Security Technologies and Hierarchical Trust

Today

1. Review/Summary of security technologies
    - Crypto and certificates
2. Combination of techniques in SSL
    - The basis for secure HTTP, *ssh*, secure IMAP, *scp*, secure *ftp*, …
    - Server authentication vs. peer/client authentication
3. Hierarchies in DNS and certificate distribution
    - Hierarchies as a basic technique for scale
    - Hierarchy of trust and autonomy

A Short Quiz

1. How does TCP rate control reflect "end-to-end" principles?
2. What is the key drawback of end-to-end rate control?
3. What is the most important advantage of symmetric crypto (DES) relative to asymmetric crypto (RSA)?
4. What is the most important advantage of asymmetric crypto relative to symmetric crypto?
5. What is the most important limitation/challenge for asymmetric crypto with respect to security?

What you really need to know, Part 1

Symmetric crypto (DES, 3DES, IDEA,…)

- Pro: cheap and fast, easily supported in hardware
- Con: requires a shared secret (private key, session key)

Asymmetric crypto (Diffie-Hellman, RSA)

- Pro: flexible: use for authentication, privacy, integrity.
- Con: slow
- Pro: solves the private key distribution problem
- Con: introduces a new public key distribution problem: secure binding of public keys to identities.

What you really need to know, Part 2

Asymmetric crypto can be used together with other techniques in a multitude of ways.

- Hybrid protocols combine advantages of both
  - Initial exchange uses asymmetric for authentication and (symmetric) session key exchange, then communicate with symmetric crypto. Example: SSL, TLS.
- Digital signatures based on secure hash functions
  - Compute a (small) hash over a (large) message efficiently.
  - MD5, SHA1: infeasible to forge another message with same hash
  - Encrypt the hash (and perhaps a nonce) with private key.

What you really need to know, Part 3

The “key” challenge today is public key distribution (and revocation).

Approach #1: trust e-mail/web (i.e., assume DNS and IP really go where you want, and authenticate the source.)

- Example: PGP, GPG, “pretty good”

Approach #2 : use a Public Key Infrastructure (PKI)

- Requires everyone to agree on a central point of trust (CA).
- Difficult to understand and deploy.
- Hierarchy helps.

Approach #3: “web of trust” in which parties establish pairwise trust and endorse public keys of third parties.

- Local example: SHARP. Involves transitive trust.

What you really need to know, Part #4

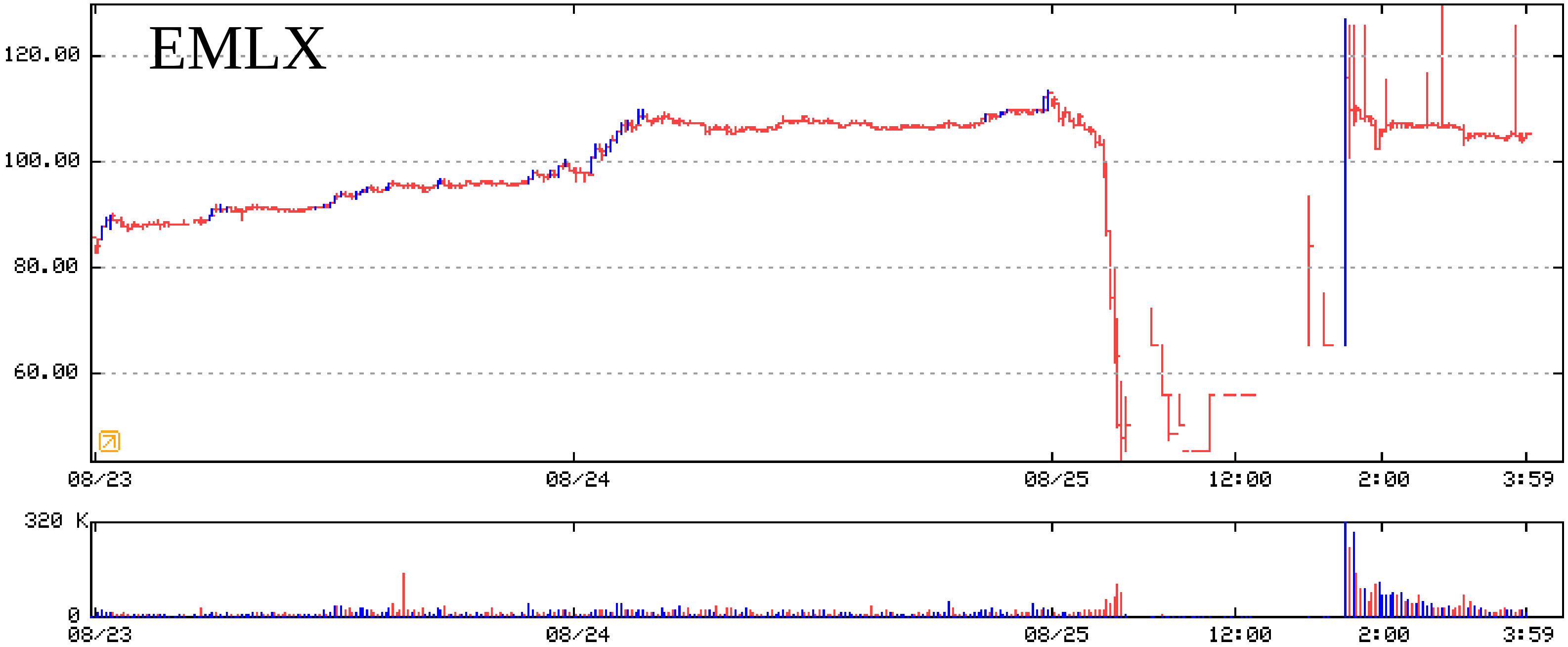
1. All of this relies on various fragile assumptions about people and communities.
   - Security technology only works if people use it.
   - Find the weakest link in the end-to-end chain.
   - Compromised key? All bets are off.
   - Beware false sense of security! (E.g., WEP)
2. Design for easy, incremental, organic deployment.
   - What layer? IPSEC or VPN vs. TLS
3. Understand full range of potential attacks.
   - Man-in-middle, replays and nonces, challenge/response
   - Useful model to guide analysis: logic of “belief” (BAN)

Projects: Resources/Ideas

- ModelNet emulation
- MACEDON
- Xen VMs/VPNs and Cereus/SIVIC
- Accountable design and SHARP
- IP/NFS interposition: instrumentation, translation
- Secure Web services, WS-Security, Shibboleth
- Computational steering
- Anypoint/XCP
- SFS

The Importance of Authentication

This is a picture of a $2.5B move in the value of Emulex Corporation, in response to a fraudulent press release by short-sellers through InternetWire in 2000. The release was widely disseminated by news media as a statement from Emulex management, but media failed to authenticate it.



[reproduced from *clearstation.com*]

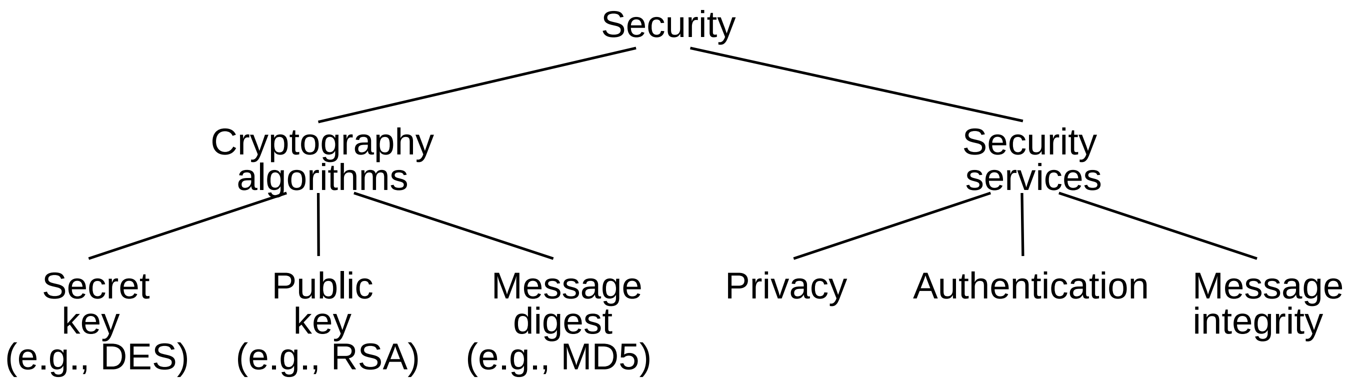
Crypto Summary

Cryptography functions

- Secret key (e.g., DES)
- Public key (e.g., RSA)
- Message digest (e.g., MD5)

Security services

- Privacy: preventing unauthorized release of information
- Authentication: verifying identity of the remote participant
- Integrity: making sure message has not been altered



Security

Cryptography algorithms

Security services

Secret key (e.g., DES)

Public key (e.g., RSA)

Message digest (e.g., MD5)

Privacy

Authentication

Message integrity

[Vahdat]

The Underpinnings of Security: Encryption

Two functions *Encrypt* and *Decrypt* with two keys K^{-1} and K

- Decrypt(K, Encrypt(K^{-1}, x)) = x
- Know *x* and Encrypt(K^{-1}, x), cannot comput K or K^{-1}

Secrecy:

- Know Encrypt(K^{-1}, x) but not K, cannot compute *x*

Integrity:

- Choose x, do not know K^{-1}: cannot compute *y* such that Decrypt(K, y) = x

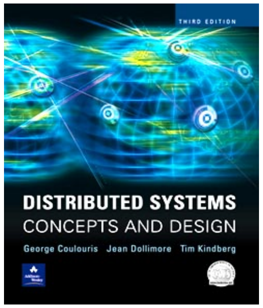
Digests are one-way (lossy) functions

- Cannot compute message from digest
- Cannot compute a second message with the same digest
- Sufficient for integrity

[Vahdat]

Figure 7.2
Familiar names for the protagonists in security protocols

Alice	First participant
Bob	Second participant
Carol	Participant in three- and four-party protocols
Dave	Participant in four-party protocols
Eve	Eavesdropper
Mallory	Malicious attacker
Sara	A server



Shared Key versus Public Key Cryptography

With shared key $K = K^{-1}$

- Mostly for pairwise communication or groups of principals that all trust one another (Data Encryption Standard or DES)

With public key cannot compute K from K^{-1}, or K^{-1} from K

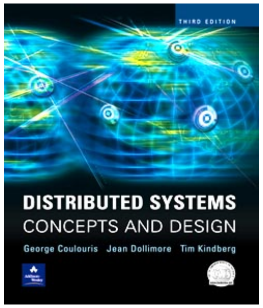
- K is made public, K^{-1} kept secret
- Can generate messages without knowing who will read it (certificate)
- Holder of K^{-1} can broadcast messages with integrity
- $(K^{-1})^{-1} = K$, send secret messages to holder of K^{-1}
- RSA (Rivest-Shamir-Adelman) most popular scheme

Secret Key much faster than Public Key

[Vahdat]

Figure 7.3
Cryptography notations

K_A	Alice’s secret key
K_B	Bob’s secret key
K_{AB}	Secret key shared between Alice and Bob
K_{Apriv}	Alice’s private key (known only to Alice)
K_{Apub}	Alice’s public key (published by Alice for all to read)
$\{M\}_K$	Message *M* encrypted with key *K*
$[M]_{\mathrm{K}}$	Message *M* signed with key *K*



Messages with both Authenticity and Secrecy

How does A send a message *x* to B with:

- Authenticity (B knows that only A could have sent it)
- Secrecy (A knows that only B can read the message)

[Vahdat]

Messages with both Authenticity and Secrecy

How does A send a message *x* to B with:

- Authenticity (B knows that only A could have sent it)
- Secrecy (A knows that only B can read the message)

A Transmits the following message *x*

- $\{\{x\}K_A^{-1}\}K_B$

What if *x* is large (performance concerns)?

- A transmits K_A to B, B transmits K_B to A
- A picks J_A, transmits $\{J_A\}K_B$ to B
- B picks J_B, transmits $\{J_B\}K_A$ to A
- Each computes secret key, $K_{sk} = \text{Hash}(J_A, J_B)$
- A transmits $\{x\}K_{sk}$ to B

[Vahdat]

Certification Authorities: Motivation

What is the problem with the previous approach?

[Vahdat]

Certification Authorities: Motivation

What is the problem with the previous approach?

- Evil router intercepts first public key exchange, imposes its own public key (with corresponding private key)
- Intercepts subsequent messages and inserts its own version
- Man in the middle attack

Solutions?

- Exchange keys over secure channel (in person)
- Trust certification authority with well-known public key

[Vahdat]

Message Digest

Cryptographic checksum

- Regular checksum protects receiver from accidental changes
- Cryptographic checksum protects receiver from malicious changes

One-way function

- Given cryptographic checksum for a message, virtually impossible to determine what message produced that checksum; it is not computationally feasible to find two messages that hash to the same cryptographic checksum.

Relevance

- Given checksum for a message and you are able to compute exactly the same checksum for that message, then highly likely this message produced given checksum

[Vahdat]

Message Integrity Protocols

Digital signature using RSA

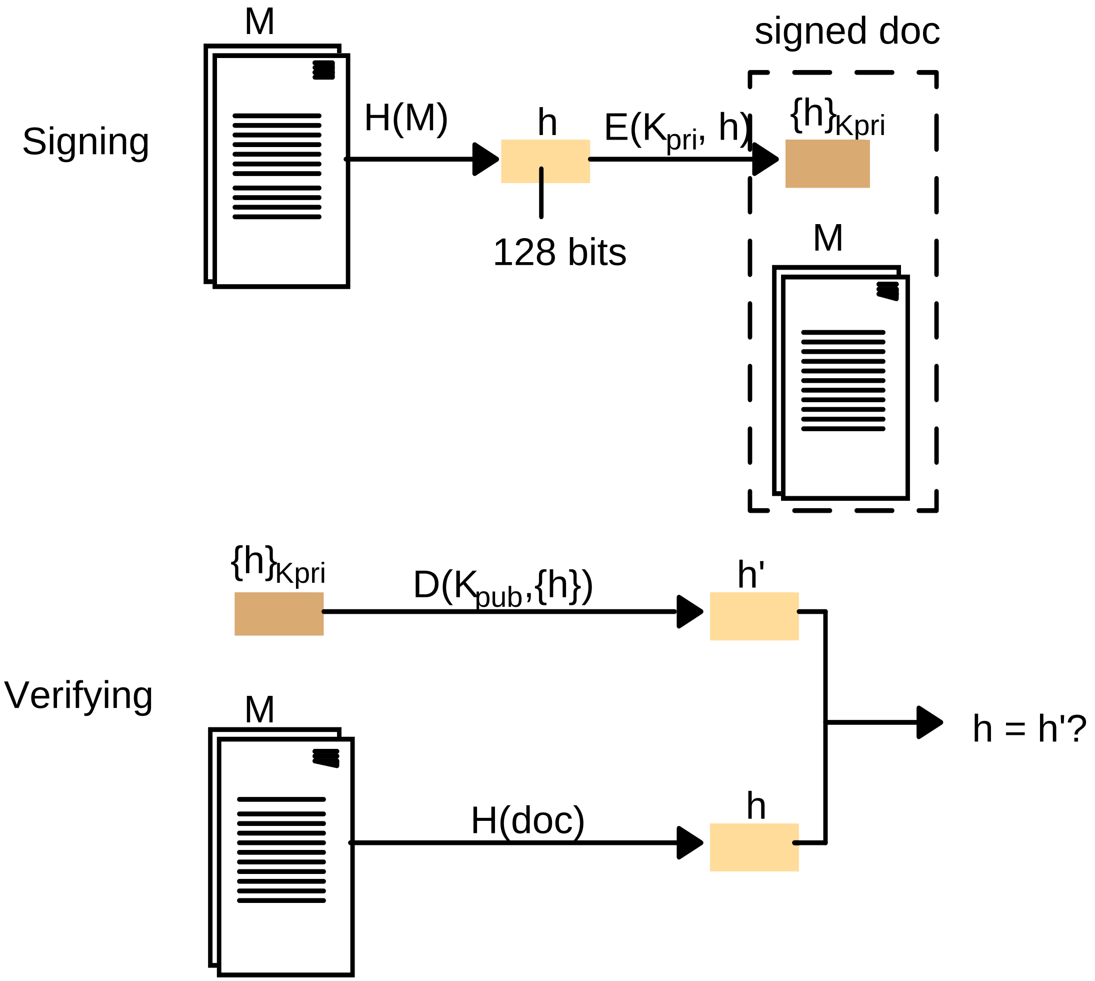
- Compute signature with private key and verify with public key
- A transmits M, $\{D(M)\}KA_{private}$
- Receiver decrypts digest using KA_{public}

Digital signature with secret key (server as escrow agent)

- $A \rightarrow server, A, \{D(M)\}_{KA}$
- $Server \rightarrow A, \{A, D(M), t\}_{KS}$
- $A \rightarrow B, M, \{A, D(M), t\}_{KS}$
- $B \rightarrow S, B, \{A, D(M), t\}_{KS}$
- $S \rightarrow B, \{A, D(M), t\}_{KB}$

[Vahdat]

Figure 7.11
Digital signatures with public keys



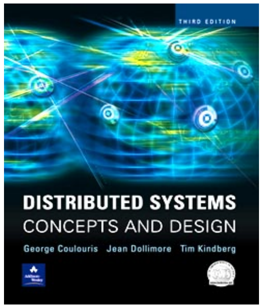
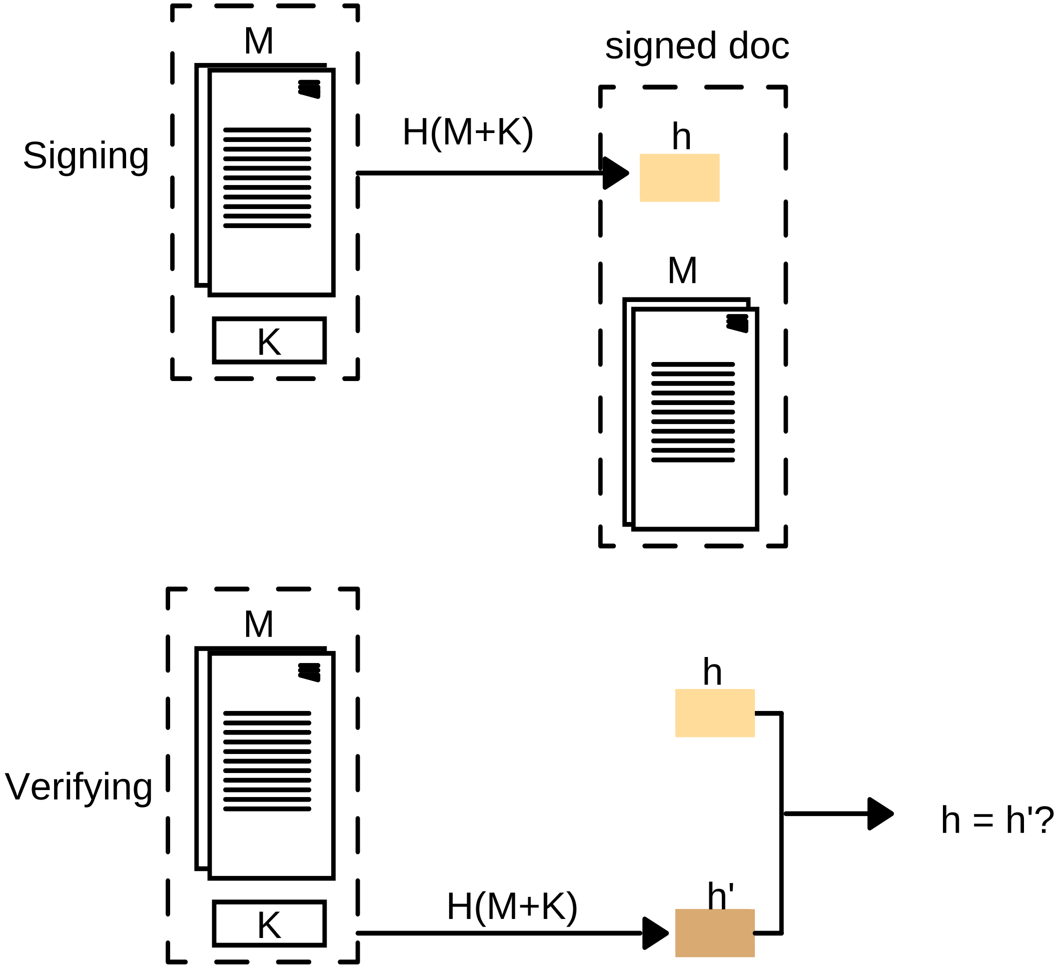
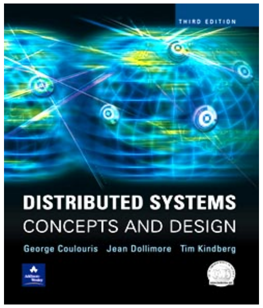


Figure 7.12
Low-cost signatures with a shared secret key

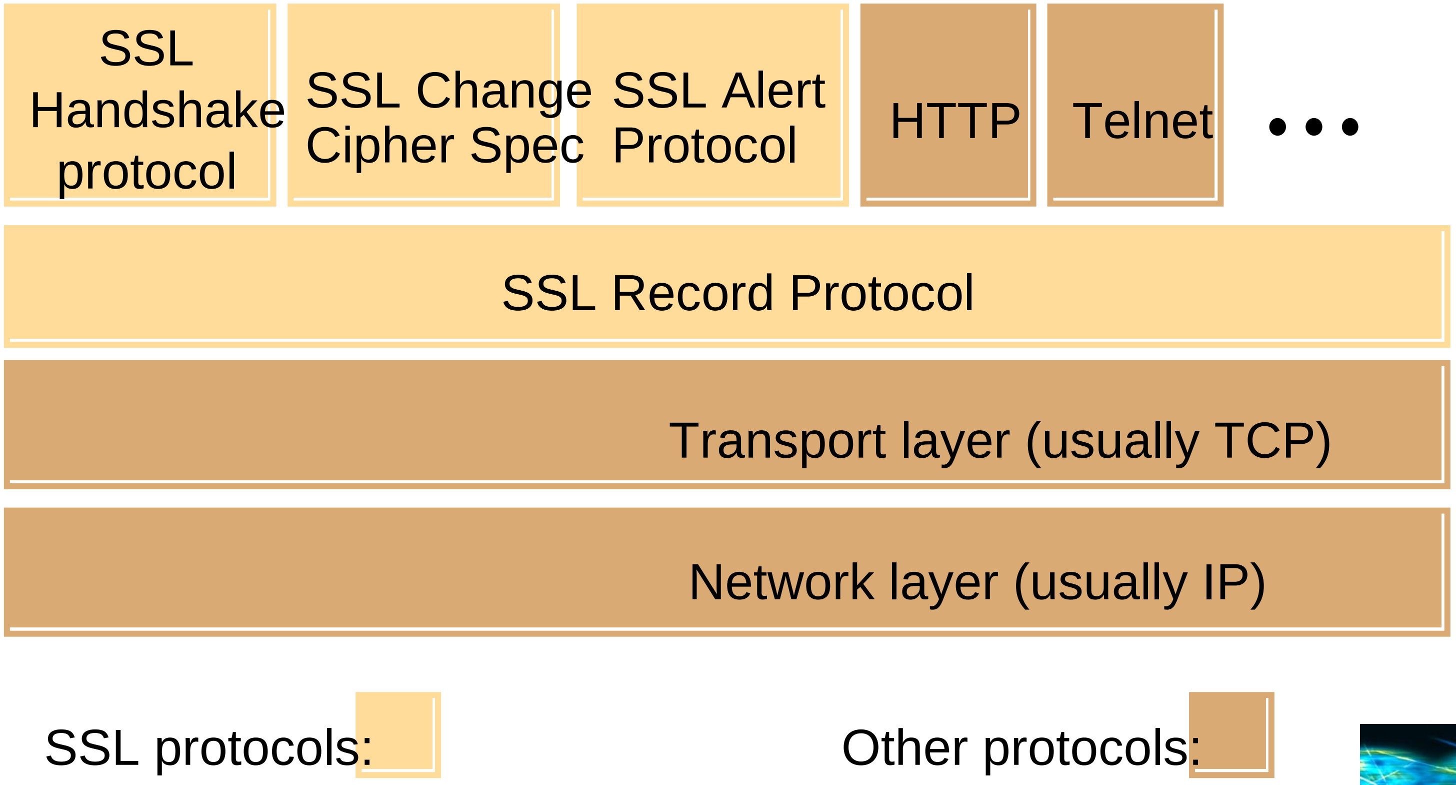




What happens…

https://www.consumefest.com/checkout.html

Figure 7.17
SSL protocol stack



SSL Handshake protocol

SSL Change Cipher Spec

SSL Alert Protocol

HTTP

Telnet

• • •

SSL Record Protocol

Transport layer (usually TCP)

Network layer (usually IP)

SSL protocols: 🟨

Other protocols: 🟧

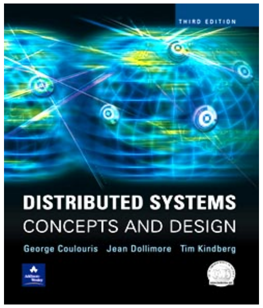
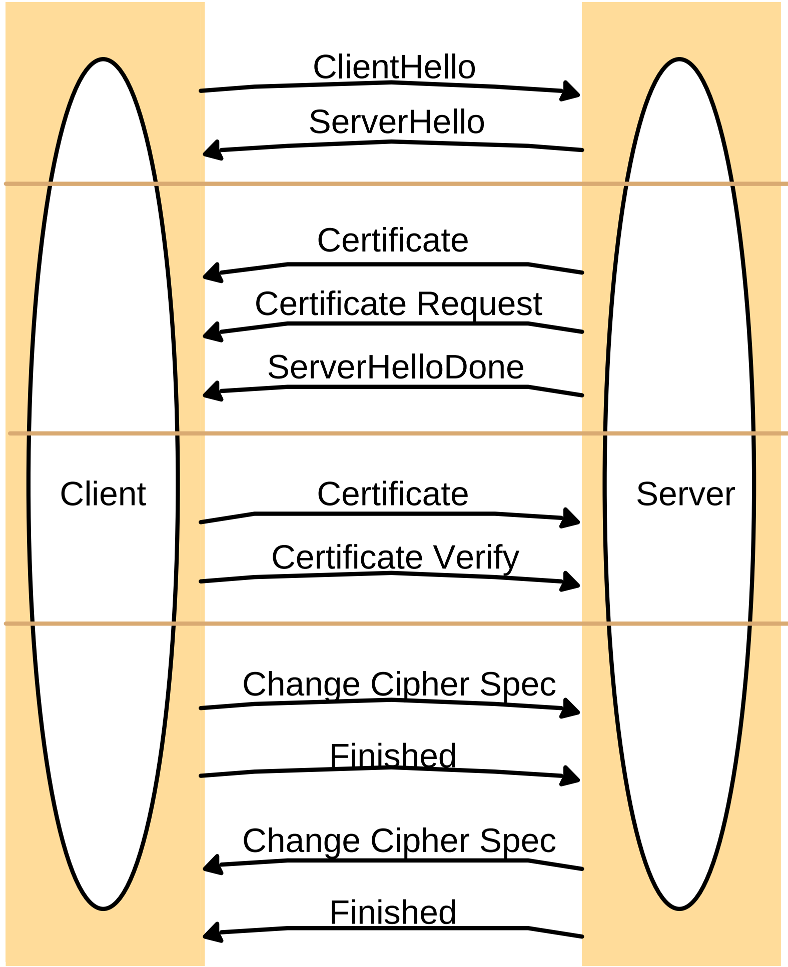


Figure 7.18
SSL handshake protocol



Client

Server

ClientHello

ServerHello

Establish protocol version, session ID, cipher suite, compression method, exchange random values

Certificate

Certificate Request

ServerHelloDone

Optionally send server certificate and request client certificate

Certificate

Certificate Verify

Send client certificate response if requested

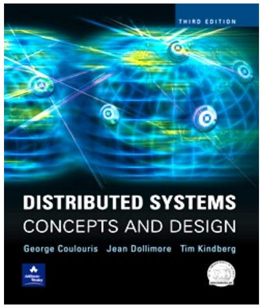
Change Cipher Spec

Finished

Change Cipher Spec

Finished

Change cipher suite and finish handshake



SSL Questions

Why doesn’t SSL need/use an authentication service like Kerberos?

How do SSL endpoints verify the integrity of certificates (IDs)?

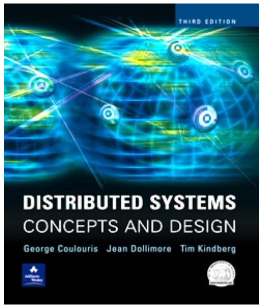
Does s-http guarantee non-repudiation for electronic transactions? Why/how or why not?

Does SSL guarantee security of (say) credit numbers in electronic commerce?

Why does SSL allow endpoints to use fake IDs?

Figure 7.13
X509 Certificate format

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of validity	Not Before Date, Not After Date
Administrative information	Version, Serial Number
Extended Information	



Hybrid Crypto in SSL

Why does SSL “change ciphers” during the handshake?

How does SSL solve the key distribution problem for symmetric crypto?

Is key exchange vulnerable to man-in-the-middle attacks?

Figure 7.14
Performance of encryption and secure digest algorithms

	Key size/hash size (bits)	*Extrapolated speed (kbytes/sec.)*	*PRB optimized (kbytes/s)*
TEA	128	700	-
DES	56	350	7746
Triple-DES	112	120	2842
IDEA	128	700	4469
RSA	512	7	-
RSA	2048	1	-
MD5	128	1740	6242
SHA	160	750	2516

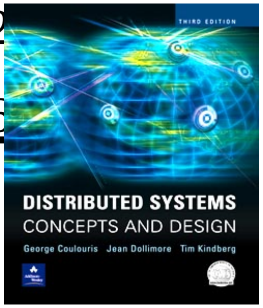
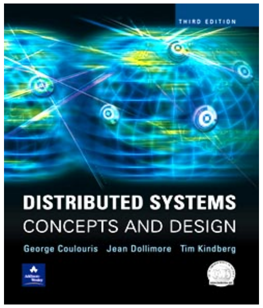


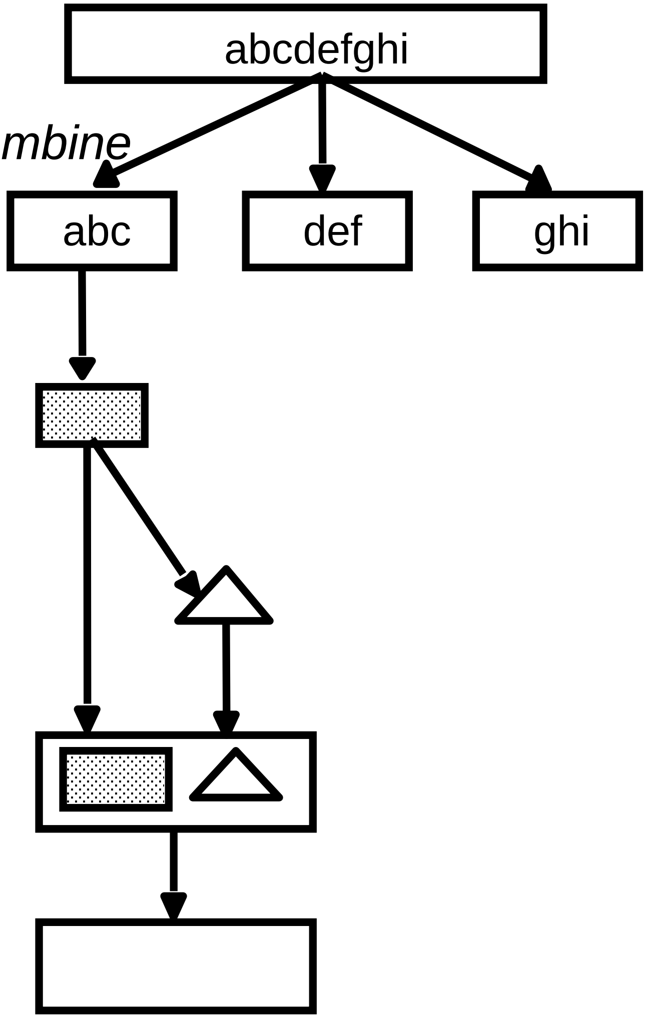
Figure 7.19
SSL handshake configuration options

Component	*Description*	*Example*
Key exchange method	the method to be used for exchange of a session key	RSA with public-key certificates
Cipher for data transfer	the block or stream cipher to be used for data	IDEA
Message digest function	for creating message authentication codes (MACs)	SHA


THIRD EDITION

DISTRIBUTED SYSTEMS
CONCEPTS AND DESIGN
George Coulouris Jean Dollimore Tim Kindberg

Figure 7.20
SSL record protocol



Application data

Fragment/combine

Record protocol units

Compress

Compressed units

Hash

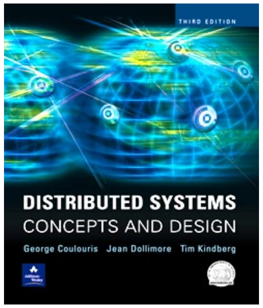
MAC

Encrypt

Encrypted

Transmit

TCP packet



www.OpenSSH.com
RLOGIN 1982-99 LIVED LIFE WITH JOY
TELNET 1971-99 WON'T CRYPT
RSH 1982-99 ++
Putting an end to unencrypted network logins

Key Distribution

Certificate

- Special type of digitally signed document:

  "I certify that the public key in this document belongs to the entity named in this document, signed X."

- Name of the entity being certified
- Public key of the entity
- Name of the certified authority
- Digital signature

Certified Authority (CA)

- Administrative entity that issues certificates
- Public key must be widely available (e.g., Verisign)

[Vahdat]

Key Distribution (cont)

Chain of Trust

- If *X* certifies that a certain public key belongs to *Y*, and *Y* certifies that another public key belongs to *Z*, then there exists a chain of certificates from *X* to *Z*
- Someone that wants to verify *Z*'s public key has to know *X*'s public key and follow the chain
- *X* forms the root of a tree (web?)

Certificate Revocation List

- What happens when a private key is compromised?

[Vahdat]

DNS 101

Domain names are the basis for the Web’s global URL space.

provides a symbolic veneer over the IP address space

names for autonomous naming domains, e.g., *cs.duke.edu*

names for specific nodes, e.g., *fran.cs.duke.edu*

names for service aliases (e.g., www, mail servers)

- Almost every Internet application uses domain names when it establishes a connection to another host.

The *Domain Name System* (DNS) is a planetary name service that translates Internet domain names.

maps <*node name*> to <*IP address*>

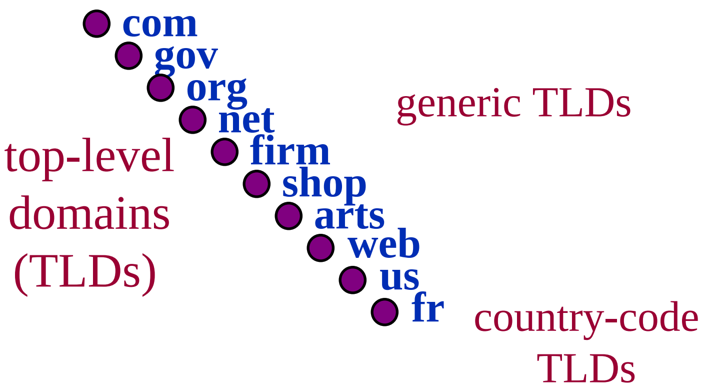
(mostly) independent of location, routing etc.

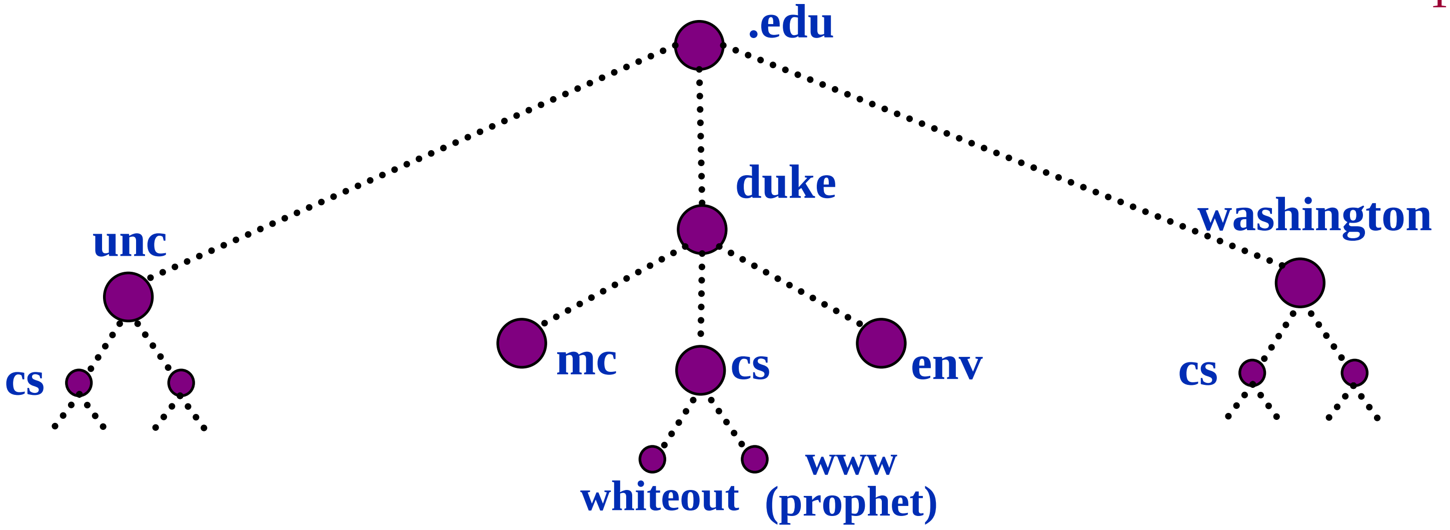
Domain Name Hierarchy

DNS name space is *hierarchical*:

- fully qualified names are “little endian”
- scalability
- decentralized administration
- domains are naming *contexts*

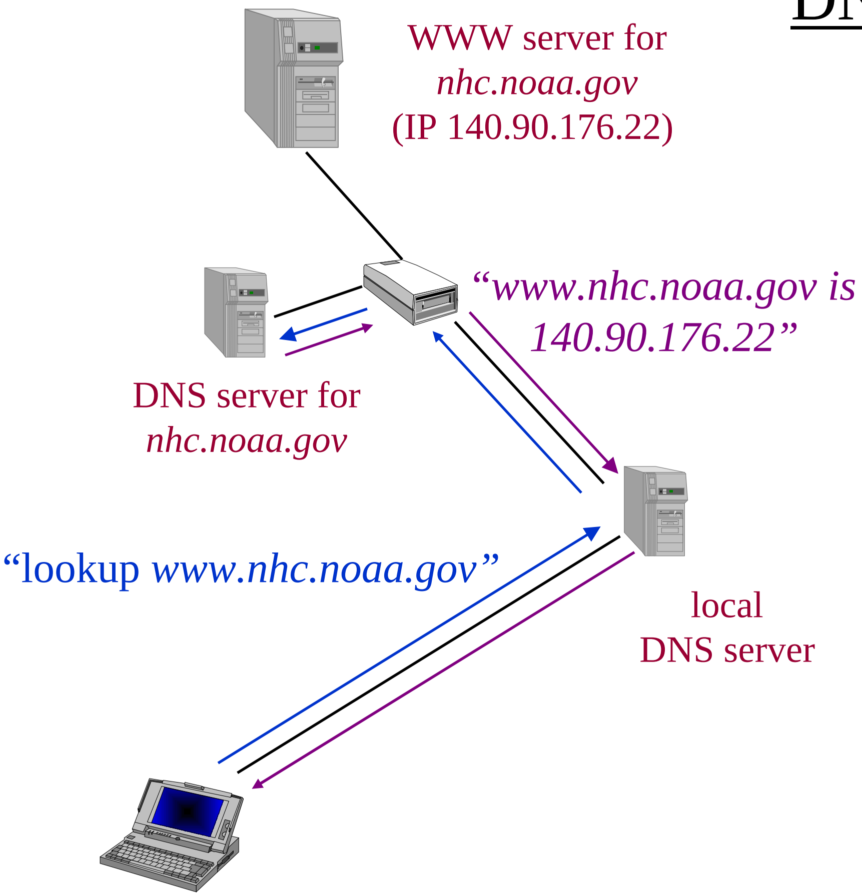
replaces primordial flat *hosts.txt* namespace





How is this different from hierarchical directories in distributed file systems? Do we already know how to implement this?

DNS Implementation 101



WWW server for
nhc.noaa.gov
(IP 140.90.176.22)

"www.nhc.noaa.gov is 140.90.176.22"

DNS server for
nhc.noaa.gov

"lookup *www.nhc.noaa.gov*"

local
DNS server

DNS protocol/implementation:

- UDP-based client/server
- client-side *resolvers*

  typically in a library

  gethostbyname, gethostbyaddr
- cooperating servers

  query-answer-referral model

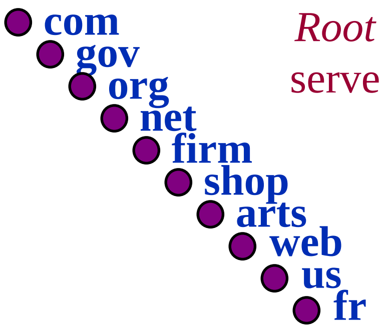
  forward queries among servers

  server-to-server may use TCP ("zone transfers")
- common implementation: BIND

DNS Name Server Hierarchy

DNS servers are organized into a hierarchy that mirrors the name space.

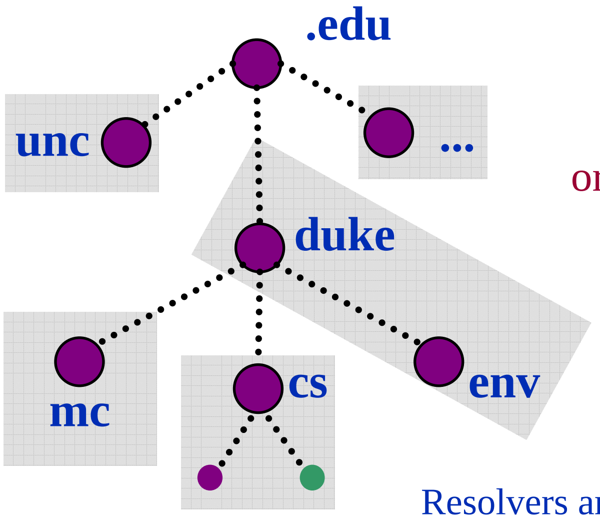
Specific servers are designated as *authoritative* for portions of the name space.



Root servers list servers for every TLD.

Servers may delegate management of *subdomains* to child name servers.

Parents refer subdomain queries to their children.



Subdomains correspond to organizational (*admininstrative*) boundaries, which are not necessarily geographical.

Servers are bootstrapped with pointers to selected peer and parent servers.

Resolvers are bootstrapped with pointers to one or more local servers; they issue *recursive* queries.

DNS: The Big Issues

1. Naming contexts

   I want to use short, unqualified names like *smirk* instead of *smirk.cs.duke.edu* when I’m in the *cs.duke.edu* domain.

2. What about trust? How can we know if a server is authoritative, or just an impostor?

   What happens if a server lies or behaves erratically? What denial-of-service attacks are possible? What about privacy?

3. What if an “upstream” server fails?

4. Is the hierarchical structure sufficient for scalability?

   more names vs. higher request rates

DNS: The Politics

He who controls DNS controls the Internet.

- TLD registry run by Network Solutions, Inc. until 9/98.
    - US government (NSF) granted monopoly, regulated but not answerable to any US or international authority.
- Registration has transitioned to a more open management structure involving an alphabet soup of organizations.

For companies, domain name == brand.

- Squatters register/resell valuable domain name “real estate”.
- Who has the right to register/use, e.g., *coca-cola.com*?