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

4. Older slides on servlets at the end for tinyserver lab.
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
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
- Sufficient for integrity

[Vahdat]
**Figure 7.2**
Familiar names for the protagonists in security protocols

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>First participant</td>
</tr>
<tr>
<td>Bob</td>
<td>Second participant</td>
</tr>
<tr>
<td>Carol</td>
<td>Participant in three- and four-party protocols</td>
</tr>
<tr>
<td>Dave</td>
<td>Participant in four-party protocols</td>
</tr>
<tr>
<td>Eve</td>
<td>Eavesdropper</td>
</tr>
<tr>
<td>Mallory</td>
<td>Malicious attacker</td>
</tr>
<tr>
<td>Sara</td>
<td>A server</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_A$</td>
<td>Alice’s secret key</td>
</tr>
<tr>
<td>$K_B$</td>
<td>Bob’s secret key</td>
</tr>
<tr>
<td>$K_{AB}$</td>
<td>Secret key shared between Alice and Bob</td>
</tr>
<tr>
<td>$K_{Apriv}$</td>
<td>Alice’s private key (known only to Alice)</td>
</tr>
<tr>
<td>$K_{Apub}$</td>
<td>Alice’s public key (published by Alice for all to read)</td>
</tr>
<tr>
<td>${M}_K$</td>
<td>Message $M$ encrypted with key $K$</td>
</tr>
<tr>
<td>$[M]_K$</td>
<td>Message $M$ signed with key $K$</td>
</tr>
</tbody>
</table>
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)
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?
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
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
Message Integrity Protocols

Digital signature using RSA
- Compute signature with private key and verify with public key
- A transmits $M, \{D(M)\}_K_{A_{\text{private}}}$
- Receiver decrypts digest using $K_{A_{\text{public}}}$

Digital signature with secret key (server as escrow agent)
- $A \rightarrow$ server, $A, \{D(M)\}_K_{A}$
- Server $\rightarrow A, \{A, D(M), t\}_K_{S}$
- $A \rightarrow B, M, \{A, D(M), t\}_K_{S}$
- $B \rightarrow S, B, \{A, D(M), t\}_K_{S}$
- $S \rightarrow B, \{A, D(M), t\}_K_{B}$

[Vahdat]
Figure 7.11
Digital signatures with public keys

Signaling:
- $H(M)\rightarrow h\rightarrow E(K_{\text{pri}}, h)\rightarrow \{h\}_{K_{\text{pri}}}$
- 128 bits

Verifying:
- $\{h\}_{K_{\text{pri}}}\rightarrow D(K_{\text{pub}}, \{h\})\rightarrow h'\rightarrow h = h'$?
Low-cost signatures with a shared secret key

Figure 7.12

Signing

Verifying

M
K

H(M+K)
h

M
K

H(M+K)
h'

h = h'?
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: [chart]
Other protocols: [chart]
Figure 7.18
SSL handshake protocol

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

Optionally send server certificate and request client certificate

Send client certificate response if requested

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**

<table>
<thead>
<tr>
<th><strong>Subject</strong></th>
<th>Distinguished Name, Public Key</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issuer</strong></td>
<td>Distinguished Name, Signature</td>
</tr>
<tr>
<td><strong>Period of validity</strong></td>
<td>Not Before Date, Not After Date</td>
</tr>
<tr>
<td><strong>Administrative information</strong></td>
<td>Version, Serial Number</td>
</tr>
<tr>
<td><strong>Extended Information</strong></td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Key size/hash size (bits)</th>
<th>Extrapolated speed (kbytes/sec.)</th>
<th>PRB optimized speed (kbytes/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEA</td>
<td>128</td>
<td>700</td>
<td>-</td>
</tr>
<tr>
<td>DES</td>
<td>56</td>
<td>350</td>
<td>7746</td>
</tr>
<tr>
<td>Triple-DES</td>
<td>112</td>
<td>120</td>
<td>2842</td>
</tr>
<tr>
<td>IDEA</td>
<td>128</td>
<td>700</td>
<td>4469</td>
</tr>
<tr>
<td>RSA</td>
<td>512</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>RSA</td>
<td>2048</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>MD5</td>
<td>128</td>
<td>1740</td>
<td>6242</td>
</tr>
<tr>
<td>SHA</td>
<td>160</td>
<td>750</td>
<td>2516</td>
</tr>
</tbody>
</table>
### Figure 7.19

**SSL handshake configuration options**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key exchange method</td>
<td>the method to be used for exchange of a session key</td>
<td>RSA with public-key certificates</td>
</tr>
<tr>
<td>Cipher for data transfer</td>
<td>the block or stream cipher to be used for data</td>
<td>IDEA</td>
</tr>
<tr>
<td>Message digest function</td>
<td>for creating message authentication codes (MACs)</td>
<td>SHA</td>
</tr>
</tbody>
</table>
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

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?
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

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.

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

Parents refer subdomain queries to their children.

Subdomains correspond to organizational (*administrative*) 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?
From Servers to Servlets

Servlets are dynamically loaded Java classes/objects invoked by a Web server to process requests.

- Servlets are to servers as applets are to browsers.
- Servlet support converts standard Web servers into extensible “Web application servers”.
- Designed as a Java-based replacement for CGI
  
  Web server acts as a “connection manager” for the service body, which is specified as pluggable servlets.

  Interface specified by JavaSoft, supported by major servers

- Servlets can be used in any kind of server (not just HTTP).
  
  Invocation triggers are defined by server; the servlet does not know or care how it is invoked.
Anatomy of a Servlet

network service (servlet container)

ServletContext

init(ServletConfig config)
String getServletInfo()
service(....)
destroy()

String getServerInfo()
Object getAttribute(name)
String getMimeType(name)
getResource*(name)
log(string)

GenericServlet

(implements)

Servlet

String getInitParameter(name)
ServletContext getServletContext()
Enumeration getInitParameterNames()

ServletConfig
Invoking a Servlet

```
service(ServletRequest, ServletResponse)
```

**Servlet**

- ServletInputStream
  - readline(...)
- ServletOutputStream
  - print(...)
  - println(...)
- ServletResponse
  - setContentType(MIME type)
  - getOutputStream()
- ServletRequest
  - getContentType(), getContentLength(), getRemoteAddr(), getRemoteHost(), getInputStream()
  - getParameter(name), getParameterValues(name),
HTTP Servlets

GenericServlet

HttpServletRequest
- service(...)
- doGet()
- doHead()
- doPost...

HttpServletResponse
- addCookie(),
- setStatus(code, msg),
- setHeader(name, value),
- sendRedirect(),
- encodeUrl()

ServletRequest

HttpServletResponse

HttpServletRequest

HttpServletResponse

HttpServletRequest

getCookies(),
getRemoteUser(),
getAuthType(),
getHeader(name),
getHeaderNames(),
HttpSession getSession()
import java.io.*;
import javax.servlet.*;

public class HelloWorld extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse response)
            throws ServletException, IOException
    {
        ...
    }
    public String getServletInfo()
    {
        return "Hello World Servlet";
    }
}
public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
    ServletOutputStream output = response.getOutputStream();
    String fromWho = request.getParameter("from");

    response.setContentType("text/html");
    if (fromWho == null) {
        output.println("<p>Hello world!</p>");
    } else {
        output.println("<p>Hello world from <em>" + fromWho + "</em>");
    }
}

HelloWorld Servlet (continued)
Example 1: Invoking a Servlet by URL

Most servers allow a servlet to be invoked directly by URL.

- client issues HTTP GET
  
  e.g., http://www.yourhost/servlet/HelloWorld

- servlet specified by HTTP POST
  
  e.g., with form data

  <FORM ACTION="http://yourhost/servlet/HelloWorld" METHOD="POST">
  From: <INPUT TYPE="TEXT" NAME="from" SIZE="20">
  <INPUT TYPE="SUBMIT" VALUE="Submit"/>
  </FORM>

  generates a URL-encoded query string, e.g., "<servletURL>?from=me"