Midterm Exam

• March 25 – Don’t miss it!

• Open book, open note
  - Bring copies of all the papers, you will need them
  - Don’t count on reading the papers during the exam

• Covers first six lectures
  - There will be at least one question on protocols

• See me after lecture if you have any questions
  - It’s my office hours (I have to sit around anyway)
  - If it seems like a stupid question, all the more reason to get it answered
  - I can meet other times, just send me mail
SPKI/SDSI naming

- Every **public key** has a local name space
  - No global names

- A **local name is an expression of the form** $K A$
  - $K$ is a public key
  - $A$ is an *identifier*, such as “Alice” or “faculty”

- An **extended name is a key $K$ followed by two or more identifiers**
  - E.g., $K$ NYU faculty, or $K$ David Security-class-TA

- A **term is a public key or local or extended name**
Name certificates

- A name certificate is a signed 4-tuple \((K, A, S, V)\)
  - \(K\) is the issuer that signed the certificate
  - \(A\) is an identifier (e.g., faculty, David)
  - \(S\), a term, is the subject (the “meaning” of local name \(K\ A\))
  - \(V\) is a validity specification

- **Given set** \(C\) **of certificates**, the value \(\mathcal{V}_C(T)\) **of term**

  \(T\) **is a set of certificates defined as follows:**
  - \(\mathcal{V}_C(K) = \{K\}\) for any key \(K\)
  - If \((K, A, S, V) \in C\), then \(\mathcal{V}_C(K\ A) \supseteq \mathcal{V}_C(S)\).
  - \(\mathcal{V}_C(K\ A_1\ A_2\ \ldots\ A_n) = \bigcup_{K' \in \mathcal{V}_C(K\ A_1)} \mathcal{V}_C(K'\ A_2\ \ldots\ A_n)\)

- **Alternate interpretation as re-write rule:** \(K\ A \rightarrow S\)
Example

• Some hypothetical certificates for NYU:
  - $K_{NYU}$ David $\rightarrow K_D$
  - $K_{NYU}$ faculty $\rightarrow K_{NYU}$ David
  - $K_{NYU}$ Margaret $\rightarrow K_M$
  - $K_{NYU}$ faculty $\rightarrow K_{NYU}$ Margaret
  - $K_{NYU}$ comp-sci-chair $\rightarrow K_{NYU}$ Margaret
  - $K_{NYU}$ faculty $\rightarrow K_{NYU}$ comp-sci-chair adjunct-faculty

• $\mathcal{V}_C(K_{NYU}$ faculty$)$ includes:
  - $K_D, K_M$
  - Keys of anyone Margaret has designated adjunct-faculty
Evaluating terms

- With only local names, evaluate by creating graph
  - Every key and local name is a vertex
  - Edges go from local name to subjects in certificates
  - $\mathcal{V}(T)$ is all $K$ such that $T \rightarrow K$

- With extended names, must define composition:
  - If $C = L \rightarrow R$ and $S = LX$, then $S \circ C = RX$
  - If $C_1 = L_1 \rightarrow R_1, C_2 = L_2 \rightarrow R_2$, and $R_1 = L_2X$, then $C_1 \circ C_2 = L_1 \rightarrow R_2X$

- The closure $C^+$ of a set of certificates $C$ is smallest superset of $C$ closed under composition.
  - Can use same graph algorithm to evaluate
  - Problem: Infinite $C^+\colon K A \rightarrow K A A$
Solution: Name-reduction closures

- **A cert** $C = L \rightarrow R$ **is reducing if** $|L| > |R|
  - All reducing certs are of form $KA \rightarrow K'$

- **Compute name-reduction closure** $C^\#$ **as follows:**
  - Initialize $C' \leftarrow C$
  - If $C_1, C_2 \in C'$, $C_1 \circ C_2$ is defined, and $C_2$ is reducing, add $C_1 \circ C_2$ to $C'$
  - Repeat previous step until no new certificates can be added

- **Theorem:** If $L \rightarrow R \in C$, then $\forall K \in V_C(R), L \rightarrow K \in C^\#$

- **Converse:** If $L \rightarrow K \in C^\#$, then $\exists L \rightarrow R \in C$ s.t. $K \in V_C(R)$
Authorization certificates

- **Auth certs are signed 5-tuples** \((K, S, d, T, V)\)
  - \(K\) is the *issuer*
  - \(S\), a term, is the *subject* \(\forall(S)\) are keys getting permission
  - \(d\) is *delegation bit* (if true, \(\forall(S')\) can further delegate)
  - \(T\) is the *authorization tag*—what is being authorized
  - \(V\) is the *validity specification* an in in name certs

- **Example: Access control lists**
  - Owner of protected resource issues one or more auth certs
  - In this common case, \(K\) can just be written Self
  - E.g., Web server key signs \((\text{Self, } K_{\text{NYU students}}, 0, T, V)\),
    where \(T = \text{(tag (* prefix http://www.nyu.edu/))}\)
Authorization tags

- Mostly opaque to certification machinery
  - Must be able to intersect two tags

- Represented as S-expressions, some special forms
  - (*) – is a wild card matching anything
  - (* set id1 id2...) – is a set of tag expressions
  - (* prefix string) – is anything with prefix string
  - (* range ordering lower-lim upper-lim) – is a range

- Can be intersected. Example:
  - (tag (pkpfs (* prefix //www.nyu.edu/) read))
  - (tag (pkpfs (* prefix //www.nyu.edu/G22.3033-001/) (* set read write)))
**Auth certs as rewrite rules**

- *(K, S, d, T, V)* can be written *K[1] → S[d]*
  - [Z] is a ticket symbol representing delegation
  - [1] means further delegation is permitted, [0] not

- **Use the same rules as before for** *C_3 = C_1 \circ C_2*
  - If *C_1 = L_1 → L_2X, C_2 = L_2 → R_2, C_1 \circ C_2 = L_1 → R_2X*
  - Note *L_2* cannot end [0], so neither can *X*
    (this restricts further delegation as desired)
  - *C_1* can be a name or auth cert, *C_3* will be same type
  - *C_2* cannot be auth cert if *C_1* is not also an auth cert
  - If *C_2* is an auth cert, *X* must be empty
The Kerberos authentication system

• **Goal: Authentication in “open environment”**
  - Not all hardware under centralized control  
    (e.g., users have “root” on their workstations)
  - Users require services from many different computers (mail, printing, file service, etc.)

• **Model: Central authority manages all resources**
  - Effectively manages human-readable names
  - User names: dm, waldman, …
  - Machine names: class1, class2, …
  - Must be assigned a name to use the system
Kerberos principals

- **Principal**: Any entity that can make a statement
  - Users and servers sending messages on network
  - “Services” that might run on multiple servers

- Every kerberos principal has a key (password)

- Central key distribution server (KDC) knows all keys
  - Coordinates authentication between other principals
Kerberos protocol

• **Goal:** Mutually authenticated communication
  - Two principals wish to communicate
  - Principals know each other by KDC-assigned name
  - Kerberos establishes shared secret between the two
  - Can use shared secret to encrypt or MAC communication
  (but most services don’t encrypt, none MAC)

• **Approach:** Leverage keys shared with KDC
  - KDC has keys to communicate with any principal
Protocol detail

- To talk to server $s$, client $c$ needs key & ticket:
  - Session key: $K_{s,c}$ (randomly generated key KDC)
  - Ticket: $T = \{s, c, \text{addr}, \text{expire}, K_{s,c}\}_{K_s}$
    ($K_s$ is key $s$ shares with KDC)
  - Only server can decrypt $T$

- Given ticket, client creates authenticator:
  - Authenticator: $T, \{c, \text{addr}, \text{time}\}_{K_{s,c}}$
  - Client must know $K_{s,c}$ to create authenticator
  - $T$ convinces server that $K_{s,c}$ was given to $c$

- “Kerberized” protocols begin with authenticator
  - Replaces passwords, etc.
Getting tickets in Kerberos

• **Upon login, user fetches "ticket-granting ticket"**
  - $c \rightarrow t: c, t$  \hspace{1cm} (t is name of TG service)
  - $t \rightarrow c: \{K_{c,t}, T_{c,t} = \{s, t, \text{addr, expire, } K_{c,t}\}_K_t}_K_c$
  - Client decrypts with password ($K_c = H(\text{pwd})$)

• **To fetch ticket for server $s$**
  - $c \rightarrow t: s, T_{c,t}, \{c, \text{addr, time}\}_K_{c,t}$
  - $t \rightarrow c: \{T_{s,c}, K_{s,c}\}_K_{c,t}$

• **Applications can use Kerberos as follows:**
  - $c \rightarrow s: T_{s,c}, \{c, \text{addr, time, } K_1, K_2, K_3, K_4\}_K_{s,c}$
  - Then $c$ and $s$ use $K_1 \ldots K_4$ as encryption and MAC keys to communicate securely in each direction.
Security issues with kerberos

- **Protocol weaknesses:**
  - Kinit could act as oracle
  - Replay attacks
  - Off-line password guessing
  - Can’t securely change compromised password

- **General design problems:**
  - KDC vulnerability
  - Hard to upgrade system (everyone relies on KDC)
Authentication in AFS

- User logs in, fetches kerberos ticket for AFS server
- Hands ticket and session key to file system
- Requests/replies accompanied by an authenticator
  - Authenticator includes CRC checksum of packets
  - Note: CRC is not a valid MAC!

- What about anonymous access to AFS servers?
  - User w/o account may want universe-readable files
AFS permissions

- Each directory has ACL for all its files
  - Precludes cross-directory links

- ACL lists principals and permissions
  - Both “positive” and “negative” access lists

- Principals: Just kerberos names
  - Extra principles, system:anyuser, system:authuser

- Permissions: rwlidak
  - read, write, lookup, insert, delete, administer, lock
Kerberos inconvenience

- Large (e.g., university-wide) administrative realms
  - University-wide administrators often on the critical path
  - Departments can’t add users or set up new servers
  - Can’t develop new services without central admins
  - Can’t upgrade software/protocols without central admins
  - Central admins have monopoly servers/services
    (Can’t set up your own without a principal)

- Crossing administrative realms a pain

- Ticket expirations
  - Must renew tickets every 12–23 hours
  - Hard to have long-running background jobs
Stretch break
SFS goal: A secure global file system

- One namespace for all files
- Global deployment (anyone can set up a server)
- Security over untrusted networks

/sfs/nyu.edu:.../dm/lec4.pdf
The security problem

- **Secure client–server communications**
  - Solution: Encrypt and MAC all network traffic

- **Authenticate users to servers**
  - Servers know what classes of users to expect in advance
  - Solution: Store users’ passwords or public keys

- **Authenticate servers to clients**
  - Would be solved if client had server’s public key
  - Clients don’t know about servers in advance
  - A user can potentially access any server in the world
  - Solution: ?
Typical systems

- Presume hypothetical cooperation of third parties
  - Echo: Assume DNS admins will manage keys

- Lack security across administrative boundaries
  - AFS provides no security to unknown users

- Penalize the creation of new administrative realms
  - Kerberos and AFS lead to inconveniently large realms

- Provide inappropriate security procedures or guarantees
  - SSL takes “one size fits all” approach to key management
SSL approach in detail

1. PubKey, $$$
2. Certificate
3. Connection request
4. PubKey, Certificate

- Everybody trusts some certification authority
- Trade-off between ease of certification and security
- Precludes other models (passwords, Kerberos, …)
Server key management

- How to get a server’s public key?
  - Use password in conjunction with SRP
  - A CA whose key you know certified the server
  - Another server in the same realm gives you the key
  - Bootstrap using another system’s key management
  - The CD-ROM for your OS contained the key
  - Correct answer: All of the above in different situations
Self-certifying File System

• Idea: Make file system security independent of key management

• Specify server keys in self-certifying pathnames:
  /sfs/sfs.mit.edu: bzcc5hder7cuc86kf6qswyx6yuemnw69/dm/
  - File name itself certifies server’s public key

• Push key management out of the file system
  - Problem reduces to finding correct file name
Different approach to key management

- Provide security without key management
- Let multiple key management schemes coexist
- Make it easy to implement new schemes
  - *Self-certifying pathnames* managed with standard file utilities
  - *SFS Agents* let external programs manage keys
  - *Secure symbolic links* like web links but secure
  - *SFS itself* allows secure sharing key management data
User’s view of SFS

- New directory `/sfs` contains global files
- Subdirectories of `/sfs` are self-certifying
  `/sfs/sfs.nyu.edu:bzcc5hder7cuc86kf6qswyx6yuemnw69/`
- Human-readable aliases give names to public keys
  `/sfs/NYU → /sfs/sfs.nyu.edu:bzcc⋯nw69`
- Ordinary naming under self-certifying pathnames
  `/sfs/sfs.nyu.edu:bzcc⋯nw69/usr/dm/mbox`
Client appears to system as NFS server for /sfs
Interprets requests for self-certifying pathnames
Agents interpret non-self-certifying pathnames
Self-certifying pathnames

- File systems lie under /sfs/Location:HostID
  \[ \text{HostID} = \text{SHA-1} (K_S, \ldots) \]
  - Location is DNS name or IP address
  - \( K_S \) is the server’s public key

- **HostID effectively equivalent to public key**
  - SHA-1 is collision-resistant
  - Client can ask server for key and check against HostID
Self-certifying pathname details

\[ /sfs/sfs.fs.net:eu4cvv6wcnzscer98yn4qjpjnn9iv6pi /sfswww/index.html \]

- **Pathnames transparently created when referenced**
  - Anyone can create a server
  - New servers instantly accessible from any client

- **Client requires server to have HostID's private key**

- **Pathname implies nothing about name of server**
  - e.g., server may not actually be the real sfs.fs.net
  - Need key management to produce the correct file name
Key management through symbolic links

- Symbolic links assign additional names to paths
  - $\text{link} \rightarrow \text{dest}$ makes $\text{link}$ another name for $\text{dest}$
  - Always interpreted locally on a file system client

- Link human-readable to self-certifying pathnames

- Example: manual key distribution
  - Install central server’s path in root directory of all clients:
    - $\text{/nyu} \rightarrow \text{/sfs/sfs.nyu.edu.bzcc5hder7cuc86kf6qswyx6yuemnw69}$
    - $\text{/nyu/README}$ designates the pathname:
      - $\text{/sfs/sfs.nyu.edu.bzcc5hder7cuc86kf6qswyx6yuemnw69/README}$
      - “The file $\text{README}$ on the server my administrator calls $\text{/nyu}$”

In SDSI terminology, “nyu’s README”
Certification authorities

- **Are simply SFS file systems**
  - Can be named by local symbolic links:
    
    `/verisign` → `/sfs/sfs.verisign.com:r6ui9gwucpkz85uvb95cq9hdhpfbz4pe`
  
  - Name other file systems with symbolic links, e.g.
    
    `/verisign/NYU` → `/sfs/sfs.nyu.edu:bzcc5hder7cuc86kf6qswyx6yuemnw69`
  
  - Servers reachable from `/verisign` can name other servers
    
    `/verisign/NYU/cs` might name server for `cs.nyu.edu`
  
  - Read-only protocol keeps private key off-line

- **Pathnames reflect trust relationships:**
  
    `/verisign/NYU/README` – “File README on the server Verisign calls NYU”

- **This is just like SDSI naming!**
  
    - Let file namespace double as “local key namespace”
Example: Getting *HostID* with a password

- **Use password to authenticate server (SRP)**
  - Force server to prove possession of secret password

- **Downloading my server’s *HostID* from Lucent:**

  ```bash
  % sfskey add dm@scs.cs.nyu.edu
  Passphrase for dm@scs.cs.nyu.edu:
  % ls -al /sfs/scs.cs.nyu.edu
  lr--r--r-- 1 root sfs 512 May 28 04:16 /sfs/scs.cs.nyu.edu ->
  scs.cs.nyu.edu:85xq6pznt4mgfvdj4mb23x6b8adk55ue
  
  [sfskey also simultaneously handles user authentication.]
  
  - **Bootstrap security using links on** scs.cs.nyu.edu
Dynamic server authentication

- Each user runs an agent program to control /sfs
- Agents can create symbolic links in /sfs on-demand
  - Agent maps names to self-certifying pathnames with arbitrary external programs
Example: Getting *HostIDs* through SSL

- **User references** /sfs/Host.ssl
- **Agent spawns SSL client to get** *HostID* securely
- **Agent links** /sfs/Host.ssl → *Host:*HostID
  - User’s file access transparently redirected
Implementing key management is trivial!

- SSL example implemented in two lines
  - Distribute pathnames from URL https://Host/sfspath.txt
  - Map /sfs/Location.ssl to path retrieved with SSL:
    ```bash
    % sfskey certprog -s ssl \n    sh -c 'lynx -source https://$0/sfspath.txt'
    ```

- Don’t like SSL? How about Kerberos?
  - Map /sfs/Location.krb to path retrieved with Kerberos:
    ```bash
    % sfskey certprog -s krb \n    sh -c 'rsh -x $0 sfskey hostid -'
    ```
  - Similar command works for SSH
• **Combine multiple certification authorities**
  - Merge your own names with those assigned by third parties

• **Make agent search multiple directories for links:**
  ~/.sfs/known_hosts, /nyu/links, /verisign, /thawte

• **Dirsearch implementation easy given file system**
Revocation

- Many links may exist to a compromised *HostID*

- Separate key revocation from key distribution
  - Announce revocation with self-authenticating certificates
    \[
    \{\text{"Path Revoke"}, \text{Location}, K_S, \ldots \} \text{ }_{K_S^{-1}}
    \]
  - Let agents search for certificates on-the-fly
Distributing revocation certificates

- **Use the file system!**
  - Publish revocation certificates as /verisign/revoked/HostID
  - *dirsearch* fetches certificates, as with certification paths

- **Benefits of separating revocation from certification:**
  - Revocation certificates require no out-of-band verification
  - No authority necessary to submit a revocation certificate
  - Revocation certificates as secure as *best* CA, not weakest
User authentication

- **Separate programs handle authentication**
  - User-authentication protocols opaque to file system proper

- **Current authd has simple public-key protocol**
  - No penalty for accessing many administrative realms
  - Use the file system to distribute user keys
Goal: A secure channel to the server for HostID

1. Client connects to server

2. Server returns its public key, $K_S$
   - Client hashes $K_S$ and verifies it matches HostID
   - Client passes connection to appropriate daemon for dialect

3. Client sends short-lived, anonymous public key, $K_C$
Session key negotiation

4. $\{x_C, y_C\}K_S$

5. $\{x_S, y_S\}K_C$

6. $k_{CS} = \text{SHA-1}(\text{dialect}, K_S, x_S, K_C, x_C, \ldots)$

   $k_{SC} = \text{SHA-1}(\text{dialect}, K_S, y_S, K_C, y_C, \ldots)$

4. Client encrypts two random key halves with $K_S$

5. Server encrypts two random key halves with $K_C$

6. Client and server compute shared session keys

Important properties of protocol:

- Efficient: Minimizes server computation, overlaps with client
- Simple: No options, always secure
User authentication protocol

\[
\text{SessID} = \text{SHA-1}(k_{CS}, k_{SC}, \ldots)
\]
\[
\text{AuthID} = \text{SHA-1}(\text{SessID}, \text{Path}, \ldots)
\]

1. SeqNo, SessID, Path, ...
2. $K_U, \{\text{SeqNo, AuthID}\}_{K_U^{-1}}$
3. SeqNo, AuthID, Credentials

1. Client notifies agent, assigns it SeqNo
2. Agent authorizes secure channel to represent user
3. authd informs file server of user’s credentials
Attacking SFS

• **Inherent dangers of a global file system**
  - Attacker’s own files visible everywhere—facilitates exploits
  - Symbolic links on bad servers can point to unexpected places

• **SFS may further expose bugs in existing software**
  - Running NFS at all can cause security holes
  - Bugs in NFS may let attackers crash machines (or worse)

• **Attacks on SFS itself**
  - Cause resource exhaustion (e.g. use up all file descriptors)
  - Cut network during non-idempotent operations
Lessons learned

- Challenge of global security is key management
- Global public key management not the answer
  - Even in a global system, key management often a local issue
- Don’t base system security on key management
  …base key management on secure systems
- Strip clients of any notion of administrative realm
Conclusions

- **SFS is first web-like system with global security**
  - Provides strong file system security
  - Realistically deployable on a global scale
    (anyone can create a server, any client can access any server)

- **SFS takes a new approach to key management**
  - Provide global security without any key management
  - Let arbitrary key management schemes coexist externally
  - Make it easy to implement new schemes

- **New key management mechanisms**
  - Self-certifying pathnames, Agents, Secure links

- **SFS is its own key management infrastructure**