7 January 2014

Responsible Authentication for APIs

Web Service Keys @ OCLC

Steve Meyer

Technical Product Manager, WorldShare Platform

Shelley Hostetler

Community Manager, WorldShare Platform

OCLC

@oclcdevnet

OCLC Developer Network API Workshops

- Covering functionality, use cases, demonstrations, and discussion
  1. WorldCat Metadata API: http://oc.lc/metadata-api-workshop
  2. WorldCat knowledge base API: http://oc.lc/kb-api-workshop
  3. Web Service Keys for APIs
  4. Coming soon! Stay tuned to http://www.oclc.org/developer/news

Workshop Goals

- Describe OCLC's WSKey system and Oauth 2.0 implementation
- Walk through sample code to demonstrate how to implement authentication

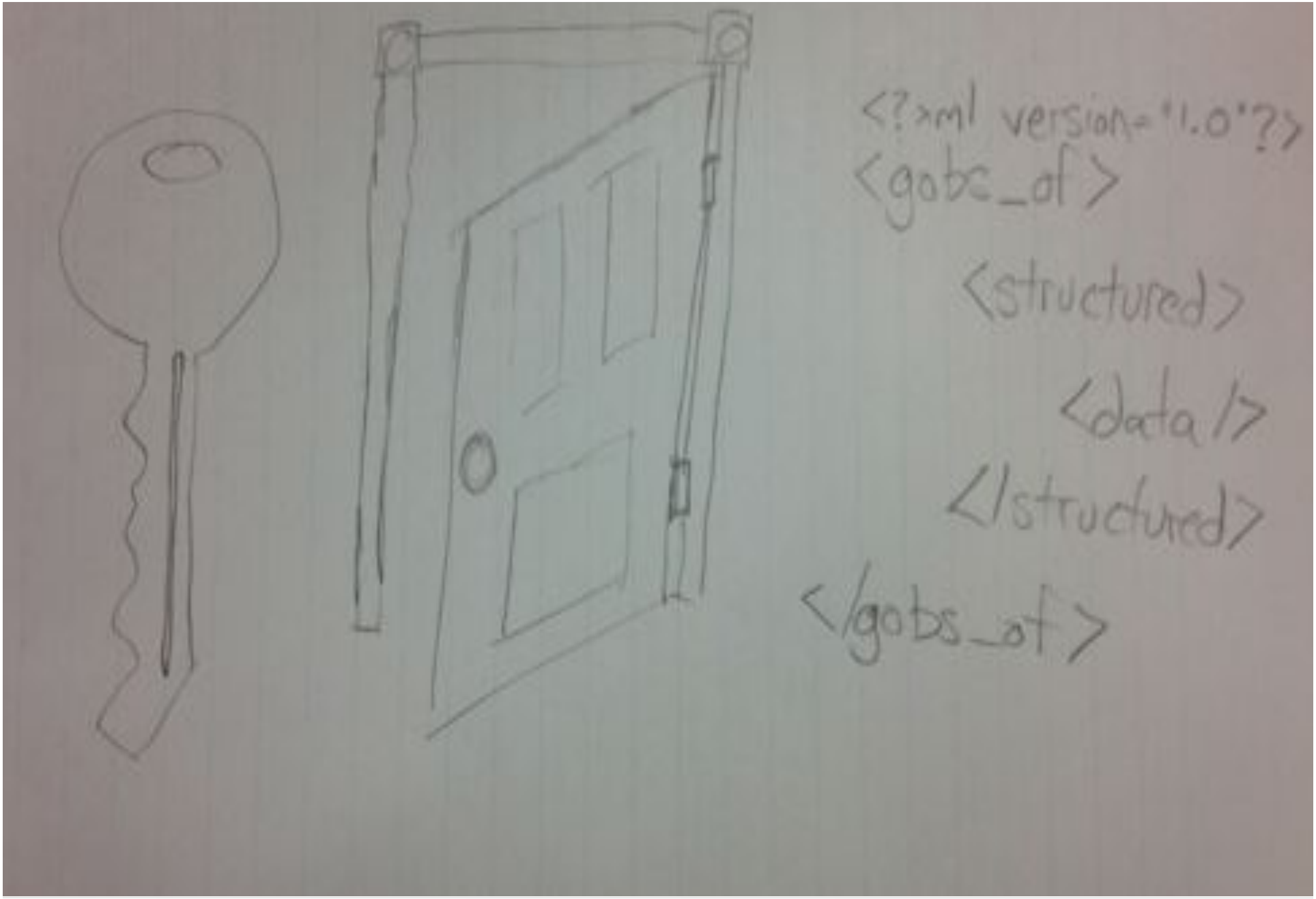
Agenda

- Intro to API Keys @ OCLC
  - what are they
  - how to request a key
  - who can get one
- API keys are a hassle. Why all the bother?
- OCLC's OAuth Implementation:
  - HMAC Signatures
  - OAuth 2 Login Flows
- OCLC Auth Libraries
- Q & A

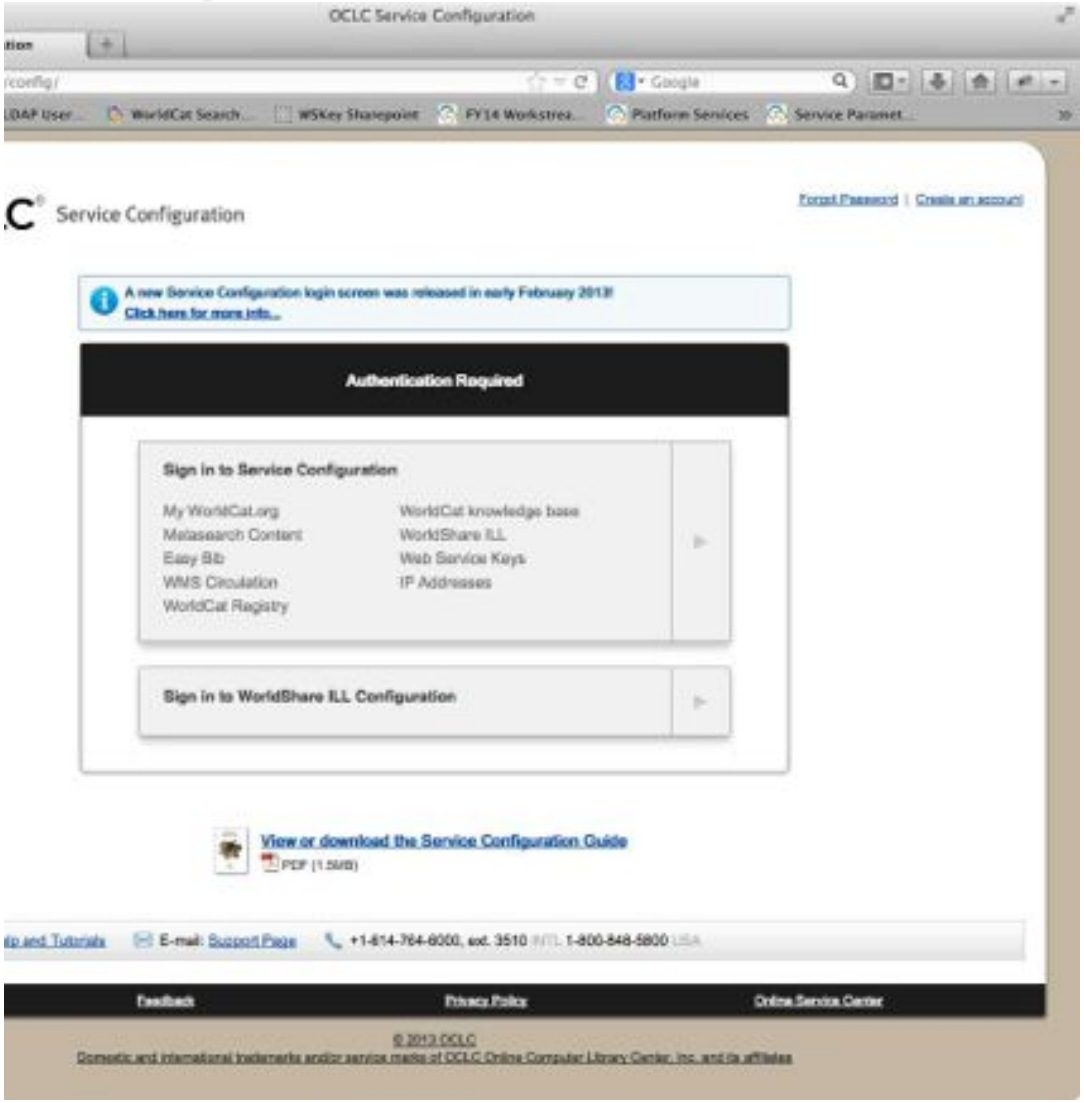
What are WSKeys?

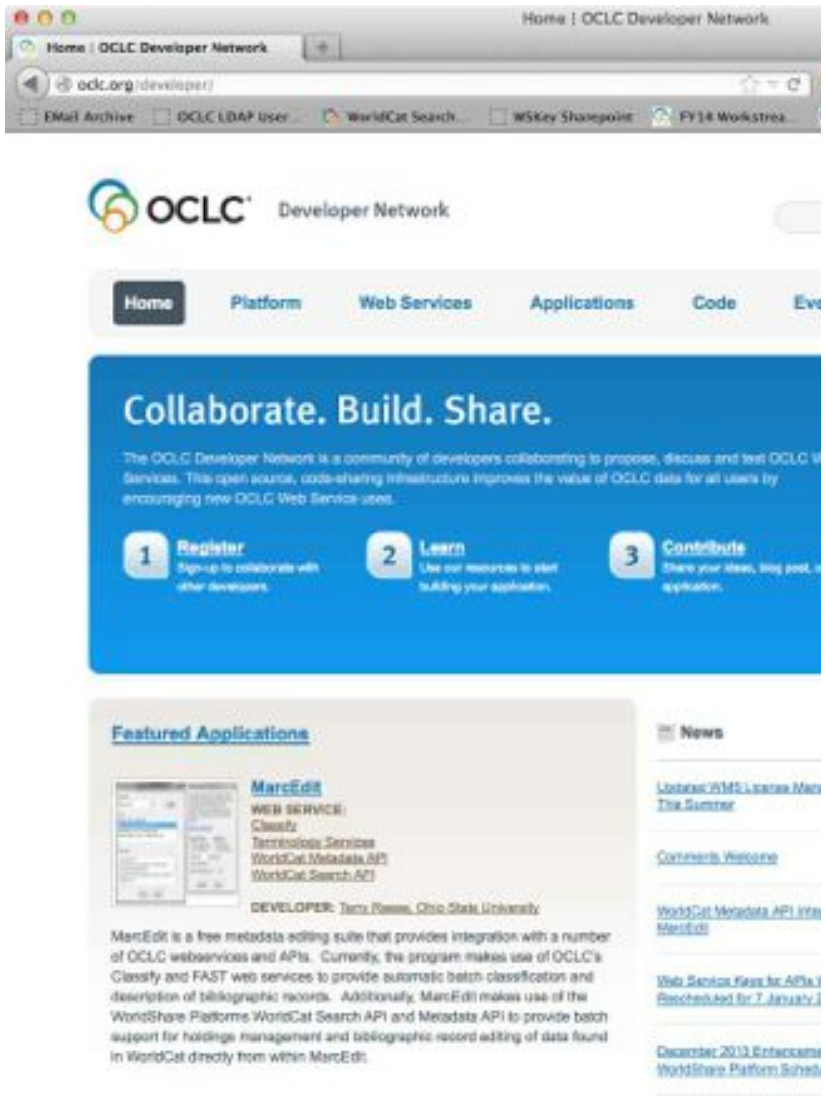
Web service keys, or WSKeys, are the primary method **used for authenticating and authorizing interactions with web services** available on the OCLC WorldShare Platform. WSKeys **authenticate clients sending requests to web services**, in effect, **creating a "secure pipe" between a remote client and the data and functionality** available on the Platform.

http://oclc.org/developer/platform/authentication/what-wskey


<?xml version="1.0"?>
<gobs_of>
<structured>
<data />
</structured>
</gobs_of>

How do I get one?





Eligibility

SANDBOX KEYS

- Available to anyone
- Typically associated with a faux institution/library
- Promoted as widely as possible

PRODUCTION KEYS

- Member libraries with access to a corresponding OCLC system/ service
- Partners that sign a business arrangement with OCLC that promotes our member libraries or extends their services

Luxor - no hassle by effeietsanders
http://www.flickr.com/photos/9435171@N03/3442747532/

What are we securing?

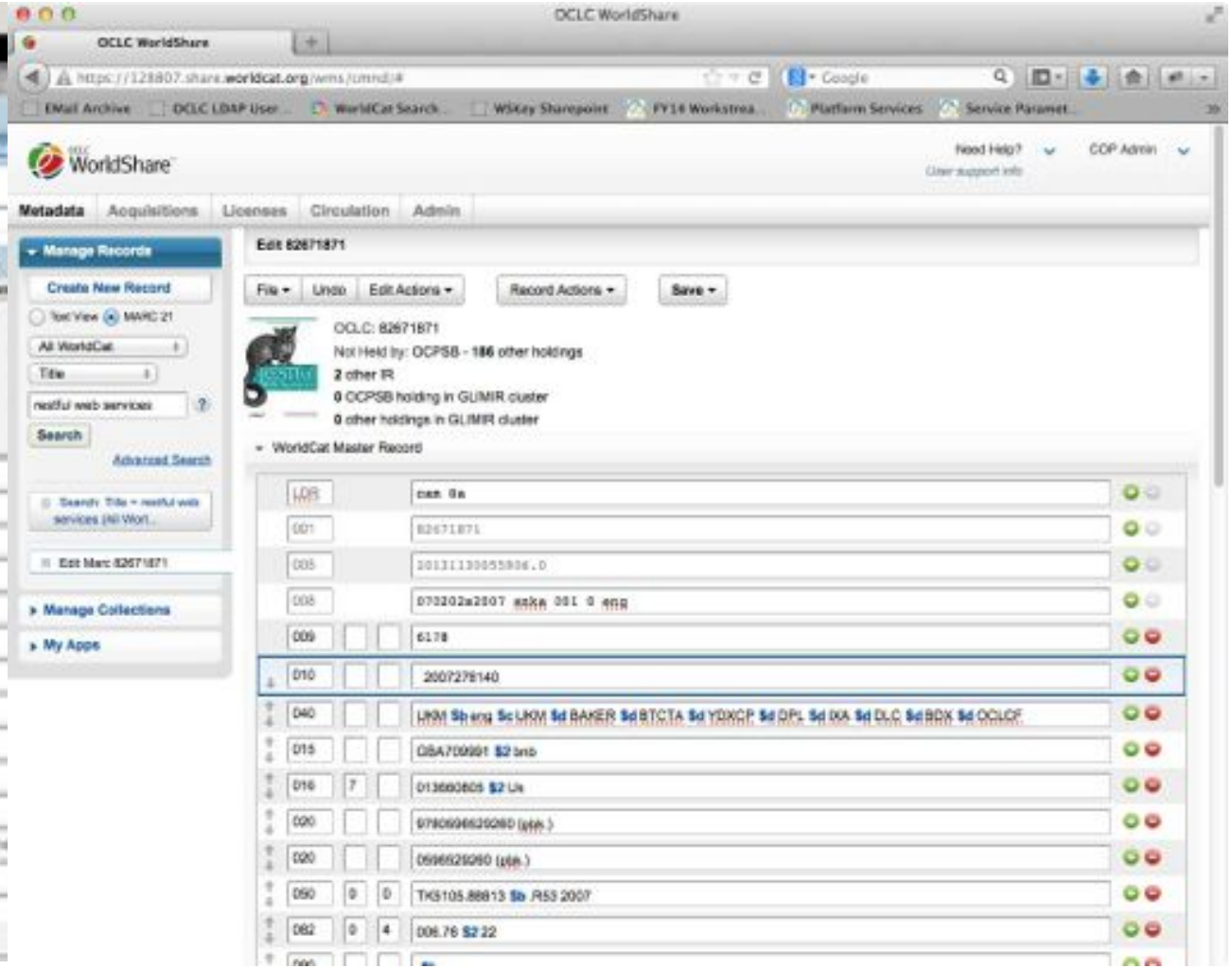
Protected data

- **Financial information**
- **Personally identifiable information**
- **Protected data**
  (e.g., governed by HIPPA or FERPA)
- **Business data**
  giving it away for free
  removes the funding that creates it

Protected activities

- **Borrow an item**
  from a peer institution as an ILL request
- **Place a hold**
  for an item so a patron can have it
  held for pickup
- **Set a holding**
  in WorldCat for an institution

HTTP 401: Authentication Required



As an agile user story: Authentication Required

As a cataloger

When I work in my library's ILS

I want to save records and set holdings in WorldCat

So that my WorldCat holdings are up to date in real time

API Key System Requirements

1. **Strong authentication:** ability to prove identity of a client

2. Optional ability to **attribute activity to a user**

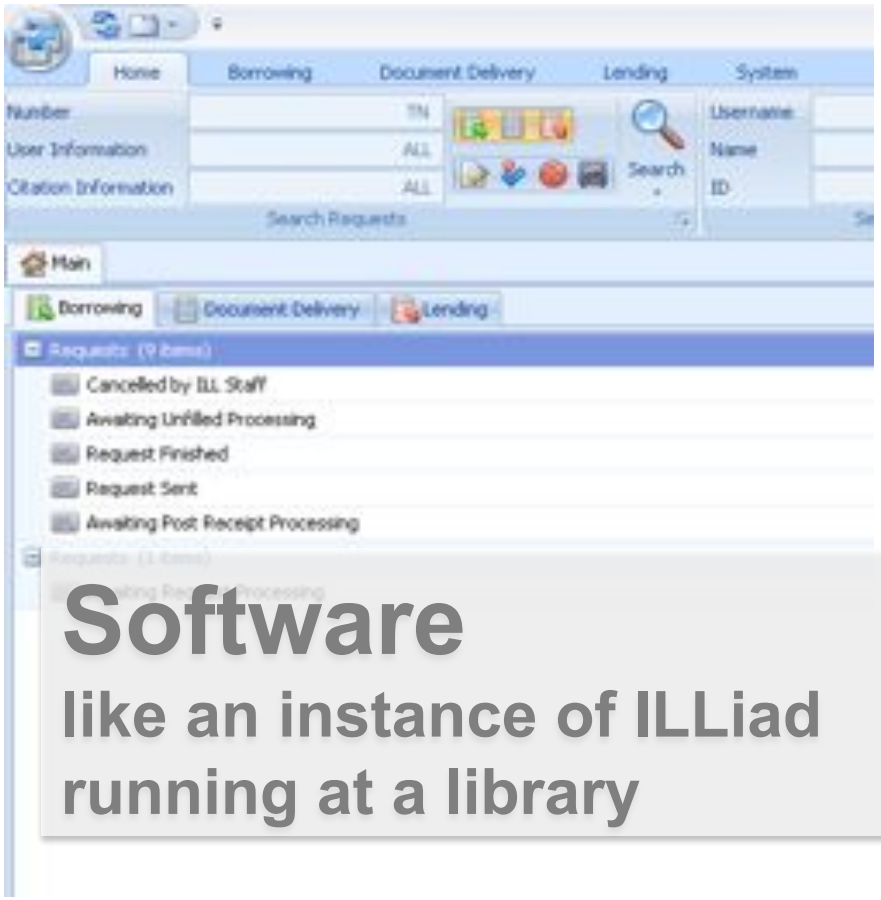
3. Enable configuration by clients so code can be shared among OCLC's Developer Network community

What is Strong Authentication?

In the context of WSKey & web services

a process in which a client's identity is verified

What Are We Authenticating?



Software

like an instance of ILLiad running at a library

```
#!/usr/bin/env ruby

$LOAD_PATH.push File.join(File.dirname(__FILE__), "/../../lib" )
require 'oclc'

require_relative '../../spec/spec_helper'

def create_payment(payment_token)
  payment = OCLC::IFM::Payment.new(payment_token)
  title = 'Logical Atomism'
  author = 'Russell'
  material_type = 'Book'
  amount = 0.05
  item = OCLC::IFM::Item.new(title, author, material_type, amount)
  payment.items << item
  payment.vendor_customer_id = '654321'
  payment.vendor_customer_id = '654321'
  payment.vendor_customer_id = '654321'
end

# Create the client WSKey
client_wskey = wskey('ifm_qaiul_client_production')
client_wskey.autho = CONFIG['ifm_qaiul_client_production']['autho']
client_wskey.principal_on_header = true

# Create the vendor WSKey
vendor_wskey = wskey('ifm_qaocl_vendor_production')

# Create the payer and payee
name = 'QAIUL PD OCLC, Inc. (5279) Test'
symbol = 'QAIUL'
ocpsb_payer = OCLC::IFM::Payer.new(symbol, name)

name = 'Online Computer Library Center (6569) TEST Change By Auto'
symbol = 'QAOCL'
ccc_payee = OCLC::IFM::Payee.new(symbol, name)

# Create the payment token as a client
payment_token = create_payment_token(100.00, ocpsb_payer, ccc_payee)
payment_token.collection_url = 'https://worldcat.org/ILL/paymentToken/data'
payment_token.debug_mode = true
payment_token.create(client_wskey)
payment_token.response_code.should == '201'

# Create the payment as a vendor
payment = create_payment(payment_token)
payment.debug_mode = true
payment.create(vendor_wskey)
payment.response_code.should == '201'
```

A script that performs a regular operation against an OCLC system

Managed-User login page

Managed-User login page

https://worldcat.authn.worldcat.org/login/mana

Google

Service Configuration Login

OCLC® Service Configuration

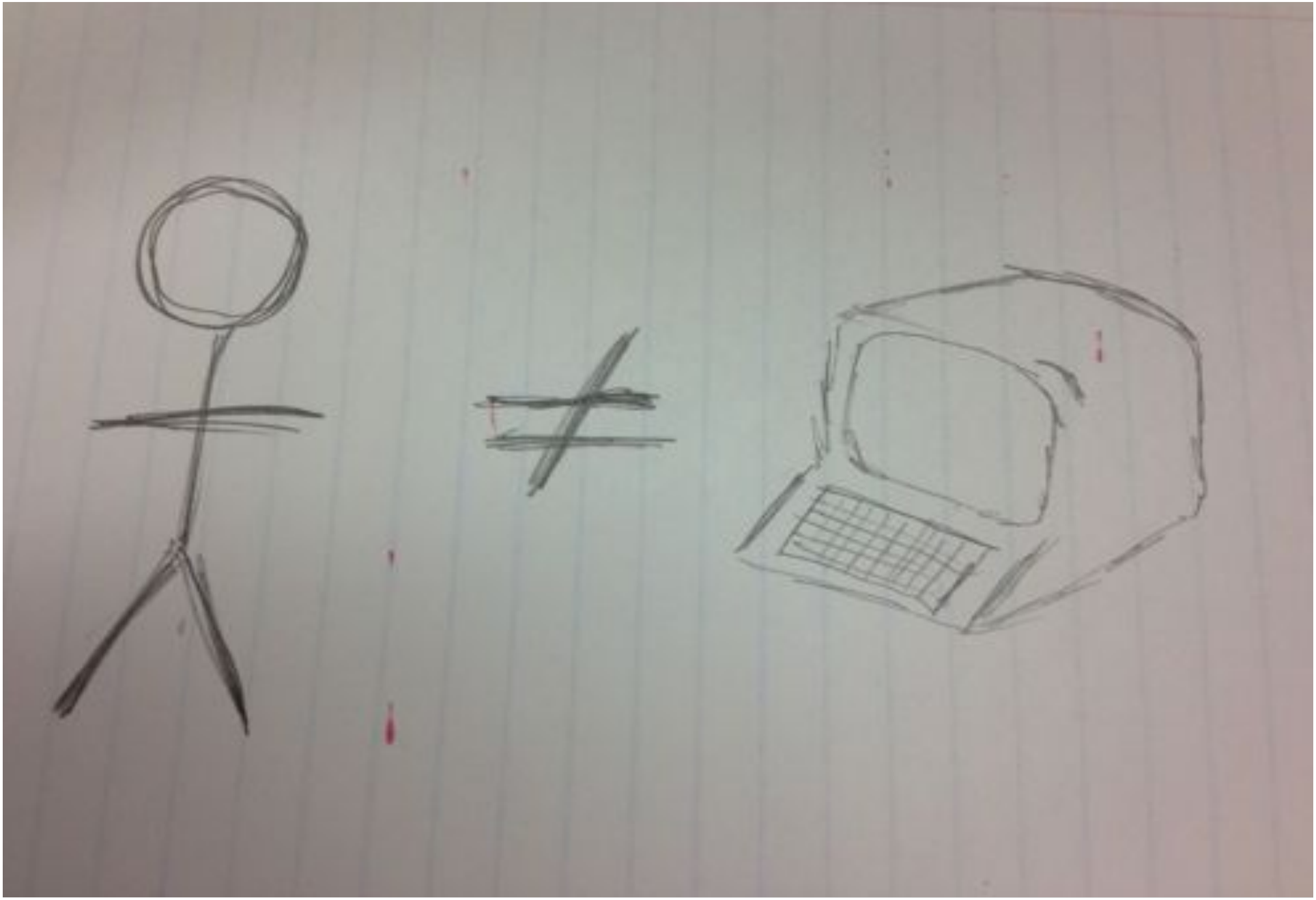
Sign In

User Name:

Password:

Sign In Cancel

What does it mean to **authenticate an application** rather than a human?



How does this work?

OAuth

An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.

http://oauth.net/

Also Used By

Google, Amazon, Facebook, Twitter, Salesforce.com …

Digital Signatures

OAuth uses digital signatures instead of sending the full credentials (specifically, passwords) with each request. Similar to the way people sign documents to indicate their agreement with a specific text, **digital signatures allow the recipient to verify that the content of the request hasn't changed in transit**. To do that, the sender uses a mathematical algorithm to calculate the signature of the request and includes it with the request.

http://hueniverse.com/oauth/guide/security/

Illiad has an OCLC WSKey with a **key** and a **secret** and is going to **POST** some data to the URL

https://worldcat.org/ILL/request/data/001?**inst**=128807&**format**=XML

1) Format a message

10rMoavKf3WTJ4Lk72wN5xvYmSCjBrWkCscR2euXBfa7ch
1370271657
34091660664936857385654714002 4

POST
www.oclc.org
443
/wskey
format=XML
inst=128807

2) Use an encryption algorithm to sign the message

```
digest = HMAC-SHA256 ( wskey_secret, formatted_message )
signature = base64 ( digest )
```

hK4vHMAIzdv4mCI9sR8c1h
dD7kHGgUj2rQ9eOMWvI7c=

URL: https://worldcat.org/ILL/request/data/001?inst=128807&format=XML

Data:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom">
    <id> http://worldcat.org/ILL/request/data/001 </id>
    <link href="http://worldcat.org/ILL/request/data/001" rel="self"/>
    <content type="application/xml">
        <ill-request id="001">
            <borrower>OCPSB</borrower>
            <lender>GZM</lender>
            <oclcNumber>30780581</oclcNumber>
        </ill-request>
    </content>
</entry>
```

Key: 10rMoavKf3WTJ4Lk72wN5xvYmSCjBrWkCscR2euXBfa7ch

Signature: hK4vHMAIzdv4mCI9sR8c1hdD7kHGgUj2rQ9eOMWvI7c=

Note: WSKey secret is not sent

ILLiad

OCLC
web service

Generates & Sends

`hK4vHMAIzdv4mCI9sR8c1hdD7kHGgUj2rQ9eOMWvI7c=`

Receives & Generates

`hK4vHMAIzdv4mCI9sR8c1hdD7kHGgUj2rQ9eOMWvI7c=`

✓

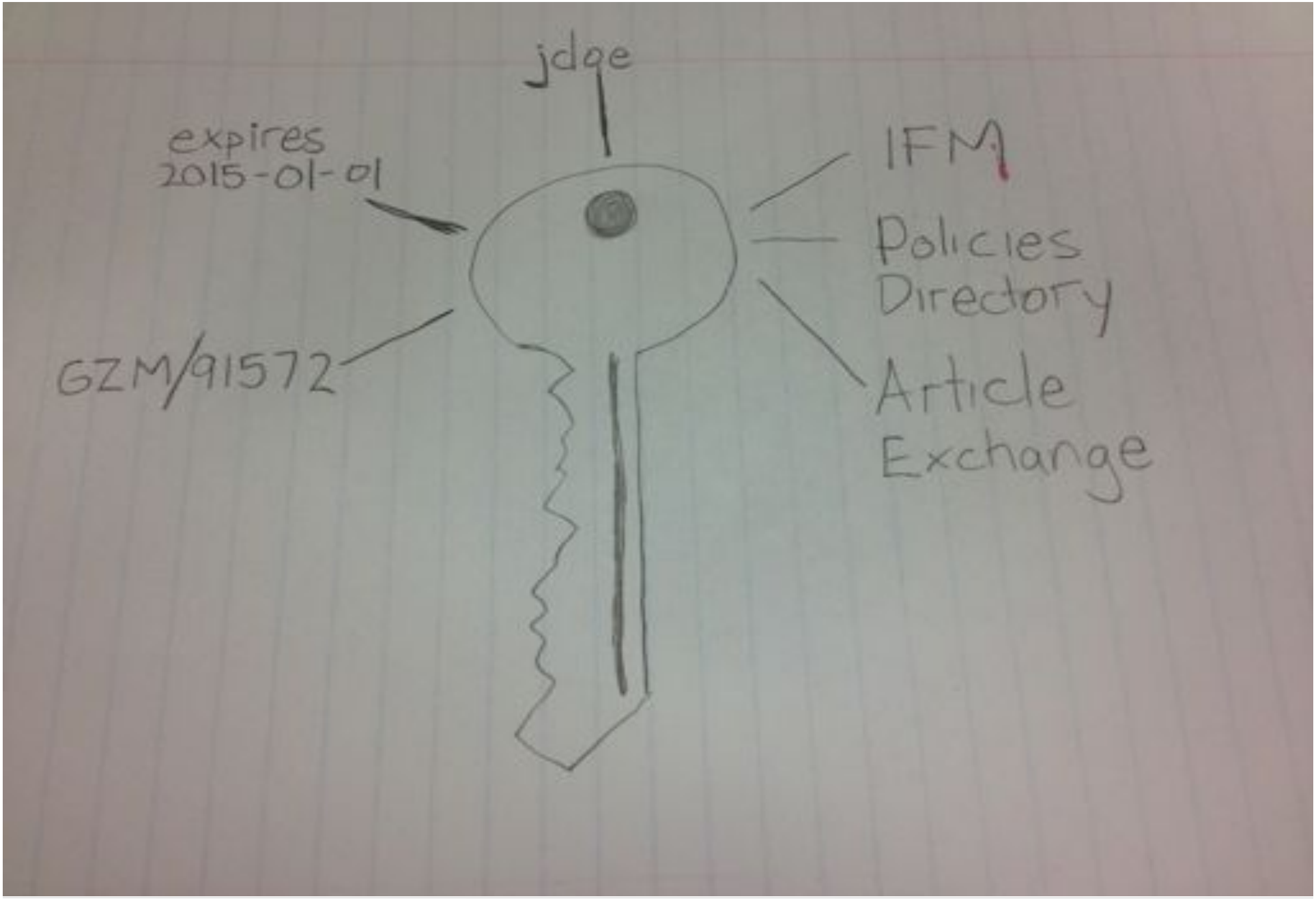
Match

I know this is ILLiad running at the University of Wisconsin-Madison

≠

No Match

I cannot verify the identity of a known client


jdoe
expires
2015-01-01
IFM
Policies
Directory
GZM/91572
Article
Exchange

OCLC-Developer-Network/oclc-auth-ruby

OCLC-Developer-Network/oclc...

GitHub, Inc. (US) https://github.com/OCLC-Developer-Network/oclc-auth-ruby

This repository | Search or type a command | Explore | Gist | Blog | Help | oclc-platform-dev

OCLC-Developer-Network / oclc-auth-ruby

Unwatch 3 | Star 0 | Fork 0

OCLC Auth Gem — Edit

9 commits | 1 branch | 0 releases | 1 contributor

branch: master | oclc-auth-ruby /

Correcting links in README

meyerst authored 37 minutes ago | latest commit 74c67718d8

lib	Modifying the example docs to use sandbox institution registry ID	44 minutes ago
spec	Adding Apache 2 license to files	an hour ago
Gemfile	Removing unnecessary comment	an hour ago
LICENSE.txt	Adding Apache 2 license to files	an hour ago
README.md	Correcting links in README	37 minutes ago
Rakefile	Adding files for first time	14 days ago
oclc-auth.gemspec	Adding files for first time	14 days ago

README.md

OCLC::Auth

Code

Issues

Pull Requests

Wiki

Pulse

Graphs

Network

Settings

HTTPS clone URL

https://github.com

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

Setup: configuration file

```
# config.yml
key: 'my-api-key-for-worldcat-metadata-api'
secret: 'api-key-secret'
principal_id: 'principal-id-for-user'
principal_idns: 'namespace-for-principal-id'
```

Script: read a bib resource in the WorldCat Metadata API

read_bib.rb — oclc-auth-metadata-api

config.yml | read_bib.rb

```ruby
require 'yaml'
require 'net/http'
require 'oclc/auth'

# Read the configuration file and construct the WSKey object
CONFIG = YAML::load(File.read("config.yml"))
wskey = OCLC::Auth::WSKey.new(CONFIG['key'], CONFIG['secret'])

# Prepare the URL for the web service
url = 'https://worldcat.org/bib/data/823520553?' +
    'classificationScheme=LibraryOfCongress&holdingLibraryCode=MAIN'
uri = URI.parse(url)

# Construct the HTTP request object and configure the request
# by setting the Authorization header with an HMAC signature
request = Net::HTTP::Get.new(uri.request_uri)
request['Authorization'] = wskey.hmac_signature('GET', url,
    :principal_id => CONFIG['principal_id'],
    :principal_idns => CONFIG['principal_idns'])

# Execute the HTTP request and save the response
http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true
response = http.start do |http|
  http.request(request)
end

# Print the XML from the HTTP response
puts response.body
```

Line: 29 Column: 19 | Ruby on Rails | Soft Tabs: 2

OCLC is providing this code for demonstration purposes and on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using the code and assume any associated risks.

Script: read a bib resource in the WorldCat Metadata API

```
1  require 'yaml'
2  require 'net/http'
3  require 'oclc/auth'
4
5  # Read the configuration file and construct the WSKey object
6  CONFIG = YAML::load(File.read("config.yml"))
7  wskey = OCLC::Auth::WSKey.new(CONFIG['key'], CONFIG['secret'])
8
9  # Prepare the URL for the web service
10 url = 'https://worldcat.org/bib/data/823520553?' +
11     'classificationScheme=LibraryOfCongress&holdingLibraryCode=MAIN'
12 uri = URI.parse(url)
13
```

Script: read a bib resource in the WorldCat Metadata API

```
13
14 # Construct the HTTP request object and configure the request
15 # by setting the Authorization header with an HMAC signature
16 request = Net::HTTP::Get.new(uri.request_uri)
17 request['Authorization'] = wskey.hmac_signature('GET', url,
18     :principal_id => CONFIG['principal_id'],
19     :principal_idns => CONFIG['principal_idns'],
20     :debug => true)
21
```

Script: read a bib resource in the WorldCat Metadata API

```
21
22 # Execute the HTTP request and save the response
23 http = Net::HTTP.new(uri.host, uri.port)
24 http.use_ssl = true
25 response = http.start do |http|
26   http.request(request)
27 end
28
29 # Print the XML from the HTTP response
30 puts response.body
```

Authorization Header

```
http://www.worldcat.org/wskey/v2/hmac/v1
clientId="dz4ES5H4qHHOs713zqL4AnmqFukywjcpT",
timestamp="1388070167",
nonce="8234471099802494338387135495 41",
signature="EF8s9oq0j8Uzrh0CxM9jaRLljzE0rESZQ=",
principalID="201dd-b197-42e1-bd36-9fea404ad",
principalIDNS="urn:oclc:wms:da"
```

HMAC signature slightly modified for readability: new lines added before each key/value pair and the clientId, signature and principalID were shortened/scrubbed.

OCLC::Auth::WSKey#hmac_signature

```
101     # Generates a digital signature for a given request according to the OAuth HMAC specification
102     #
103     # [http_method] the HTTP method, GET, POST, PUT, DELETE
104     # [url] the URL the request will be made to
105     # [options] a hash of optional parameters described below
106     #
107     # Options
108     #
109     # [:principal_id] the ID that represents a user
110     # [:principal_idns] the ID namespace context for the user
111     def hmac_signature(http_method, url, options = {})
112       req_timestamp = timestamp
113       req_nonce = nonce
114       signature_base = signature_base_string(req_timestamp, req_nonce, http_method, url)
115
116       auth  = ""
117       auth += "#{scheme_url} "
118       auth += "clientId=\"#{client_id}\", "
119       auth += "timestamp=\"#{req_timestamp}\", "
120       auth += "nonce=\"#{req_nonce}\", "
121       auth += "signature=\"#{signature(signature_base)}\""
122
123       if options[:principal_id] and options[:principal_idns]
124         uri = URI.parse(url)
125         if uri.query
126           params = CGI::parse(uri.query)
127           unless params.has_key?("principalID")
128             auth += ", principalID=\"#{options[:principal_id]}\", principalIDNS=\"#{options[:principal_idns]}\""
129           end
130         end
131       end
132       auth
133     end
```

OCLC::Auth::WSKey#hmac_signature

PATTERN	EXAMPLE
{API Key}\n	10rMoavKf3WTJ4Lk72wNvYmSCjB\n
{Unix Timestamp}\n	1370271657\n
{nonce}\n	apo3r7md7fjyn\n
{request body hash}\n	\n
{request HTTP method}\n	POST\n
www.oclc.org\n	www.oclc.org\n
443\n	443\n
/wskey\n	/wskey\n
{request URL param(s)}\n	format=XML\n
	inst=128807\n

OCLC::Auth::WSKey#hmac_signature

```
def signature( base_string )
  digest = OpenSSL::Digest::Digest.new( 'sha256' )
  hmac = OpenSSL::HMAC.digest( digest, @secret, base_string  )
  Base64.encode64( hmac ).chomp.gsub( /\n/, '' )
end
```

Signature = Strong Authentication

1. Produced a message describing the client/ sender and the request it wants to make

2. Encrypted the message in a way only the client and OCLC can verify the message is authentic

Signature = Strong Authentication

```
require 'oclc/auth'

wskey = OCLC::Auth::WSKey.new('key', 'secret')

wskey.hmac_signature('GET', url,
    :principal_id   => 'principal-id',
    :principal_idns => 'principal-idns')
```

OAuth 2

Authorization Server
&
Access Tokens

Access Tokens

```
Authorization: Bearer tk_HYKExLu0ByQ2Co0QjPq
```

Access Tokens

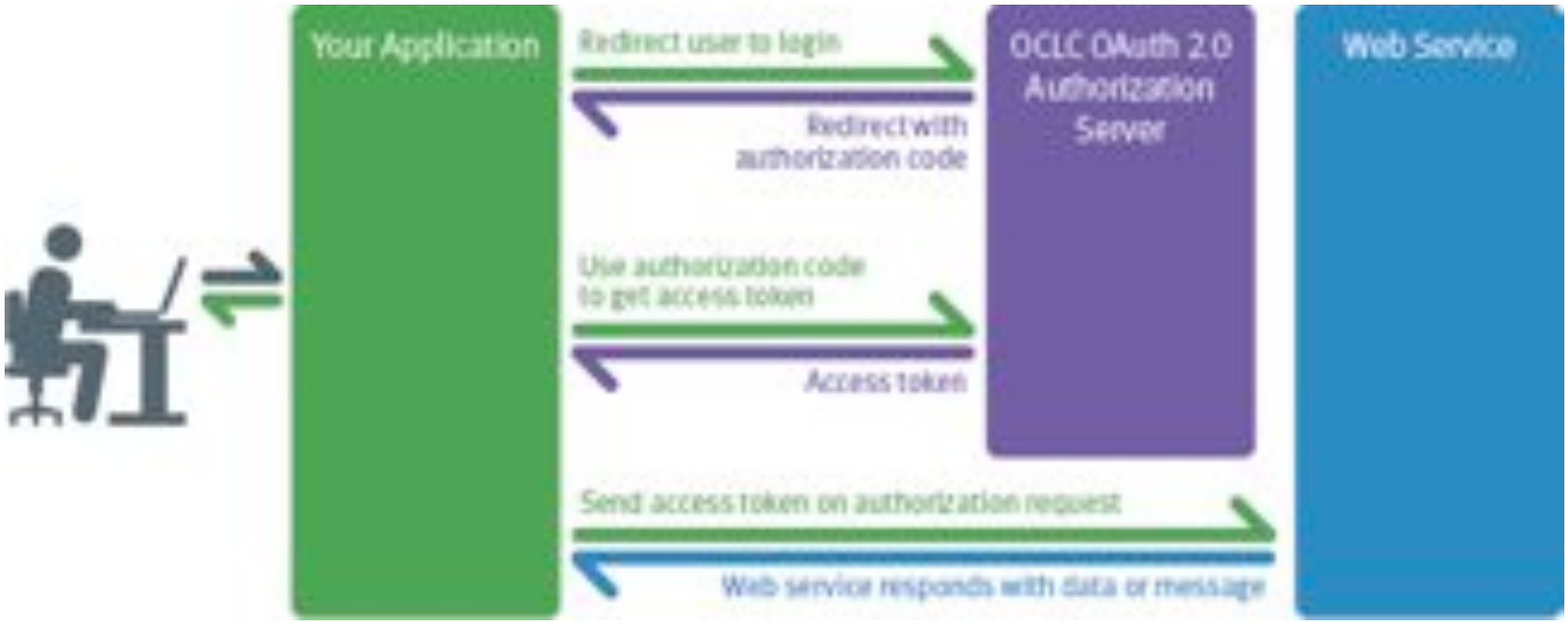
1. Are obtained by a client with a WSKey

2. Are limited to specific OAuth “scope(s)”
   for OCLC a scope = a web service

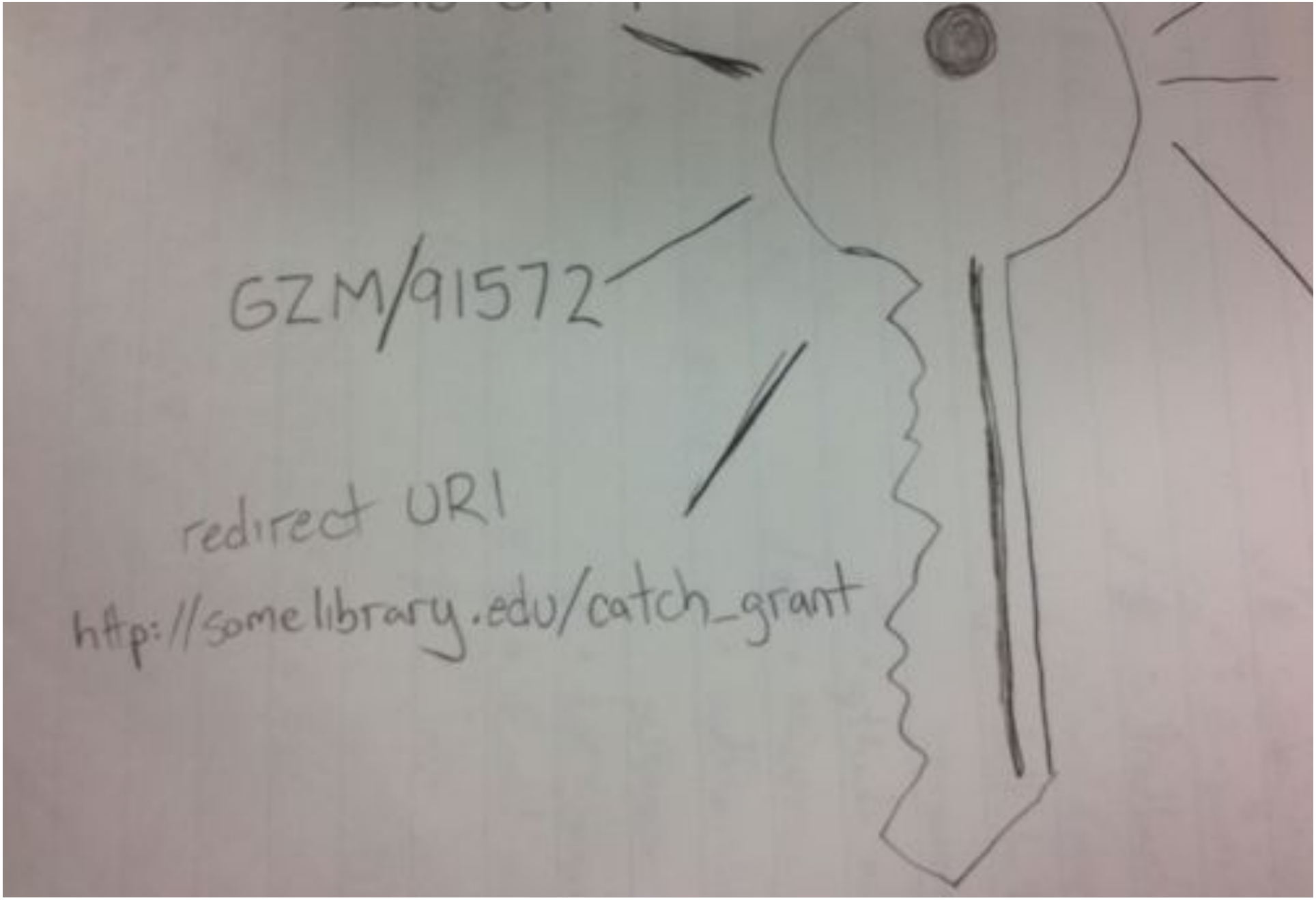
3. Are time sensitive
   typically will expire after 20 minutes

4. Are *granted* to the client

Explicit Authorization Code



http://www.oclc.org/developer/platform/explicit-authorization-code


GZM/91572
redirect URI
http://somelibrary.edu/catch_grant

Explicit Auth Use Cases

WorldShare Management System

- Integrating campus portal with patron account info
- Integrating acquisitions data with financial/billing system

Explicit Authorization Code: Reference Example

https://github.com/OCLC-Developer-Network/oclc-auth-ruby#example-sinatra-classic-style-app-protected-by-an-oauth-2-explicit-authorization-login

Steve Meyer

meyerst@oclc.org

Questions?

Resources

- Authentication and Authorization documentation
  - http://oclc.org/developer/platform/authentication-documentation
- How to request a WSKey
  - http://oclc.org/developer/platform/authentication/how-request-wskey
- Assistance
  - DevNet@oclc.org

Get Started

- Sign up for a WorldCat account
  - http://worldcat.org
- Request a WSKey in OCLC Service Configuration
  - https://worldcat.org/config
- Join the OCLC Developer Network and listserv
  - www.oclc.org/developer/register
  - http://oc.lc/subscribewc-devnet-l

This Workshop Brought to You By

http://www.oclc.org/developer