

Secure Distributed Computing

Distributed Systems Department
April 15, 2005

Mary Thompson
Abdelilah Essiari
Olivier Chevassut



Outline

- **Concepts in distributed security**
- **Small domain security**
 - Kerberos
 - SSH
 - PKI
 - SSL
 - Tools for implementing these solutions
- **Cross domain security**
 - Single sign-on, delegation
 - Virtual organizations
 - Authentication and authorization
- **One Time Passwords**
- **Advanced Topics**



Security concepts in a distributed environment

Characteristics of a distributed environment

- **Users, resources and stakeholders spanning multiple hosts**
 - **multiple workstations and servers**
 - **Access across LAN**
- **... spanning multiple physical sites**
 - **Enterprise computing**
 - **Access across WAN**
- **... spanning multiple administrative domains**
 - **Web, Grids, Virtual Organizations**
 - **Authorization to use resources needs to be coordinated with multiple authorities**



Basic security requirements in a distributed environment

- **Users need to be authenticated at each site**
- **Information needs to be transmitted “securely” between sites**
 - **No unnoticed changes (integrity)**
 - **No eavesdropping (confidentiality)**
 - **Known sender and receiver (authenticated)**
- **Users need to be authorized to use resources at various sites**

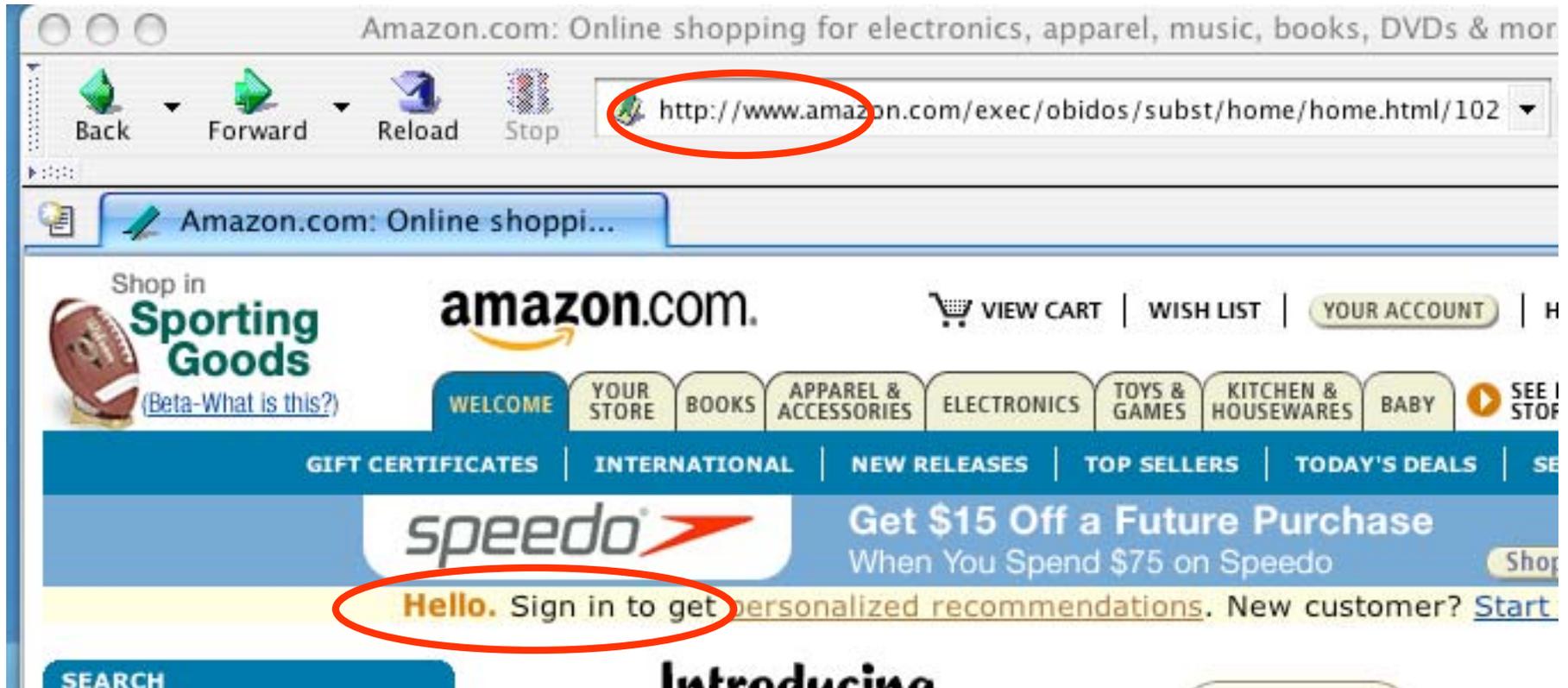


Example of Web interactions

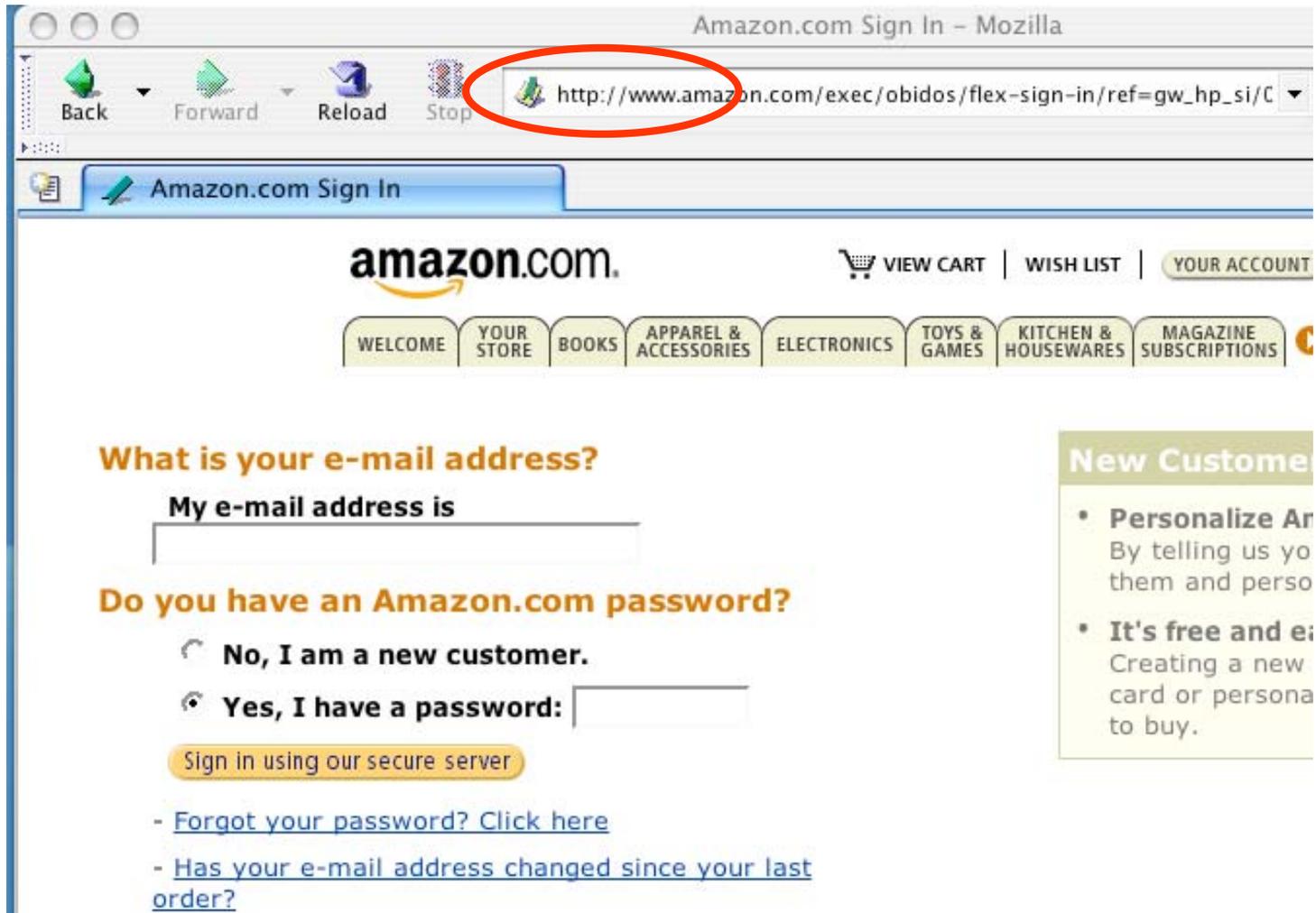
- Start with a several familiar scenarios
- Environment is multi-domain
- Parties need to establish trust
- In some cases communication must be authenticated and secure
- User interface is *simple*



Client-server communication no security



Asking for password over insecure connection



The screenshot shows a Mozilla browser window titled "Amazon.com Sign In - Mozilla". The address bar contains the URL http://www.amazon.com/exec/obidos/flex-sign-in/ref=gw_hp_si/C, which is circled in red. The browser's navigation buttons (Back, Forward, Reload, Stop) are visible. The page content includes the Amazon.com logo, navigation links (VIEW CART, WISH LIST, YOUR ACCOUNT), and a menu of product categories (WELCOME, YOUR STORE, BOOKS, APPAREL & ACCESSORIES, ELECTRONICS, TOYS & GAMES, KITCHEN & HOUSEWARES, MAGAZINE SUBSCRIPTIONS). The main content area asks for the user's email address and password. The "Do you have an Amazon.com password?" section has two radio button options: "No, I am a new customer." and "Yes, I have a password:". A button labeled "Sign in using our secure server" is present. Below this are two links: "Forgot your password? Click here" and "Has your e-mail address changed since your last order?". A "New Customer" sidebar on the right lists benefits like "Personalize Ar" and "It's free and e".

Amazon.com Sign In - Mozilla

Back Forward Reload Stop http://www.amazon.com/exec/obidos/flex-sign-in/ref=gw_hp_si/C

Amazon.com Sign In

amazon.com. [VIEW CART](#) | [WISH LIST](#) | [YOUR ACCOUNT](#)

WELCOME YOUR STORE BOOKS APPAREL & ACCESSORIES ELECTRONICS TOYS & GAMES KITCHEN & HOUSEWARES MAGAZINE SUBSCRIPTIONS

What is your e-mail address?

My e-mail address is

Do you have an Amazon.com password?

No, I am a new customer.

Yes, I have a password:

[Sign in using our secure server](#)

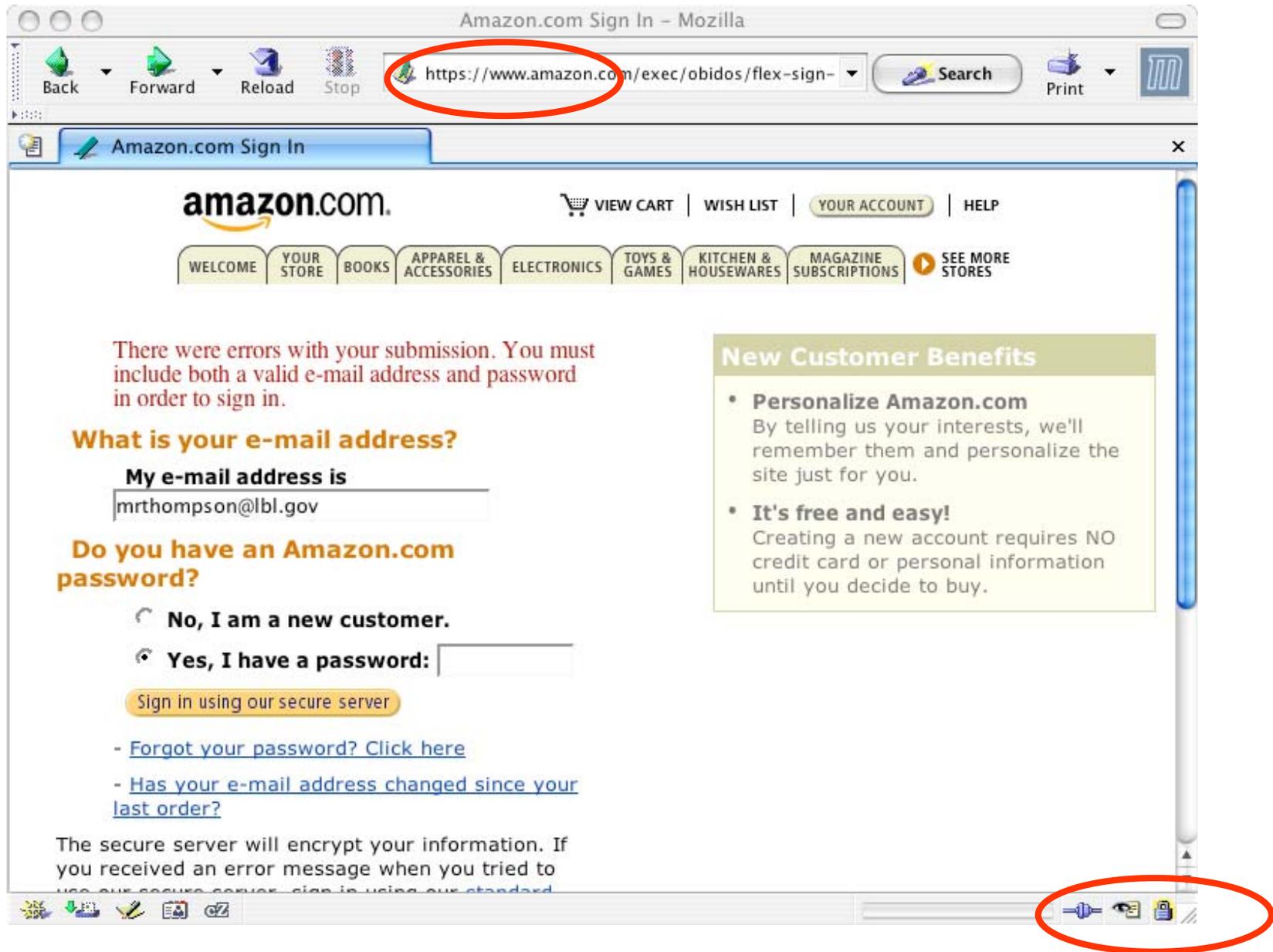
- [Forgot your password? Click here](#)

- [Has your e-mail address changed since your last order?](#)

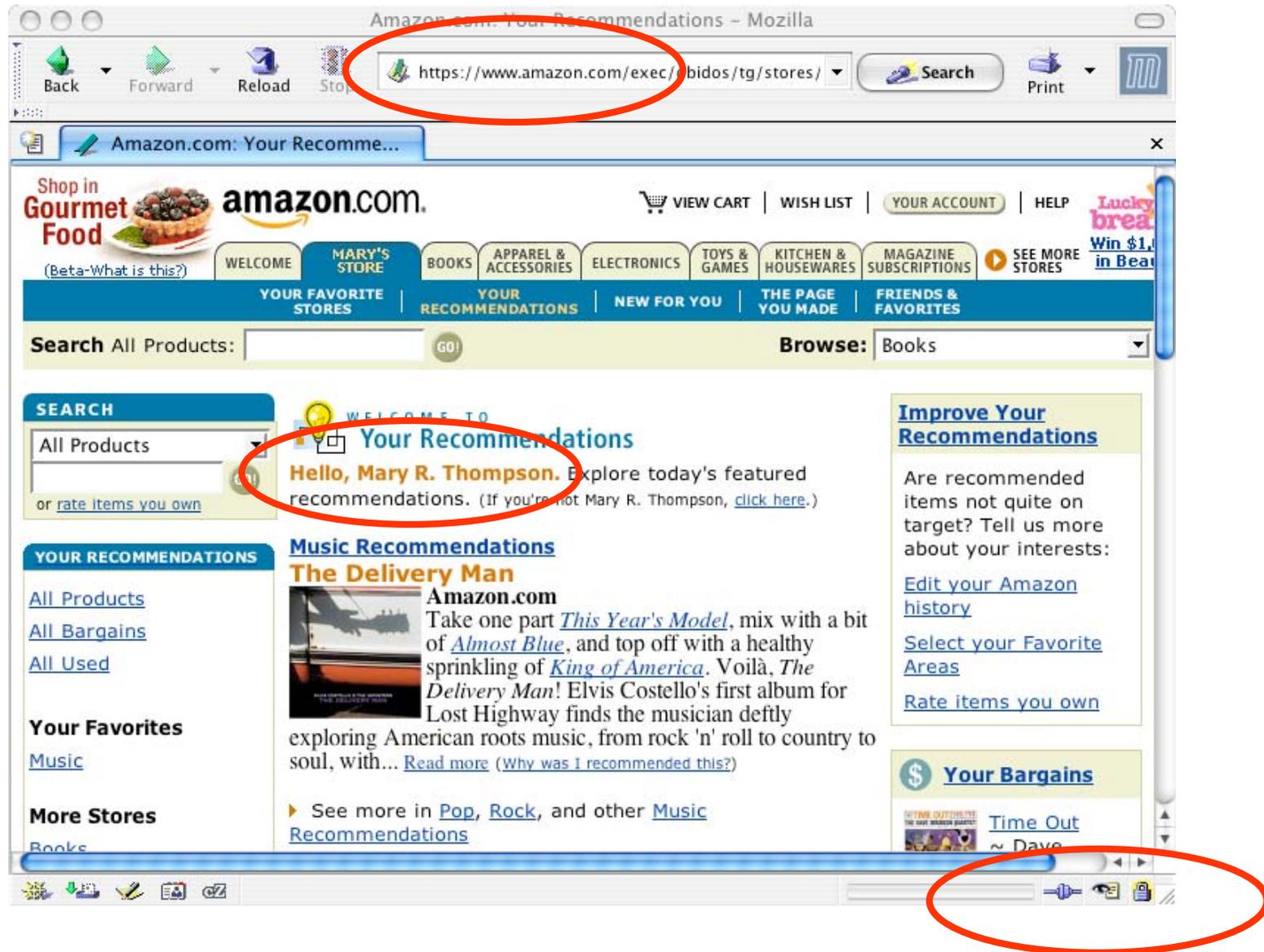
New Customer

- **Personalize Ar**
By telling us yo them and perso
- **It's free and e**
Creating a new card or persona to buy.

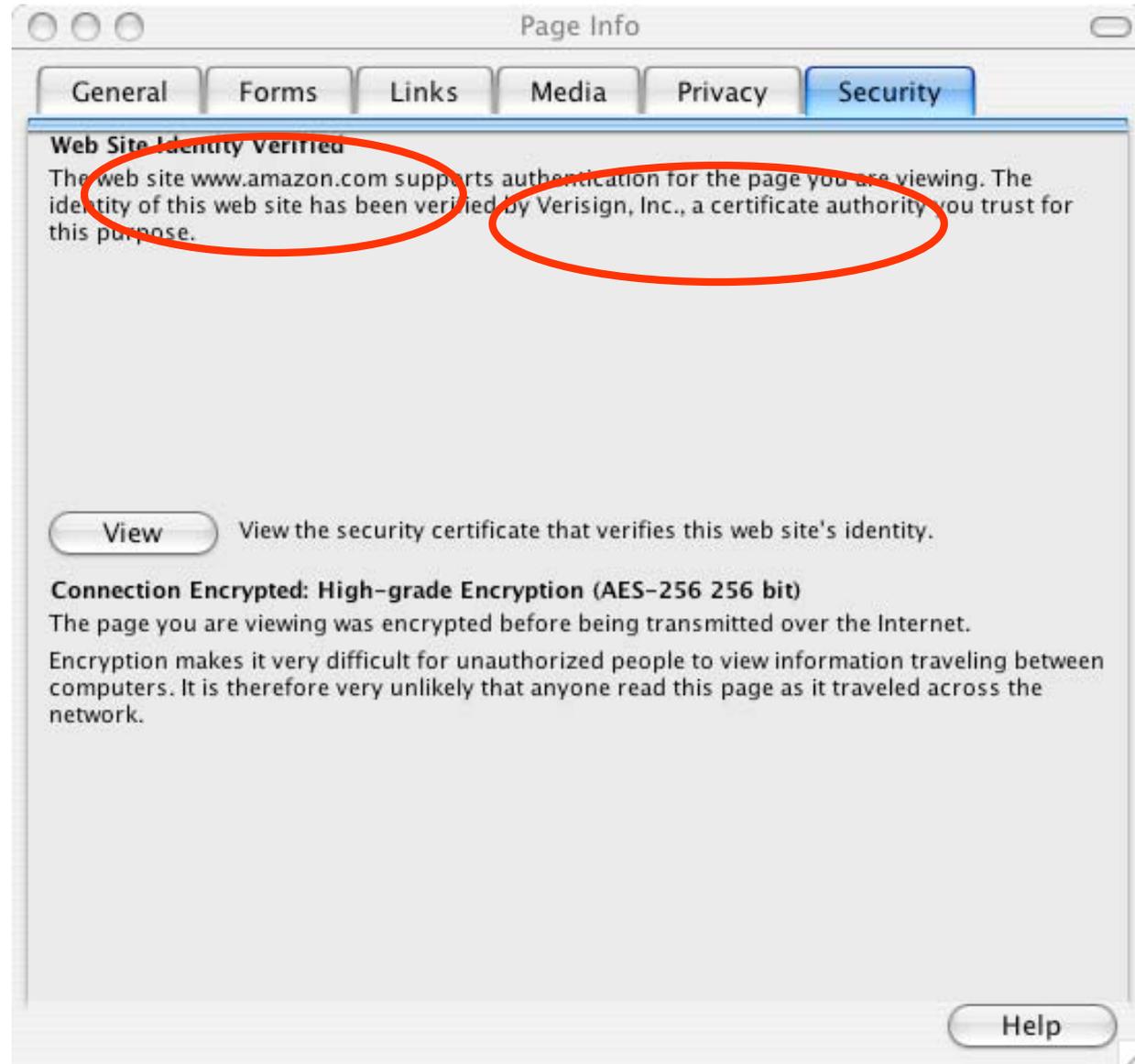
Connection is now secure



Mutual authentication with a secured connection



Clicking on the lock shows...



Clicking on view shows

Web Site

CA

The screenshot shows a window with two tabs: 'General' (selected) and 'Details'. Below the tabs, a message states: 'This certificate has been verified for the following uses:' followed by a list box containing 'SSL Server Certificate'. The main content area is divided into several sections:

- Issued To** (circled in red):
 - Common Name (CN): www.amazon.com
 - Organization (O): Amazon.com Inc.
 - Organizational Unit (OU): <Not Part Of Certificate>
 - Serial Number: 0E:A5:09:3E:35:7E:74:DB:8A:D3:7D:44:83:20:F9:DD
- Issued By** (circled in red):
 - Common Name (CN): <Not Part Of Certificate>
 - Organization (O): RSA Data Security, Inc.
 - Organizational Unit (OU): Secure Server Certification Authority
- Validity**:
 - Issued On: 1/5/05
 - Expires On: 1/6/06
- Fingerprints**:
 - SHA1 Fingerprint: 1E:52:BF:E8:3D:B2:7E:A2:B5:C2:A2:C7:B5:24:3B:5E:42:57:F9:81
 - MD5 Fingerprint: B9:C1:B9:A3:40:3A:4A:93:A8:03:E8:78:1D:E3:9F:20

At the bottom right, there are two buttons: 'Help' and 'Close'.

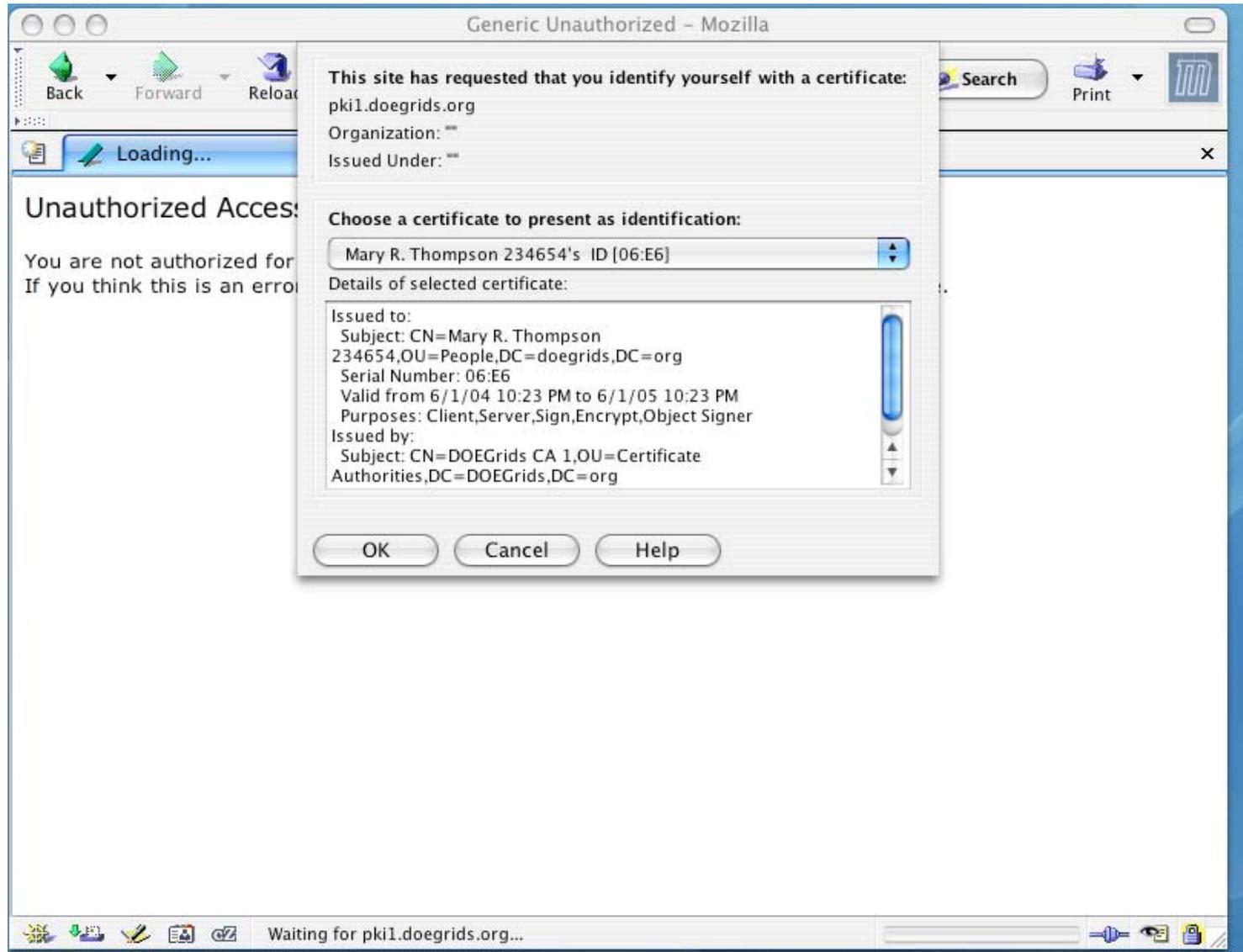
We have seen

- **Unsecured client server connection**
- **Secure connection to an authenticated server**
- **Client authentication via a password**
- **Mutual authentication on a secure connection**

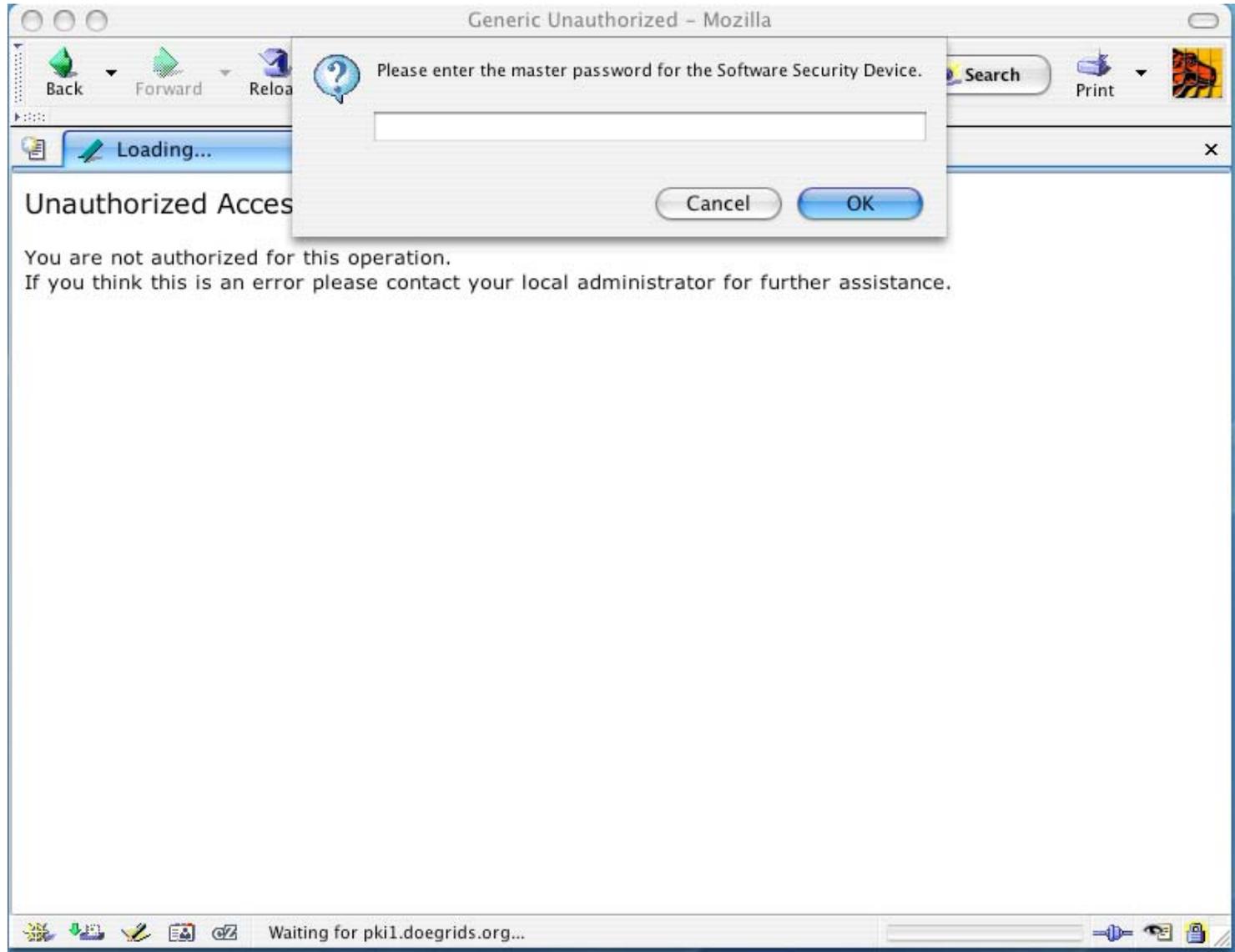


Accessing a site that requires a client certificate

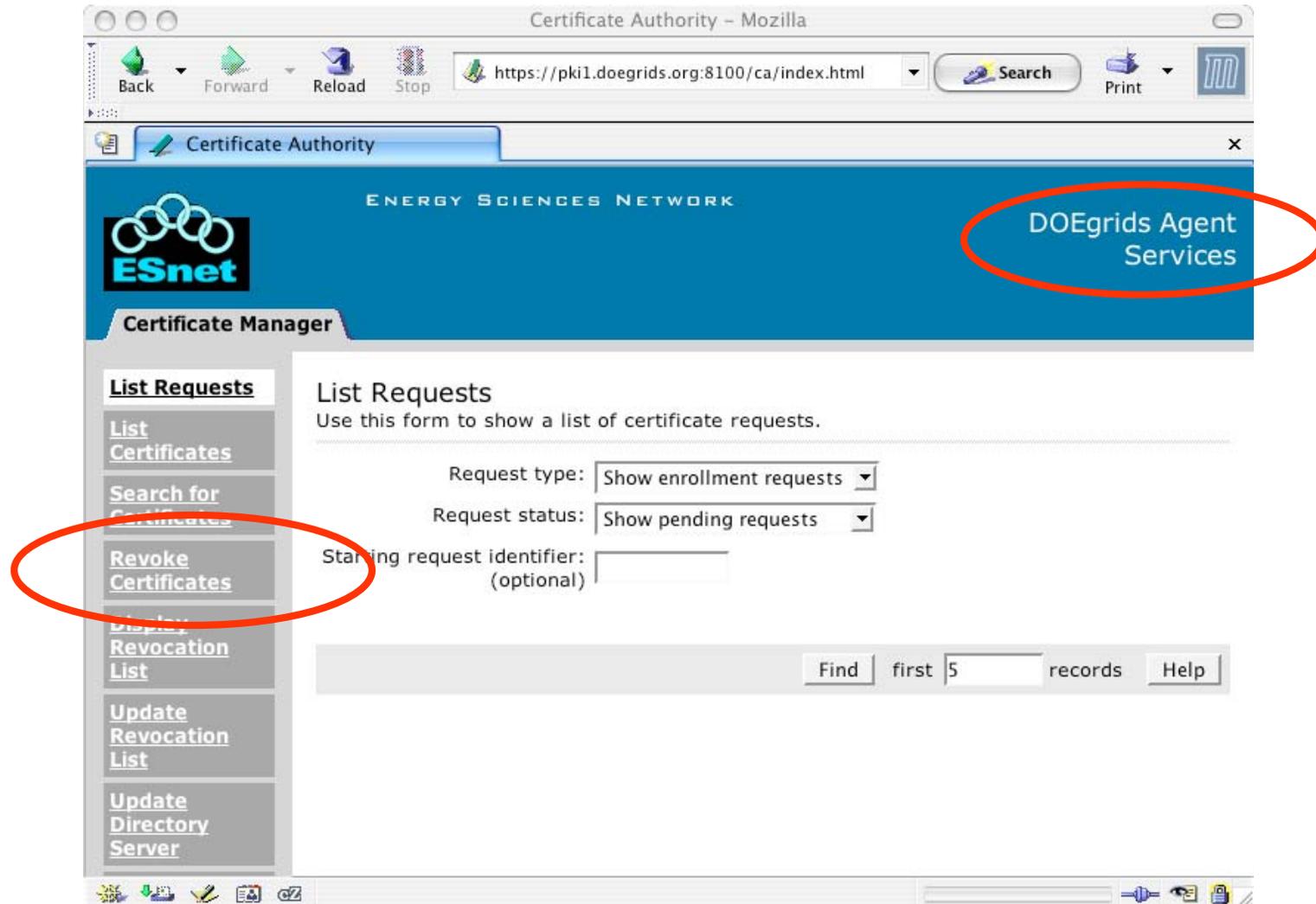
<https://pki1.doe grids.org:8100/ca/>



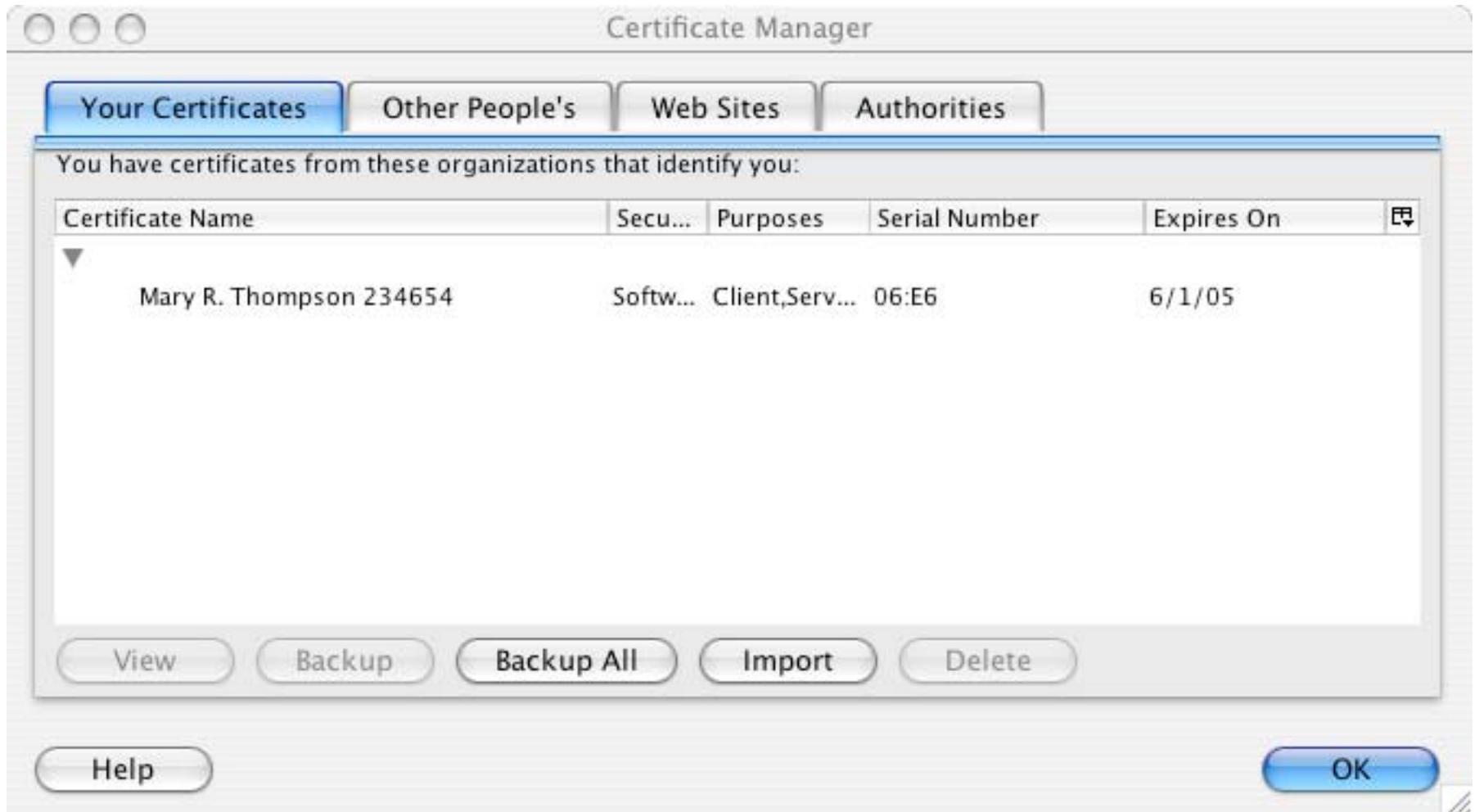
My credential is kept encrypted by Mozilla



Strong client authentication was required



Preferences -> security -> certificates -> manage certificates



Trusted CAs



Trust

- *“assured reliance on the character, ability, strength, or truth of someone or something”*
- Trust is generally orthogonal to the technology behind the scenes
- A distributed environment requires explicit statements of trust
 - Who is trusted to do what
 - Obligations of all the parties



Security properties and the threats they counter

- Authentication: Assurance of identity of person or originator of data (**impersonation**)
- Integrity: Maintaining data consistency (**false information**)
- Confidentiality: Protection from disclosure to unauthorized persons (**eavesdropping**)
- Authorization: Rights to perform some action (**misuse of resources**)
- Non-repudiation: Originator of communications can't deny it later
- Availability: Resources available for authorized parties (**denial of service**)



Authentication

- Assurance of identity of person/originator of data
- Real life
 - Picture id - e.g. passport or badge - issued by trusted source
 - Face must match picture on the id
- Digital
 - Prove knowledge of some individually assigned secret that is also known by the challenger, e.g. password
 - Prove knowledge of a private key that matches a public key known to belong to an identity.
 - Biometrics – useful for person-to-machine authentication, but not machine-to-machine (forwarding a person's identity)



Passwords

- **Shared Secret** between user and server
 - **Distribution is hard**
- **Reusable passwords**
 - **Memorable, hard to guess, short, single purpose**
- **One time passwords (OTP)**
 - **Can't be stolen and reused (prime motivation)**
 - **Client and server need access to a coordinated password source, e.g. keys, hardware tokens initialized with server.**
 - **2 factor protocol**



Using a Password

- **Sent from client to server**
 - **Sent in clear text (NOT)**
 - **Sent encrypted - need to set up an encrypted channel**
- **Used to encrypt a known string**
 - **Time based**
 - **Challenge string sent by server, through encrypted channel to client**
- **Used to encrypt the “answer”, e.g. session key from the server**



Integrity

- Assurance that the data hasn't changed since it was written (e.g., Prevent man-in-the-middle from changing messages)
- Integrity can be checked using
 - Check-sums (modulo arithmetic), XOR all the bits
 - Hash functions – any one-way function that reduces variable sized data to a fixed length “hash code”



Hash Algorithms

- **Used to**
 - **Produce fixed-length fingerprint of arbitrary-length data**
 - **If the hashes of two documents differ, then the documents differ**
 - **Distill passphrases down to fixed-length encryption keys**
- **Also called message digests or fingerprints**



Hash Algorithms

- Reduce variable-length input to fixed-length (usually 128 or 160 bit) output (hash)
 - Too short a hash allows an attacker to find a different input message that produces the same hash
- Requirements of good hash
 - Can't deduce input from the output (one-way)
 - Can't choose an input that will generate a given output
 - For 128 bit hash requires 2^{128} tries
 - Can't easily find two inputs which produce the same output
 - This is twice as easy as matching the input requires 2^{64} tries



Confidentiality

- **No one but authorized parties (who know some secret) can understand the data**
- **Encryption**
 - **an algorithm or function (encrypt) that transforms plain text to cypher text where the meaning is hidden, but which can be restored to the original plain text by another algorithm (decrypt)**
- **Steganography**
 - **Hiding a message in another message or image.**
 - **Used when it is necessary to conceal the fact that a secret message is being transmitted.**



Keys

- **Symmetric Keys**
 - Both parties share the same secret key
 - Problem is securely distributing the key
 - DES (Data Encryption Standard) - developed by IBM, adopted by NIST 1973
 - 3 DES - uses 56 bit key applied 3 times - equivalent of 128 bit key - 1995
 - AES (American Encryption Standard) - NIST replacement for DES - public algorithm - 2001
- **Public/Private keys**
 - One key is cryptographically related to the other
 - Private keys are known only to the owner
 - Public key are stored in public servers, usually in a X.509 certificate.
 - RSA (patent expired Sept 2000), Diffie-Hellman, DSA
- **Encryption/Decryption is about 1000 times faster with symmetric keys than asymmetric**



Digital Signatures

- **Can provide**
 - **message integrity**
 - **message authentication**
 - **Non-repudiation**
- **Digital signatures on documents are an attempt to provide the same level of assurance to digital documents that a notarized wet-ink signature provides to a paper document.**



Digital Signatures - how do they work

- **Combines a hash with a encryption algorithm**
- **To sign**
 - **hash the data**
 - **encrypt the hash with the sender's private key**
 - **send data, signer's name and signature**
- **To verify**
 - **hash the data**
 - **decrypt the signature with the sender's public key**
 - **the result of which should match the hash**



Authorization

- **Rights to perform some action**
- **Access control lists**
 - **For some set of resources, a list of user names and their rights**
 - **Real life examples**
 - **Guest list for an event**
 - **Door locks that read your badge**
 - **Digital examples**
 - **Unix file permissions**
 - **AFS access control lists**



Authorization Tokens

- **Ticket or Capability**

- **A capability (unforgeable ticket), aka authorization token, grants the holder to rights to a resource**

- **Real life examples**

- **Entry visa allows entry to a country, while passport just proves identity (and that you are the rightful owner of the visa)**

- **Movie ticket**

- **Digital examples**

- **Used in academic operating systems .e.g. Hydra, Chorus circa 1975-80**

- **Digitally signed assertions**

- **Include the user's name and rights and are signed by a trusted authority**



Attribute or Authorization Assertions

- **Attributes - characteristic of a user on which authorization may be based**
 - Roles, groups, credit-card number
 - Independent of any resource
 - Assertion needs to be signed by a trusted authority for the attribute.
- **Authorization assertion**
 - Rights of a user to a specified resource
 - Signed by trusted authority for the resource
- **XML Schema - Security Assertion Markup Language (SAML) - OASIS V.2 2005**



Small Domain Security

Kerberos

SSH

Public Key Infrastructure (PKI)

SSL/TLS

What's a small domain

- A collection of server sites and users
- Connected by LAN and/or WAN
- One administrative domain
 - Common usage policies



Kerberos

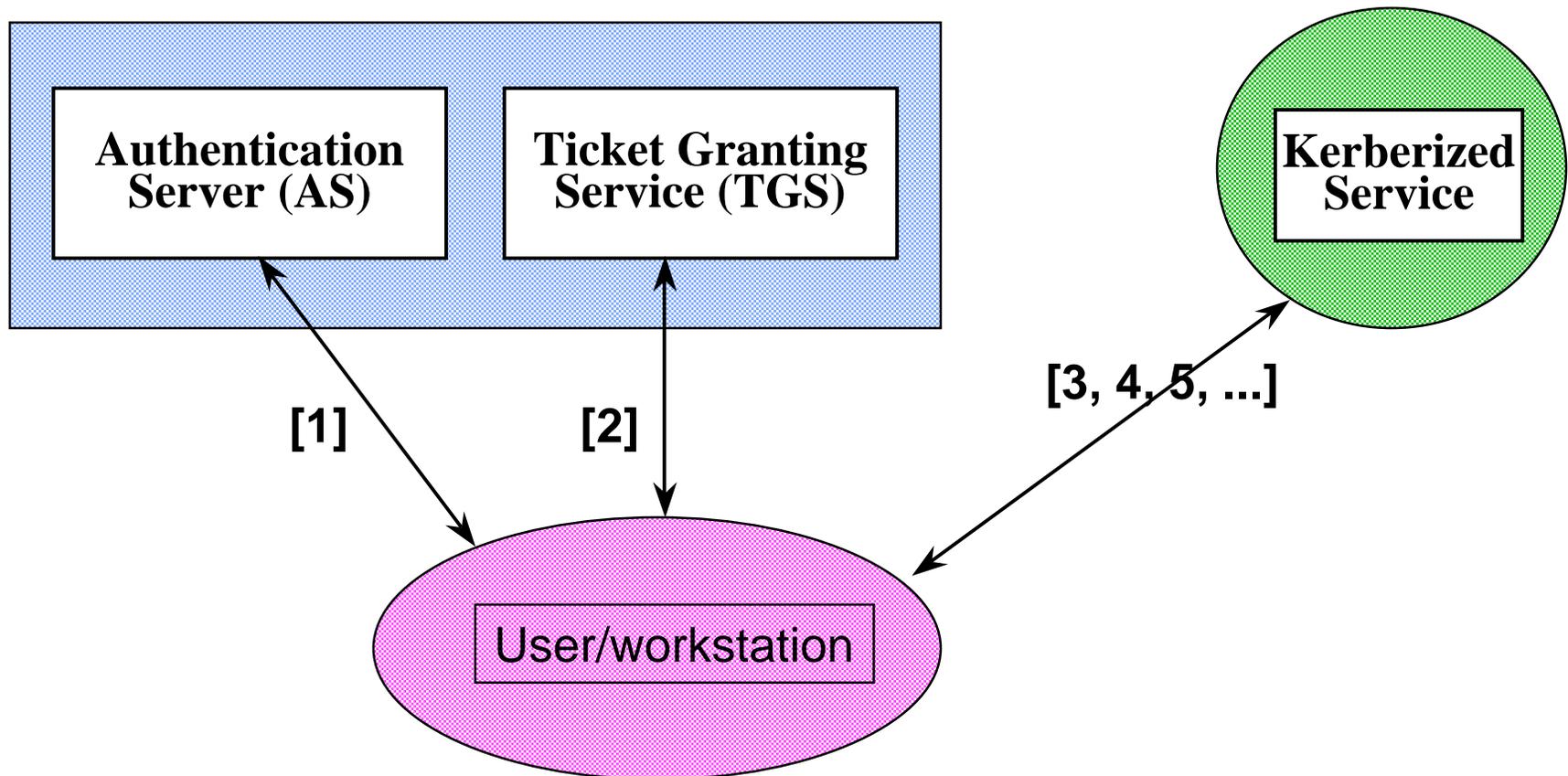
Kerberos Authentication Service

- **First example of system to enable secure authenticated connections in a distributed environment using symmetric cryptology**
- **Developed by MIT's Athena project**
 - **Version 4 deployed at MIT in 1987**
 - **Version 5 1991-96 (RFC 1510)**
- **Goals:**
 - **Support secure authenticated communication between users and servers.**
 - **Provide a transparent solution to the user**
 - **Provide single sign-on**



Components in a Kerberos site

Kerberos System



Kerberos Keys

- **Each service has a secret key**
 - **Known by the TGS**
 - **Stored in the file system of the service (long-lived)**
- **Each human has a password**
 - **Known by the AS**
 - **Computed by the human as a function of a user's password**



John Logs In



1. *I am John*



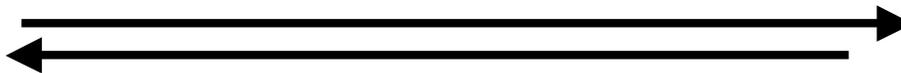
**Authentication
Server (AS)**

2. *AS sends a session key ($sk1$) encrypted using John's password and a ticket ($t1$) encrypted using TGS's secret key.*

Note: John retrieves $sk1$ and holds on to ticket $t1$. It would use ticket $t1$ whenever it needs to use any service for the first time.

John Wants to Use a Printer

3. John sends $t1$ and tells TGS it wants to use printer $p1$.



Ticket Granting Service (TGS)

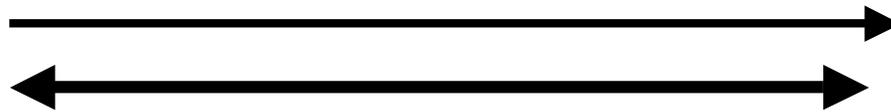
4. TGS retrieves $sk1$ and sends session key ($sk2$) encrypted using $sk1$ and a ticket ($t2$) encrypted using printer $p1$'s secret key.

Note: John retrieves $sk2$ using $sk1$

Finally John Gets to Use the Printer



6. John sends ticket t_2



**Kerberized
Printer p1**

7. John retrieves sk_2 using sk_1 . The printer retrieves sk_2 using its secret key. They both use sk_2 to encrypt subsequent messages.

Note: John is able to use the printer numerous times as long as ticket t_2 has not expired.

Other Kerberos Details

- Typical ticket lifetime: 8 or 12 hours
- Tickets can be forwardable, renewable
- Cross-realm
 - N^2 relationships
 - Hierarchy and forwarding
- Clocks must be synchronized (within 5 minutes)
- In summary this system is nice for one organization or a number of organizations that answer to a single authority.
 - Key management/distribution is a nightmare.



Secure Shell (SSH)

SSH Protocol

- **Protocol to protect data transmitted over a network**
 - **First adopted in 1995**
 - **Designed to replace the Unix remote commands rlogin, rsh and rcp commands.**
 - **ssh2 became an IETF standard in 1997**
- **Authenticating servers**
 - **Public keys**
- **Authenticating users**
 - **Passwords/Public keys**
 - **Integration with other authentication systems**
 - **Unix passwords, Kerberos, ...**



SSH: Encrypted Session

- **SSH negotiates an encrypted session**
 - **Server sends two public keys**
 - A permanent host key
 - A temporary server key
 - **The client accepts the host key if host key has not already been accepted**
 - Generates a random key
 - Sends it back to server double encrypted with both server's public keys
- **Now they both have a session key unknown to anyone else**
- **Server authentication**
 - **Authenticator (Use of the session key)**



SSH: Client Authentication

- **Password**
 - Password is ‘securely’ transmitted over the secure connection
 - Use RSA key pairs whenever possible
- **RSA**
 - User has a public/private key pair
 - Server must have the user’s public key in its cache
 - A distribution headache
 - Client sends user’s public key over the secure connection
 - Server sends a challenge
 - Random number encrypted with the user’s public key
 - Client echoes back the random number
 - I have the right private key



SSH: Private Key Management

- **Users should keep their private key encrypted on disk and only readable by the user.**
- **SSH-agent**
 - **Allows clients to add their private keys**
 - **Useful when connecting to multiple hosts**
 - **SSH agent forwarding**
 - **Start agent on machine A**
 - **Forward it to machine B**
 - **On machine B, login into machine C without ever typing a pass phrase!**



Public Key Infrastructure (PKI) and the SSL protocol

Public Key Infrastructure (PKI)

- **X.509 Public Key Certificates**
 - A digitally signed document
 - Binds a name to a public key
- **Certificate Authorities (CA)**
 - A trusted party that certifies that a public key belongs to a unique entity (person, host or service)
- **Simplifies key distribution**
 - If I 'trust' a CA then I 'trust' all the identities it vouches for.
 - I only need to store the CA's certificate in a local database (or a file).
- **Why the word infrastructure?**
 - Revocation mechanisms.
 - Certificate Renewals
- One can use X509 certificates without using the infrastructure.



X.509 Identity Certificates

- X.509 Name of the user
 - C=US, O=Lawrence Berkeley National Laboratory, OU=DSD, CN=Mary R. Thompson
- X.509 Name of the issuer (CA)
 - C=US, O=Lawrence Berkeley National Laboratory, CN=LBNL-CA
- Validity dates: Not before <date>, Not after <date>
- User's public key
- Extensions
 - Allowed usage
- Signed by CA



Certificate Chain Validation

- In the web model:
 - A list of trusted CAs.
 - Once presented with a chain:
 - CA1-CA2-John Doe
 - Check expiration of each certificate
 - Check validity of a chain
 - Signatures
 - Extensions
 - Check that one of the CAs in the chain is either
 - Trusted (listed in the trusted ca list)
 - Has been vouched for by one of the trusted CAs



SSL/TLS protocol

- **Developed by Netscape to provide secure authenticated connections between browsers and Web servers (published protocol SSL 2.0 by Feb 1995)**
- **Made an open standard (SSL 3.0, TLS) by IETF 1999**
- **Open source software (SSLeay, OpenSSL) available from the start**
- **Standard for all browsers now**
 - **Whenever you see https, you are talking TLS**
 - **Browsers come supplied with many CA certificates**



TLS/SSL

- Provides secure connections between two entities at the transport layer
 - Secret session key
- Supports two modes of operation
 - Server-only authentication
 - Server-client authentication
- Has been tweaked to support wireless apps (WTLS)
- Can be tweaked to support password based authentication



Authentication in TLS/SSL

- **Server authentication**
 - **X509 certificates**

- **Client authentication**
 - **X509 certificates**
 - **At the application level (passwords, credit card numbers, ...)**
 - **Passwords at the SSL level?**
 - **Would be nice**



Server-only Authentication

1. Connects to server



2. Here is my *certificate*



3. Client verifies server's certificate

4. Client Generates a secret key and encrypts it with server's public key



5. Here is the encrypted *secret key*



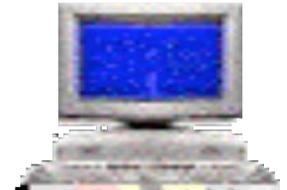
6. I am finished (encrypted)



8. I am finished (encrypted) *key.*



Application messages (encrypted)



7. Server retrieves the secret

key.

Server-Client Authentication

1. Connects to server



2. Here is my **certificate**



3. Client verifies server's certificate

4. Client Generates *pre-master-secret* and encrypts it with server's public key

5. Here is the encrypted **secret key**

Here is my **certificate** and the proof that it is my certificate



6. I am finished (encrypted)



8. I am finished (encrypted)



Application messages (encrypted)



7. Server decrypts secret key, verifies the client's certificate and the proof.

Implementing these Solutions

SSH

OpenSSL

Java - JCE, JSSE



SSH

- **SSH client executes remote commands**
 - `ssh boris.lbl.gov -l aes ls`
 - **Scripts?**
- **SSH-agent**
 - `ssh-agent tcsh`
 - `ssh-add myPrivateKeyFile`
 - `ssh -k boris.lbl.gov`
- **SSH Port Forwarding**
 - **Create secure tunnels**
 - TCP applications
 - **Client<->ssh<->sshd<->server**
- **Scp/sftp to transfer files securely**



OpenSSL (C/C++/Python)

- **Openssl command line tool**
 - **Generate certificates/key pairs**
 - RSA, DH, DSA, ...
 - **Validate certificate chains**
 - **Encrypting/Decrypting private keys**
 - **Convert from one format to another**
 - *PKCS12, PKCS8, Binary (ASN.1), PEM, Pretty text*
 - *Nicely documented*
- **Crypto Library**
 - **Engines (digest, cipher, signature, ...)**
 - **No comments/Very little documentation**
 - **Tools provide nice sample code**
 - **Strategy to find what you need.**
- **SSL library**
 - **Did I mention “No comments”?**
 - **Sample SSL server/client**
 - **Pretty complex but complete**



PEM Encoded Certificate

-----BEGIN CERTIFICATE-----

```
MIICyjCCAjOgAwIBAgIBADANBgkqhkiG9w0BAQQFADBTMQswCQYDVQQGEwJVUzEL
MAkGA1UECBMCQ0ExDDAKBgNVBAoTA0xCTDEMMAoGA1UECxMDRFNEMRswGQYDVQQD
ExJTY2lzaGFyZSBDbXR0b3JpdHkwHhcNMDQwNDEyMDUxMDEwWWhcNMTQwNDEwMDUx
MDEwWjBTMQswCQYDVQQGEwJVUzELMAkGA1UECBMCQ0ExDDAKBgNVBAoTA0xCTDEM
MAoGA1UECxMDRFNEMRswGQYDVQQDEExJTY2lzaGFyZSBDbXR0b3JpdHkwZ8wDQYJ
KoZlhvcNAQEBBQADgY0AMIGJAoGBALW0826zuyIaK2WB0Gi5iVQOajWS7QIIXSdb
5blEOgNtQmyYL8XtfG+m7OcE5ZxB3m1Plcq+NuuBQzkHKOjlrh+bol3IZWkEh2vR
7PsGwWE6/ndkBipQwq7TZNXELzxRUhElzIMVIAVHbsNiVr+RmriFREFDn/2m4Sei
mqw7pR0XAgMBAAGjga0wgaowHQYDVR0OBBYEFgwRb1jntCOX64JzR0JG9FHFpi/B
MHsGA1UdIwR0MHKAFGwRb1jntCOX64JzR0JG9FHFpi/BoVekVTBTMQswCQYDVQQG
EwJVUzELMAkGA1UECBMCQ0ExDDAKBgNVBAoTA0xCTDEMMAoGA1UECxMDRFNEMRsw
GQYDVQQDEExJTY2lzaGFyZSBDbXR0b3JpdHmCAQAwDAYDVR0TBAUwAwEB/zANBgkq
hkiG9w0BAQQFAAOBgQCi9ILpPVHkbZ3ekDecVMdjxZMQb4/5E8CDDroQCkSIMpd8
OYiDfXR+z5nBDWoj7bINmfTiw9KYCF1mQLPdTL86vGmDFo/KbSxKhDXsKO4BhhY5
O3TNki9PJ6e2YffyzltJ6zXffHzqyFUfS+rjDsVEZ3/F0IPgU4/u9pJCI6STkw==
```

-----END CERTIFICATE-----



Pretty Print an X509 Certificate

```
>openssl x509 -in cert.pem -text
```

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1 (0x1)

Signature Algorithm: sha1WithRSAEncryption

Issuer: C=US, OU=LBL, CN=TESTCA I

Validity

Not Before: Jun 8 18:39:08 2004 GMT

Not After : Apr 4 18:39:08 2005 GMT

Subject: C=US, OU=LBL, CN=TESTCA I

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (512 bit)

Modulus (512 bit):

00:ab:13:8d:be:f8:ae:56:48:2b:df:06:ab:69:8c:
05:f6:6d:a5:cf:05:51:00:c1:22:3d:5a:0b:ed:57:
b3:7c:8e:dd:38:ad:53:37:2b:ba:cc:2b:cd:8c:d4:
73:57:b9:21:78:a6:0c:db:2c:e4:9f:d7:a7:6b:e5:
f0:8e:d3:28:a7

Exponent: 3 (0x3)

Signature Algorithm: sha1WithRSAEncryption

0e:8e:bd:85:28:c9:6b:45:e4:7f:89:5e:f9:82:16:16:74:d1:
d4:2d:ca:43:0c:e0:f9:51:f3:25:76:0c:17:85:af:01:22:2c:
db:86:15:19:ea:99:f6:7a:4a:e9:1b:b0:59:92:04:12:a1:ff:
63:47:26:e6:2b:0f:b0:a5:27:ae



PKCS12 (P12)

- A user's p12 file contains:
 - The user's certificate chain
 - The user's private key
 - Encrypted using a passphrase
- `>openssl pkcs12 -in john.p12 -out john.pem`
- `>openssl pkcs12 -export -in john.pem -out john.p12`
- `>openssl pkcs12 -in john.p12 -out john.pem -nodes`



Encrypted RSA Private Key

-----BEGIN RSA PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: DES-EDE3-CBC,FDE0A813ED203FD0

v8CLtR2sfQbe2s3cipbCkAh0IAWq2MEymM4VDDjQDAnhU+XuwLFkzn6+5AxJByLM
hbhJ7Cwx+zBdyrs3ZRojTCleoalnw7Df67Xn7rjJnzTmBuyJeObMyz8OFH3F/Am4
l8lCmyKytEUY66QSdgq3Y/dzIRYDnge4bjCWTnS54yDPB9cITBCNGHSHgPs0l/9
tybXBikTMEEViXQm2lq7pq/Olw/XBLjiWVuMNxyfykdRrKWUcgLnJzQOqu2cjYJL
VEo23yw//NZErsPNERkJYml3ahFbLhqHSP78VLiD+8fmg4gRW1kQnhn0QLgba+QP
36kFIQW9QD4rbyp3hsl9FZvX2Km9knTjFwUFImVqyIZCjSlrzpubLCibFVqrWqlt
l5pOc9iXjV9+EjATYQVfdo/uuGMbHIAgyXN5QTXmBLAV36rMK2cOsLr0QyI7FSuC
l0hVh5t6UKHj6xlnFmKwWUVoPJ6OO12FL1MQBOE9SOaOQrEZq5liQ41+UsHoxSp6
CqJ/FVhn7CgO8qqS70+NwVNdUIPQROpBs6kXrcVcPqDDqXvZuzjxbLcL+B+OBSvv
YUFbP5wjCWllvkdFgwtRSsVSeWy9mAQ0vBJtrsLnDR0ezmidomwFIBzR3sbsTWLz
CmVI9KCKyUF504eYbB+ItsH1Phr33936XQBpXU224lq+fD7+k+DX8kukjEA7DfYi
W5B1Cd4HzbGLoQ+yIzm/wp+/omsQwll9rtGd/UoaEUAGZP8Vty4J3HvgX5gL/axg
A5qsfCLVf298e/qZeZlnJVBET/JFEFAEIUMxbVyVzAfsOtSRq2auPA==

-----END RSA PRIVATE KEY-----



PEM Private Key In The Clear

-----BEGIN RSA PRIVATE KEY-----

```
MIICXAIBAAKBgQCylBv7mOI8J3nvZni9ClbAc8B8OfViQSpV6vdIQLoLXVd/Fa4
bs2s2bjrPlyr+HAC3OYzBAVlwRpvmtes/w+Vhh+Mv/fZ4mFq+kfgZA5vL0I96cb1
YAz8TFhI/oOMb9qKV3Pr369JlgOrcP6W1Q5t8ejODHhLI83oLc4Lr90xZwIDAQAB
AoGARBJsi/bZnzYUPcsUFb8LoKHKQoWVLOGSjFhYUr0f7TAg3O1g/ElozpbHrSeO
3syljpJyaB8rHaaYwPYabT9s5iGaCNzbe1Gbz/2RknOvc0XYhiU1hUgaNjutwUAh
noMhuXZC+Fm1n2ww8BJzi+DCySoe91m0IOIXfdpmer1+yIECQQDI/4JAA9hisWqB
YW0z+Yk+IQgN0RbgQbUK7dibl9i1Tj5UFoYuHDPI5aa6xXvV/xVy9FsnL3n5uEJh
dHAXDP2hAkeAxIJxXoQBXWyLgsLT5fnLKMUivgGU01p/xwd07dMWttgoBe/d2DPs
SeEOt7MEvqjWBcX0EloK9Vz2FPC2sScCBwJAWdsNA/18mHI/7fQmOdwPpoeK8fry
OQLyvCZYLq/kfXTm/6AxZr3EFHIZceprd1Of+6/ffuqZ8I5BaH7+MV2gQJATOGc
7WUDGnJ/yts4RgkGBvIVO89IIGIO8tsjKycPwSYerd6FA9FtlI7K08/kYP0Wqvpl
WTkKcn/tLcT7LcXPdQJBAJfJIJb5q1+3jdWUNA+ykR/iPI8g4CH/Sy4dwQ0preRP
dE4EOpTDpOORSC24RAd4FKQgoipO5inE6YbBhMuPi68=
```

-----END RSA PRIVATE KEY-----



Java

- **SunJCE provider provides a lot of stuff but not all**
 - **Signatures/Cipher/Digest engines**
 - **Can load a p12 file/Cannot generate one**
 - **Can read an X509 certificate/cannot generate one**
 - **Nothing on X509 attribute certificates**
 - **Sun hides its implementation**
 - **API nicely documented**
 - **A good exercise in software design**
- **Bouncy castle provider is open source and provides a lot of stuff**
 - **Pkcs12 generation**
 - **Generating/reading Attribute certificates**
- **IAIK provider (cost some money)**



SSL In One day

- **SSL is complex**
- **But as a black box**
 - **Credentials (server or client)**
 - Private key
 - Certificate chain
 - Separate or together in a P12 file
 - **List of trusted CAS.**
- **Transparency By design**
 - **Hand in these two inputs and you should be in the clear**



Java: TLS/SSL

```
// Keep an eye on the ball (Loading SSL inputs)
KeyStore ks = KeyStore.getInstance("pkcs12");

ks.load(new FileInputStream(p12FileName), passphrase);

CertificateFactory cfac = CertificateFactory.getInstance("X509");
BufferedInputStream bis = new BufferedInputStream(
    new FileInputStream(casFileName) );
do {
    // Reading CA certificates.
    X509Certificate cert = (X509Certificate) cfac.generateCertificate(bis);
    String alias = cert.getSubjectDN().toString();

    ks.setCertificateEntry(alias, cert);
} while (bis.available() > 0);
```



Java: TLS/SSL

```
KeyManagerFactory kfac = KeyManagerFactory.getInstance("SunX509");  
  
kfac.init(ks, passphrase);  
  
TrustManagerFactory tfac = TrustManagerFactory.getInstance("SunX509");  
  
tfac.init(ks);  
  
SSLContext ctx = SSLContext.getInstance("TLS");  
  
ctx.init(  
    kfac.getKeyManagers(), // Used to authenticate to server.  
    tfac.getTrustManagers(), // Used to verify the server.  
    null ); // Here you should pass in a random number  
  
// We almost done
```



Java: TLS/SSL

// Once the context has been initialized

// SSL server

```
ServerSocketFactory ssf= ctx.getServerSocketFactory();  
ServerSocket serverSocket = ssf.createServerSocket(port);
```

```
while (true) {  
    Socket clientSocket = serverSocket.accept();  
    OutputStream os = clientSocket.getOutputStream();  
    InputStream is = clientSocket.getInputStream();  
    ...  
}
```

// SSL client

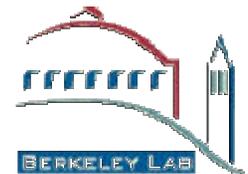
```
SocketFactory sf = ctx.getServerSocketFactory();  
ServerSocket clientSocket = sf.createSocket(host, port);  
OutputStream os = clientSocket.getOutputStream();  
InputStream is = clientSocket.getInputStream();  
...
```

// HTTPS client

```
HttpsURLConnection.setDefaultSSLSocketFactory(sf);  
InputStream is = new URL("https://portnoy.lbl.gov:8000/mypictures").openStream();  
...
```



Cross Domain Security



Complex distributed environments

- **These environments need the mechanisms that we have explained so far and much more**
- **Users and resources span multiple administrative domains**
- **Interactions involve multiple entities**
 - **in diverse administrative domains**
 - **playing the role of both clients and servers**
 - **acting on behalf of the original user**
 - **may be long lived**
- **For example**
 - **grids, collaboratories, sharing university library resources**



Security Requirements/Solutions

- **Authentication and secure communication**
 - Users will come from different administrative domains
 - Need global names and credentials
- **Virtual Organizations become relevant**
 - a logical grouping of the users and resources in a collaboration
 - may be the central trust authority
- **Single sign-on**
 - Users should be able to login once and get access to all sites
 - X509 Proxy Certificate
- **Delegation**
 - Remote processes running on behalf of a user need to be able to authenticate to other processes or services
 - X509 Proxy Certificate
- **Scalable and fine-grained authorization**
 - Virtual organizations (VO)
 - Plus local control at resource site

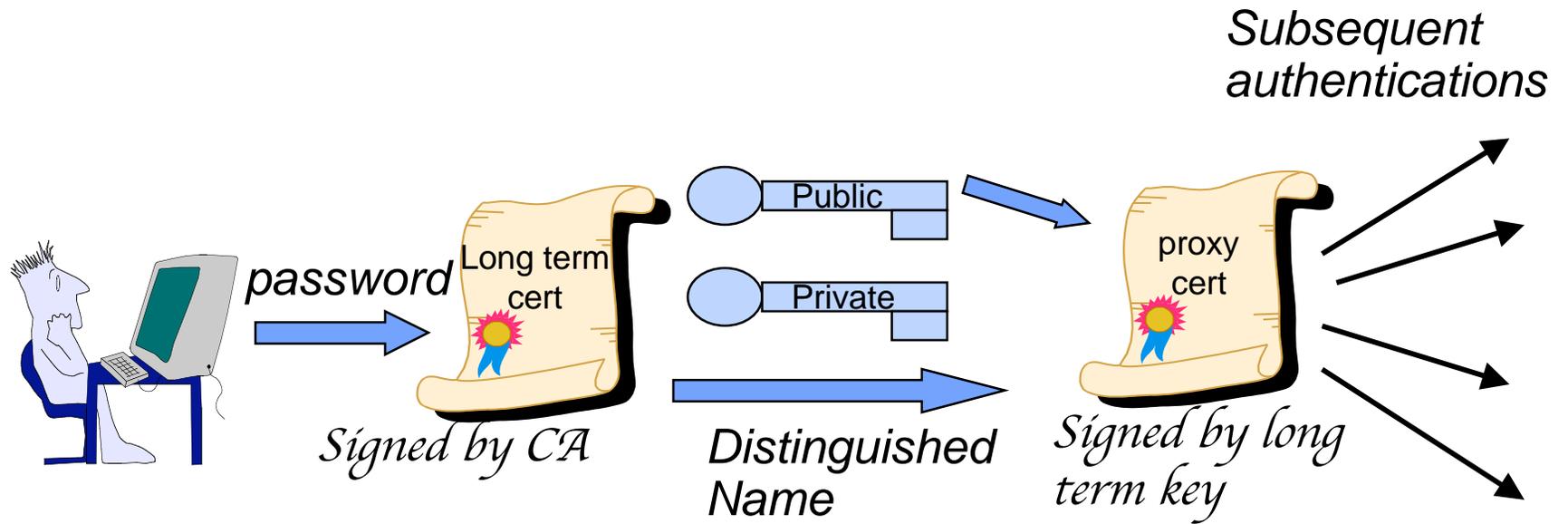


Single Sign-on

- Enables the user to use his passphrase once, after which all the authentications to various sites take place automatically
- Long term public key should be encrypted so every time it is used a passphrase must be used
- Create a short term “proxy certificate” with an unencrypted key that can be used instead.



Single sign-on by creating a proxy certificate

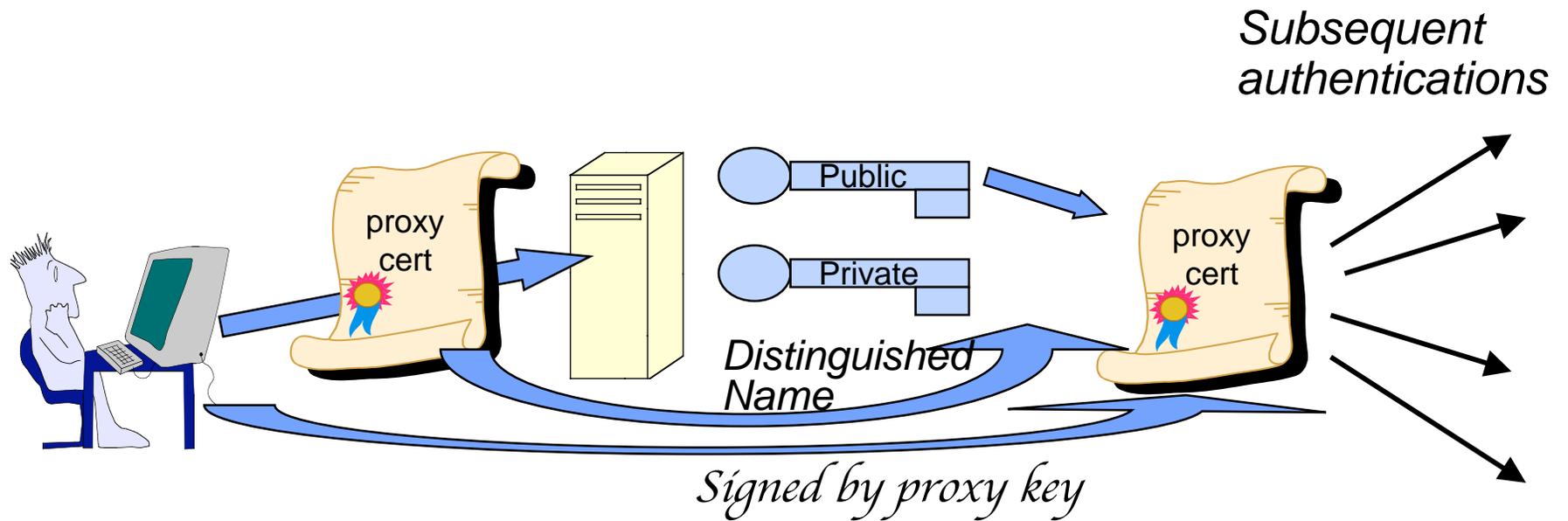


Delegation

- Delegation is needed to allow a service that is running on my behalf, access resources that are available to me.
- Client establishes an authenticated connection to a server
 - Server-Client authentication mode (TLS)
- Server generates a public/private key pair
 - Private key stays on the server side
- Server sends a certificate request
 - Process is no different than requesting a certificate from a Certificate Authority
- Client generates and signs a proxy certificate
- Client sends back the certificate



User delegates a proxy to a server



Authorization Issues

- **Who has authority to grant access**
 - **Local resource providing site**
 - **“Owner” of data or code**
 - **Virtual Organization administrators**
- **Must be scalable**
- **How/where is access decision made and enforced**
 - **PDP - policy decision point (can be separate service)**
 - **PEP - policy enforcement point - server application**
 - **Need standardized callouts from the middleware**
- **How does user present credentials - proof of identity and attributes**



Authorization Authority

- **Local to site**
 - Can be local protected data set by site admins
 - E.g. Globus grid-map file + traditional ACLs
 - Local ACLs based on a global identity
- **“owners” of data or code**
 - Require a secure interface for owners remote from the resource site to check and set access



Separation of access decision and enforcement

- Centralizes the access policy decisions in one place. (PDP)
- Simple interface for resource provider (PEP), just a callout
 - GGF OGSA Authorization WG has developed a profile for a SAML definition of an authorization callout.
- Works best when there are few access checks, e.g.
 - On file open, but not file access.
 - When job is started check that client is allowed and has access to all the resources the job will use.

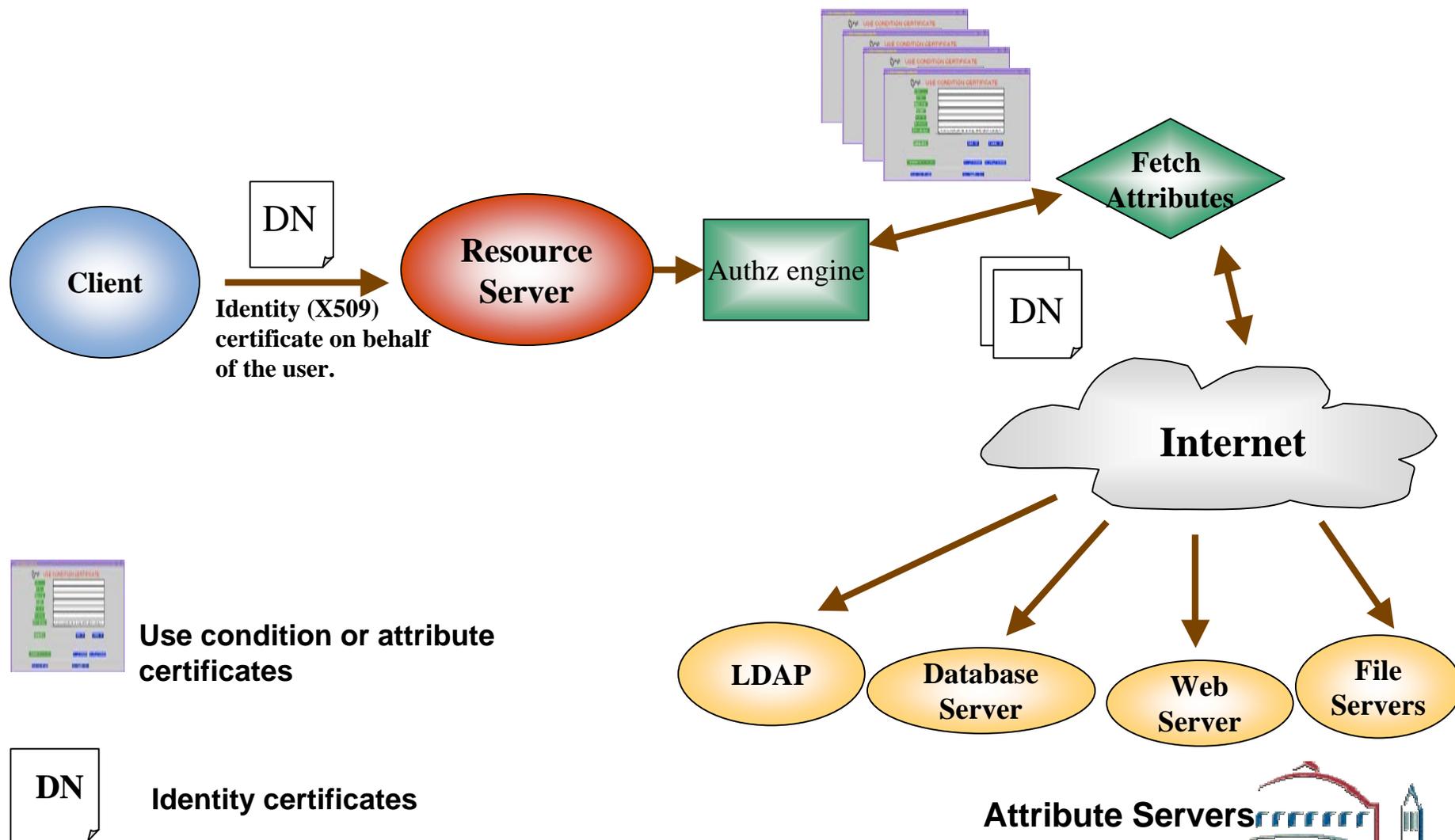


Presenting Credentials

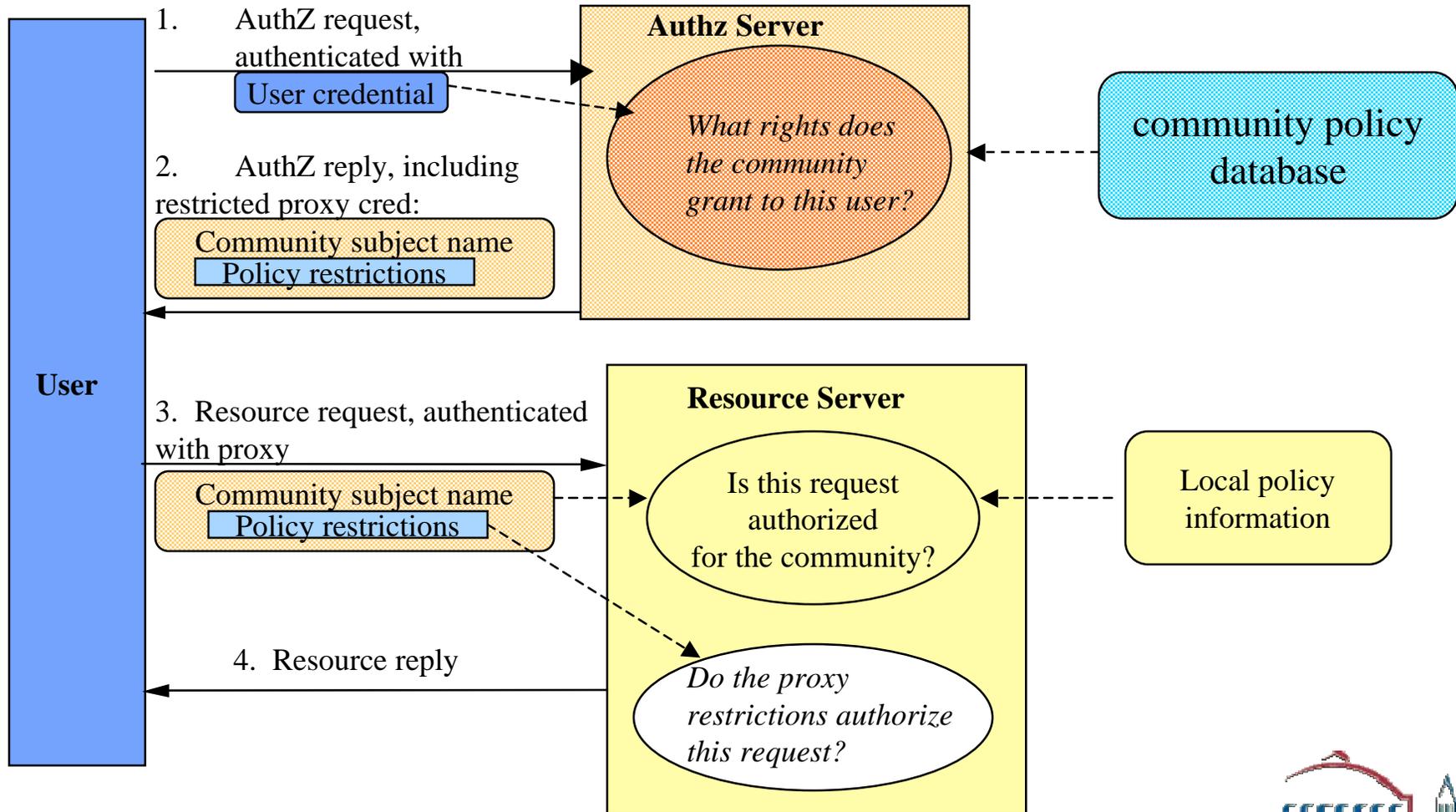
- **Attributes - characteristic of a user on which authorization may be based**
 - **Roles, groups, credit-card number**
- **Client can just authenticate and the PDP can discover any of his attributes**
 - **Simpler for client, harder for PDP**
- **Client can send his attribute assertions to the server.**
 - **Harder for client, simpler for server**
- **Attribute assertions have to come from an accepted authority.**



Pull Model



Push model



Community Authorization Service

- **User contacts CAS**
 - **Gets a proxy certificate containing its rights as a certificate extension**
- **Uses that proxy to contact server**
- **Server verifies proxy credential and extracts rights**
- **May do some local rights checking as well**



Virtual Organization Management System

- The user contacts a VOMS server
 - User can specify the roles he/she wants to play
 - User can contact more than one VOMS server
- The user receives attribute certificates signed by the VOMS server
 - X.509 attribute certificates
- The user generates an X.509 proxy certificate
 - Containing attribute certificates as a non-critical extension
 - Subject is the user's distinguished name
 - Signed by the subject
- The user contacts the resource provider(RP) using the proxy certificate



Authentication Servers for the Wide-area

.NET Passport

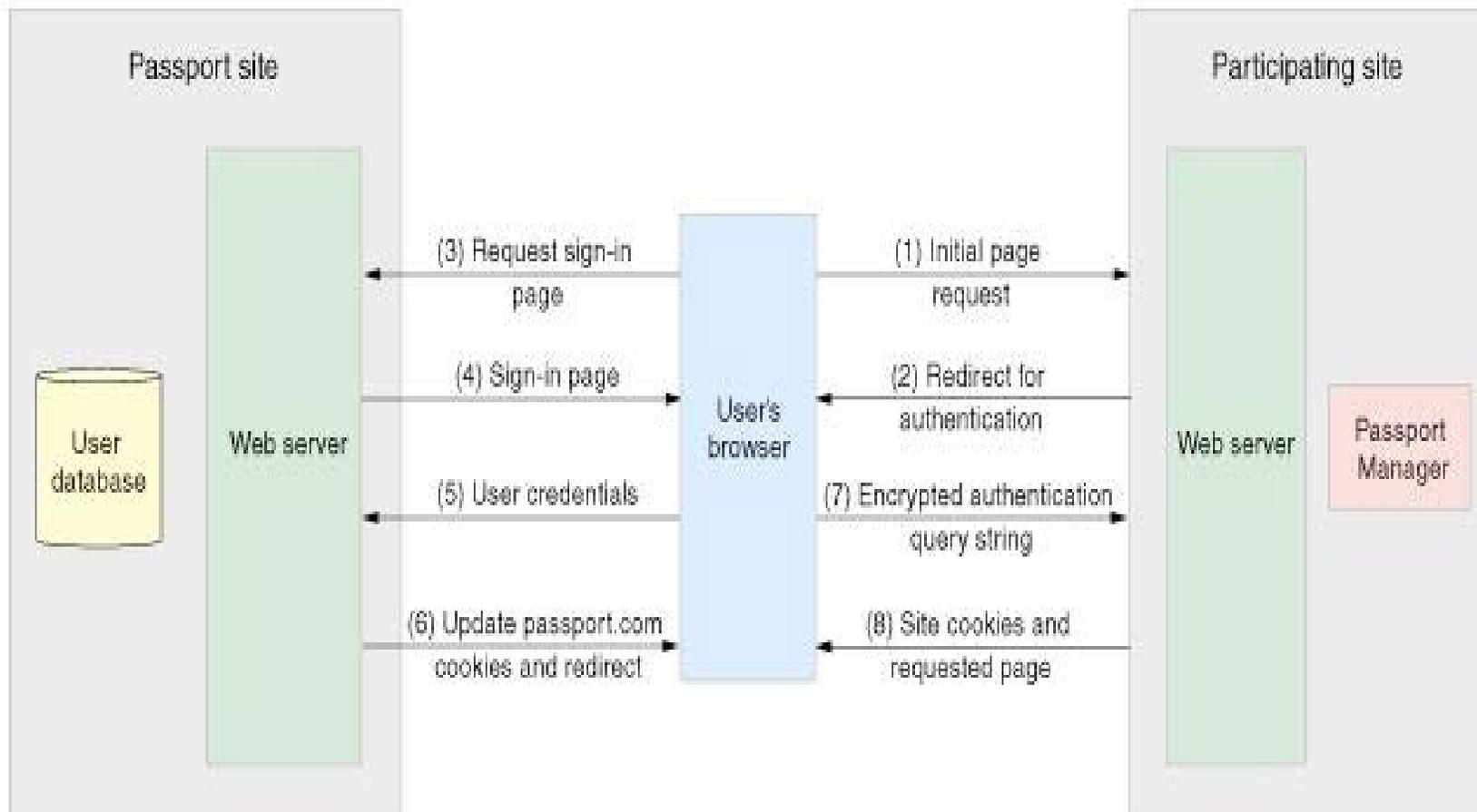
- **Single-sign-on service run by Microsoft for its MSN and .Net sites**
 - **Users include all msn.com sites and hotmail accounts**
- **Based on email address and password**
 - **Stores a number of attributes about each user**



How Passport works

- **User creates an account from any signon site**
 - **Passport Unique Identifier (PUID)**
 - **Email or phone number**
 - **Optionally name, zip code, gender ...**
 - **Credential (password or pin, 1-3 secret questions)**
- **Single signon to all .NET Passport sites**
 - **.net site redirects authorization to Passport server**
 - **Passport server asks for and checks credentials**
 - **Creates a cookie encrypted with requesting site's public key – includes PUID and other information**
 - **Cookie get forwarded to Passport site who keeps your profile under your PUID.**





Picture from Extremeexperts website

<http://www.extremeexperts.com/Net/Articles/ImplementingPassportAuth.aspx>

Liberty Alliance

- Sun, HP, IBM, RSA, MasterCard, AMEx, Entrust, Verisign, AOL - Formed Sept 2001
- Goal - to develop and deploy open, federated solution for network identity
 - Allow a individuals and business to keep personal information securely
 - provide a universal, open standard for "single sign-on"
 - To provide an open standard for network identity
- Many Specifications
 - <http://www.projectliberty.org/resources/specifications.php>
- Many Liberty enabled products
- No sign of any open source libraries



Shibboleth and Anonymous Credentials

- Part of the Internet2 middleware project
- User authenticates at home site
- References another co-operating site which can ask the originating site for a handle for the user
- Asks server at originating site for attributes about the handle which it can use for authorization
 - E.g. user is a faculty member.
- Currently implemented with modified browsers and web servers.



Firewalls

- network layer: based on source, destination, and port contained in IP packet
- application layer: hosts running proxy servers (can function as NAT) or “stateful inspectors”
- NAT (network address translation)
 - Hides a number of hosts behind one network address
 - typically only responses to requests originating behind firewall are allowed back in



Concerns when Designing a Distributed Application

- **Firewalls break end-to-end authentication**
 - **Make host certs less useful for host authentication**
- **Grid Services behind firewall (probably) cannot choose arbitrary IP address/ports**
 - **Some firewalls only route HTTP traffic (not TCP/IP directly)**
 - **Need to limit use of ports, especially ports that clients can connect back to**
- **Grid clients behind NATs can't accept incoming port connections**



One Time Passwords

Olivier Chevassut



Secure One-time Password Authentication



Motivation

- **Most common cause of recent break-ins to well run sites has been from stolen passwords:**
 - **passwords are used in multiple places**
 - **passwords are often static and not change very often**
 - **password sniffers are installed on compromised sites**
- **Many long-lived credentials are not well-managed:**
 - **long-lived credentials will no longer be stored on users' machines**
 - **long-lived credentials will be stored on data centers' servers**
 - **users will obtain short-lived credentials after successful authentication**
- **Many DOE sites are seriously considering mandating one-time password authentication for their users.**



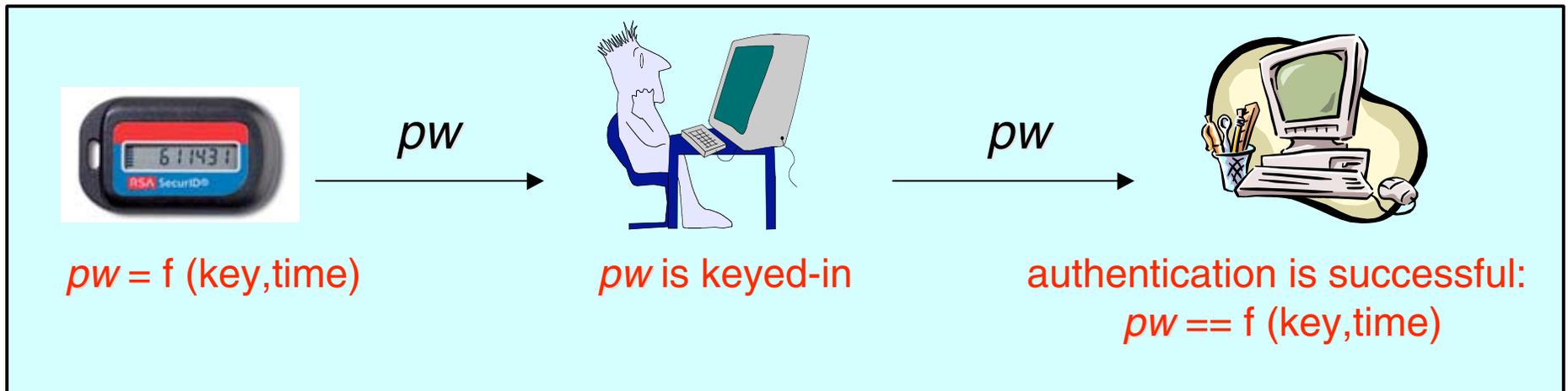
Objectives

- A technology allowing data centers to securely authenticate a user connecting from un-trusted terminals (e.g, a CyberCafe):
 - protects against replaying of a captured user's password
 - protects against exhaustive searching for a user's password
- A technology allowing data centers to securely communicate a short-lived credential to a user:
 - protects against hijacking a session
 - provides data integrity and message confidentiality



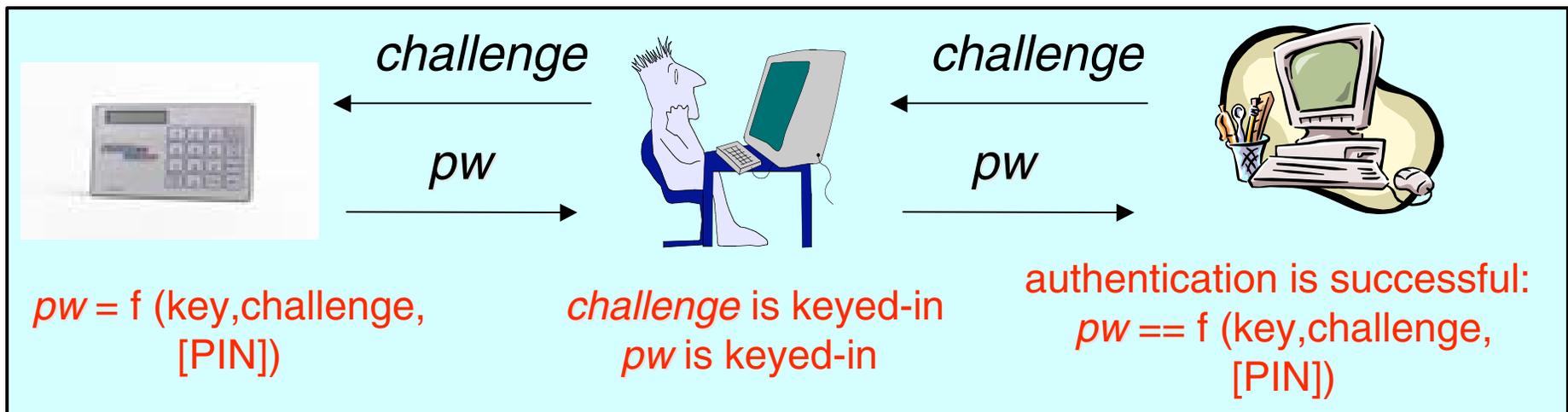
Technology for One-time Password authentication based on time

- The hand-held device derives a password pw as a function of its key and its clock
- The user keys pw into the terminal to authenticate himself to the server
- The server compares pw to an independently computed password
- RSA SecureID is an example of this technology:
 - advantages: simplicity of utilization, ...
 - drawbacks: synchronization, secure-channel needed,...



Technology for One-time Password authentication based on a challenge

- The hand-held device derives a password pw as a function of its key, a keyed-in challenge [and secret pass-phrase]
- The user keys in a challenge [and pass-phrase] to obtain pw , and then in turn keys pw into the terminal
- The server compares pw to an independently computed password
- CryptoCard is an example of this technology:
 - advantages: no synchronization, ...
 - drawbacks: key-in a challenge, ...



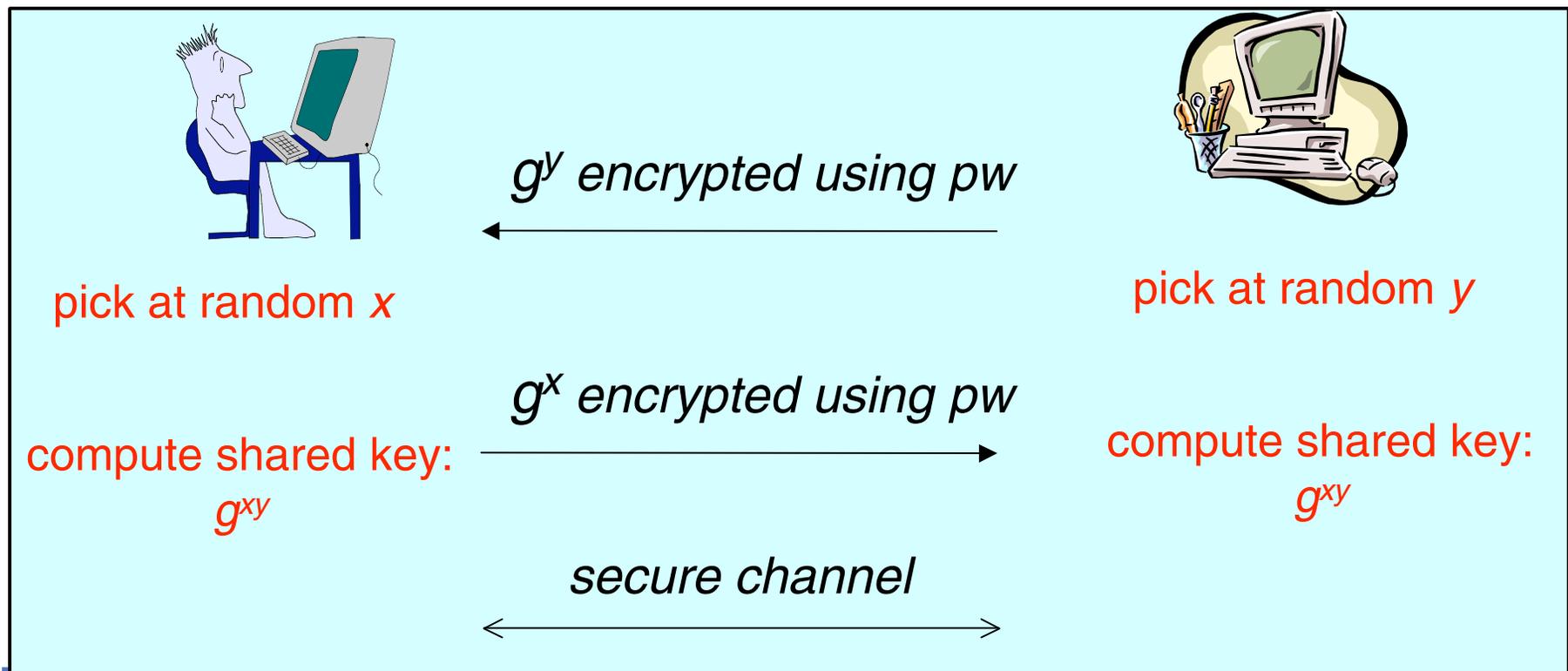
Issues with One-time Password Security

- “Many” implementations send one-time passwords in the clear
 - time-window or man-in-the-middle where hackers can exploit and impersonate/hijack an OTP authentication
 - “Many” implementations use OTP-authentication without mutual authentication
 - “anyone” can pose as a OTP-authentication server
 - phishing attacks exposure
 - “Many” implementations use the Transport Layer Security (TLS) to provide OTP-server authentication
 - requires bootstrap CA trust-configuration
 - explicit trust in OTP-authentication server name
- => Bundle OTP-authentication with key exchange**



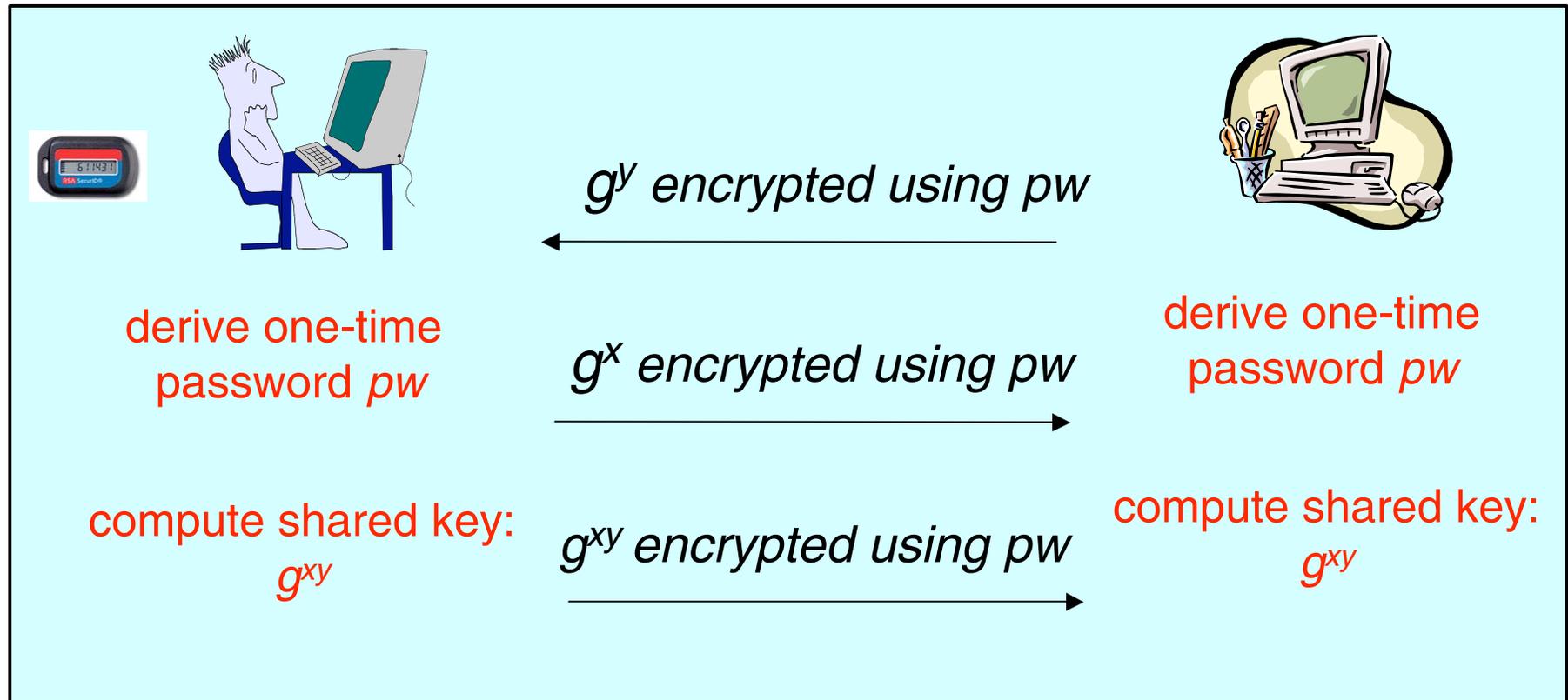
Technology for Key eXchange based on a short password

- The password pw is used to encrypt the Key eXchange algorithm's flows
- The KeyX algorithm allows the two-parties to agree on a session key sk
- The session key sk implements an encrypted and authenticated channel



One-time Password authentication and Key eXchange (OPKeyX)

- A one-time password pw is derived and used to encrypt the flows of the Key eXchange algorithm
- The KeyX algorithm computes a session-key allowing the two parties to implement an encrypted and authenticated channel



The OPKeyX Algorithm: Security Measurement

- The algorithm relies on the impossibility of solving a hard problem
 - the Diffie-Hellman problem (DH)
- The algorithm is secure against dictionary attacks:
 - the attacker does not gain any information about the user's password by mounting an exhaustive searching attack
- The algorithm is secure against replaying of a captured user's password assuming a secure one-way-function for the password-derivation
- The algorithm provides mutual authentication:
 - the two parties prove each other's identity by proving that they know the session key

$$\text{Adv}^{\text{ake}}(t, q, \dots) \leq 2 \cdot q/N + 4 \cdot \text{Succ}^{\text{dh}}(t, \dots) + \text{Cte}$$

, where number of interactions is q and the size of the dictionary is N .



Using OPKeyX in the Grid Security Infrastructure

Grid Security Infrastructure (GSI)



Transport Layer Security (TLS)

symmetric-key
algorithms

key-exchange
algorithms



Transport Control
Protocol (TCP)

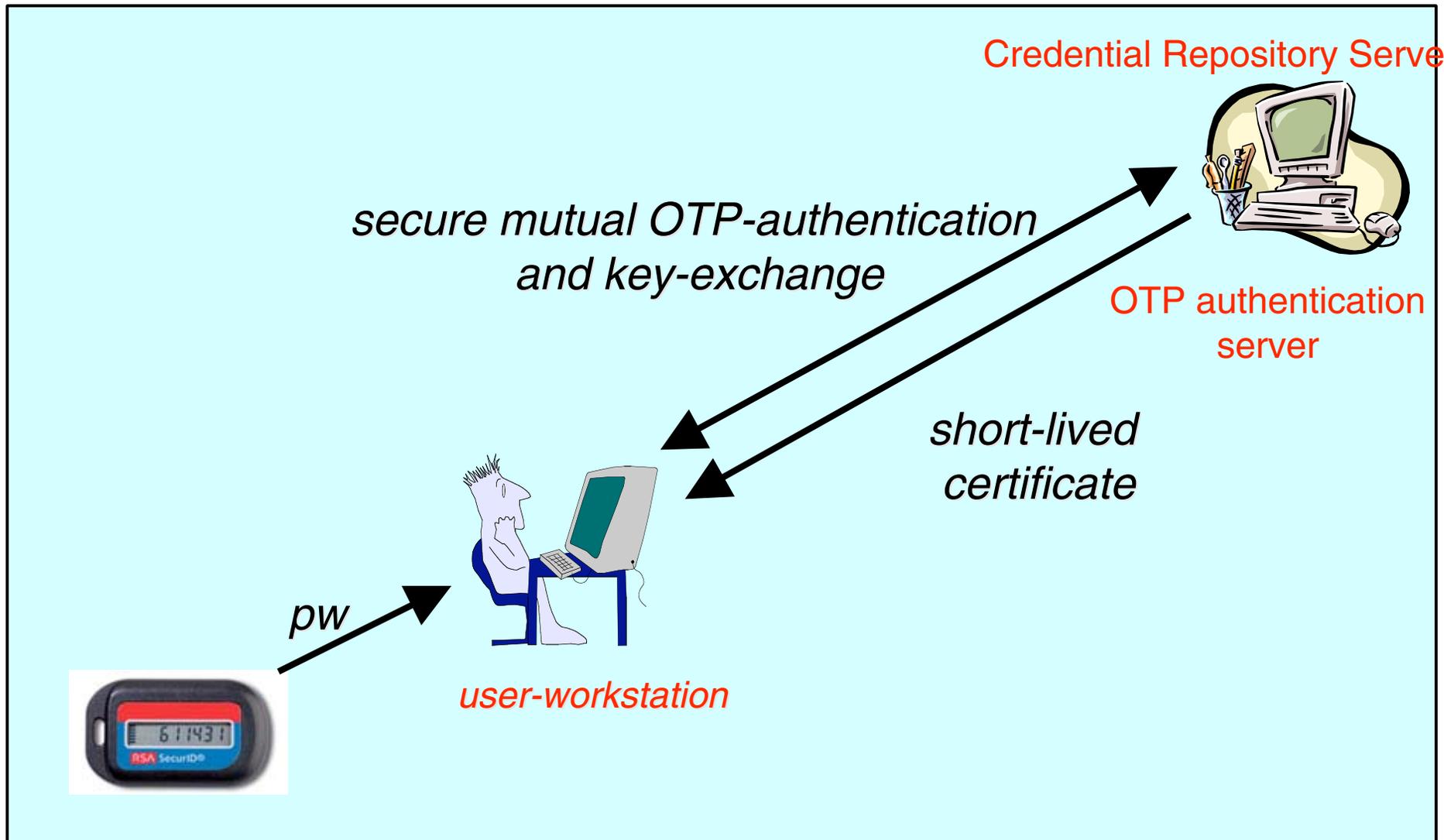


Using OPKeyX in the Grid Security Infrastructure

- The TCP protocol provides the reliable communication channel between the client and the server
- The TLS protocol provides the secure communication channel between the client and the server
 - confidentiality, authenticity, and integrity
 - authorization and access control
- The TLS protocol uses OPKeyX as the key-exchange method
 - OPKeyX computes the shared key and hands it to the symmetric-key algorithms for data encryption and integrity



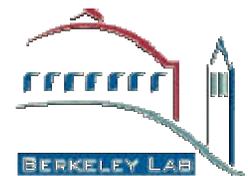
Using OPKeyX in Grid environments



Advanced Topics

Secure Group Communication Scishare: PKI-Based security in a P2P application

Secure Group Communication



Motivation

- **An increasing number of distributed applications need to communicate among peers:**
 - collaboration and videoconferencing tools (Access Grid)
 - distributed computations (Grid)
 - replicated servers
- **An increasing number of distributed applications have security requirements**
 - privacy of data
 - protection from hackers
 - protection from viruses and trojan horses
- **Group communication must address security needs**

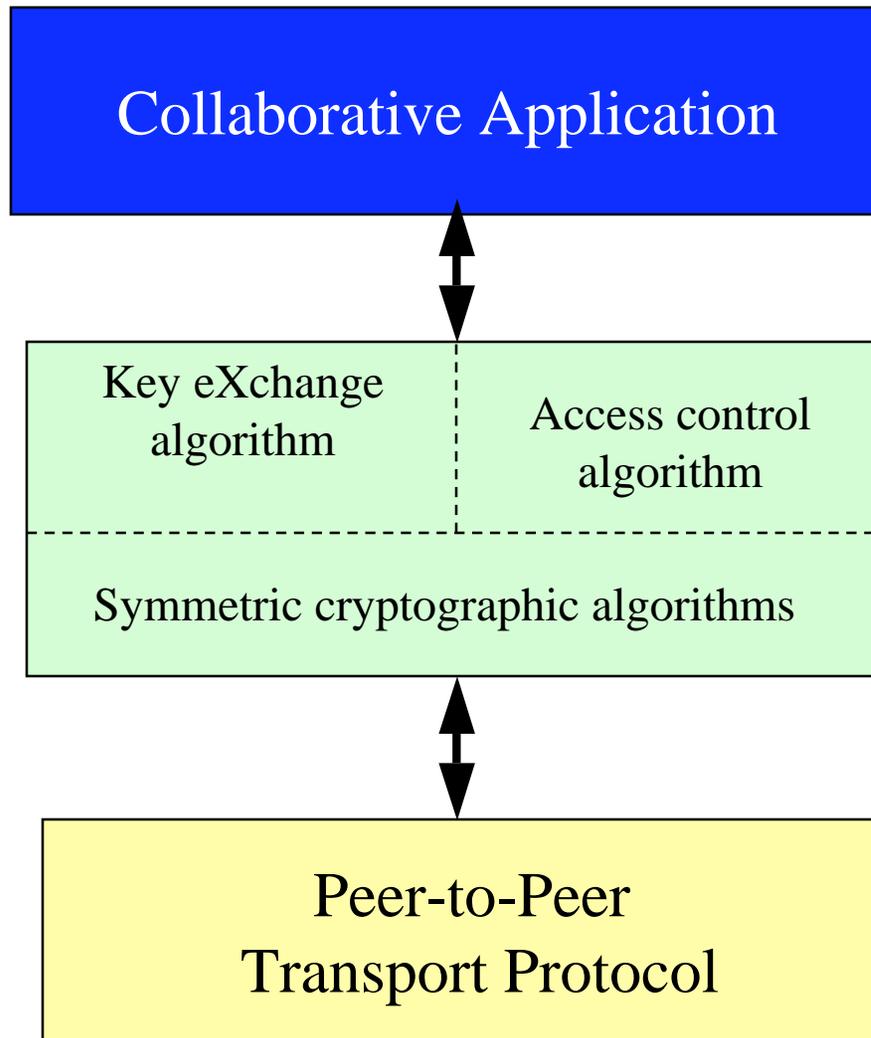


Objectives

- **Provide an efficient and reliable communication between participants aggregated into a group**
 - communication channel directly connecting the participants (no intermediary server)
 - remove dependence on centralized servers (bottleneck, scalability)
 - support peers spread across the Internet
- **Provide a secure communication among the peers**
 - support confidentiality, authenticity, and integrity
 - support access control based on certificates
 - security services optional



Architecture



The Components

- **Symmetric crypto algorithms (e.g. Rijndael and HMAC)**
 - **implement an authenticated and encrypted channel**
- **A group key exchange algorithm to enable peers to establish a session key**
- **An access control mechanism makes sure that only the legitimate parties have access to the session key**
 - **certificate-based: off-line (does not participate in key exchange)**
 - **password-based**



The Secure Group Layer (SGL)

- **Provide reliable and ordered delivery of messages**
 - data messages are delivered in order - FIFO, partial, and total
 - at each member of the group
- **Provide membership notifications**
 - membership changes delivered in order with respect to data messages
- **Several systems provide a reliable multicast layer**
 - e.g., Isis, Ensemble, Totem and InterGroup



SciShare

Example of a secure Peer to Peer application

Peer-To-Peer, ad hoc Applications

- **No central trusted authorities**
- **All participating entities are equal**
 - **Run same software**
- **Highly dynamic**
 - **Users and resources join and leave frequently**
- **Examples**
 - **Chat, file sharing, audio/video conferencing**



Traditional Security Model

- Authorized users are predefined
 - Either you are in or you are out
 - Harder to meet 'new people' online in a collaboration
- Policies are managed by third party entities (administrators)
 - Hard to start a spontaneous collaboration
 - Setup takes time
 - Hard to invite a person to an established collaboration
 - Must contact resource administrators
- Security becomes a nuisance
 - File sharing, chat, shared spaces, ...



A Flexible Security Model

- Partition the collaboration into two types of secure components:
 - **Public**
 - Capture users' identities
 - Allows for gradual trust in the collaboration
 - Turn off public components => traditional model
 - **Protected**
 - Authorized users only
 - Give invitation/escort powers to some of these users
- Example of components:
 - Communication channels, online instruments, chat rooms, shared spaces, files, ...

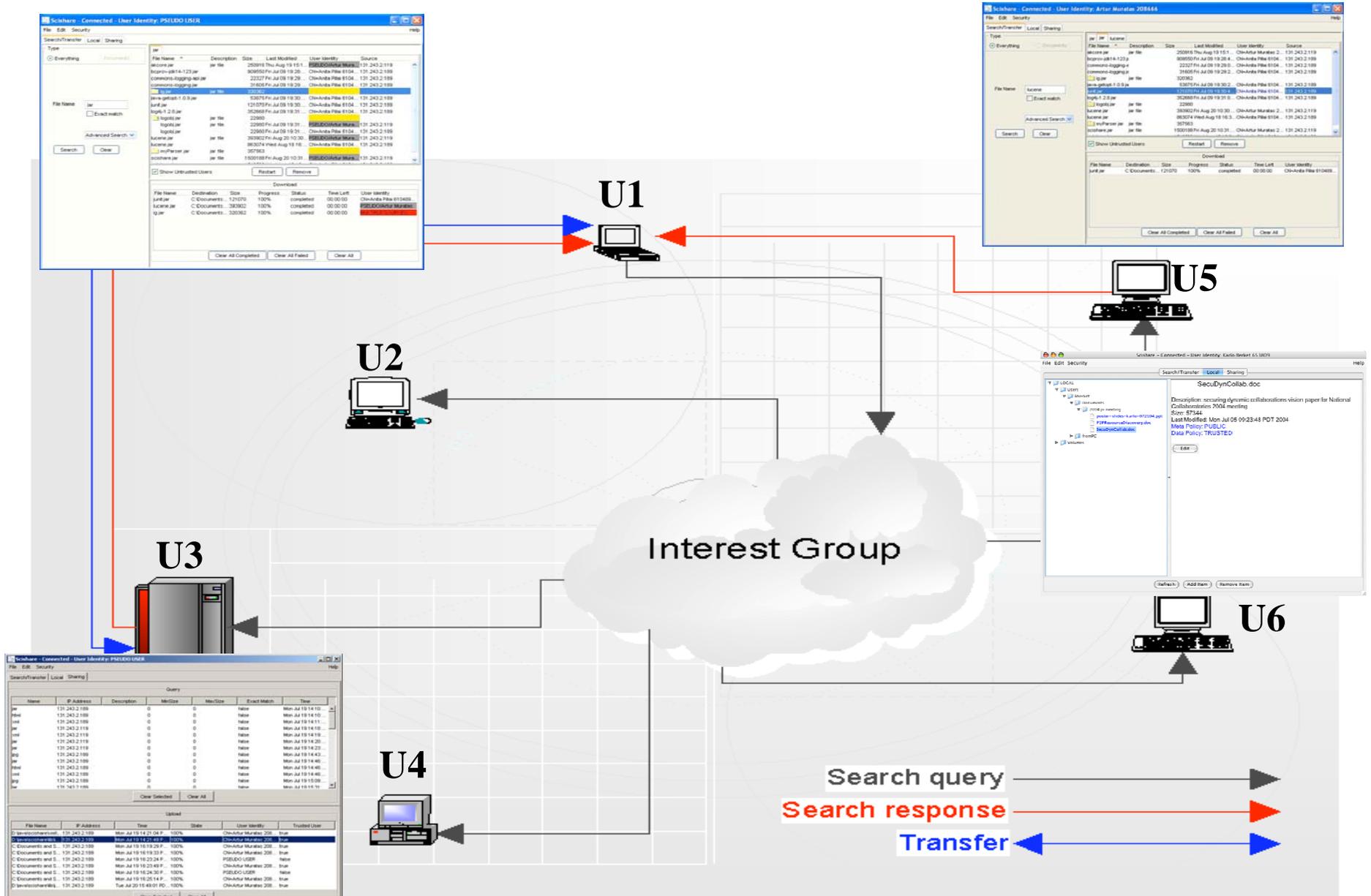


Scishare: Information Sharing

- **Search for information**
 - **Multicast communication**
- **Share and retrieve information**
 - **Unicast communication**
- **Security approach**
 - **PKI in order to scale large user population**
 - **X509 certificates (Pseudo)**
 - **Existing security solutions.**
 - **Modifications are external in order to reduce the risk of introducing security holes**
- **Pure peer-to-peer**
 - **Rely less on servers to retrieve X509 certificates/attribute certificates**



Scishare Architecture



Securing Unicast (SSL/TLS)

- Provide users with pseudo (self-signed) X509 certificates if they don't have any
 - SSL requires servers to have certificates.
- Public channel
 - Everyone has access
 - Custom trust manager
 - Accepts any valid chain
 - Marks users as trusted if user and chain verify
 - Remember un-trusted users
 - Users can later authorize un-trusted users based on their experience with those users
 - One channel provides access to all public and protected resources
 - Simplifies development



Securing Multicast

- Need a secure group communication channel with properties similar to SSL
 - Implements an authenticated and encrypted group channel
 - Enables group members to establish a group session key
 - Supports Certificate-based access control
- SGL fits the description
 - Sits on top of Inter Group Protocol (IGP)
 - Designed after SSL but for groups of more than 2.
 - An initial prototype is now ready
- Need two channels? <ip, port>
 - Public/protected
 - Invitation to the protected channel



Securing Files and Metadata

- **Policies**
 - **Simple and flexible (set by owner of file)**
 - **PUBLIC, PRIVATE, TRUSTED**
 - **Allowed groups**
 - **Support distributed groups**
 - **Rejected users**
 - **A policy can be mapped to many resources**
 - **Hierarchical resources inherit parent policy by default**
- **Revoked X509 certificates**
 - **Pseudo users?**



References

- **SSL/TLS**
 - **TLS v 1.0 RFC** - <http://www.ietf.org/rfc/rfc2246.txt>
 - **SSL-v3** <http://wp.netscape.com/eng/security>
 - **openSSL** <http://www.openssl.org/>
- **IETF PKIX working group**
 - **Standards for X.509 certificates, Attribute Certificates, OCSP**
<http://www.ietf.org/html.charters/pkix-charter.html>
- **GSI, CAS**
 - <http://www.globus.org/security/>
- **SSH**
 - <http://www.openssh.org>
- **Kerberos**
 - <http://web.mit.edu/kerberos>
- **Internet2** - <http://middleware.internet2.edu>
- **Liberty Alliance** <http://www.projectliberty.org>

