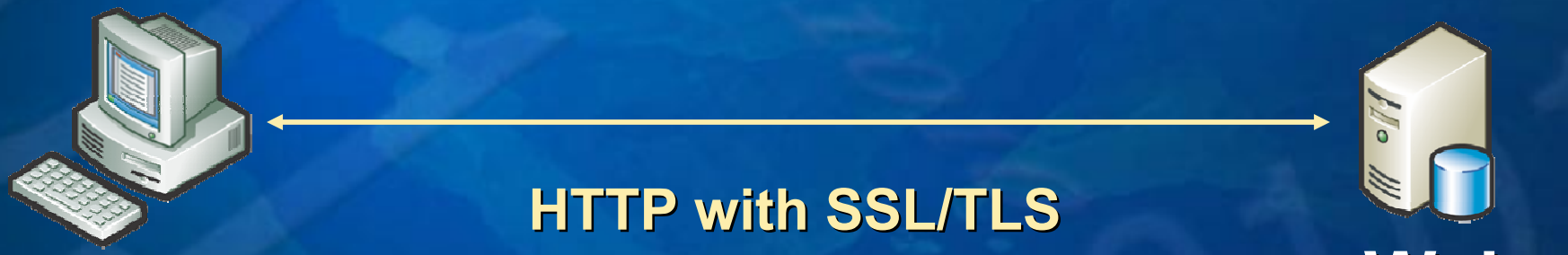
More on this in a moment…

Phase 5: Authenticate previously exchanged data

- "Change ciphersuites" message
  - Time to start sending data for real…
- "Finished" handshake message
  - First protected message, verifies algorithm parameters for the encrypted channel
  - 12 bytes from:
    PRF(master_secret, "client finished",
    MD5(handshake_messages) +
    SHA-1(handshake_messages))

Why do I trust the server key?

- How do I know I'm really talking to Amazon.com?
- What defeats a man-in-the-middle attack?



Why do I trust the server key?

- How do I know I'm really talking to Amazon.com?
- What defeats a man-in-the-middle attack?

SSL/TLS

You (client) **Merchant (server)**

Let's talk securely.
Here are the protocols and ciphers I understand.

I choose this protocol and ciphers.
Here is my public key and
some other stuff that will make you
trust this key is mine.

Here is a fresh key encrypted with your key.

What's the "some other stuff"

How can we convince Alice that some key belongs to Bob?

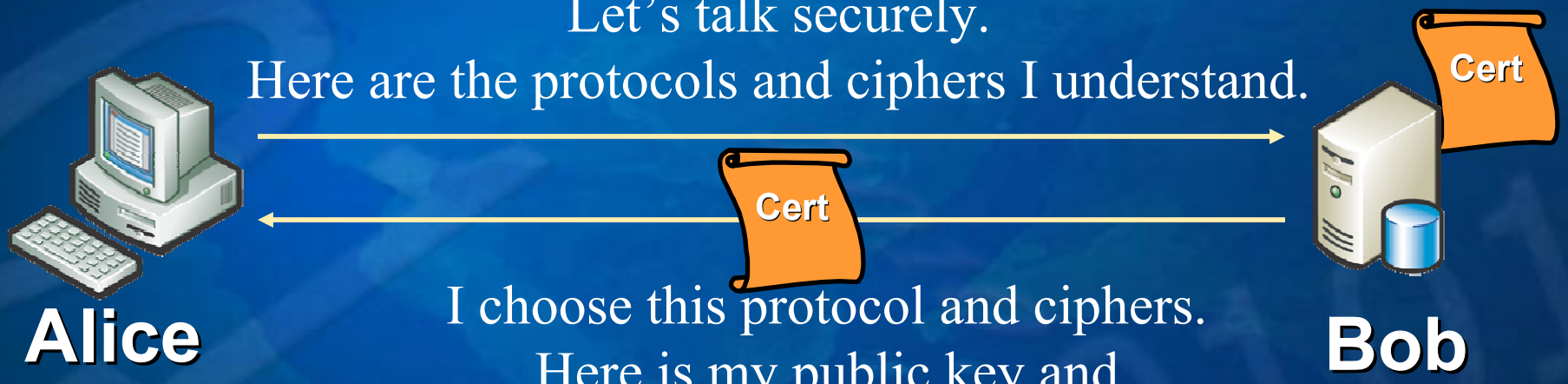
- Alice and Bob could have met previously & exchanged keys directly.
  - *Jeff Bezos isn't going to shake hands with everyone he'd like to sell to…*
- Someone Alice trusts could vouch to her for Bob and Bob's key
  - *A third party can certify Bob's key in a way that convinces Alice.*

What is a certificate?

- A certificate is a digitally-signed statement that binds a public key to some identifying information.
  - The signer of the certificate is called its issuer.
  - The entity talked about in the certificate is the subject of the certificate.
- That's all a certificate is, at the 30,000' level.

Defeating Mallet

Bob can convince Alice that his key really does belong to him if he can also send along a digital certificate Alice will believe & trust



Server & Client Authentication with Certificates

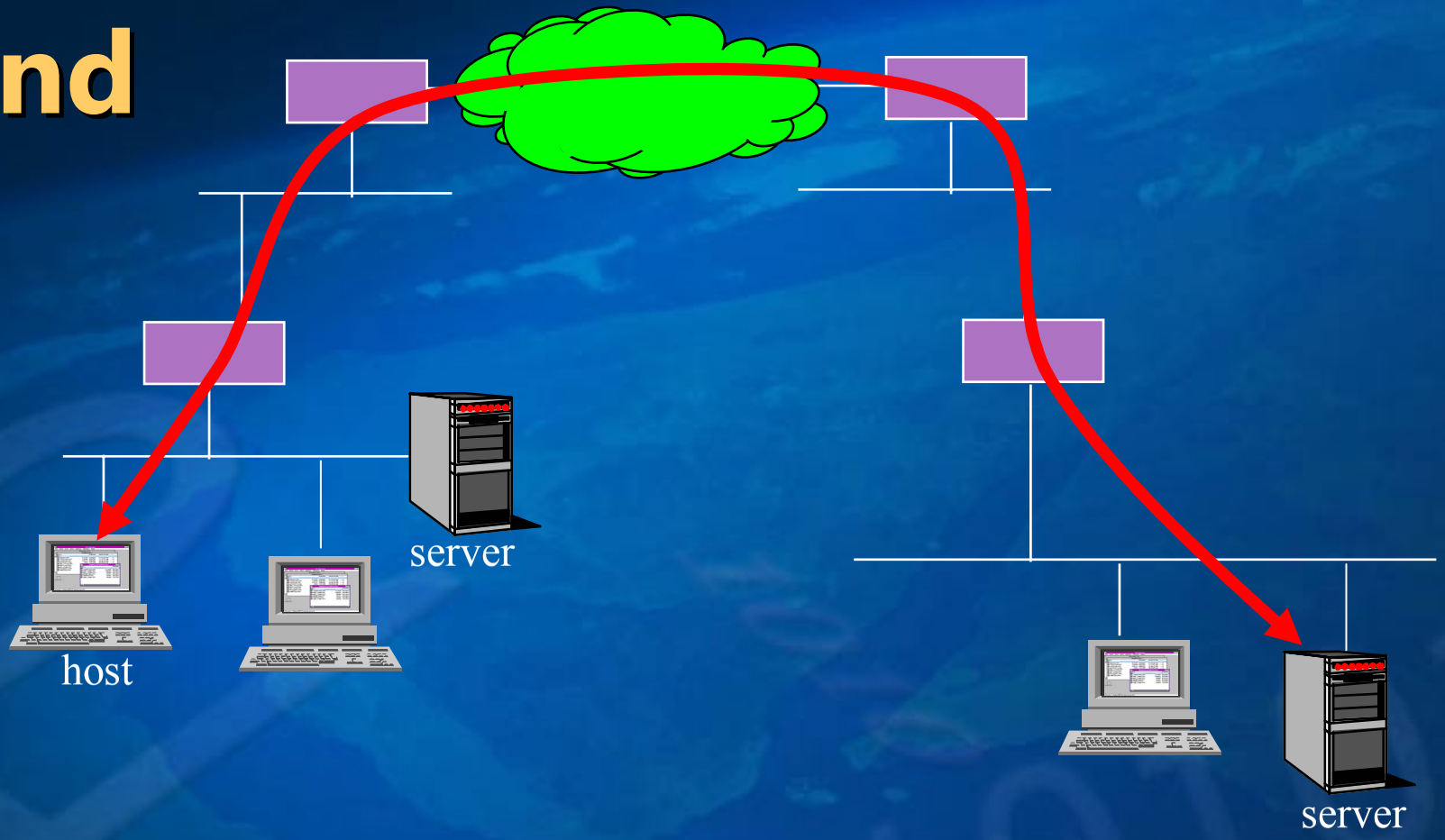
- We're going to talk a lot more about how you determine whether you trust a name-key binding later in the course
  - Lecture #8: Trust, Public Key Infrastructure (PKI) and Key Management
- For now, simply assume that each client and server can:
  - Cryptographically validate a certificate to verify its integrity
  - Decide whether a validated certificate should be *believed* according to its *trust policy*

IPSEC

Protocol-Level Security: IPSEC

- Application-level security protocols work great for particular applications
  - But they only work for that application
- SSL/TLS requires lots of infrastructure to work; how many protocols can we do that for?
- Ideally, we'd like all the security features of SSL/TLS available for every Internet protocol/application
  - "Security at the IP layer"

Ideal Protection: End-to-End



- SSL/TLS does this at the application layer (TCP)
- IPSEC does this for any IP packet, at network layer
- Apps must be aware of/control SSL, don't have to be for IPSec

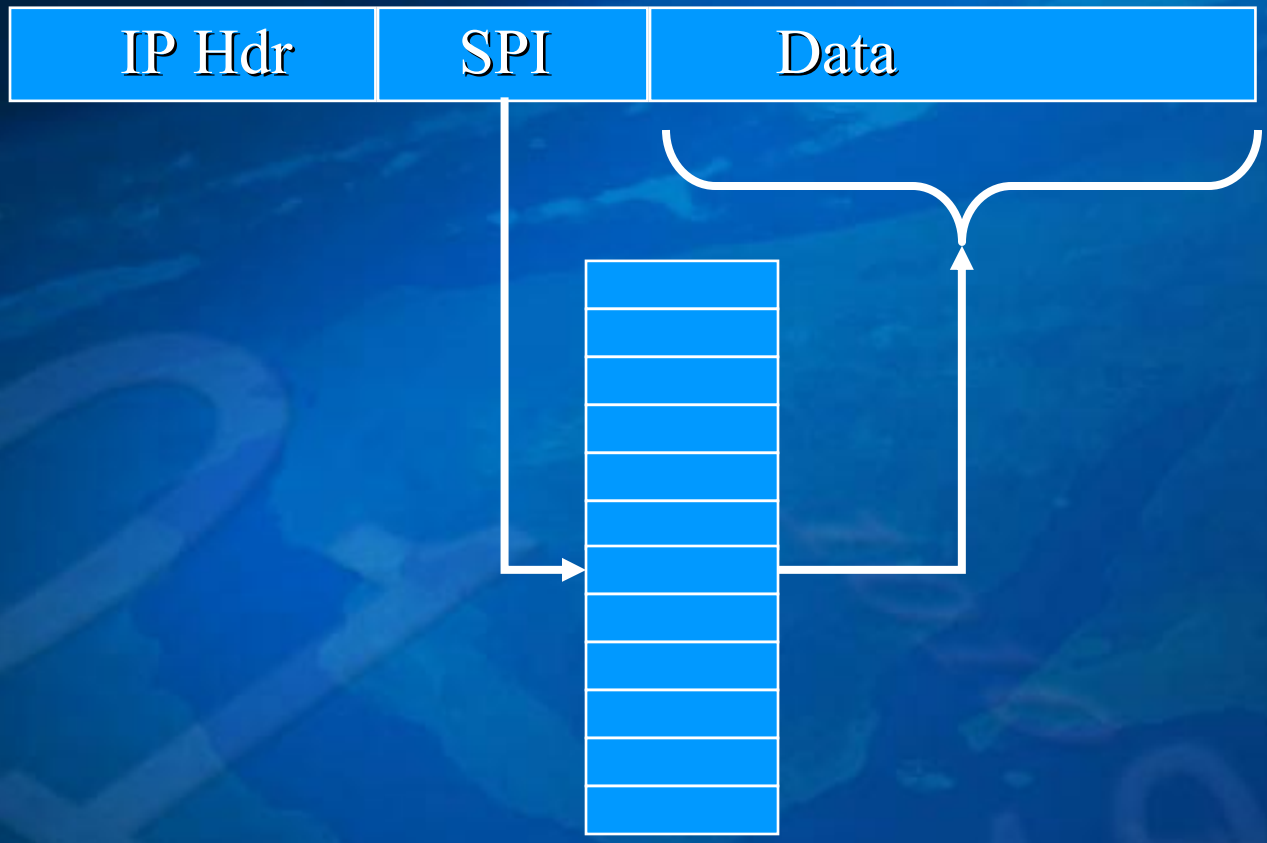
IPSEC

- IPSEC = IP (Internet Protocol) Security
  - Suite of protocols that provide encryption, integrity and authentication services for IP packets
  - Mandatory-to-implement for IPv6, optional (but available) for IPv4
- Consists of two main components:
  - IPSEC key management
  - IPSEC protection protocols
    - Encryption & auth of IP packets

IPSEC Key Management

- Establishes a Security Association (SA) for a session
  - Think "shared secret key" for each pair of communicating parties
  - SA used to provide authentication and confidentiality services for that session
  - SA is referenced via a security parameter index (SPI) in each IP datagram header

IPSEC Architecture



Security information maintained by host

IPSEC Protection Protocols

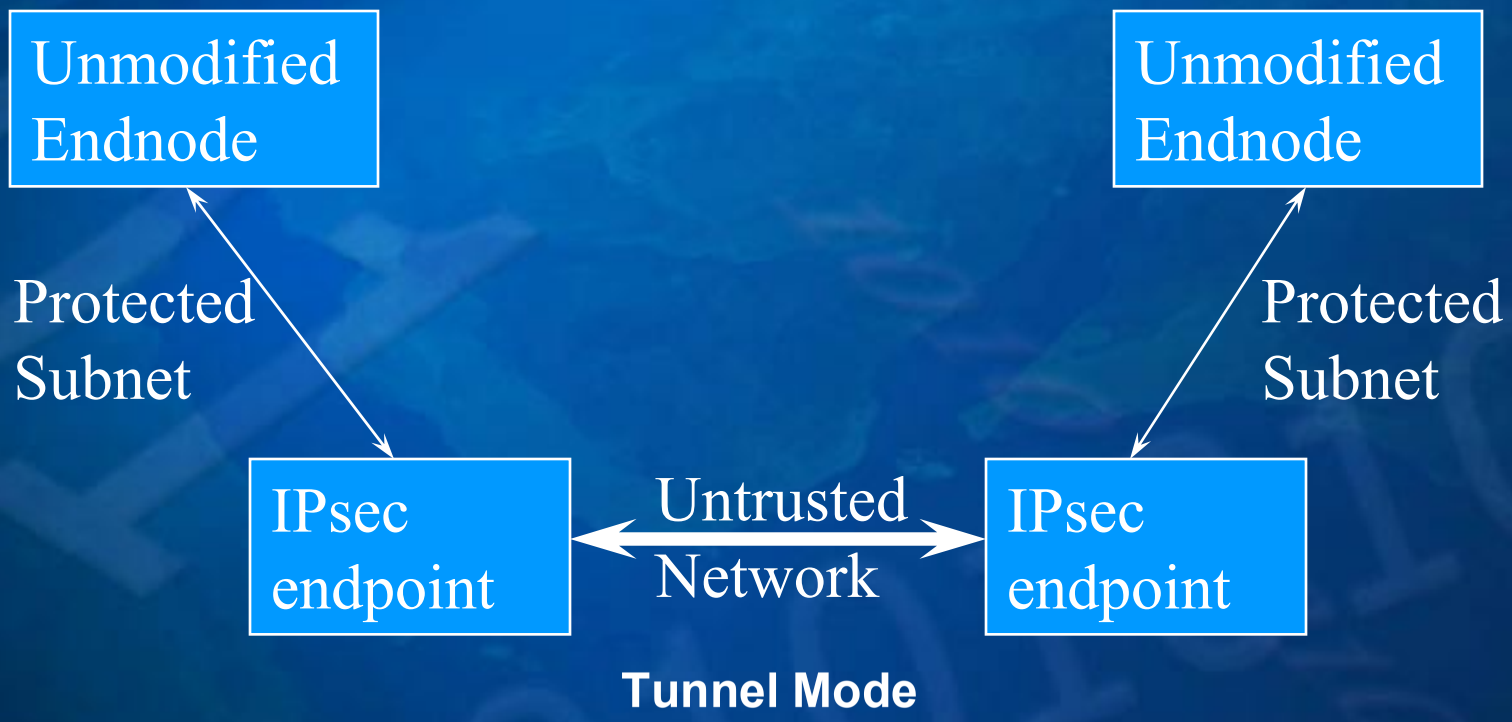
- Authentication Header (AH)
  - Authenticates payload data
  - Authenticates network header
  - Gives anti-replay protection
- Encapsulated Security Payload (ESP)
  - Encrypts payload data
  - Authenticates payload data
  - Gives anti-replay protection

IPSEC Modes of Operation

- Tunnel Mode
  - Encapsulates the entire IP packet within IPSC protection
  - Tunnels can be created between several different node types
    - Gateway to gateway
    - Host to gateway
    - Host to host
- Transport Mode
  - Encapsulates only the transport layer information within IPSEC protection
  - Can only be created between host nodes

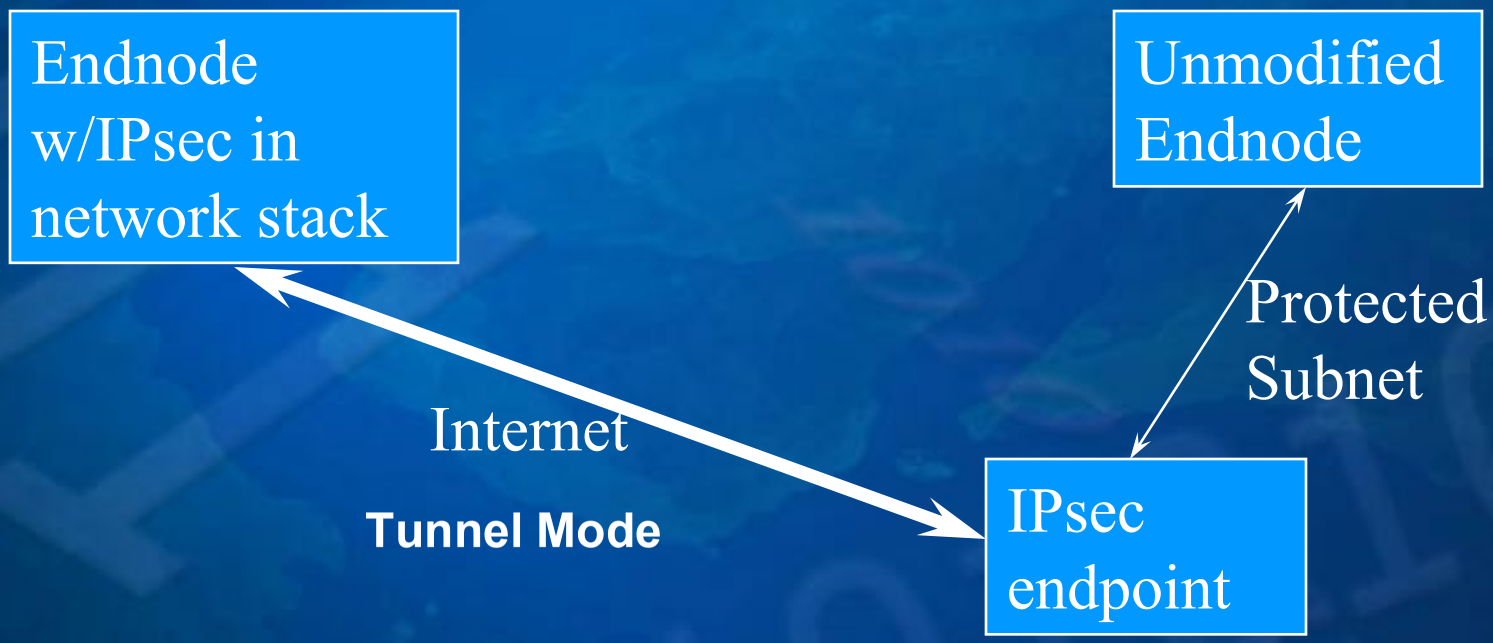
IPsec Scenario 1 Firewall to Firewall

- Corporate network connected through Internet



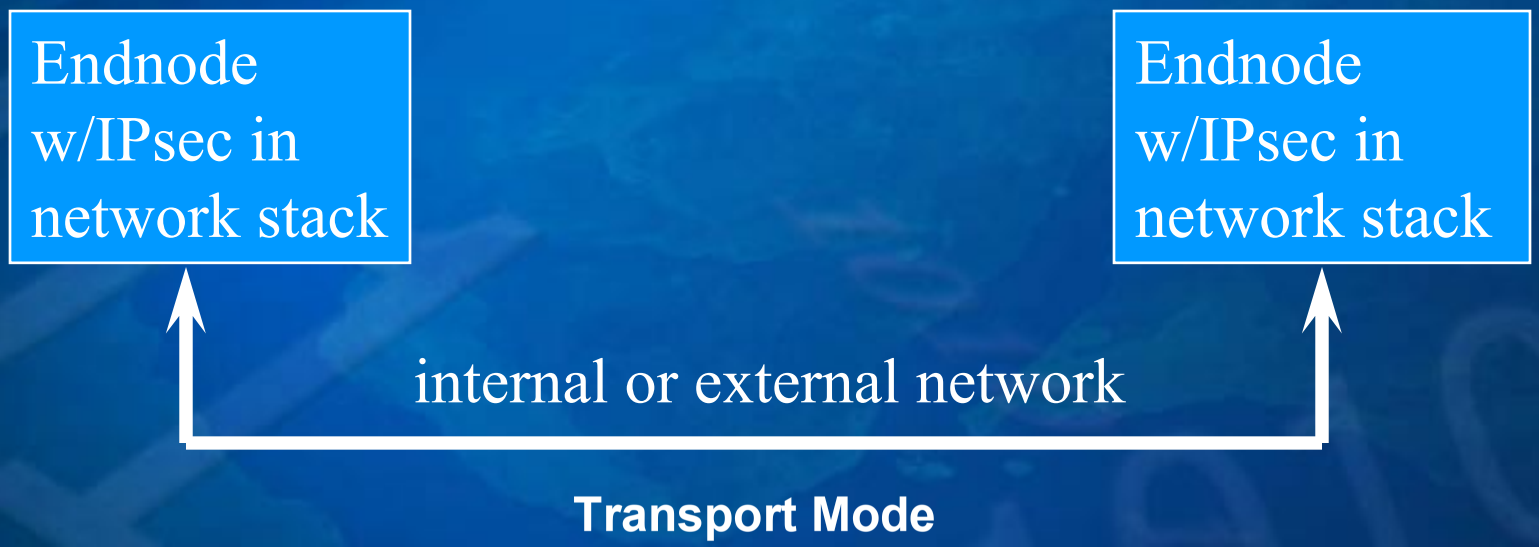
IPsec Scenario 2 Endnode to Firewall

- Mobile node connects home through Internet



IPsec Scenario 3
End to End

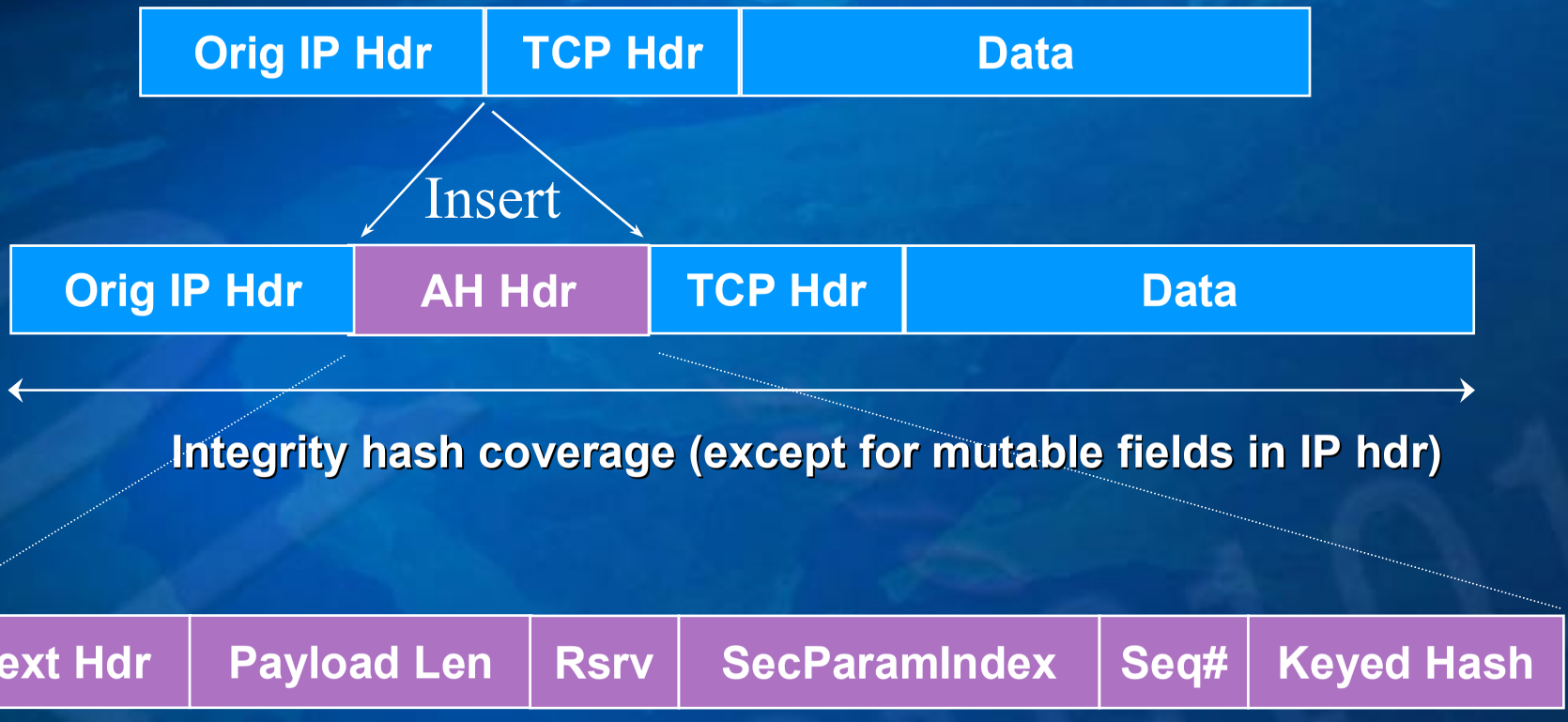
- Two nodes don't need to trust the network



Authentication Header (AH)

- Authentication is applied to the entire packet, with the mutable fields in the IP header zeroed out
- If both ESP and AH are applied to a packet, AH follows ESP

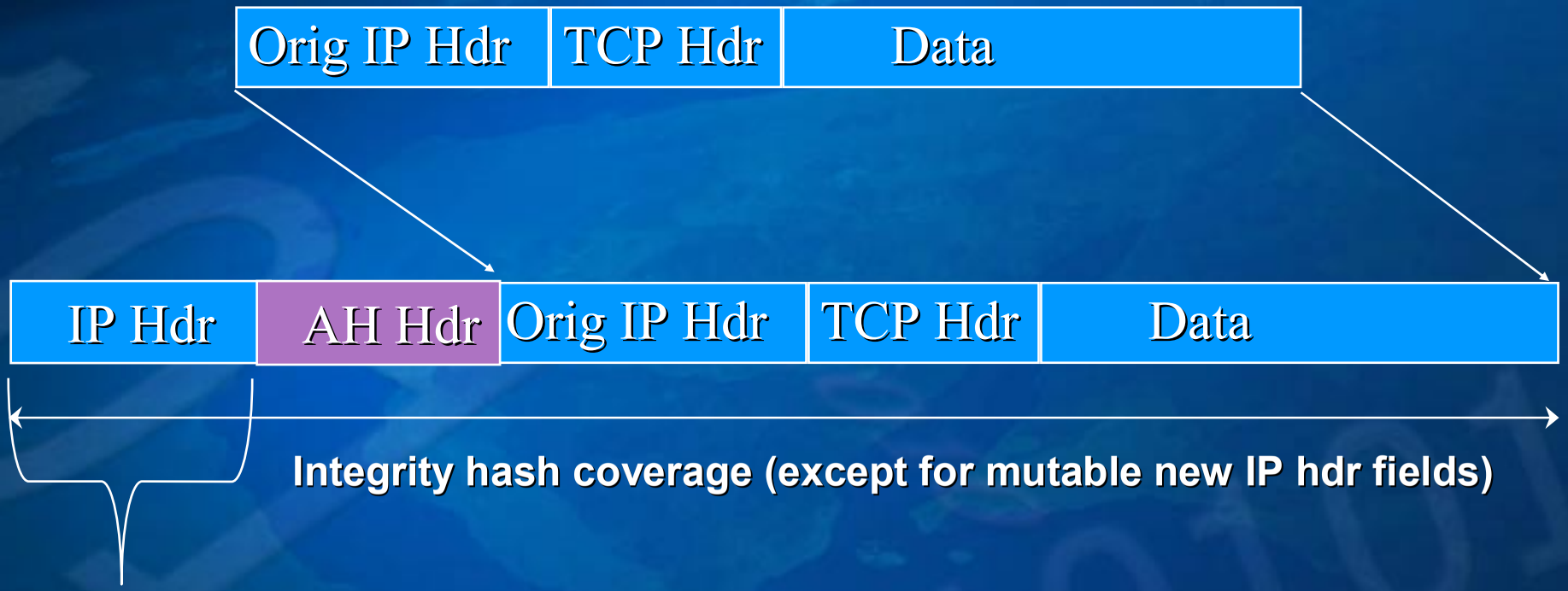
IPSEC Authentication Header (AH) in Transport Mode



AH is IP protocol 51

24 bytes total

IPSEC AH in Tunnel Mode

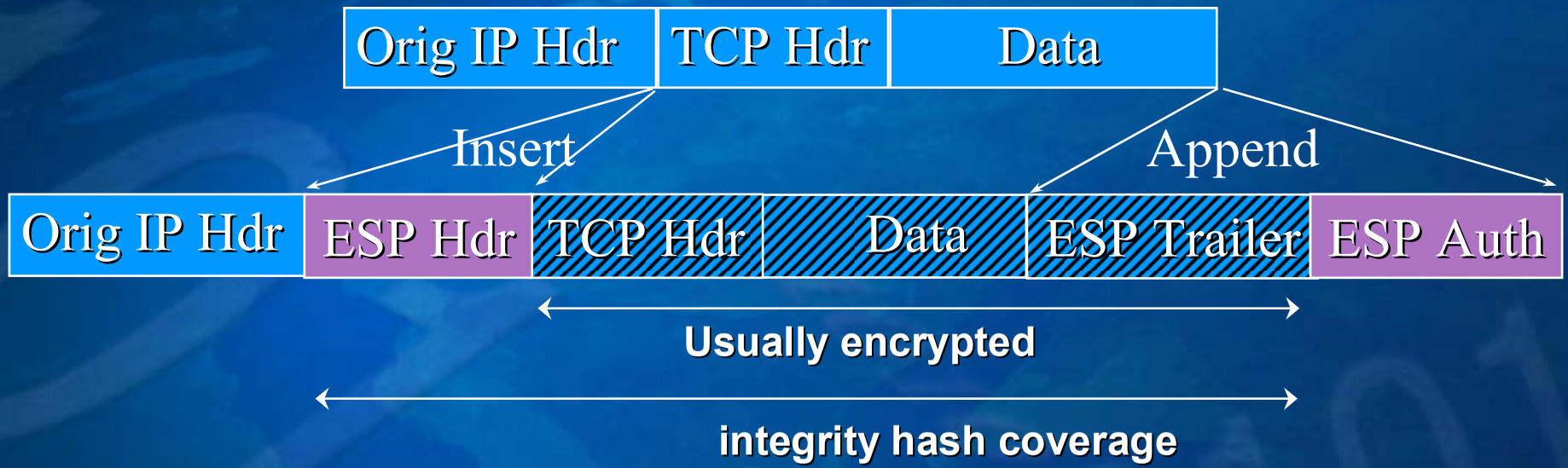


New IP header with source & destination IP address

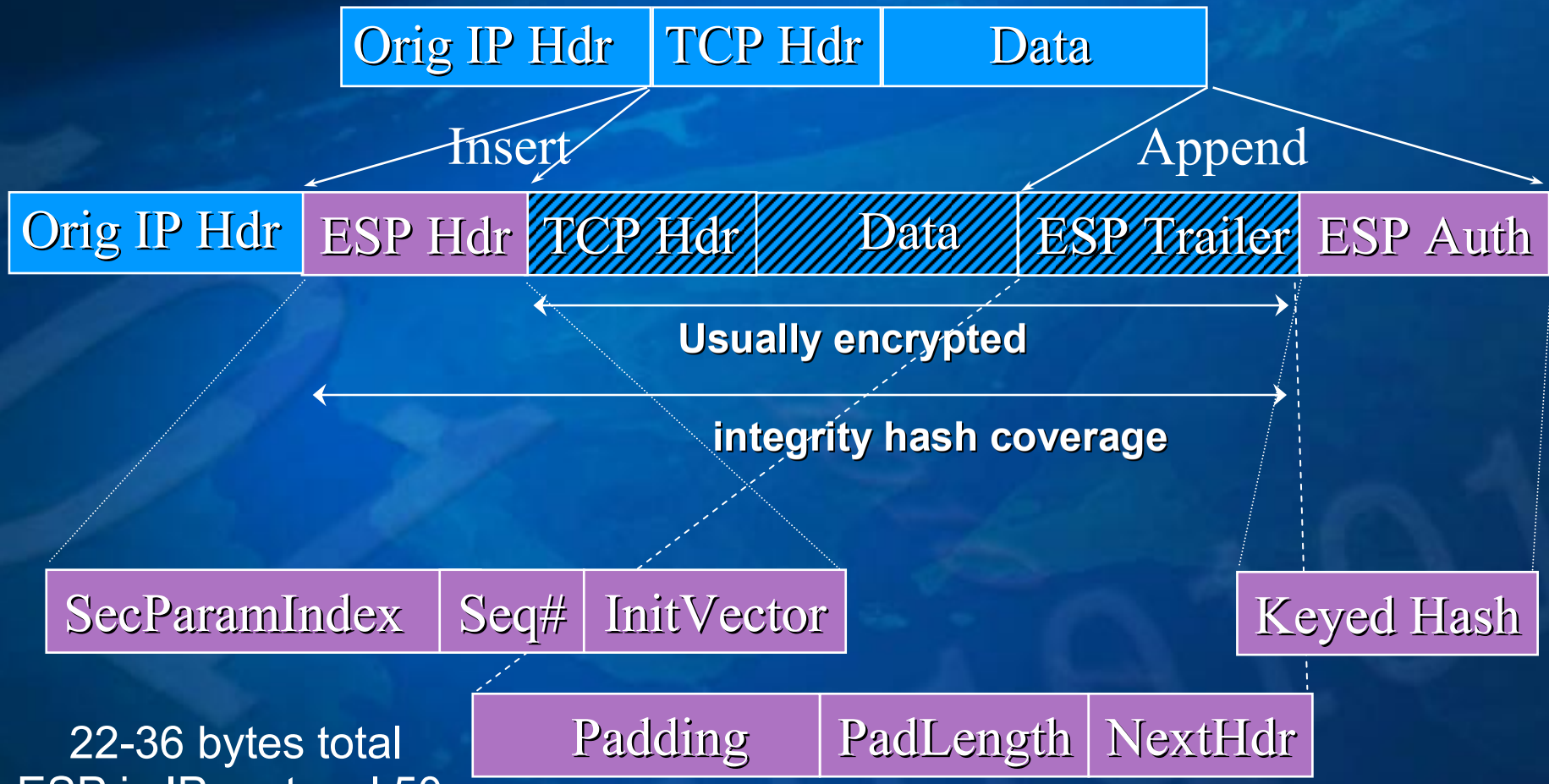
Encapsulated Security Payload (ESP)

- Must encrypt and/or authenticate in each packet
- Encryption occurs before authentication
- Authentication is applied to data in the IPSEC header as well as the data contained as payload

IPSEC ESP in Transport Mode

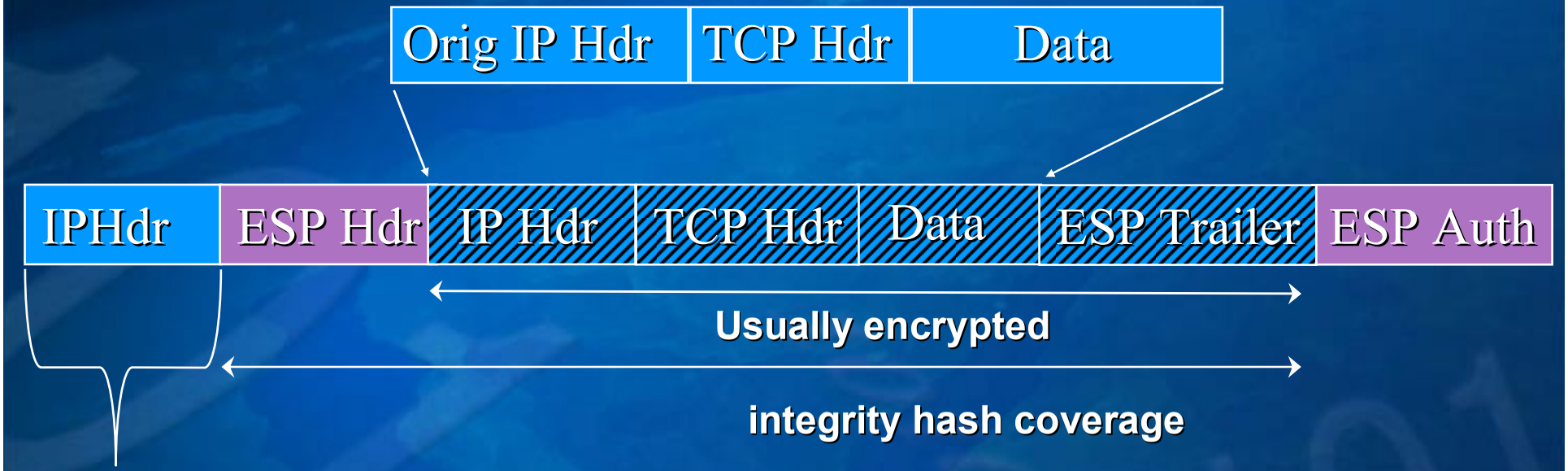


IPSEC ESP in Transport Mode



22-36 bytes total
ESP is IP protocol 50

IPSEC ESP Tunnel Mode



New IP header with source & destination IP address

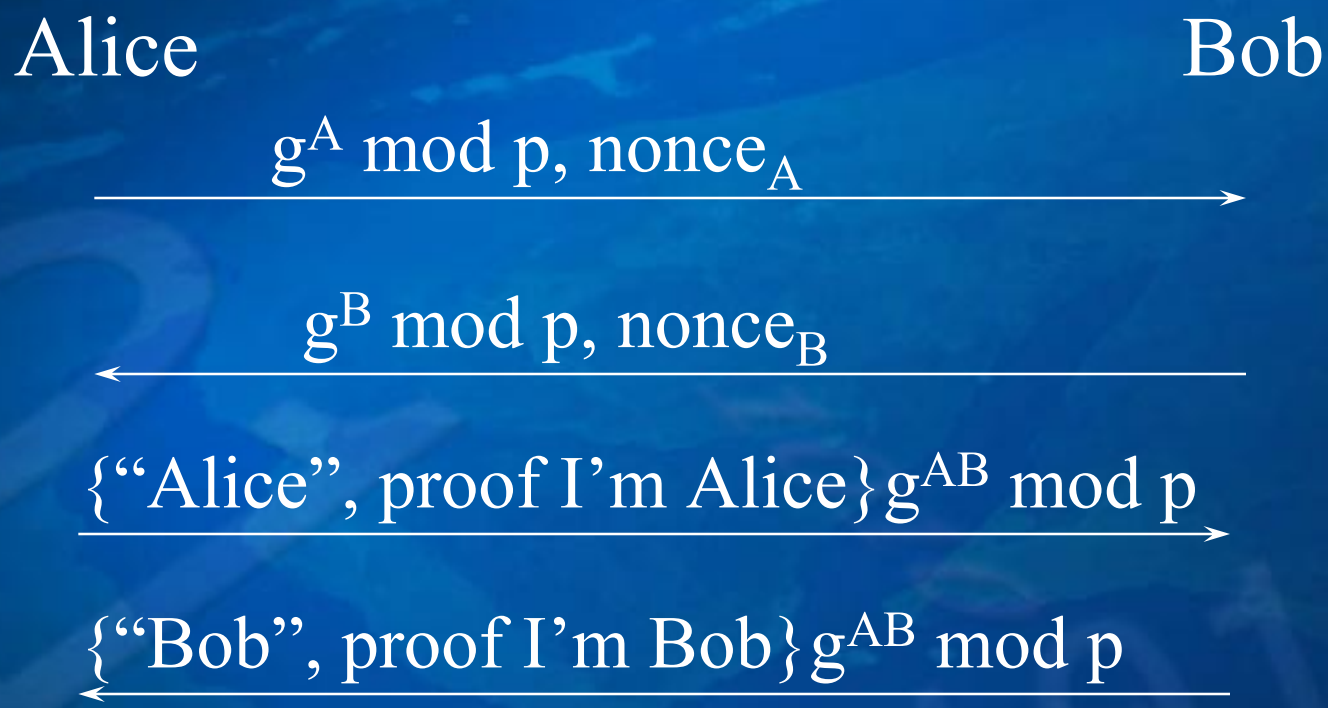
IPSEC Key Management

- IPSEC Key Management is all about establishing and maintaining Security Associations (SAs) between pairs of communicating hosts

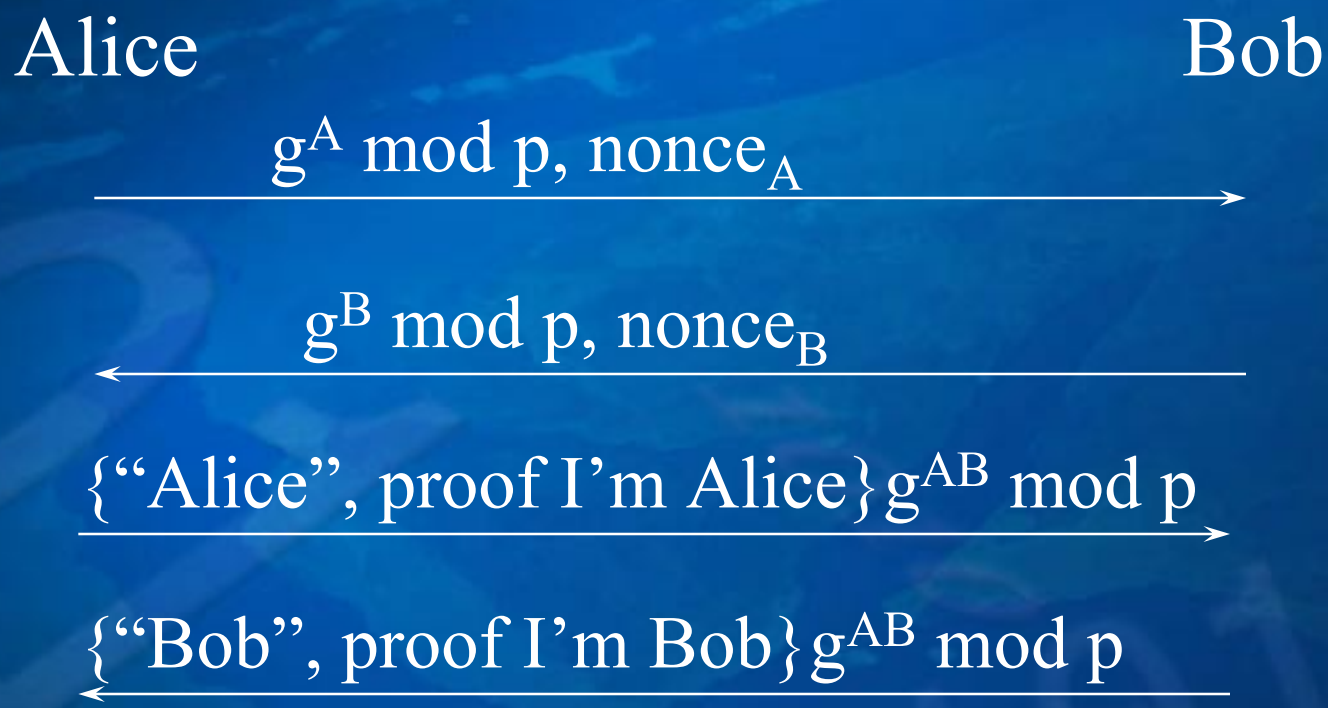
Security Associations (SA)

- New concept for IP communication
  - SA not a "connection", but very similar
  - Establishes trust between computers
- If securing with IPSEC, need SA
  - IKE protocol negotiates security parameters according to policy
  - Manages cryptographic keys and lifetime
  - Enforces trust by mutual authentication

General idea of IKEv2



General idea of IKEv2



- **It's just Diffie-Hellman Key Exchange!**

Internet Key Exchange (IKE)

- Resynchronize two ends of an IPsec SA
  - Choose cryptographic keys
  - Reset sequence numbers to zero
  - Authenticate endpoints
- Simple, right?
  - Design evolved into something very complex

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}
- And the winner was…

ISAKMP

- Internet Security Association and Key Management Protocol
- Gift to the IETF from NSA
- A “framework”, not a protocol. Complex encodings. Flexible yet constraining.
- Two “phases”. Phase 1 expensive, establishes a session key with which to negotiate multiple phase 2 sessions

Internet Key Exchange (IKE)

- Phase I
  - Establish a secure channel (ISAKMP SA)
  - Authenticate computer identity
- Phase II
  - Establishes a secure channel between computers intended for the transmission of data (IPSEC SA)

Internet Key Exchange (IKE)

- IKEv1 authors tried to fit academic papers (SKEME, OAKLEY) into ISAKMP
- Mostly a rewriting of ISAKMP, but not self-contained. Uses ISAKMP
- Since both so badly written, hadn't gotten thorough review
  - Really 3+ specs (ISAKMP, IKE, DOI)
  - Plus a few more (NAT traversal, etc.)

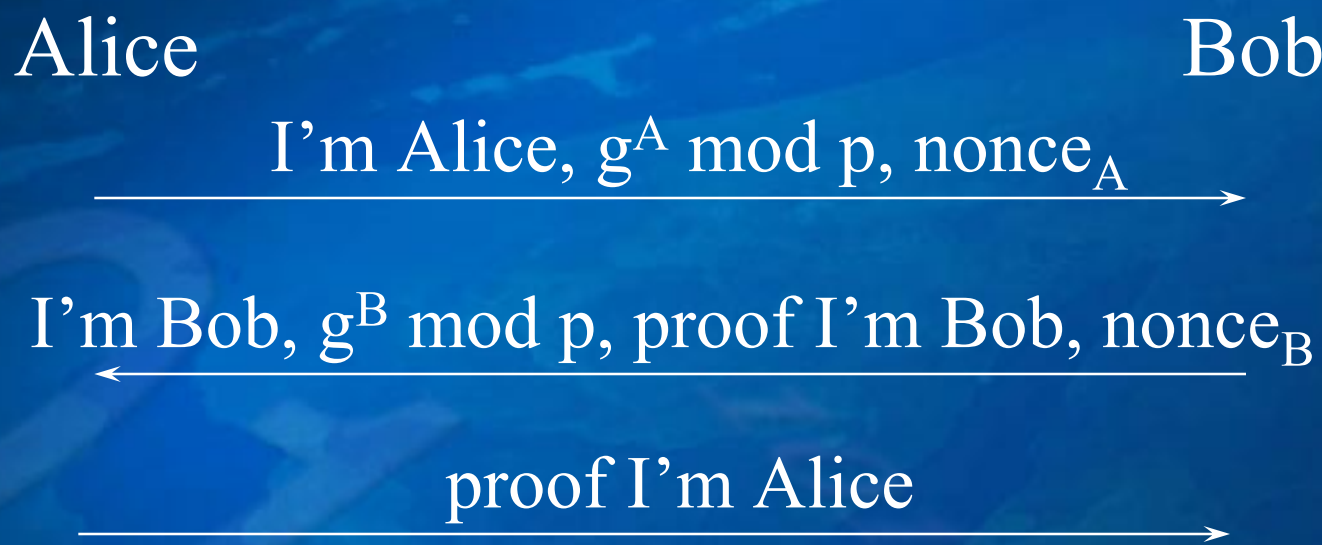
Imagine 150 pages of this!

- While Oakley defines “modes”, ISAKMP defines “phases”. The relationship between the two is very straightforward and IKE presents different exchanges as modes which operate in one of two phases. —RFC 2409

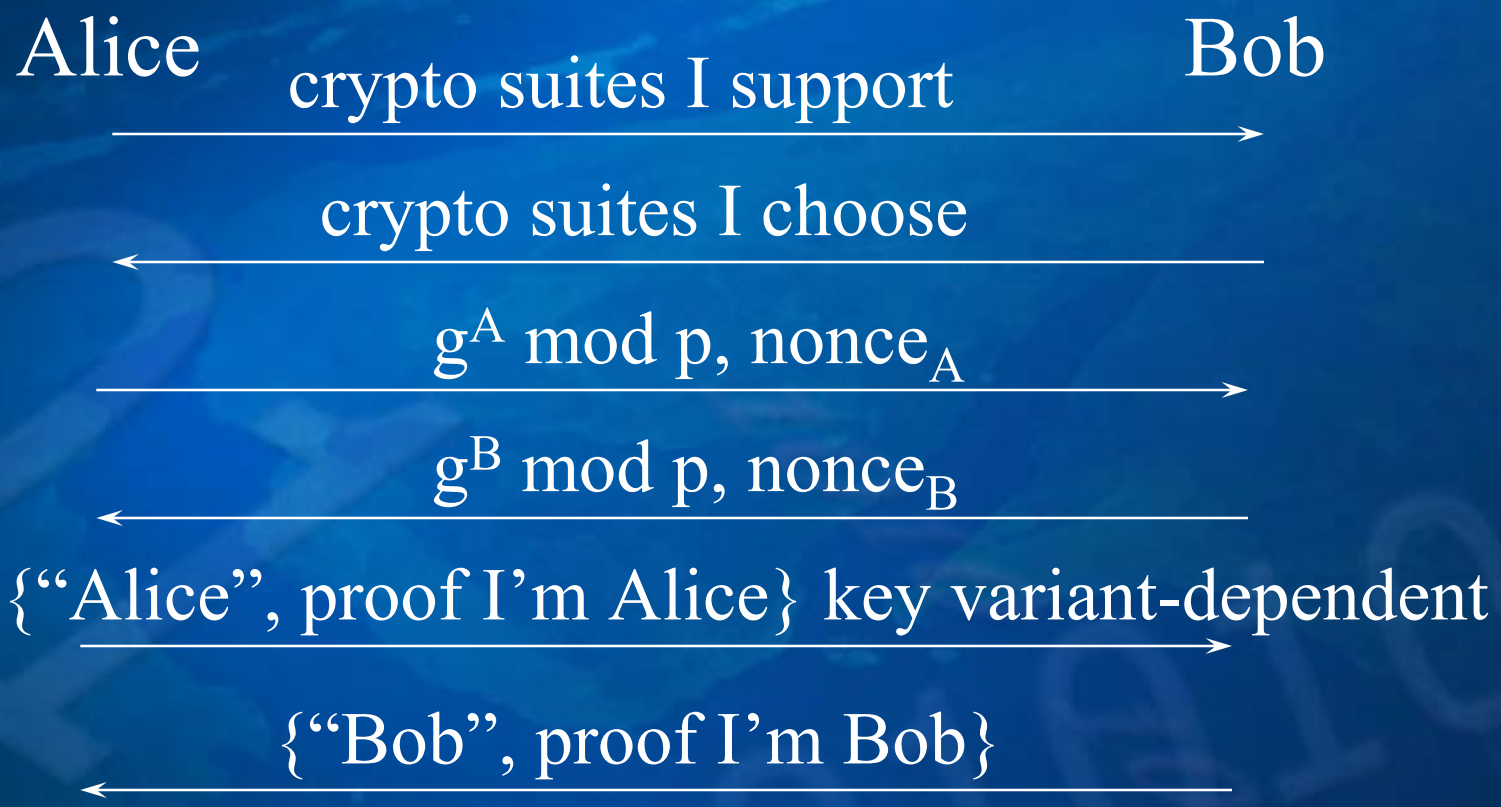
IKE

- Two phases, like ISAKMP
- Phase 1 is 8 protocols!
  - Two "modes": aggressive (3 msgs), and main (6 msgs)
  - Main does more, like hiding endpoint identifiers
- Phase 2 known as "quick mode"
- So 9 protocols (8 for phase 1, + phase 2)

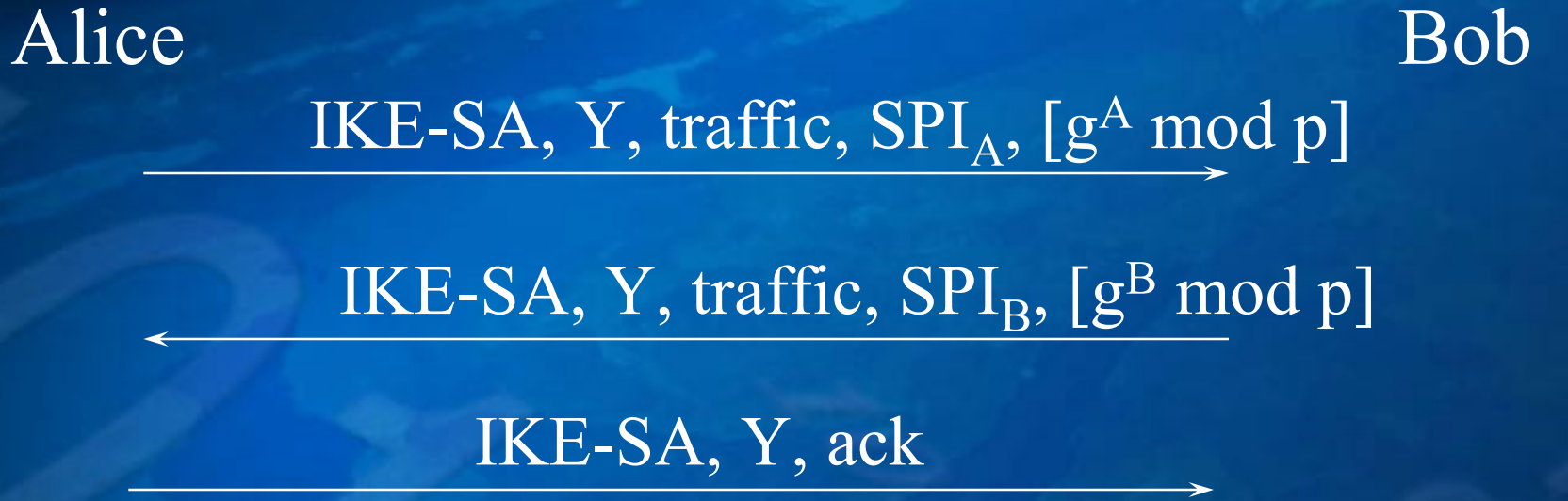
General Idea of Aggressive Mode



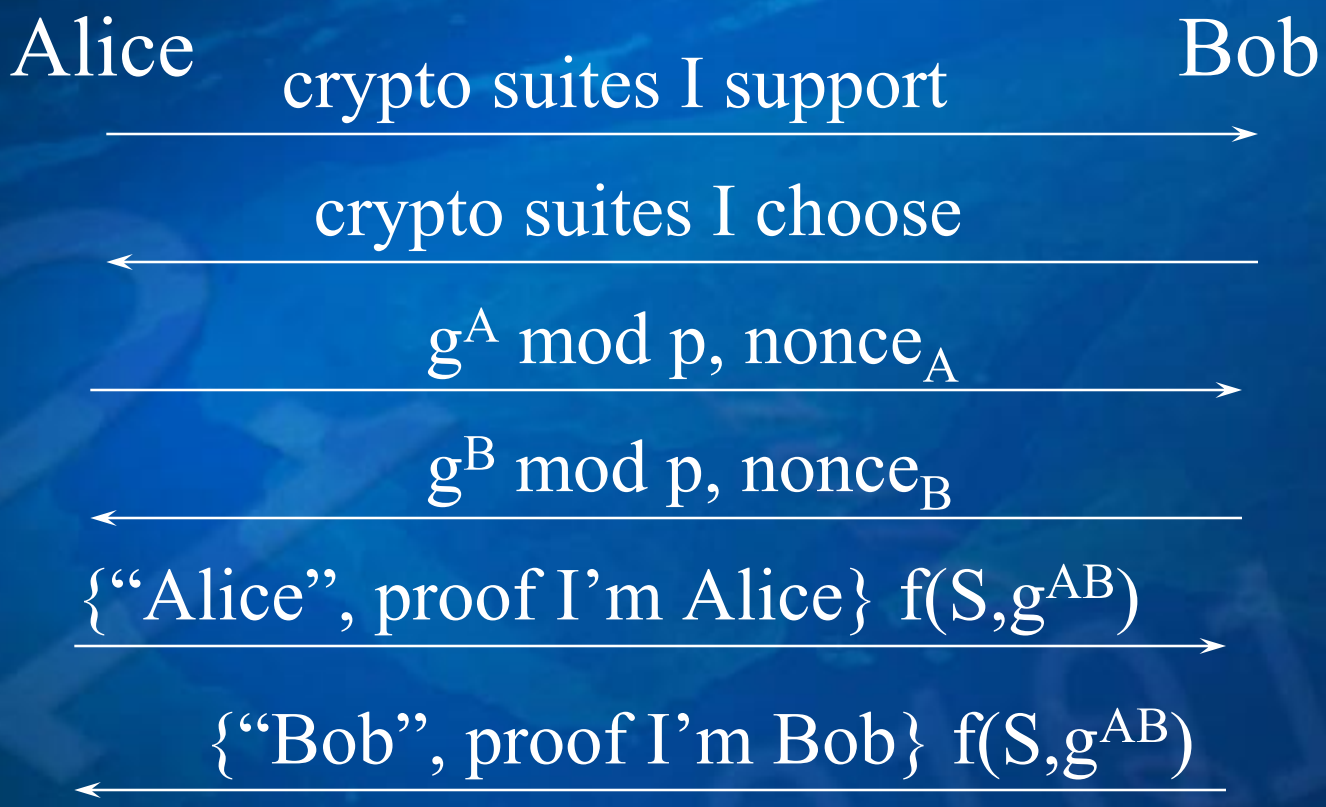
General Idea of Main Mode



General idea of Quick Mode



Main-Mode-Preshared key S

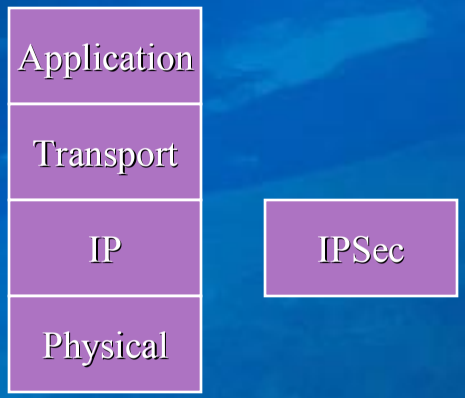


Additional IPSEC Topics
(if we have time)

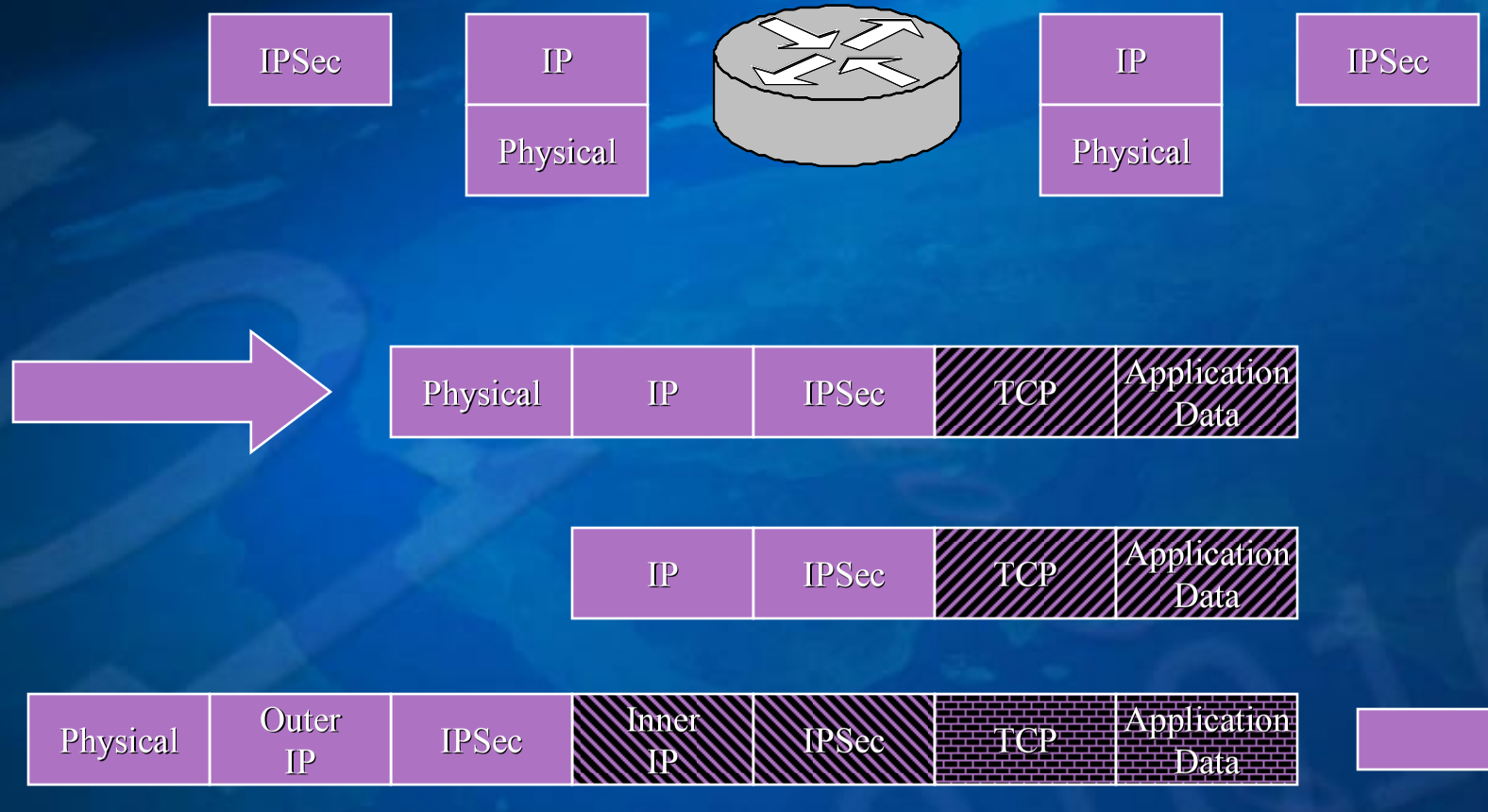
IPSEC Bundling/Wrapping

- Multiple IPSEC transforms may be wrapped successively around a single IP datagram
  - Example: IPSEC transport sent over an IPSEC tunnel

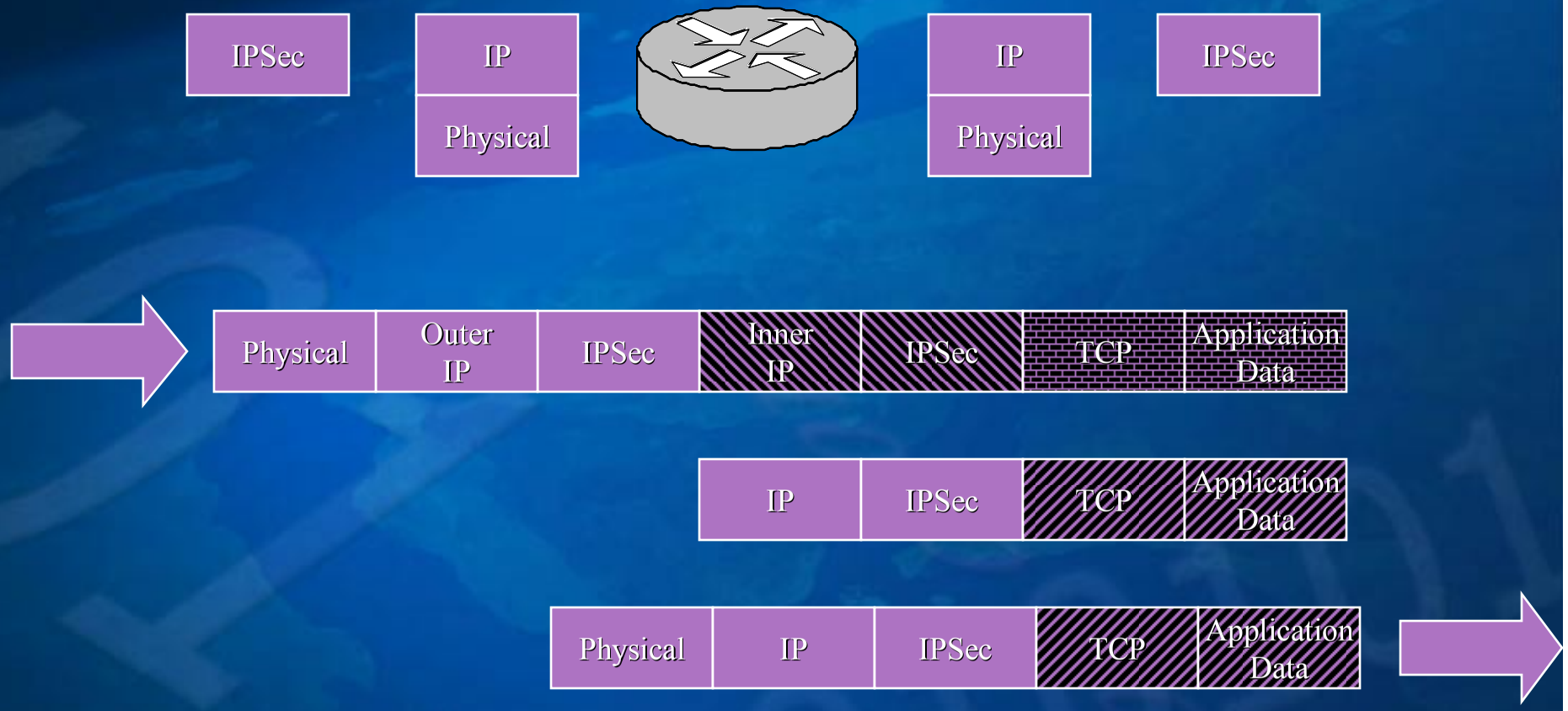
Sending in Transport Mode



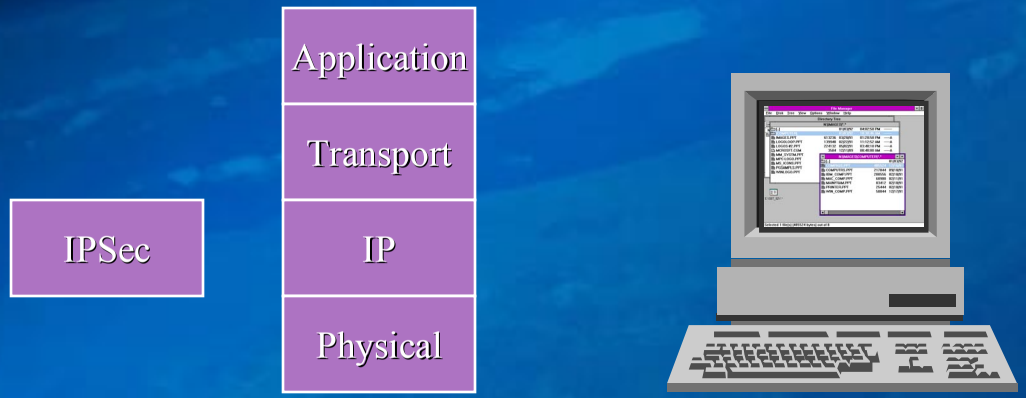
Sending in Tunnel Mode

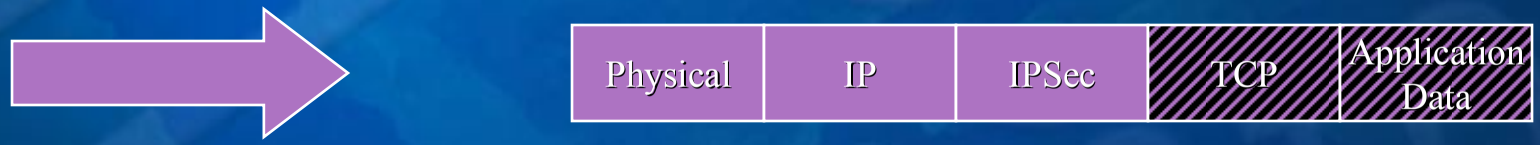


Receiving in Tunnel Mode



Receiving in Transport Mode

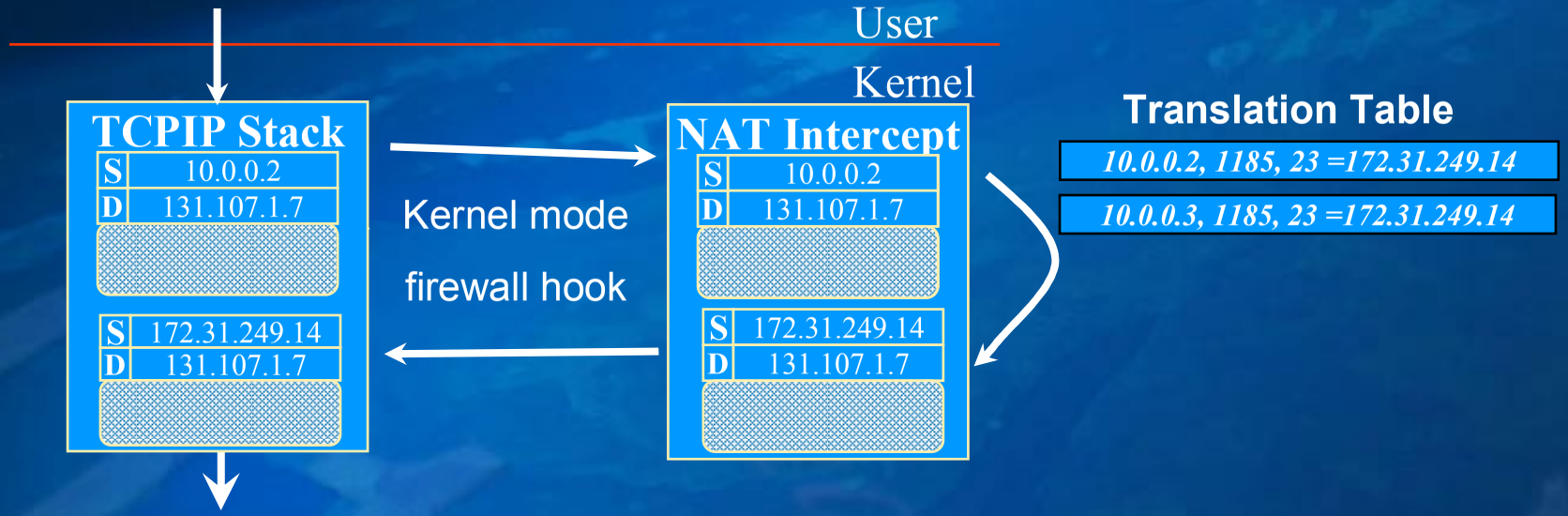




What is Network Address Translation (NAT) ?

- Network Address Translation (NAT)
  - Dynamically modifies source address
  - Dynamically recomputes interior UDP/TCP checksums
- Port Address Translation (PAT)
  - Dynamically modifies TCP/UDP source address and port
  - Dynamically recomputes interior UDP/TCP checksums

NATs Rewrite Address/Port Pairs



IPSEC AH and NAT

- Change in address or port will cause message integrity check to fail
  - Packet will be rejected by destination IPSEC
  - AH cannot be used with NAT or PAT devices

IPSEC ESP and NAT

- Can change IP header in special cases only
  - Special TCP/UDP ignores pseudo header used in checksum calculation
- Port information encrypted!
- Can't change ESP header because integrity hash coverage