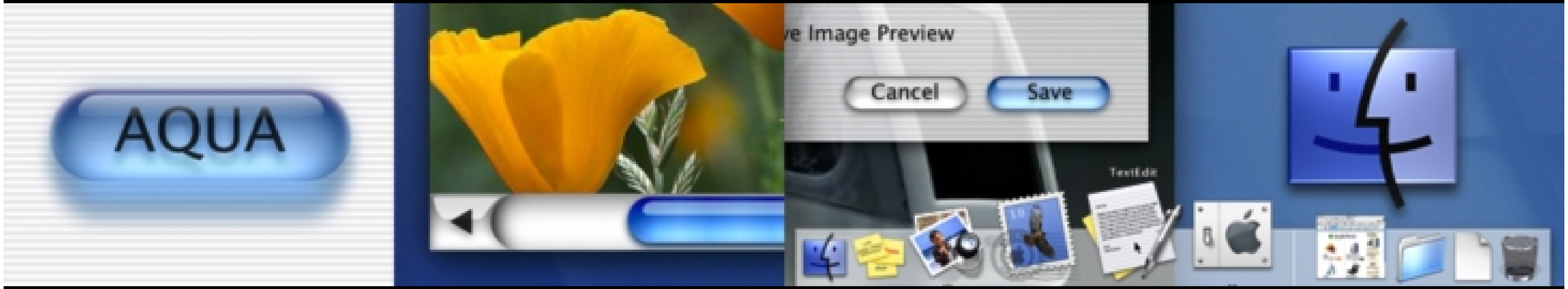
Session 409

WebObjects and Security



David Neumann
System Engineer

Introduction

- Security Concepts
- Coding techniques
- Discussion of a new security kit for WebObjects
  - WOSecurityKit including frameworks, WOAdaptors, and a demo
- B2B applications

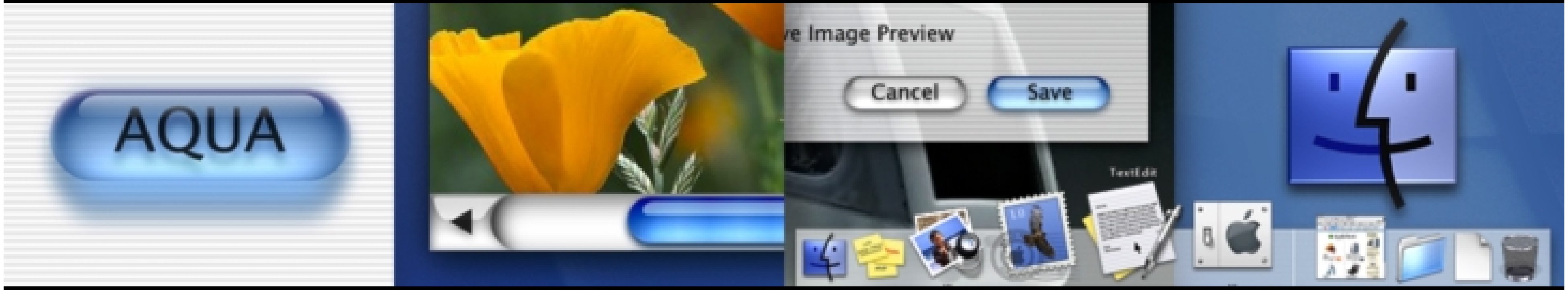
What Is Security

- Secrecy
  - The focus of this talk
- Integrity
  - Some detail
- Availability
  - Not covered: things like DoS

Outline

- Cryptography
- Authentication Techniques
- Access Control in EOs
- Integrity of Transactions

Cryptography

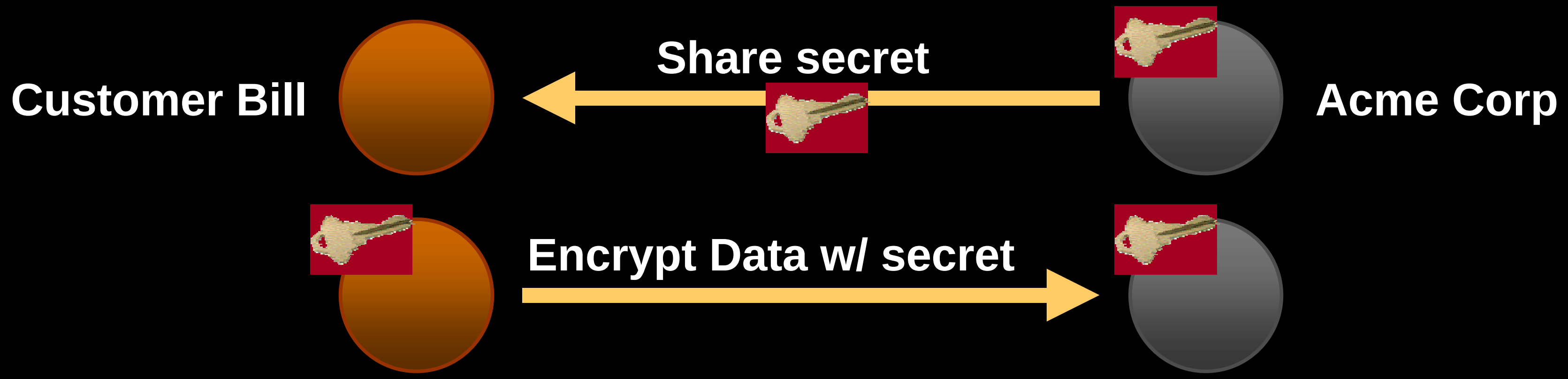


Cryptography

- Crypto Primer
  - Secret Key Crypto
    - You share a single secret,
      a different secret with each user
  - Public Key Crypto
    - You share a public secret with all users,
      but keep a private secret only you know

Secret Key Crypto

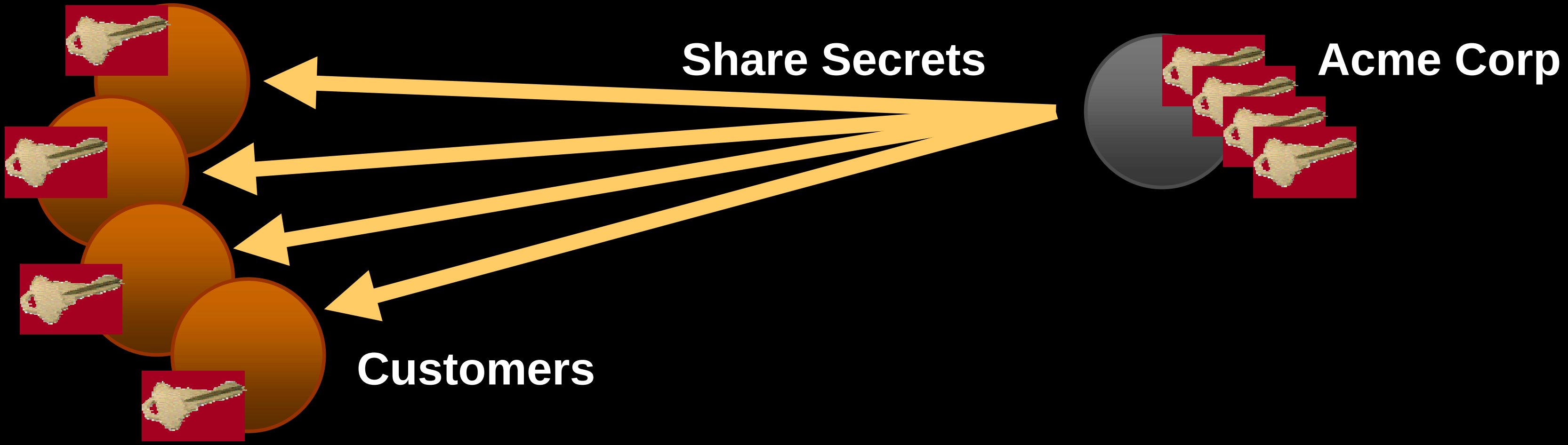
The Secure Channel Problem



- You encrypt data for a secure channel
- But to get a secure channel you must exchange a secret
- So you need a secure channel to get a secure channel

Secret Key Crypto

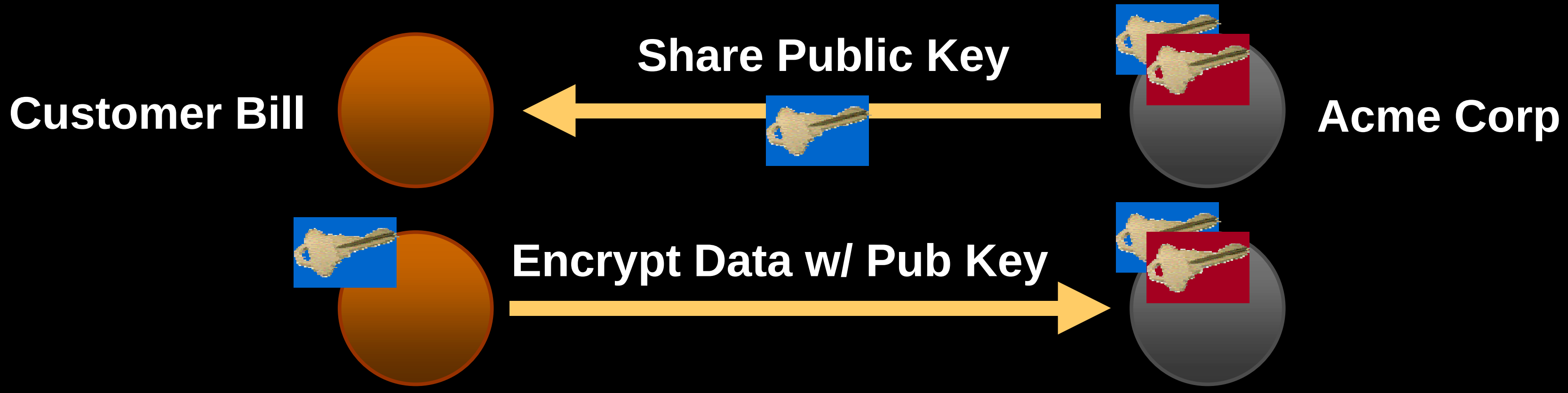
The Key Distribution Problem



- Sharing the secret must be done carefully
  - Meeting face-to-face at a ‘registration authority’
- Acme has to do it (differently with different key) for every Customer
  - The same secret shared by all isn’t much of a secret

Public Key Crypto

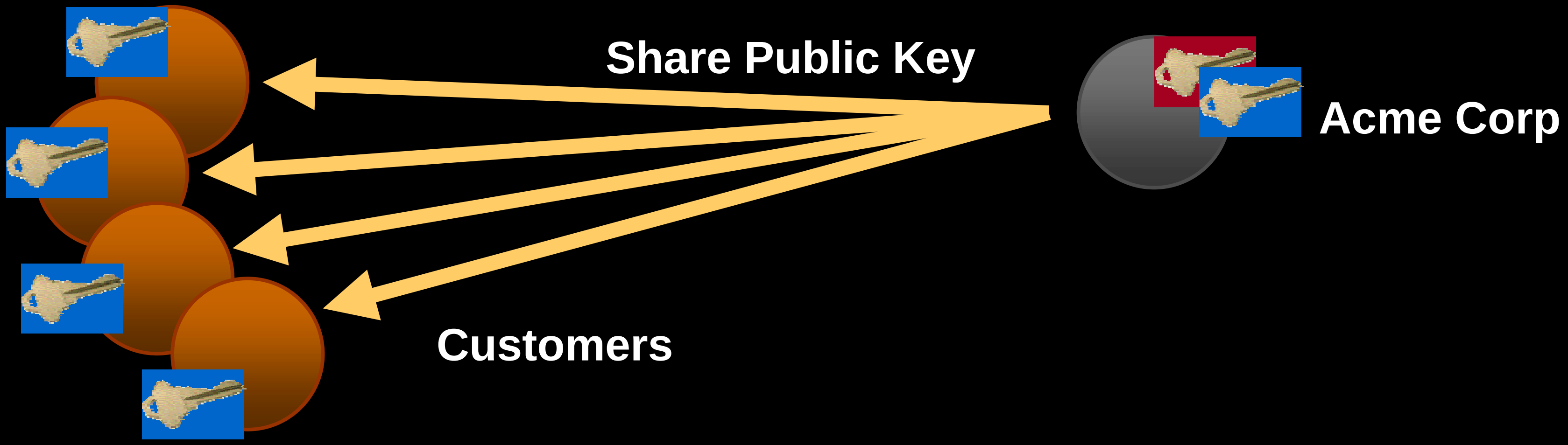
No Secure Channel Problem



- You encrypt data with public key
- Key can be shared in the clear
- Only private key can read the data
  - Public key cannot decrypt what it encrypts

Public Key Crypto

No More Key Distribution Problem (Almost)



- You can publish the same non-secret to world
  - No special meetings
- Every customer can use the same key
- Still two loopholes

Public Key Crypto Loophole #1

- **Q**: How do you know that the key for Acme is for the *real* Acme?
- **A**: You don't—unless you have some credentials that say so
- Solution:
  - A trusted third-party that assures Acme is Acme by issuing an ID binding Acme's public key to its address, name, D&B number, etc.
  - DoT issues driver's license; Certificate Authority issues digital ID—Both require a registration process
    - The more involved this process, the better the ID

Questions

- What is a secure hash?
- What is a digital signature?
- How do you know which CAs to trust?
- How can you tell a fake ID from a real one?
- How can I get an ID for encrypting a message?
- What is the second public key crypto loophole?

SSL

- SSL is an implementation of public key crypto on the web
  - Acme.com's web server presents its Digital ID
  - Your browser checks that the ID is issued by a trusted CA
  - Your browser encrypts a random secret key to the server using the server's public key
  - Browser and server exchange further info encrypted using secret key crypto

Using SSL in WebObjects

- You don’t have to lift a finger in some cases
  - A sysadmin however will need to:
    - Get a digital ID (server certificate) from a CA like VeriSign, Entrust.net, etc.
    - Configure web site to use it
      - For test ID: **http://digitalid.verisign.com/server/trial/index.html**
- WOApp has to run behind a web server such as Apache, iPlanet/Netscape, or IIS
- Resources are accessed using https:// instead of http://
  - Doesn’t that sound easy?

Using SSL in WebObjects

- WebObjects generates partial URLs by default

/cgi-bin/WebObjects/App.woa/wo/F00000EXSA/1.2

  - If you access site over secure URL, this link will be secure

- To force SSL you need to
  - Access your app from a secure link, **or**
  - Force WebObjects to generate full URLs

https://wosite.com/cgi-bin/WebObjects/App.woa/wo/F00000EXSA/1.2

Forcing Access over SSL

- Use private Obj-C API to force full URLs

  http://til.info.apple.com/techinfo.nsf/artnum/n70101

- Create custom WOHyperlink and WOForm implementaions
- Use a redirect technique
  - Method used in the Technotes in the WOInfoCenter
  - Method lets you use normal elements and doesn’t require private API

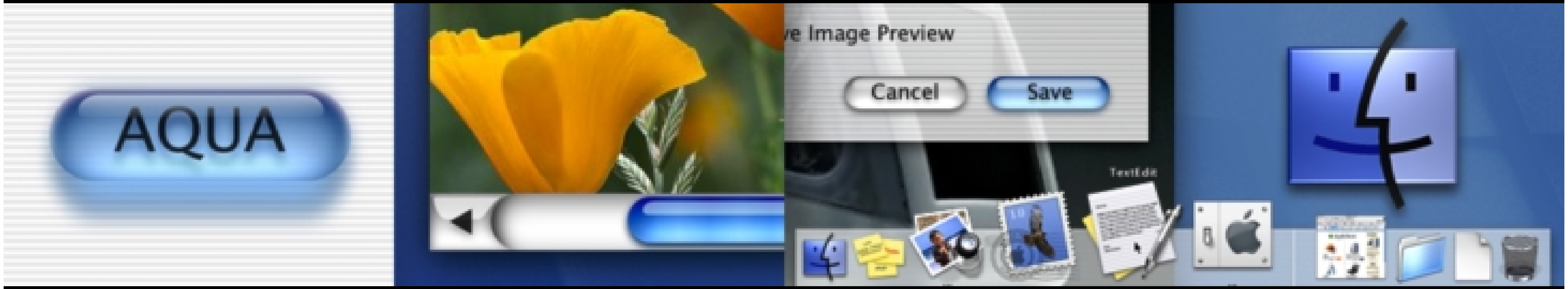
SSL URLs via Custom Component

```
HyperlinkContainer: WOGenericContainer {
    elementName = "a";
    invokeAction = ^action;
    href = href;   }
```

- ‘action’ is the method on your page to invoke
- ‘href’ is the actual URL WebObjects generates
- See WXHyperlink for a starting point
  - Use method like this for href in your version:

```
public String href(){
     return “https://hostname” +
    context().componentActionURL();    }
```

DEMO



SSL using Redirect—Introduce WebObjects AuthPolicy

Encrypting Programmatically

- **Why?** Some stuff should be secret
  - Passwords, credit card numbers, personal data…
- **How?**
  - Buy a crypto lib such as BSAFE (C-lib) and JSAFE (Java lib) from RSA
  - Download a free lib such as SSLeay, Intel's CDSA, Microsoft's CyptoLib

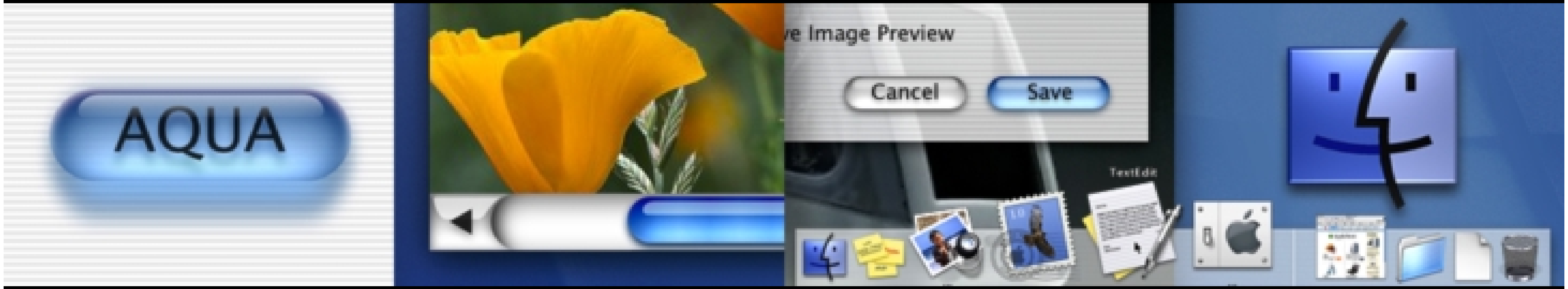
Encryption Techniques

- Explicitly call crypto functions
- Implicitly encrypt/decrypt
  - Use custom accessor methods:
    - Encrypt in setMethods
    - Decrypt in getMethods
  - For performance
    - cache on get
    - Reset cache on set

Encryption Questions

- What key size?
- Does my data get less secure as computing power increases?

Authentication Techniques



How to Login

- There are two logical, and two physical aspects
  - Logical
    - Are you whom you claim to be?
    - Do you have access?
  - Physical
    - Gathering credentials (presentation specific)
    - Processing credentials (business policy specific)

When to Login

- No pages allowed unless logged in
- Allow surfing until login required
  - Show link to login, and re-navigate to protected page
  - Prompt for login then immediately access protected page
- Prompt for login on WOSession timeout

When to Login

- Access Posture
  - Default to allow/deny all pages?
  - Default to allow/deny all DirectActions?
  - Must access all pages in private (over SSL)?
  - Exceptions if any to the default posture?

Login Panels

- Simple, right?
  - Many ways to gather username/password
    - HTML page, HTTP login panel, Certificate, Cookie
  - Many ways to verify credentials
    - RDBMS? LDAP? File? ERP App?
  - WOAuthPolicy provides
    - Three presentation styles
    - Delegation hooks for custom verification business logic

Sessionless Login

- Benefits
  - Allows login page to be bookmarked
  - No “session expired” on login!
  - Less resource impact on you (sessions can be heavyweight)
- For HTML page, use WOForm and DirectAction

Sessionless Login

- Use the DirectAction action handler as the “default action handler”
- Force WebObjects to goto your LoginPage page instead of Main
- In your LoginPage, do *not* call session() anywhere
  - This goes for any subcomponents or sub-subcomponents used on your LoginPage
  - Be wary of session.foo bindings in any wod files

Sessionless Login

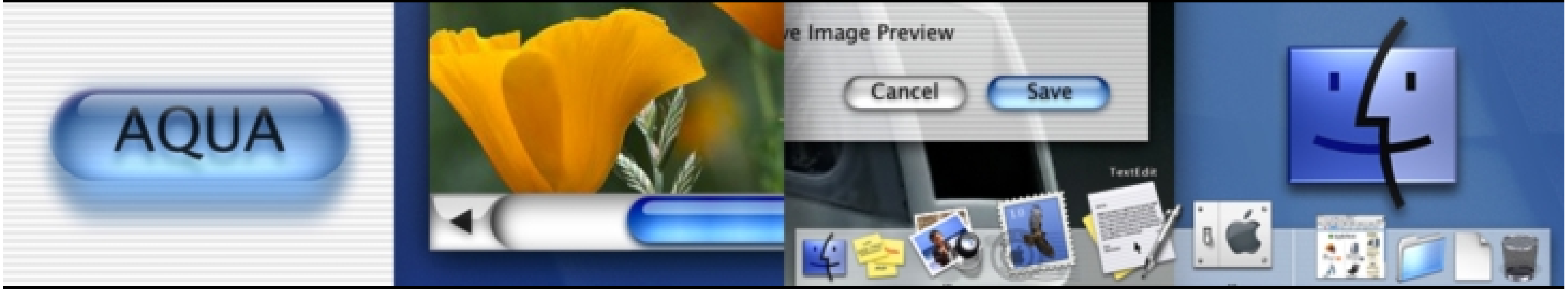
- In your DirectAction subclass of WODirectAction, override defaultAction

```
public WOActionResults defaultAction() {
    return pageWithName("LoginPage");    }
```

- In your Application subclass of WOApplication, enter this line into the constructor

```
setDefaultRequestHandler(
     requestHandlerForKey(
WOApplication.directActionRequestHandlerKey()
) );
```

DEMO



HTML Login Page

Using HTTP Challenge Panel

- Really tricky to do in WebObjects…
  - See the technote in the WOInfoCenter for details
  - Your WOResponse must emit certain statuses and headers, and look for certain headers in WORequests
  - Your web server might not work
  - You have to parse Base64 encoded data

Using HTTP Challenge Panel

- Getting Browser to prompt the panel

```
aResponse.setStatus(401);
aResponse.setHeader(“Basic realm=\”” +
   aRealm + “\””, ”WWW-Authenticate”);
```

- To interpret the response you need to look for a header in the WORequest named “*authorization*”
- Your web server must use an interface that passes this header to the WOAdaptor
  - CGI with Netscape does not
  - NSAPI does

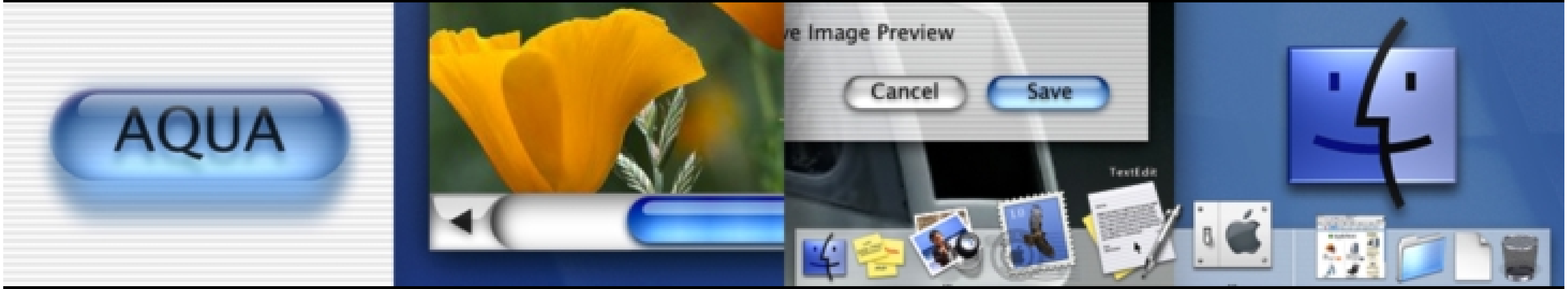
Using HTTP Challenge Panel

- To decode the authorization header, use the JDK’s Base64 support

```
decoder = new sun.misc.BASE64Decoder();
```

- Once you have a normal character string, you can parse it to find the username and password

DEMO



HTTP Challenge Login Page

Logging in Without a Login Panel

- Cookies
  - On successful login once, you might return a cookie
    - Then look for that cookie when a user returns
  - Can be dangerous if
    - User logs in from some other user’s computer
    - User uses IE and Cookies are attacked

Logging in Without a Login Panel

- Digital Certificates
  - The ultimate in user security
  - Reverse role from username/password
    - Web server identifies user
    - WOApp merely authorizes access
      (no password store need be consulted)
  - Requires HTTPS

Digital ID and WebObjects

- Manipulating the ID
  - Find it under a header as an ASN.1 BLOB encoded in Base64 format
  - Parse it Using the Java security package (sun.security.x509.*)
  - Validate it's status via a CRL or VA
    - WOSecurityKit includes a wrapper for ValiCert's online cert status software/service

Digital Certificates and Granting Access

- Web server can be configured to grant access to certain digital certificates
- Or your WOApp can perform this duty
  - Needs the certificate to see if you are allowed access
    - Unfortunately the WOAdaptors shipped with WebObjects either do not even ask for the cert, or they truncate it
    - As part of the WOSecurityKit, you will find source code for CGI and NSAPI adaptors that process a client certificate properly

The Second Loophole

- Just because a unexpired digital ID is issued by a trusted CA does *not* mean it should be trusted
  - The ID may have been revoked
  - You should check a CRL or contact a VA before accepting any digital ID
  - Do merchant’s trust your VISA card? Or do they scan it for validation?

Digital Certificates and the User

- User’s private key must perform an operation (signing or encryption)
  - To perform signing, a user must unlock their private key (usually with a passphase)
  - The private key is usually stored in a file encrypted with the access passphrase

Digital Certificates and the User

- Why bother with a Digital ID to avoid passwords, when you use a password anyway to unlock it?
  - Unlike a username/password, this password does not leave your computer
  - The passphrase is something you created
    - It wasn’t issued by anyone
    - So only you know it

Digital Certificates and the User

- Storing a private key in a file has downsides
  - Unlike a username/password, it’s not portable (unless you carry a floppy)
  - It should be extraordinarily well protected and files don’t cut it

Digital Certificates and the User

Smartcards to the Rescue

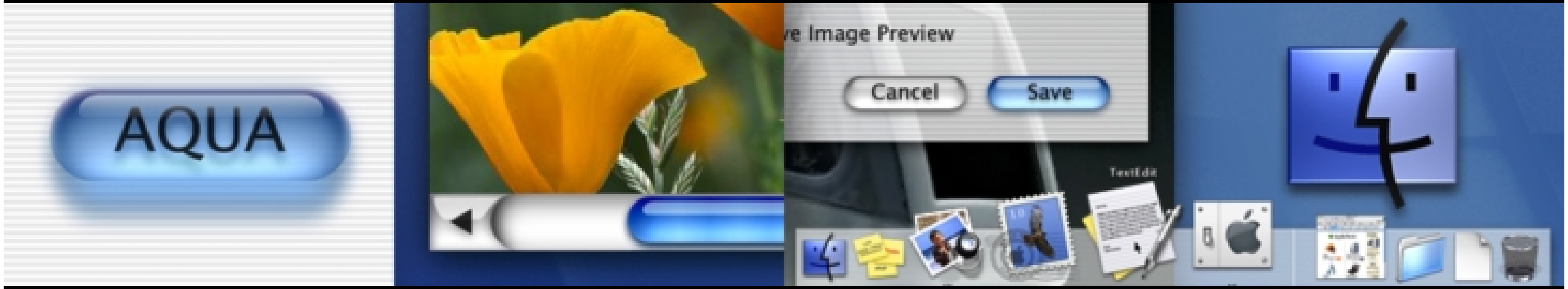
- A private key can be stored on a Smartcard
  - Smartcards are as portable as credit cards
  - Smartcards have a CPU that performs the actual operations
    - The private key never leaves the card
    - Hacker would need to physically steal your card
  - Smartcards can be attached to devices that accept your passphrase directly

Digital Certificates and the User

For the Truly Paranoid…

- Some smartcards can be equipped with a biometric passphrase
- You feed the passphrase data through a biometric device
  - Existing readers for: palm, finger, voice, face, or retina
  - Imagine logging into a web site like this:
    - Insert your smart card
    - Place your thumb on it when prompted
- To digitally impersonate you, someone needs your smartcard, and some part of your body

DEMO



Digital Certificate Login and ValiCert.framework

Blocking Access to Your App

- Override WOComponent’s *appendToResponse()*
  - Not necessarily OK to goto an action’s destination
  - Prevent page display no matter how page is accessed:
    - Initial app access, DA, or ComponentAction
  - If you can see a ComponentAction it (usually) means it’s OK to execute it
    - If not, don’t show it

Blocking Access to Your App

- Override WODirectAction’s *performActionNamed()*
  - DAs can be accessed from anywhere
    - Whether you gen the page or not (can’t hide them)
  - Protecting *appendToResponse()* does not prevent the DA from executing
    - But does hide the result

Blocking Access to Your App

- Your version of *appendToResponse*()
  might look like:

```
public void appendToResponse(WOResponse r,
WOContext c){
    if(shouldDenyPageGen(aContext)){
         WOComponent *p = WOApplication.application().
              pageWithName(“LoginPage”, c);
         r.setContent(p.generateResponse().content());
    }else{
         super.appendToResponse(r, c);
    }
}
```

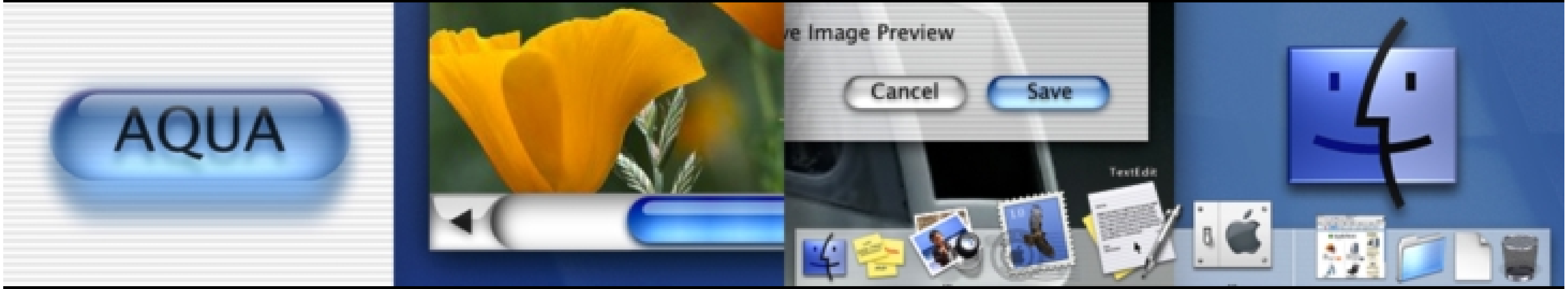
Blocking Access to Your App

On Demand Login

- Before generating the LoginPage response, push it the page name of the intended destination
  - Get name from context passed into *appendToResponse()*

    aContext.page().name()
  - Name is better than instance:
    - Lighter weight
    - No side effects
- Your LoginPage then should goto to this destination on successful login

DEMO



On-demand Login

WOSecurityKit

- What is it?
  - Modified WOAdaptors including source
  - A security whitepaper
  - WXAuthPolicy framework
  - Celo Digital Sig plug-in support framework
  - ValiCert Digital Cert Validation support framework
  - A Demo app that uses all of the above

WXAuthPolicy.framework

- What is it?
  - Three credential gathering schemes
    - HTML page, HTTP challenge, and Certificate
  - Hooks for custom auth biz logic
  - Access posture for pages, actions, and privacy
  - SSL access toggling support
  - Sessionless login
  - More…

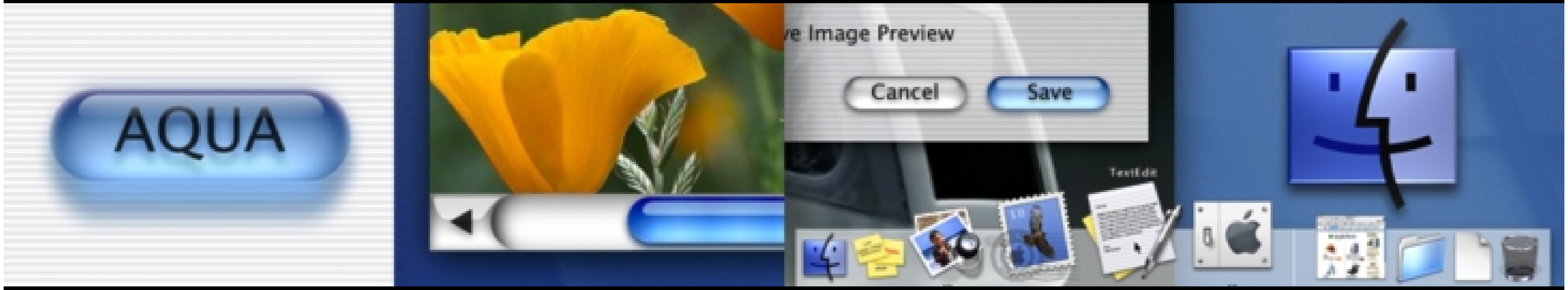
WXAuthPolicy.framework

- How to use it?
  - See the demo application CFN.app
  - Involves inheriting your Components, Session, DirectAction, and Application from WXAuthPolicy superclasses
  - Policy can be set in code or via GUI component

WXAuthPolicy.framework

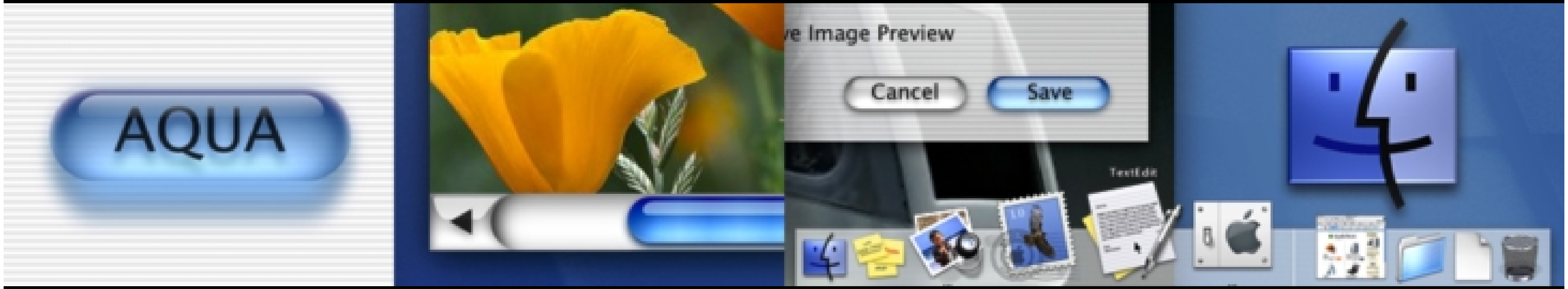
- Where to get it?
  - WOSecurityKit is available online at:
    - http://enterprise.apple.com/wwdc2000

DEMO



WXAuthPolicy: Access Posture, SSL Detection, Fallback Login, On-the-Fly Policy Config

Access Control



Access Control

- Degree of access granted **after** they login
- The question is:
  - Given an instance of Entity A, Can User B
    - See it?
    - Edit it?
- Access depends on the state of both A and B
  - What kind of EO is being edited?
  - What kind of user is attempting to edit it?

Access Control

Techniques

- Have all your EOs implement an interface like this:

```
public boolean canShow(User usr);
public boolean canEdit(User usr);
```

Access Control

Techniques

- An example inheritance chain might look like this:

```
GenericEO
    SecuredEO
        Product
```

- GenericEO contains default access policy
- SecuredEO dictates certain schema
- Product is an example of an EO that might need secured access

Access Control

Techniques

- Implementation of GenericEO might be:

```
public boolean canShow(User usr){
    return true;
}
public boolean canEdit(User usr){
    return true;
}
```

Access Control

Techniques

- Implementation of SecuredEO might be:

```
public boolean canShow(User usr){
    if(usr.equals(creator()))
        return true;
    else if(owners().containsObject(usr))
        return true;
    return false;
}

public boolean canEdit(User usr){
    return canShow(usr);
}
```

Complex Access Control

Ex: Discretionary Access Control

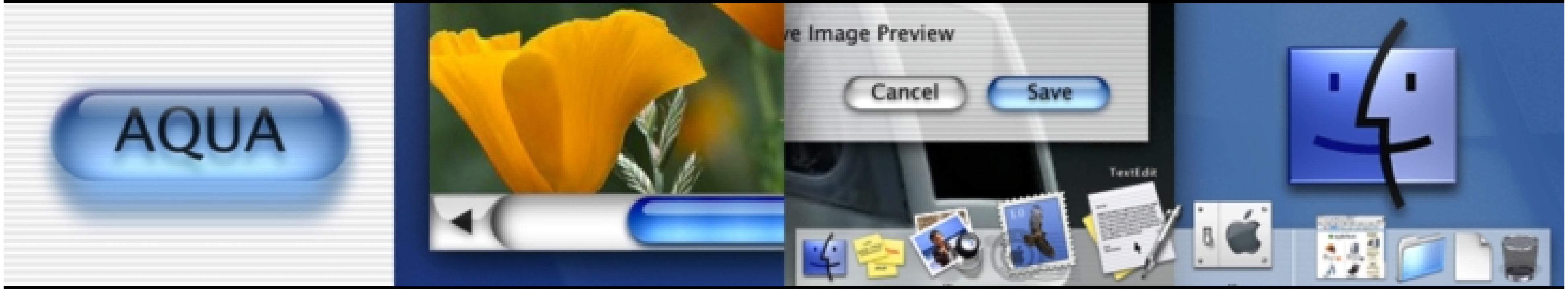
- To mimic DAC
  - Your SecureEOs might have relationships like these
    - creator(): To-one to a User
    - owners():To-many to a set of User objects
    - groups(): To-many to a set of Group objects
    - permission(): To-one to a Permission object
    - Permission objects would have Y/N state assigned to columns like: ownerRead, ownerEdit, groupRead, groupEdit, etc.
  - The Unix file system uses DAC

Complex Access Control

Ex: Mandatory Access Control

- To mimic MAC
  - Your permission table might have level names like
    - "Secret", "Confidential", "Unclassified"
  - Instead of a groups you would have compartments with entries like
    - "Accounting", "Shipping", "Marketing"
  - Implement EOEditingContext delegates to intercept object creation calls
    - Your delegate would disallow insertions unless they had the right permission, compartment assigned
    - Unlike DAC, MAC means users with, say **Secret** permission could not write to a lower permission level like **Unclassified**

Integrity



Integrity

- Aspects of Integrity
  - Data corruption can be tested
  - Data tampering can be detected
  - Origin of data can be proved
- Integrity is usually based on
  - Digital signatures
  - Public key crypto

What Is a Digital Signature?

- You hash a message
- You use your private key to sign the hash
- You append the signed hash to the message

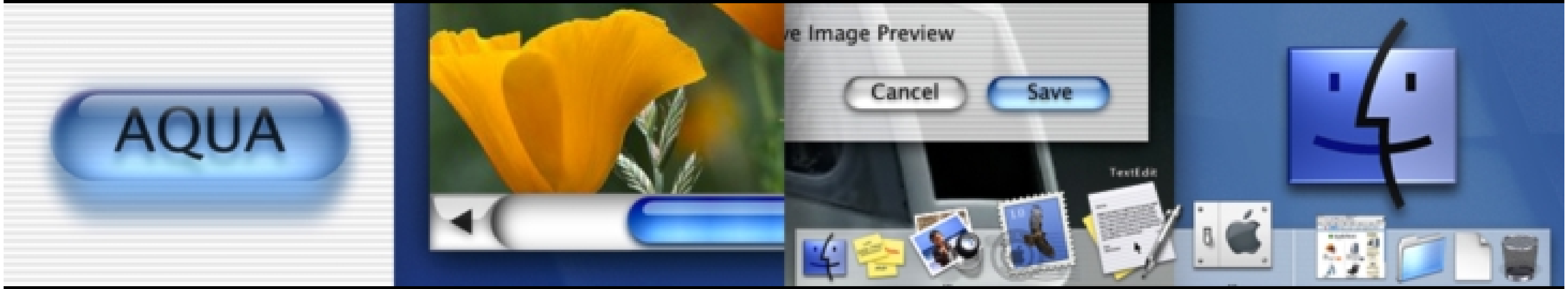
Nonrepudiation

- You have it if you can prove an event happened
  - In the paper world, it’s via ink signatures
  - In the electronic world, it’s via digital signatures

B2C Digital Signatures

- Clients require a browser plug-in
- Example Applications
  - Employee forms processing
  - Brokerage enrollment
  - Paperless workflow with authorization

DEMO



Digital Signature in a Browser

B2B Digital Signatures

- When machines send and receive digitally signed messages
- Ex: DropShip order, PO, any EDI message

B2B Infrastructure in WebObjects

- WebObjects 4.0 added DirectActions
  - Which turn WOApps into services easily callable by other programs
  - But it was still hard to talk to another WOApp programmatically
- WebObjects 4.5 adds additional B2B-oriented support
  - You can programmatically send WORequests to remote apps and get their answers as WOResponses
  - XML support included
    - Help generate XML to be sent over the net
    - Help interpret XML received

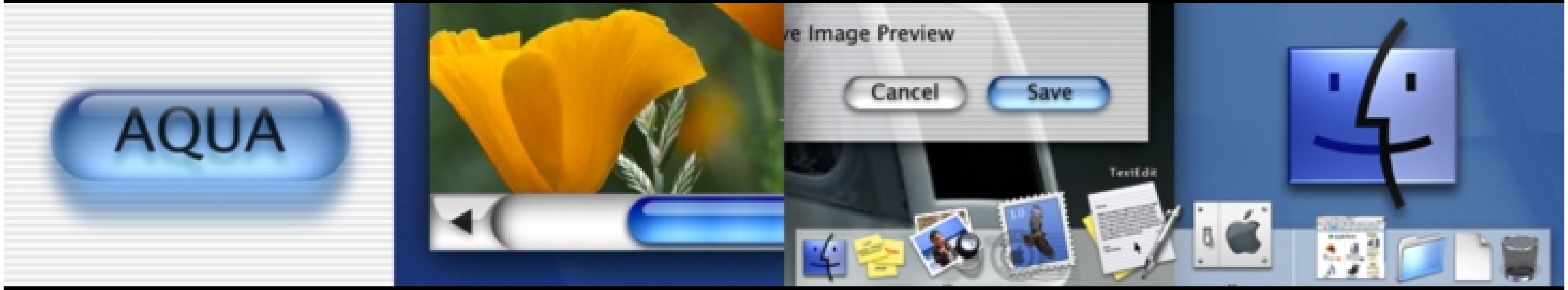
B2B Scenario

- Acme issues PO to WidgetCo
  - Creates an XML document
  - Signed using the Java’s sun.security.* package
  - Encrypted using WidgetCo’s public key
  - Sent using WOMessage API

B2B Scenario

- WidgetCo receives PO from Acme
  - Decrypts with private key
  - Verifies Acme digital signature is valid
  - Verifies Acme digital ID is valid
    - Using a CRL or ValiCert VA
  - Creates a “digital receipt” by
    - Combining Acme’s signed request with a “digital timestamp”
    - And signing it all with WidgetCo’s private key
  - Digital receipt returned to Acme

Summary



Summary

- Cryptography
  - Primer on how it works and usage (SSL and by-call)
- Authentication Techniques
  - Meat of the talk, demos, and area addressed by the WXAuthPolicy.framework built for this talk
- Access Control in Eos
  - Controlling what they see after they login
- Integrity of Transactions
  - Using digital signatures in B2C and B2B messaging, helped along via Celo.framework built for this talk

Roadmap

413 WebObjects: XML Useful for B2B applications	Room J2 **Thurs., 3:00 p.m.**
415 WebObjects: Advanced EOF Place to learn more about biz objects	Room J2 **Fri., 9:00 a.m.**

For More Information

http://www.rsa.com—and get the FAQ

http://www.valicert.com—leading VA

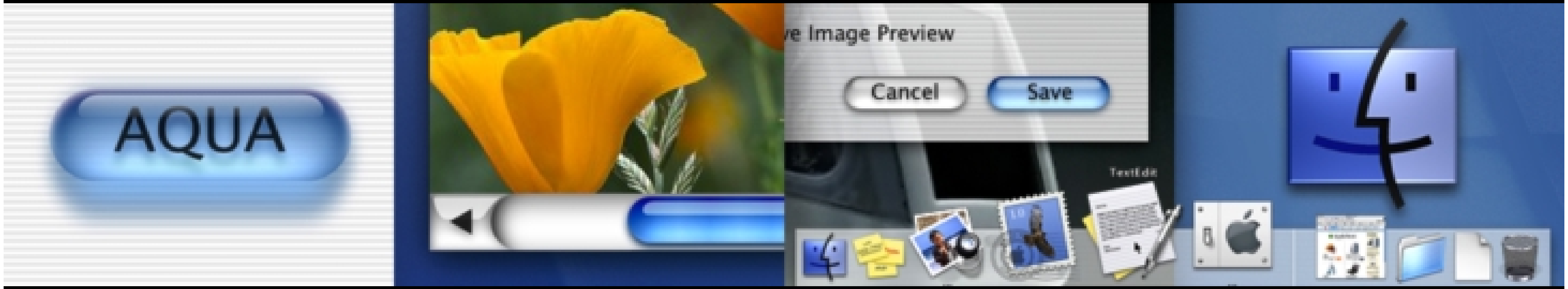
http://www.verisign.com—leading CA

http://www.celocom.com—signing plug-in

See the whitepaper and look over the demo in WOSecurityKit

Session 409

Q&A



David Neumann
SE, ValiCert

Who to Contact

Toni Trujillo Vian
Director, WebObjects Engineering
wofeedback@group.apple.com

Ernest Prabhakar
Product Line Manager, WebObjects
webobjects@group.apple.com

WWDC

Worldwide Developers Conference 2000

Think different.

Digital Certificates—Supplemental

- The SSL protocol has a second optional phase
  - Client Authentication
    - Like the server proves itself to the user, the user proves itself to the server
    - User does so by signing something, a signature the server can verify
    - If the web server trusts the CA that issued your digital ID and the signature verifies OK, only then do you even get access to the WOApp!

Access Control

- You can implement this logic in 2 ways
  - Top down (in your pages)
  - Bottom up (in your EOs)
- Top down you replicate your logic everywhere
- Bottom up you put the policy in once place
  - Your pages don’t have the policy
  - Your pages only ask the questions, your EOs answer them

B2C Digital Signatures

Sample process flow

- User fills out HTML form and submits
- WOApp processes action,
  - Gens document summarizing what user typed
  - Returns page with a plug-in embedded in it
- Src attribute on plug-in retrieves document
- User uses plug-in to select signing cert, enters passphrase, and submits
- Plug-in signs document and sends it to the server

Secure Channel for eBusiness

- WOMessage + WebObjects XML support + public key crypto = secure channel for nonrepudiable B2B communication
  - Crypto signing provided by Java’s sun.security.*
  - Crypto encryption by SSLeay, RSA, Intel, etc.
  - Credential validation with ValiCert.framework