TITANIUM SPONSORS

epiQ

PAIGE
TECHNOLOGIES
INTELLIGENT PAIRING. PERPETUAL SUCCESS.

ADAPTIVE™
SOLUTIONS GROUP

okta

Platinum Sponsors

VinSolutions
Make every connection count.

jack henry
& ASSOCIATES INC.®

VALOREM

NAIC
National Association of Insurance Commissioners

NIPR
NATIONAL INSURANCE
PRODUCER REGISTRY

TEKsystems
Our people make IT possible.

KEYHOLE
SOFTWARE

smartsheet
Work Better™

Couchbase

EquipmentShare

KOCH™

PLURALSIGHT

Google Cloud

VML

mylo™
POWERED
BY LOCKTON

Gold Sponsors

Advantage Tech
IT Staffing & Recruiting Services

PeopleAdmin

CENTRIQ
TRAINING

Cerner

KU
EDWARDS
CAMPUS
The University of Kansas

twilio

{Track:js}
JAVASCRIPT ERROR MONITORING

OBJECT
PARTNERS

smg
service
management
group®

stackify

GARMIN

Balance
Innovations

Allied Global
SERVICES

dsi

LRS
CONSULTING SERVICES

Aviture
ingenuity. engineered.

TouchNet
+ Heartland

Mediware®

An affiliate of CSA Industries, Inc.
AUREUS
GROUP
Finance | Systems | Executive

Children's Mercy
KANSAS CITY

Commerce Bank

SECURITY PS
STRATEGIC INFORMATION SECURITY

ProKarma

bdemers
BrianDemers
shiro.apache.org
ApacheShiro

APACHE SHIRO
Simple. Java. Security.

Brian Demers
Apache Shiro PMC
bdemers@apache.org

okta

Who is this guy?

What is Apache Shiro ?

- Application Security Framework
- Quick and Easy
- Simplifies Security Concepts

APACHE SHIRO SECURITY

“Talk is cheap. Show me the code.”

–Linus Torvalds

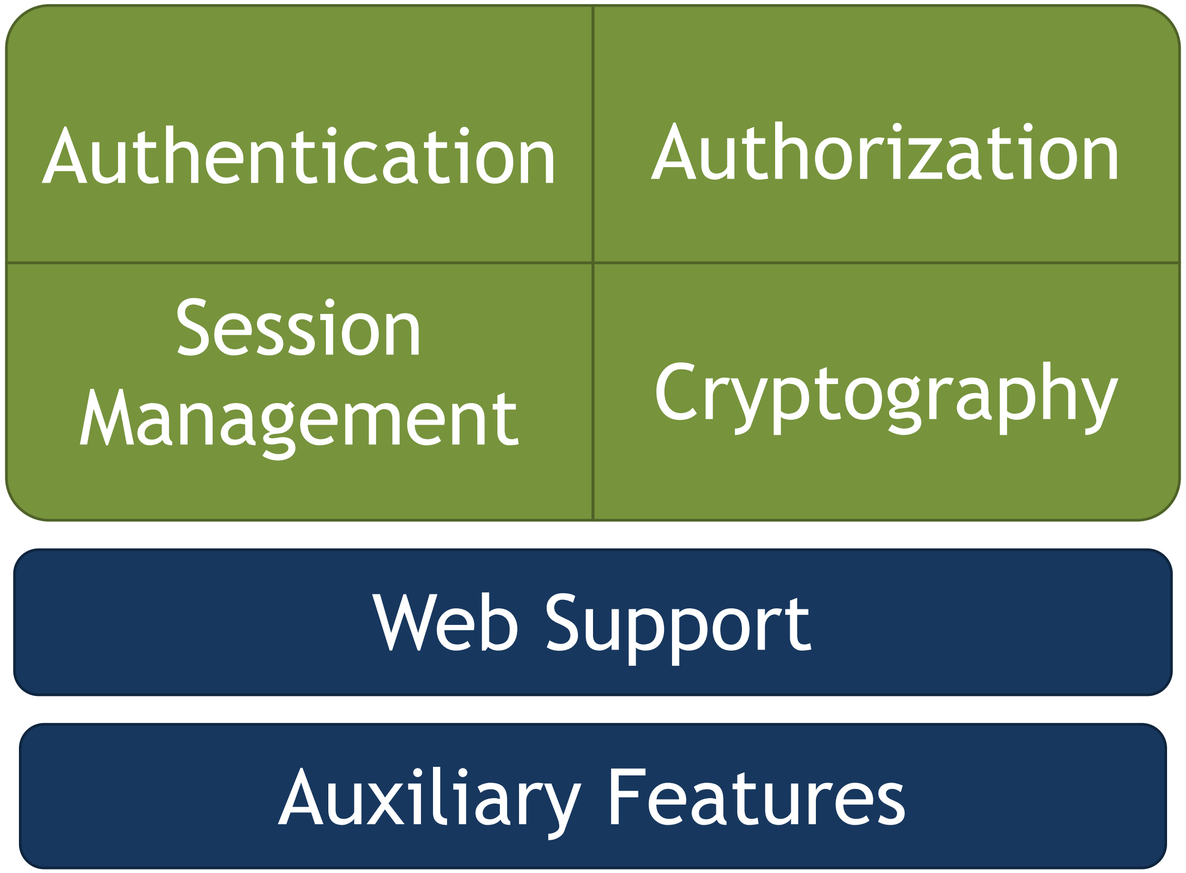
JAX-RS via Annotations

```
@GET
@Path("/{id}")
@RequiresPermissions("troopers:read")
public Stormtrooper getTrooper(@PathParam("id") String id) {

    Stormtrooper stormtrooper = trooperDao.getStormtrooper(id);
    if (stormtrooper == null) {
        throw new NotFoundException();
    }
    return stormtrooper;
}

@POST
@Path("/{id}")
@RequiresPermissions("troopers:update")
public Stormtrooper updateTrooper(@PathParam("id") String id,
                                  Stormtrooper updatedTrooper) {

    return trooperDao.updateStormtrooper(id, updatedTrooper);
}
```

Spring Controller via Annotations

```
@GetMapping(path = "/{id}")
@RequiresPermissions("troopers:read")
public Stormtrooper getTrooper(@PathVariable("id") String id) {

    Stormtrooper stormtrooper = trooperDao.getStormtrooper(id);
    if (stormtrooper == null) {
        throw new NotFoundException();
    }
    return stormtrooper;
}


@PostMapping(path = "/{id}")
@RequiresPermissions("troopers:update")
public Stormtrooper updateTrooper(@PathVariable("id") String id,
                                  @RequestBody Stormtrooper updatedTrooper) {

    return trooperDao.updateStormtrooper(id, updatedTrooper);
}
```

Four Cornerstones



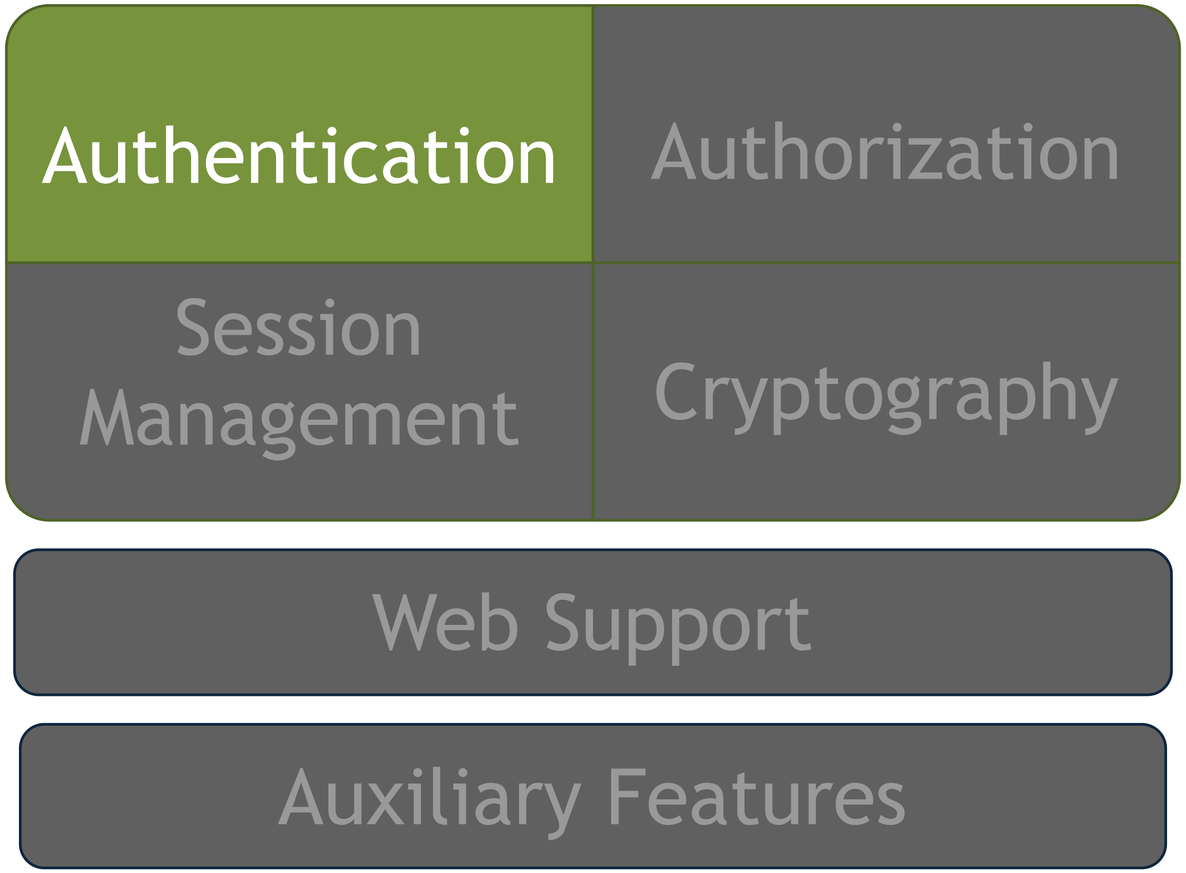
Terminology

- Subject - An Account - A security-specific user ‘view’
- Principals - Subjects identifying attributes
- Credentials - Secret values that verify identity
- Realm - Security-specific DAO

Example Realm

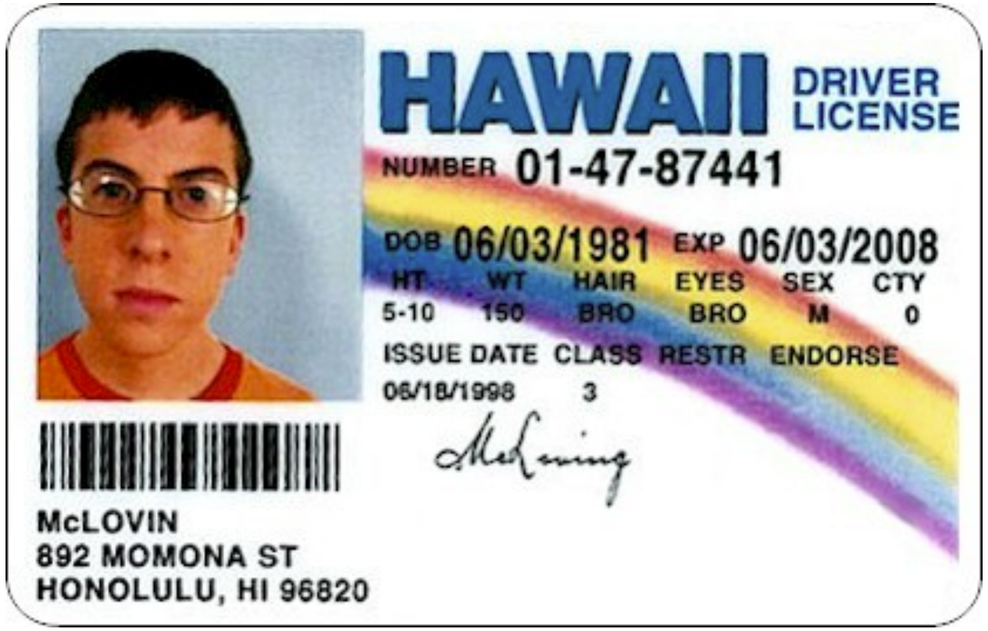
```
protected AuthenticationInfo doGetAuthenticationInfo(
                                  AuthenticationToken token)
                                      throws AuthenticationException {
    // Validate token, build and return AuthenticationInfo
    // Returning null or throwing exception are invalid login attempts
}

protected AuthorizationInfo doGetAuthorizationInfo(
                                 PrincipalCollection principals) {
    // Look up user by principal
    // return AuthorizationInfo
    SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
    info.addRole("admin");
    info.addStringPermissions("trooper:read");

    return info; // null is also valid
}
```

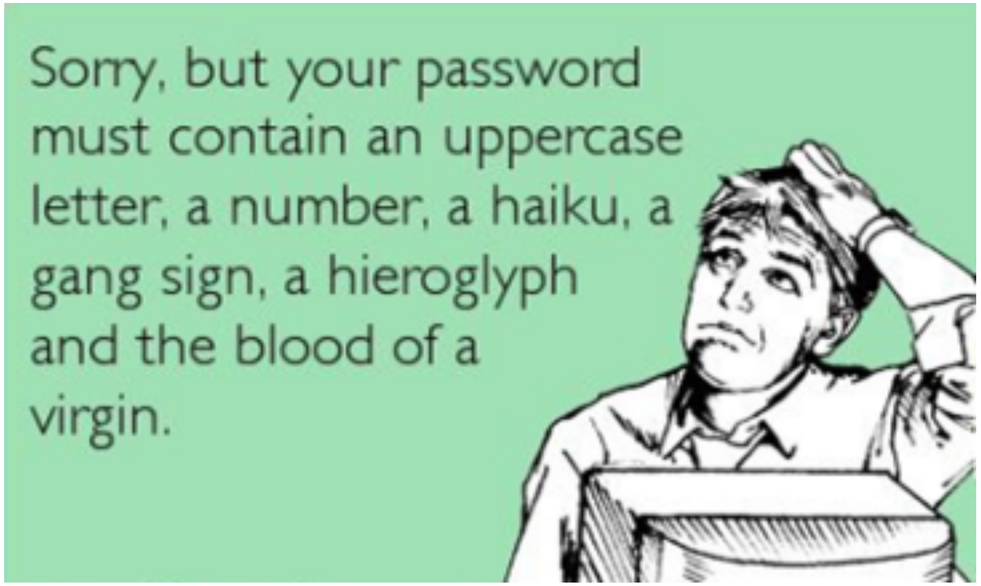
Authentication



Authentication == Identity Verification

Principals: User ID, Email Address, etc





Credentials (Secret data): Password, Certificate, etc

APACHE SHIRO SECURITY

Shiro Authentication Features

- Subject-based (current user)
- Single method calls
- Rich Exception Hierarchy
- “Remember Me” built in
- Event Listeners / Bus

How to Authenticate with Shiro

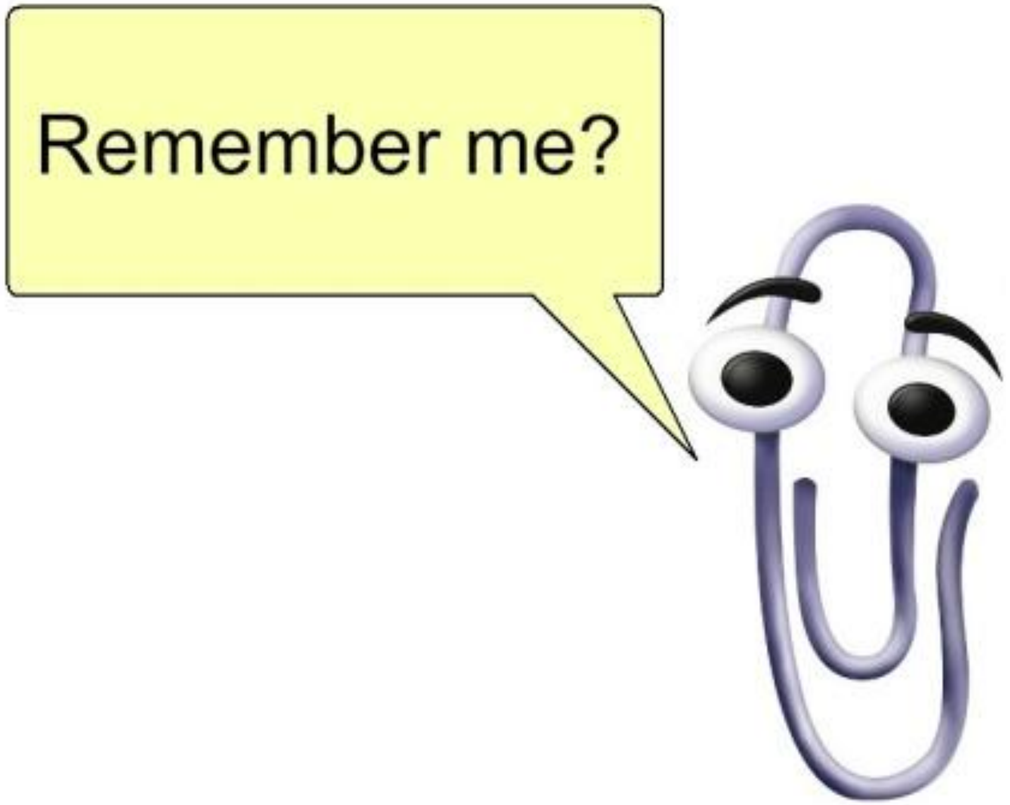
1. Collect principals & credentials
2. Submit to Authentication System
3. Allow, retry, or block access

APACHE SHIRO SECURITY

Step 1: Collect Principals & Credentials

```
UsernamePasswordToken token =
    new UsernamePasswordToken(username, password);

//"Remember Me" built-in:
token.setRememberMe(true);
```

Remember Me != Authenticated



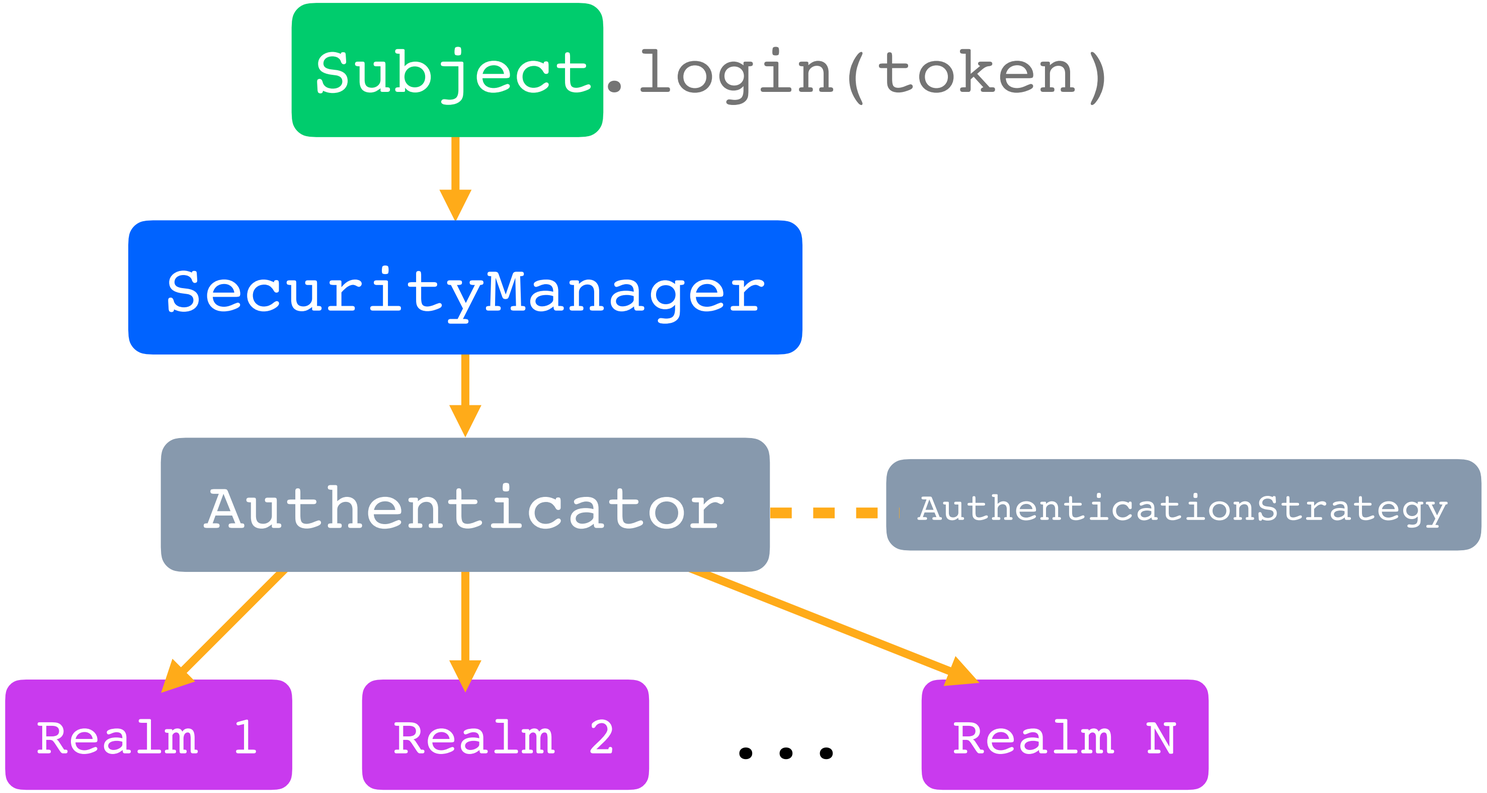
Step 2: Submission

```
// get current Thread's Subject
Subject currentUser = SecurityUtils.getSubject();

currentUser.login(token);
```

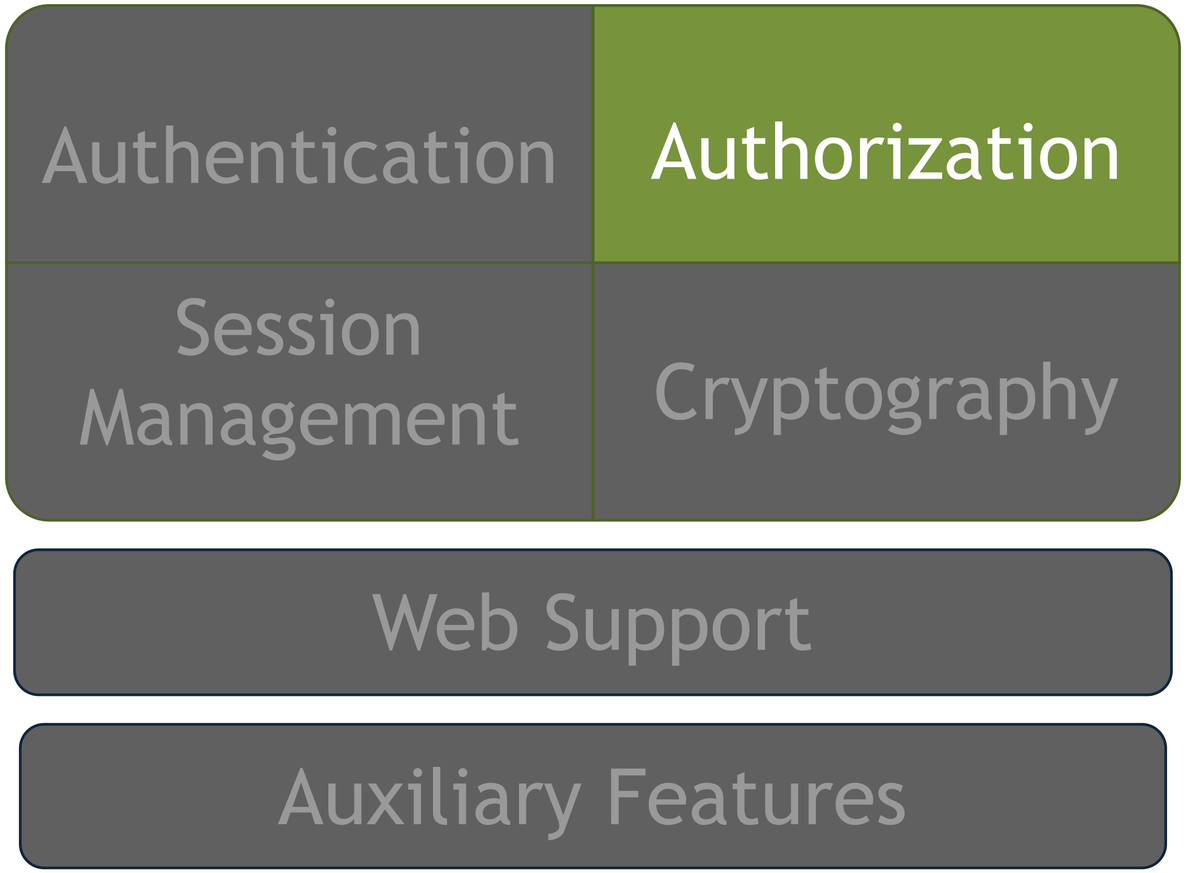
Step 3: Grant Access or Handle Failure

```
try {
    currentUser.login(token);
}
catch (UnknownAccountException uae ){ ... }
catch (IncorrectCredentialsException ice { ... }
catch (LockedAccountException lae ) { ... }
catch (ExcessiveAttemptsException eae ) { ... }
catch ( ... your own ) { ... }
catch (AuthenticationException ae ) {
    //unexpected error?
}
//No problems, show authenticated view...
```

How does it work ?



Authorization



Authorization Defined

Access Control - “Who can do what”

- Users - “Who”
- Permissions - “What”
- Roles - “What”/“logical group”

Authorization Features

- Subject-centric (current user)
- Checks based on roles or permissions
- Any data model - Realms decide
- Powerful out-of-the-box **WildcardPermission**

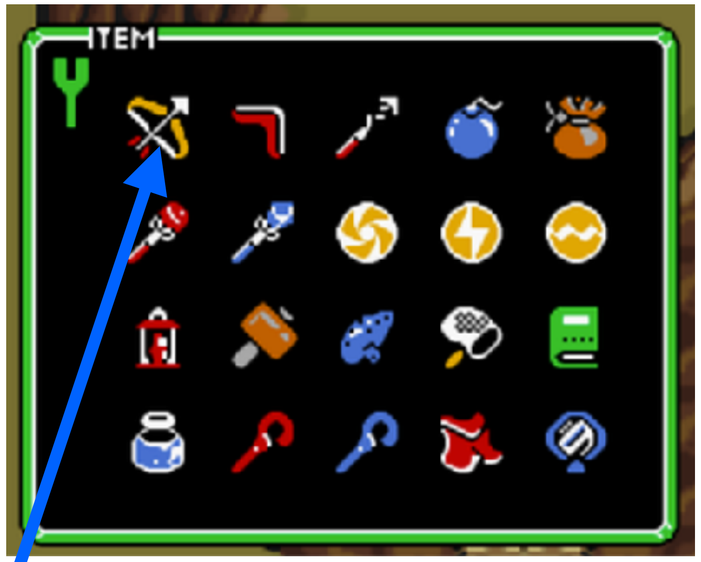
Permissions

- Most atomic security element
- Describes resource types and behavior
- The “what” of an application
- **NOT** the “who”
- AKA “rights”

Permissions -> User/Role

Resource

Fully Qualified Permission String



`zelda:bow:equip`

User/Role

Any Permission String

`zelda:*:equip`

Wildcard Permissions

Multiple Parts (no limit)

- domain:action
- domain:topic:action

- domain:*
- *:action
- domain:action1,action2

Implies

- domain:topic ⇒ domain:topic:*
- *:* ⇒ *

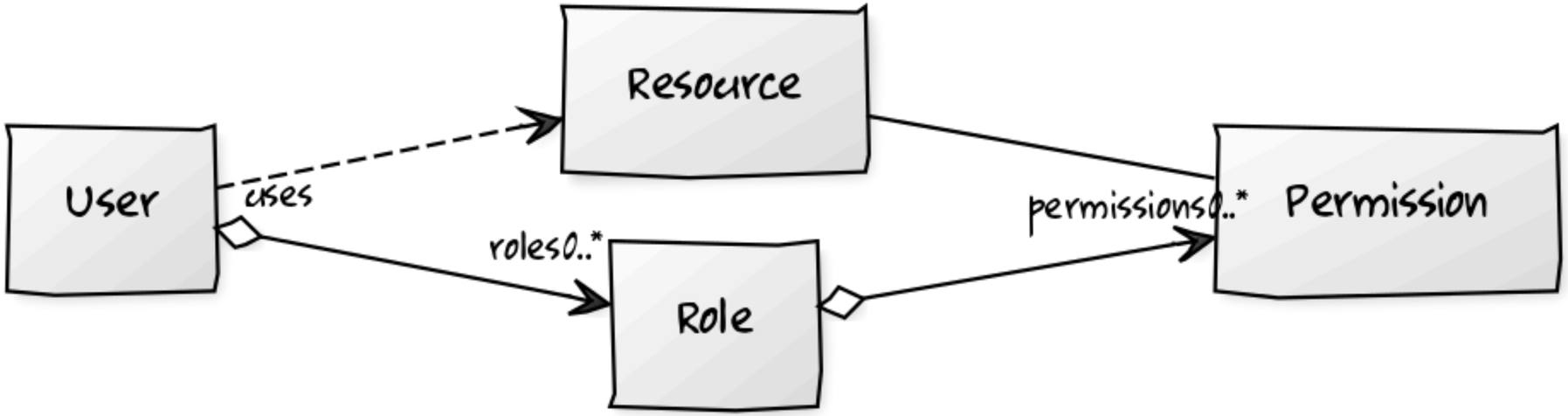
- domain:action **!=** domain:*:action

Wildcard Permissions (cont.)

- trooper:create,read,update,delete
- trooper:*

- deathstar:*
- deathstar:*:read
- deathstar:trooper:read
- deathstar:trooper:read:TK-421
- deathstar:tiefighter:read

Users have Roles
Roles have Permissions



Resources are associated with Permissions

How to Authorize with Shiro

Multiple ways of checking access control:

- Programmatically
- Annotations && AOP
- JSP/GSP TabLibs
- Community support for JSF & Thymeleaf

APACHE SHIRO SECURITY

Programmatic Authorization (Role Check)

```
//get the current Subject
Subject currentUser = SecurityUtils.getSubject();

if (currentUser.hasRole("administrator")) {
    //do one thing (show a special button?)
} else {
    //don't show the button?
}
```

Programmatic Authorization (Permission Check)

```
Subject currentUser = SecurityUtils.getSubject();

Permission deleteWidget =
    new WidgetPermission("foobar","delete");

if (currentUser.isPermitted(deleteWidget)) {
    //show the 'delete widget' button
} else {
    //don't show the button?
}
```

Programmatic Authorization (String Check)

```
String perm = "widget:delete:foobar";

if (currentUser.isPermitted(perm)){
    //show the 'delete widget' button
} else {
    //don't show the button?
}
```

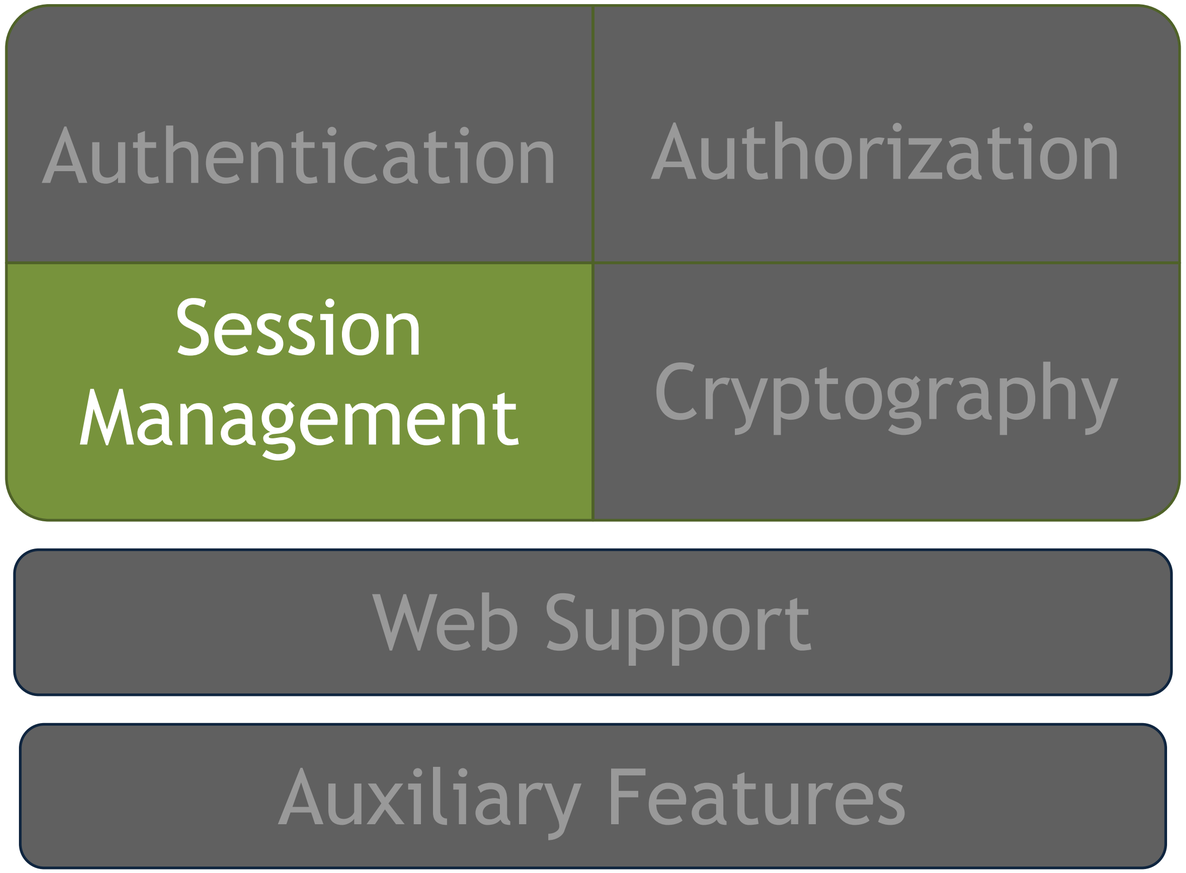
Annotation Authorization (Role Check)

```
@RequiresRoles( "teller" )
public void openAccount(Account a) {
    //do something in here that
    //only a 'teller' should do
}
```

Annotation Authorization (Permission Check)

```
@RequiresPermissions( “account:create” )
public void openAccount(Account a) {
    //create the account
}
```

Session Management



Session Management Defined

- Managing the lifecycle of Subject-Specific temporal data context

Session Management Features

- Heterogeneous client access
- POJO based (IoC friendly)
- Event Listeners
- Host address retention
- Inactivity / expiration support: **`touch()`**
- Transparent web use - HttpSession
- Container-Independent Clustering

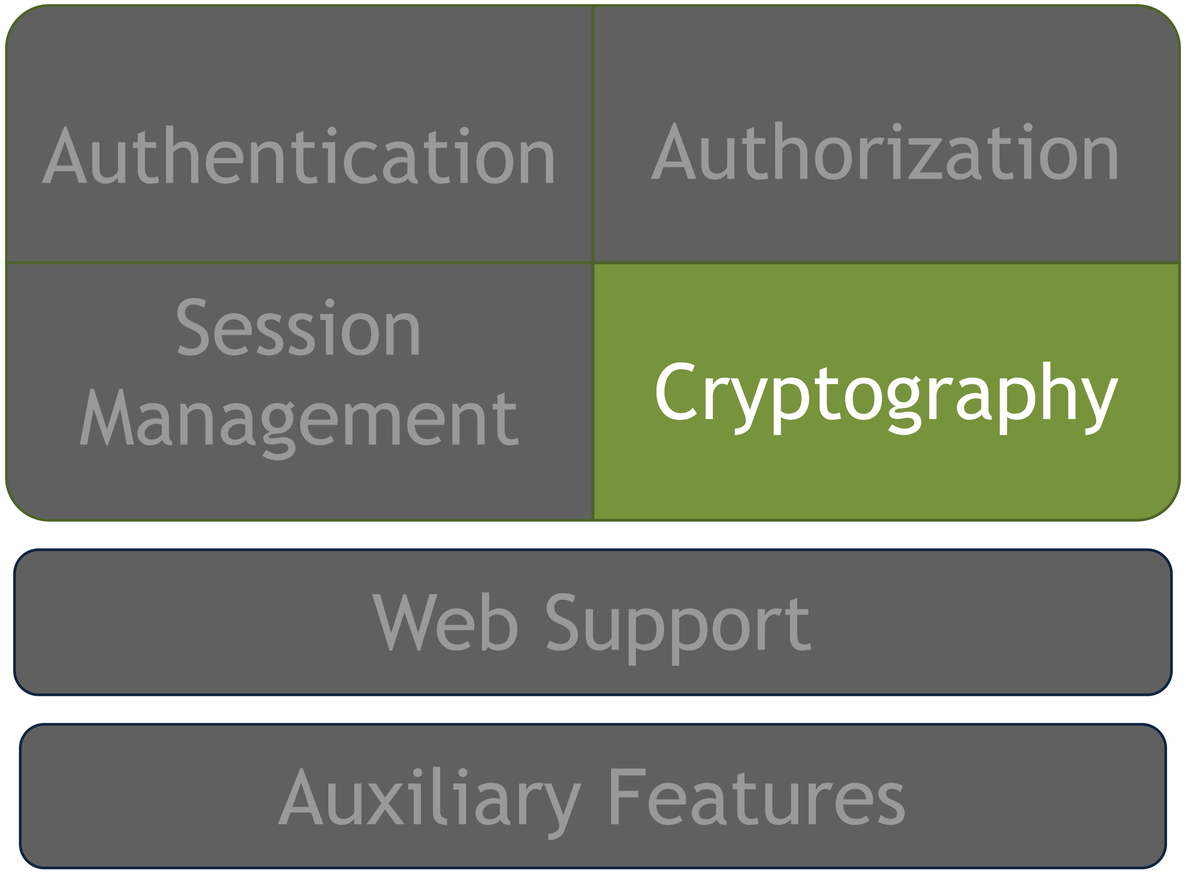
Session API

```
getStartTimestamp()
getLastAccessTime()
getAttribute(key)
setAttribute(key, value)
getTimeout()
setTimeout(long)
touch()
...
```

Acquiring and Creating Sessions

```
Subject currentUser = SecurityUtils.getSubject()

//guarantee a session
Session session = subject.getSession();

//get a session if it exists
subject.getSession(false);
```

Cryptography



Cryptography Defined

Protecting information from undesired access by hiding it or converting it nonsense.

Elements of Cryptography

- Ciphers
- Hashes

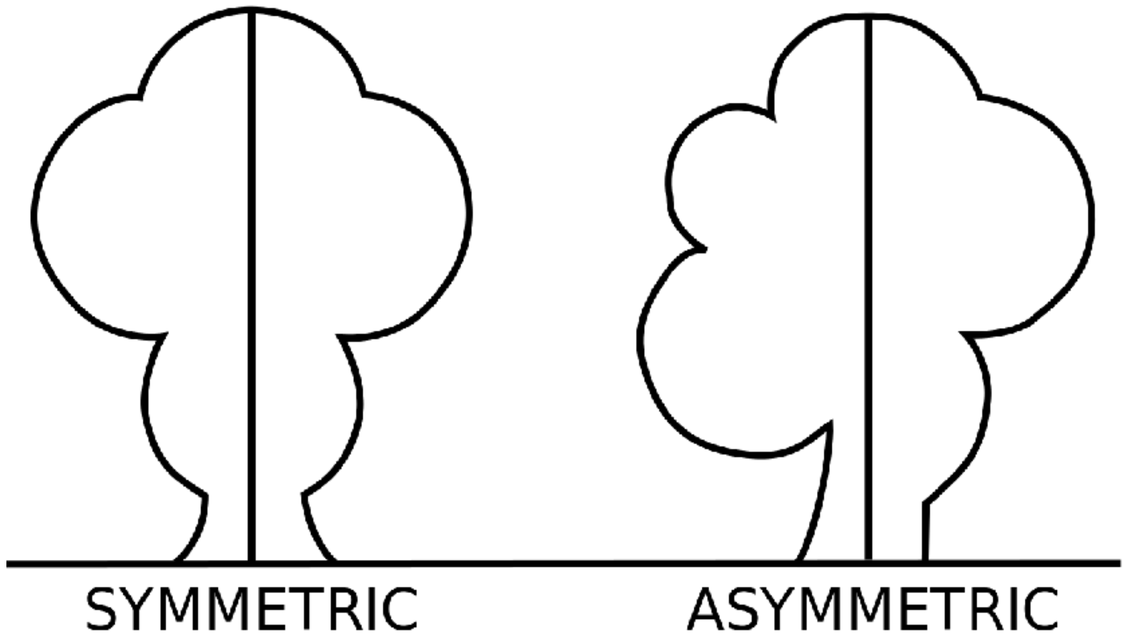
Cryptography Features

- Simplicity
- Interface-driven, POJO based
- Simplified wrapper over JCE infrastructure.
- “Object Orientifies” cryptography concepts
- Easier to understand API

Ciphers Defined

Encryption and decryption data based on shared or public/private keys.

- Symmetric Cipher - same key
  - Block Cipher - chunks of bits
  - Stream Cipher - stream of bits
- Asymmetric Cipher - different keys



Hashes Defined

One-way, irreversible conversion of an input source (e.g. `MessageDigest`)

Used for:

- Credentials transformation, Checksum
- Data with underlying byte array

  Files, Streams, etc

Two kinds of people

Cipher Features

- Object Oriented Hierarchy

  JcaCipherService, AbstractSymmetricCipherService,

  DefaultBlockCipherService, etc

- Just instantiate a class

  No “Transformation String” / Factory methods

- More secure default settings then JDK !????

  Cipher Modes, Initialization Vectors, etc

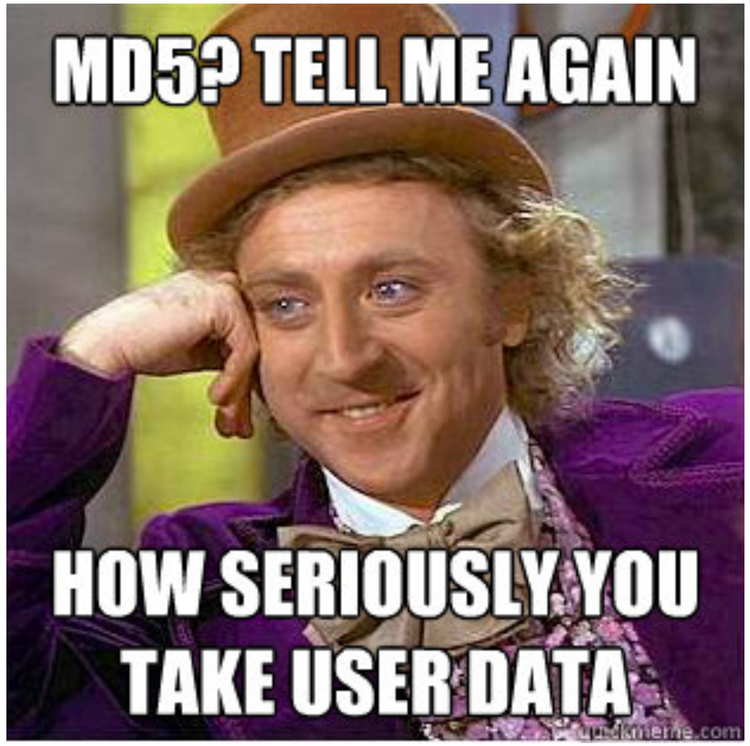
Shiro's CipherService Interface

```
public interface CipherService {

  ByteSource encrypt(byte[] raw, byte[] key);

  void encrypt(InputStream in, OutputStream out, byte[] key);

  ByteSource decrypt( byte[] cipherText, byte[] key);

  void decrypt(InputStream in, OutputStream out, byte[] key);
}
```

Hash Features

- Default interface implementations

  MD5, SHA1, SHA-256, SHA-512, etc
- Built in Hex & Base64 conversion
- Built-in support for Salts and repeated hashing



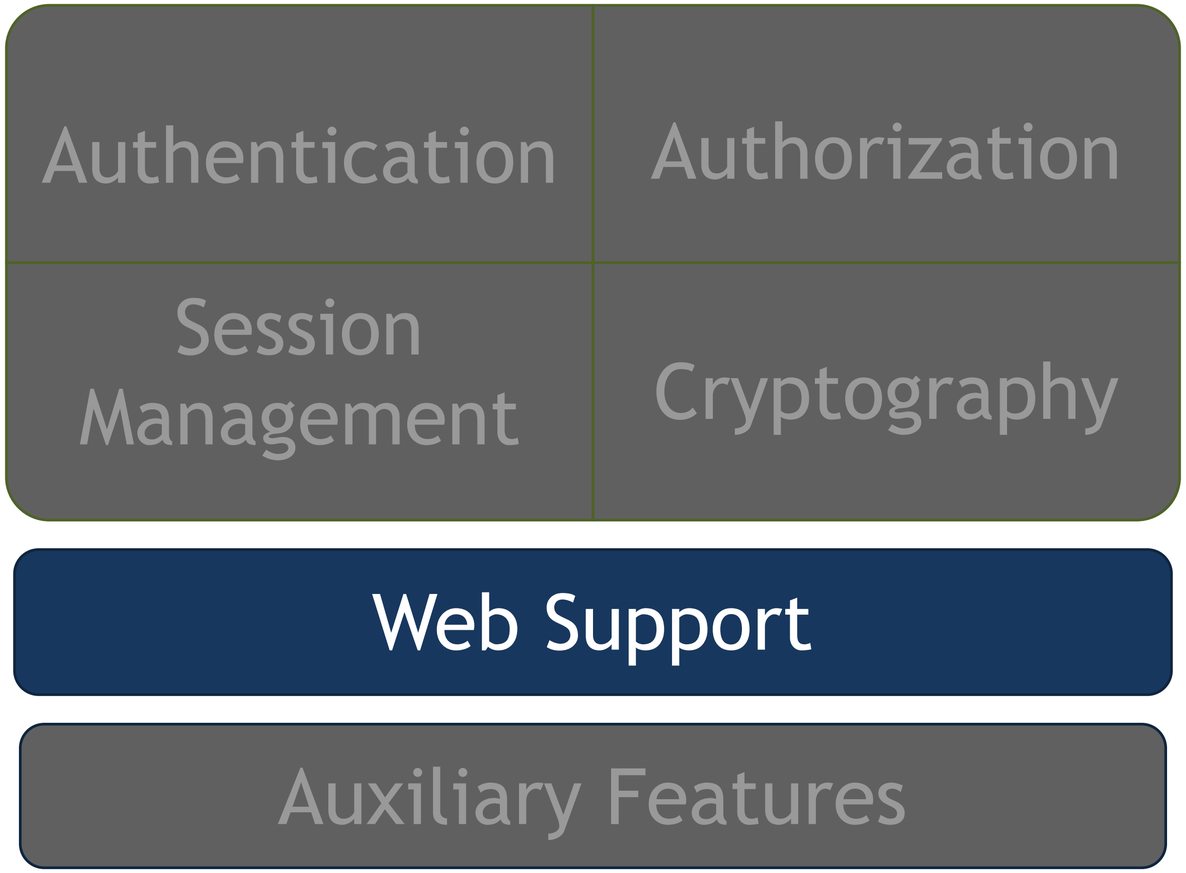
Shiro's Hash Interface

```
public interface Hash {
    byte[] getBytes();
    String toHex();
    String toBase64();
}
```

Intuitive Hash API

```
//some examples:
new Md5Hash("foo").toHex();

//File MD5 Hash value for checksum:
new Md5Hash( aFile ).toHex();

//store password, but not plaintext:
new Sha512(aPassword, salt, 1024).toBase64();
```

Web Support


Authentication
Authorization
Session Management
Cryptography
Web Support
Auxiliary Features

Web Support Features

- Optional Servlet Fragment (`shiro-servlet-plugin`)
- Servlet Filter based (simple configuration)
- Protects all URLs
- Configurable URL-specific chaining
- Transparent `HttpSession` support
- Configure with an Spring, Guice, or an INI file

shiro.ini

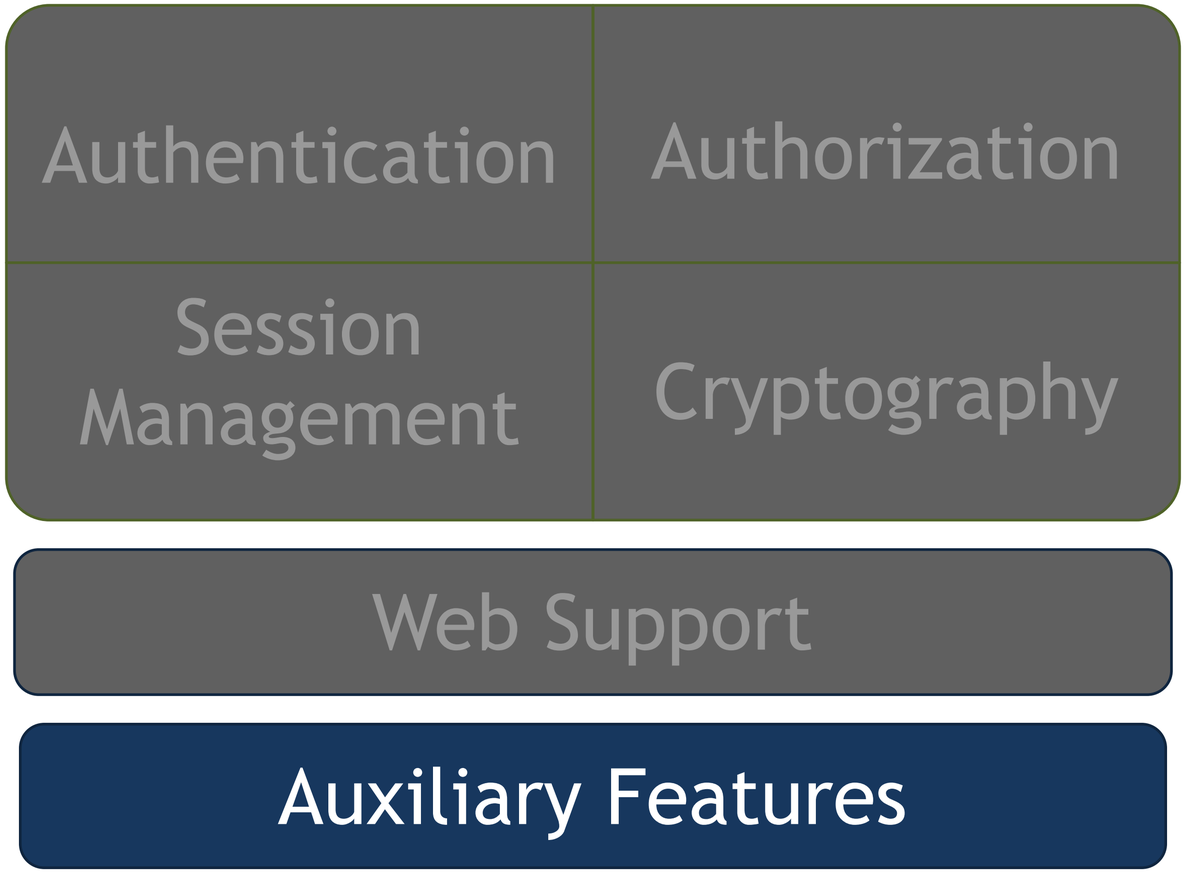
```
[main]
ldapRealm = org.apache.shiro.realm.ldap.JndiLdapRealm
ldapRealm.userDnTemplate = uid={0},ou=users,dc=mycompany,dc=com
ldapRealm.contextFactory.url = ldap://ldapHost:389

securityManager.realm = $realm

[urls]
/images/**  = anon
/account/** = authc
/rest/** = authcBasic
/remoting/** = authc, roles[b2bClient], ...
```

JSP TagLib Authorization

```
<%@ taglib prefix="shiro" uri=http://shiro.apache.org/tags %>

<shiro:guest/>
<shiro:user/>
<shiro:principal/>
<shiro:hasPermission/>
<shiro:lacksPermission/>
<shiro:hasRole/>
<shiro:lacksRole/>
<shiro:hasAnyRoles/>
<shiro:authenticated/>
<shiro:notAuthenticated/>
```

APACHE SHIRO SECURITY

Auxiliary Features



Auxiliary Features

- Threading & Concurrency

  `Callable`/`Runnable` & `Executor`/`ExecutorService`

- “Run As” support
- Ad-hoc Subject instance creation
- Unit Testing
- Remembered vs Authenticated

Using a Subject in other frameworks

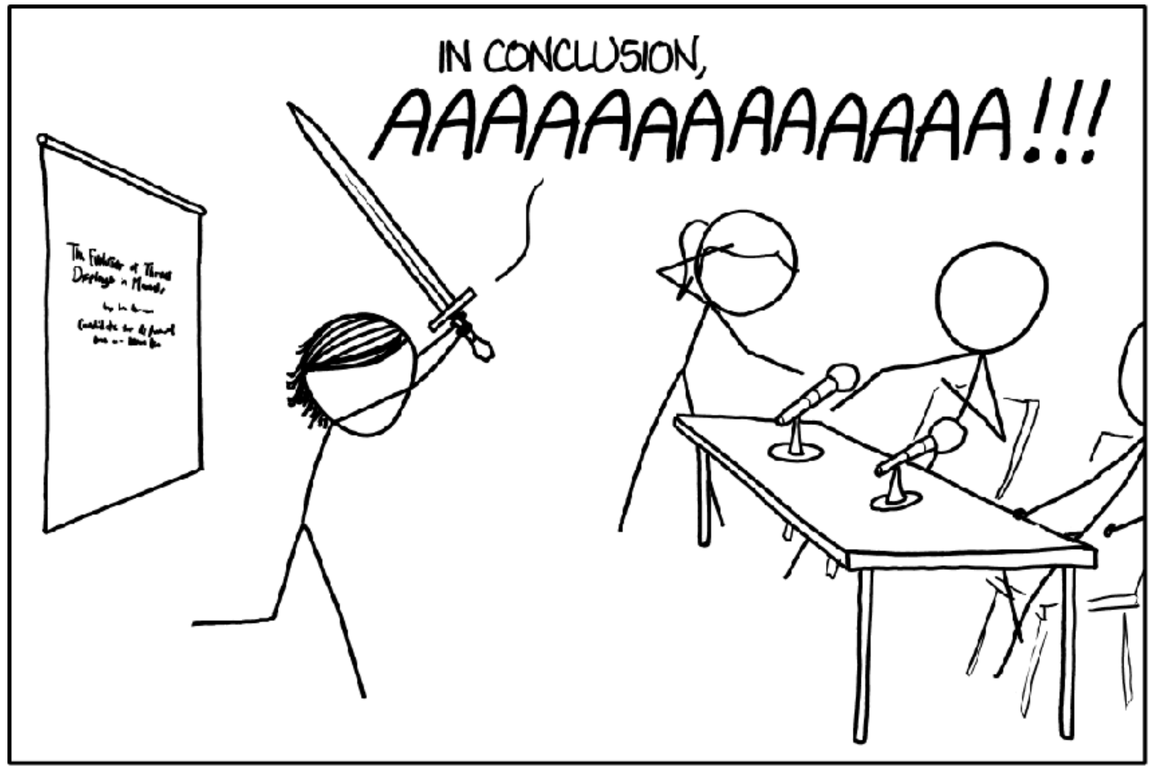
```
Subject subject = //build or acquire subject

MyResult result = subject.execute( new Callable<MyResult>()
{
    public MyResult call() throws Exception {
        //subject is 'bound' to the current thread now
        //any SecurityUtils.getSubject() calls in any
        //code called from here will work
        return myResult;
    }
});
//At this point, the Subject is no longer associated
//with the current thread and everything is as it was before
```

Logging Out

```
//Logs the user out, relinquishes account
//data, and invalidates any Session
SecurityUtils.getSubject().logout();
```

Thanks!

- https://shiro.apache.org
- bdemers@apache.org
- https://developer.okta.com
- We are hiring! okta.com



Credits

- https://commons.wikimedia.org/wiki/File:130608_Matsumoto_Castle_Matsumoto_Nagano_pref_Japan02bs4.jpg
  - Author: 663highland
  - License: GFDL + Creative Commons 2.5
- https://commons.wikimedia.org/wiki/File:Chocolate_Chip_Cookies_-_kimberlykv.jpg
  - Author: Kimberly Vardeman
  - License: Creative Commons Attribution 2.0
- xkcd
  - https://www.xkcd.com/1363/
  - https://xkcd.com/1403/
- https://yuml.me/edit/f18ceb64
- http://maybenull.github.io/angular-authz/ (front end permission strings)