Easy Application Security with Apache Shiro

Les Hazlewood
Apache Shiro Project Chair
CTO, Stormpath, stormpath.com

- Identity Management and Access Control API
- Security for ***your*** applications
- User security workflows
- Security best practices
- Developer tools, SDKs, libraries

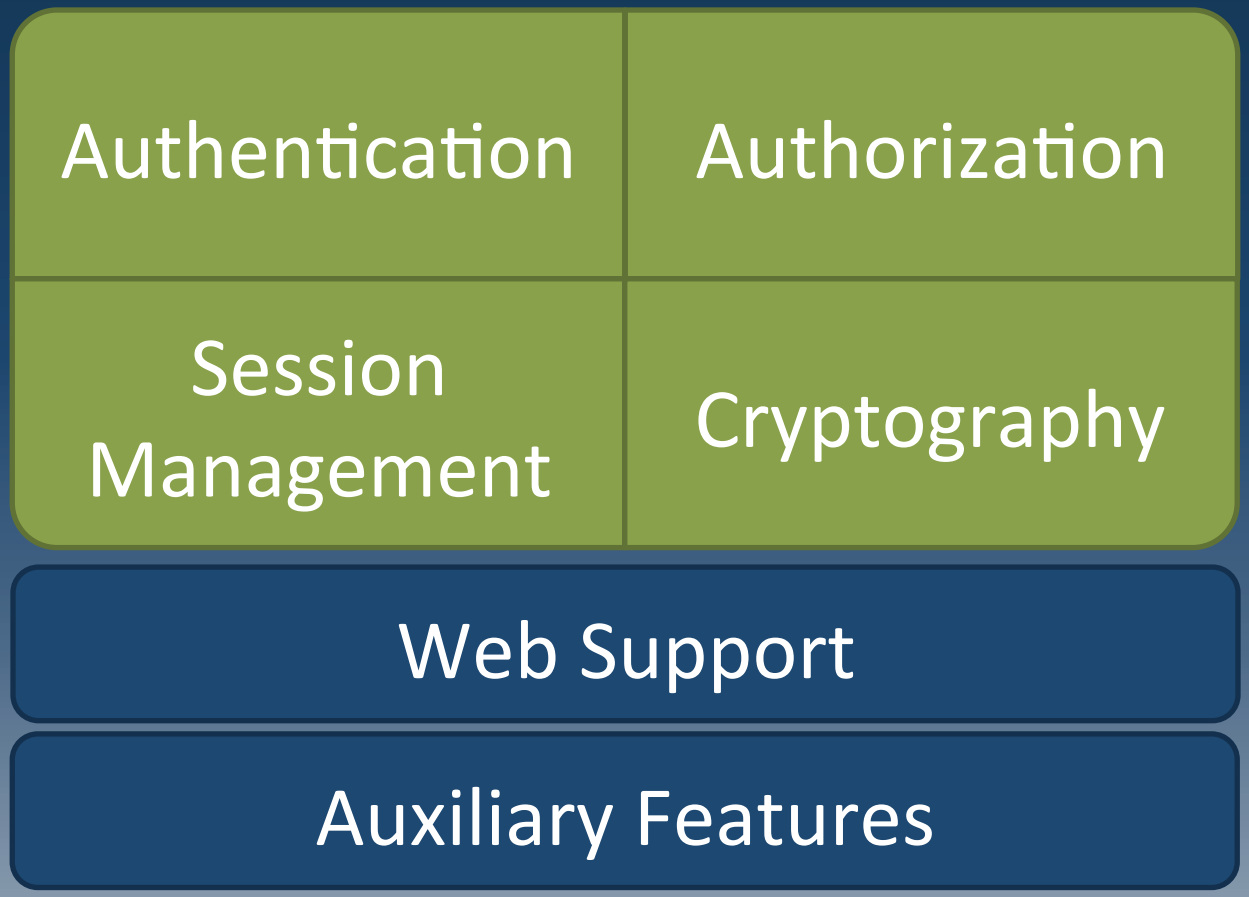
What is Apache Shiro?

- Application security framework
- ASF TLP http://shiro.apache.org
- Quick and Easy
- Simplifies Security

SHIRO

Stormpath

Agenda



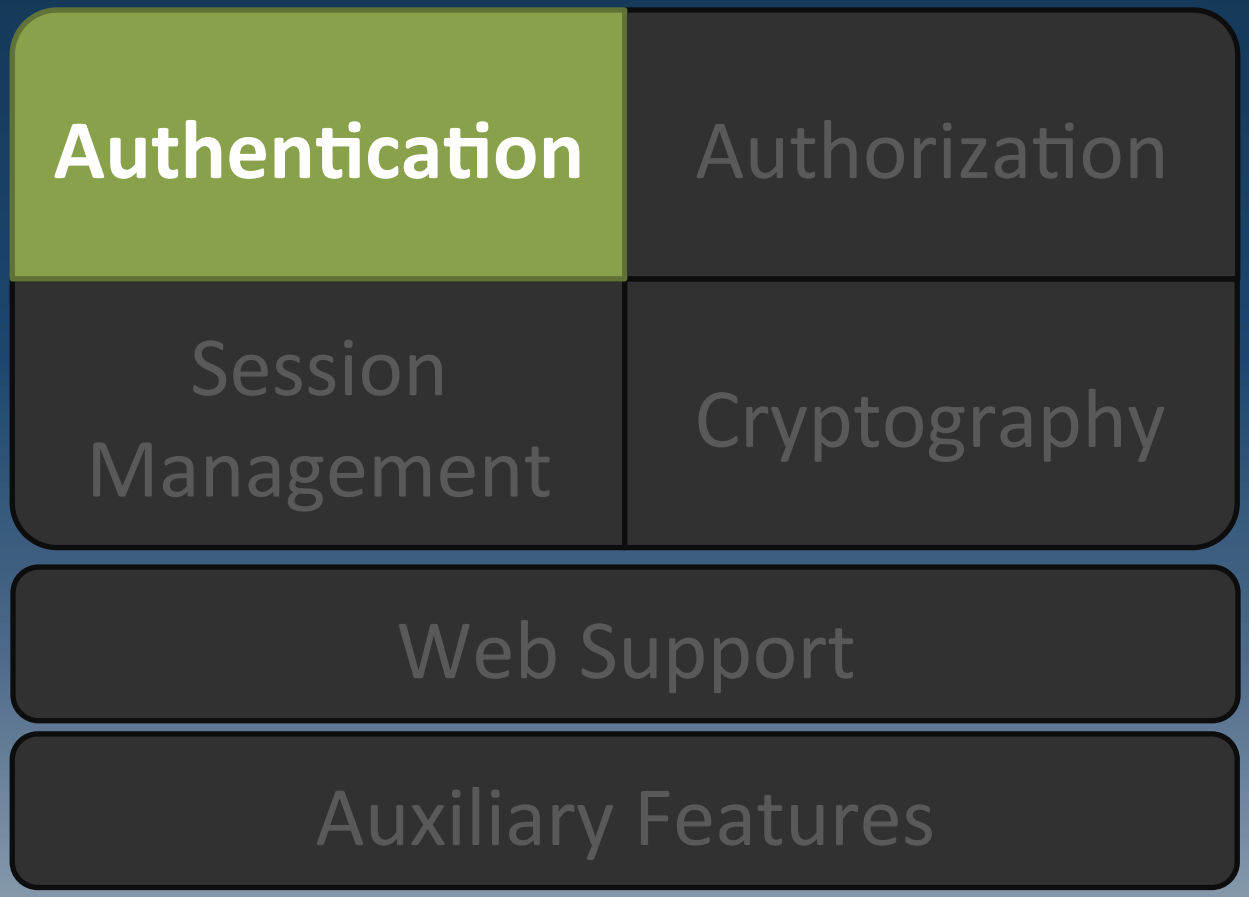
SHIRO

Stormpath

Quick Terminology

- **Subject** – Security-specific user 'view'
- **Principals** – Subject's identifying attributes
- **Credentials** – Secret values that verify identity
- **Realm** – Security-specific DAO

Authentication



SHIRO

Stormpath

Authentication Defined

Identity verification:

Proving a user is who he says he is

Shiro Authentication Features

- Subject-based (current user)
- Single method call
- Rich Exception Hierarchy
- ‘Remember Me’ built in
- Event listeners

How to Authenticate with Shiro

Steps

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

Step 1: Collecting Principals & Credentials

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

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

Step 2: Submission

```
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…
```

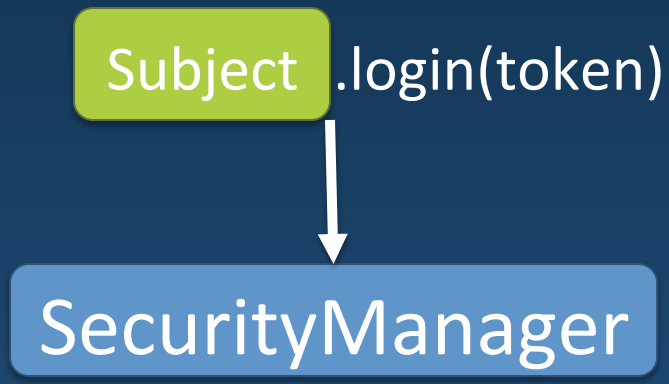
SHIRO

Stormpath

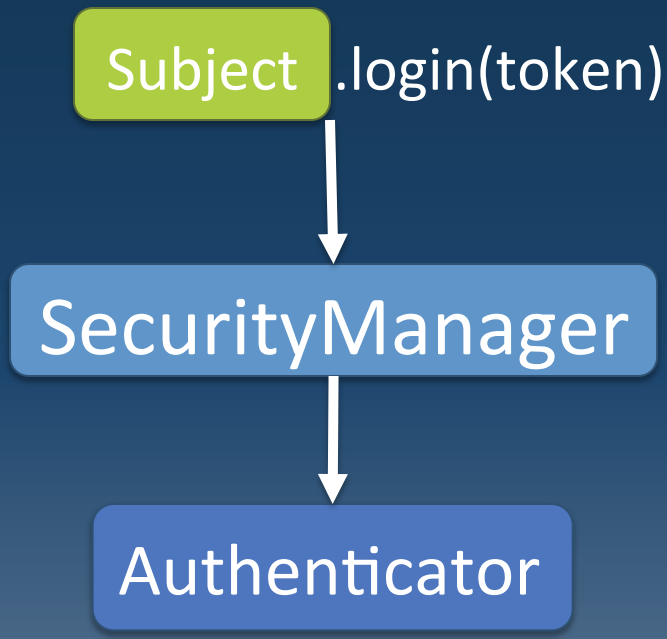
How does it work?

Subject .login(token)

How does it work?



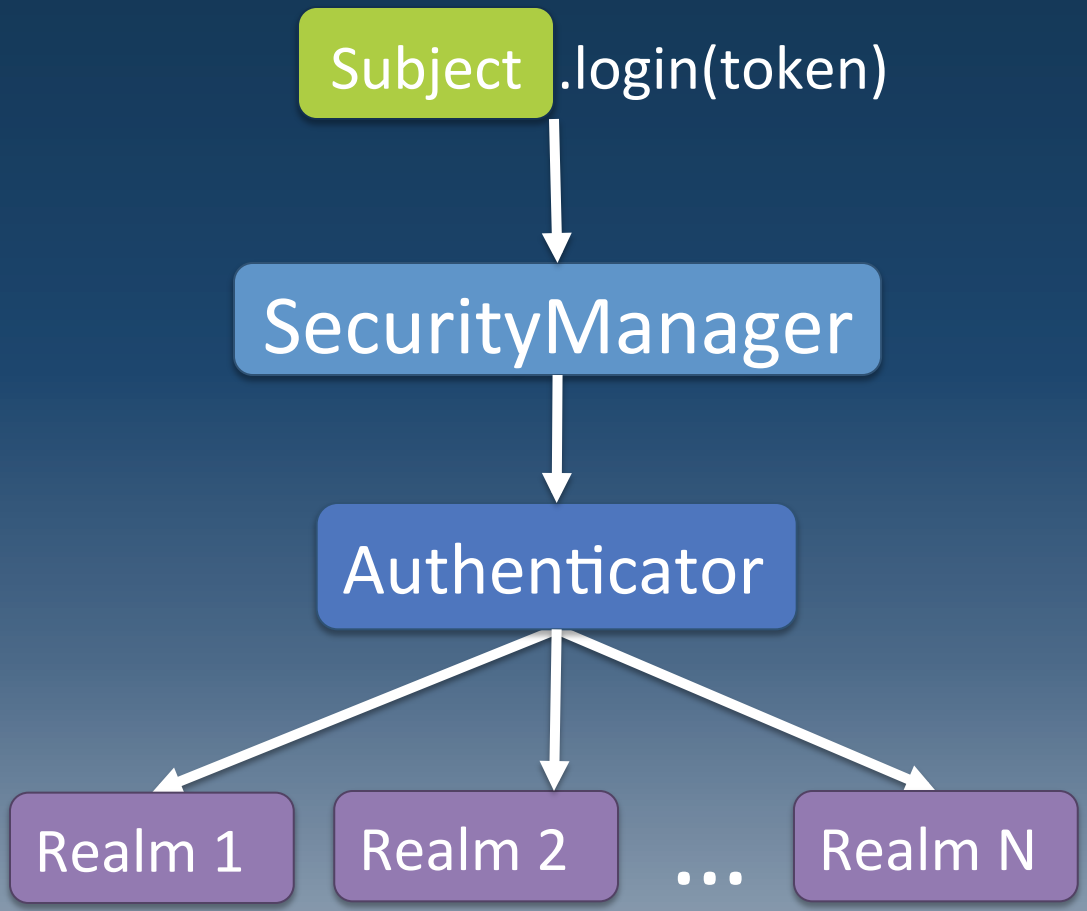
How does it work?



SHIRO

Stormpath

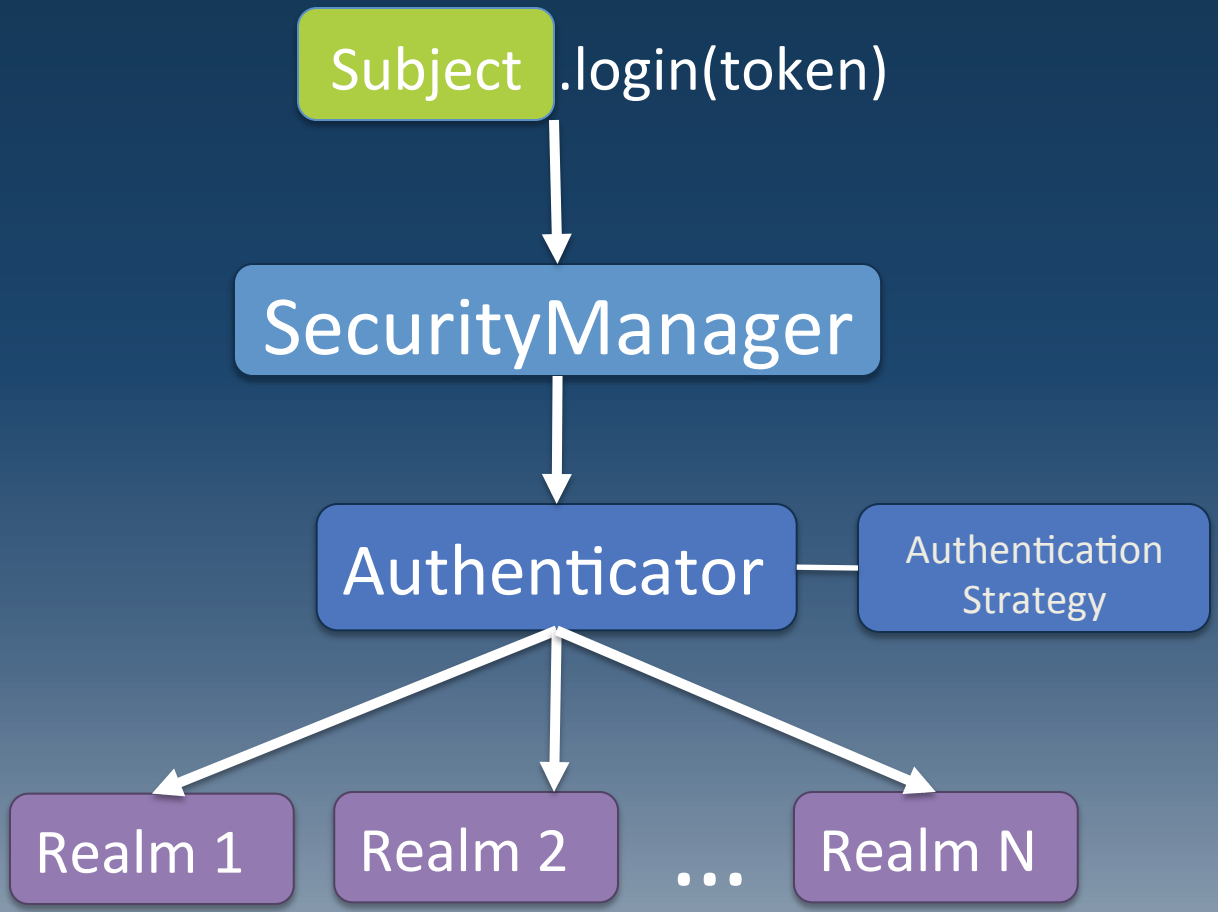
How does it work?



SHIRO

Stormpath

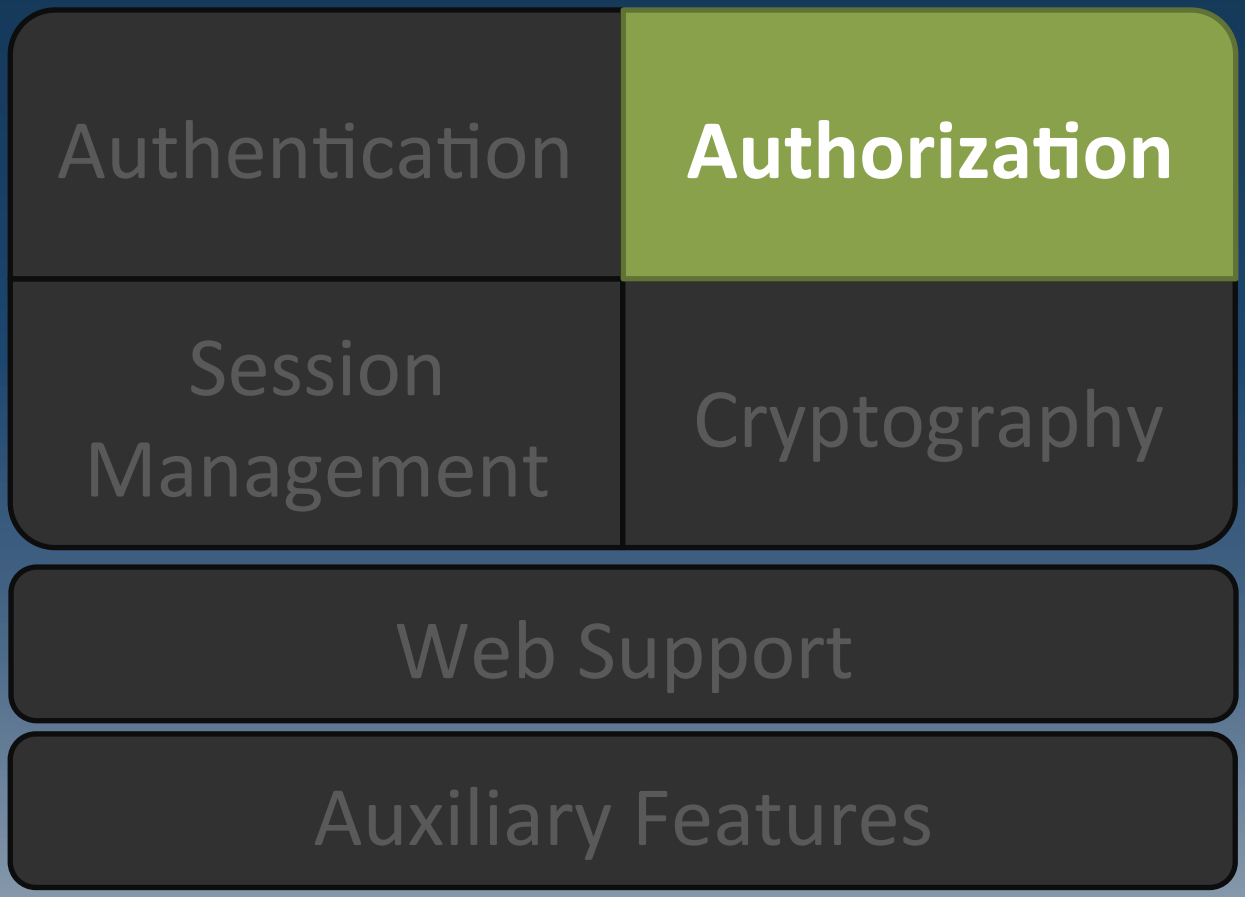
How does it work?



SHIRO

Stormpath

Authorization



SHIRO

Stormpath

Authorization Defined

Process of determining “who can do what”
AKA Access Control

Elements of Authorization

- Permissions
- Roles
- Users

Permissions Defined

- Most atomic security element
- Describes resource *types* and their behavior
- The “what” of an application
- Does not define “who”
- AKA “rights”

Roles Defined

- Implicit or Explicit construct
- Implicit: Name only
- Explicit: A named collection of Permissions

  Allows behavior aggregation

  Enables dynamic (runtime) alteration of user abilities.

Users Defined

- The “who” of the application

- What each user can do is defined by their association with Roles or Permissions

Example: User’s roles imply PrinterPermission

Authorization Features

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

How to Authorize with Shiro

Multiple means of checking access control:

- Programmatically
- JDK 1.5 annotations & AOP
- JSP/GSP/JSF* TagLibs (web support)

Programmatic Authorization

Role Check

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

if (currentUser.hasRole("administrator")) {
    //show the 'delete user' button
} else {
    //don't show the button?)
}
```

SHIRO

Stormpath

Programmatic Authorization

Permission Check

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

Permission deleteUser =
new UserPermission("jsmith","delete");

If (currentUser.isPermitted(deleteUser)) {
    //show the 'delete user' button
} else {
    //don't show the button?
}
```

SHIRO

Stormpath

Programmatic Authorization

Permission Check (String-based)

```
String perm = "user:delete:jsmith";

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

Annotation Authorization

Role Check

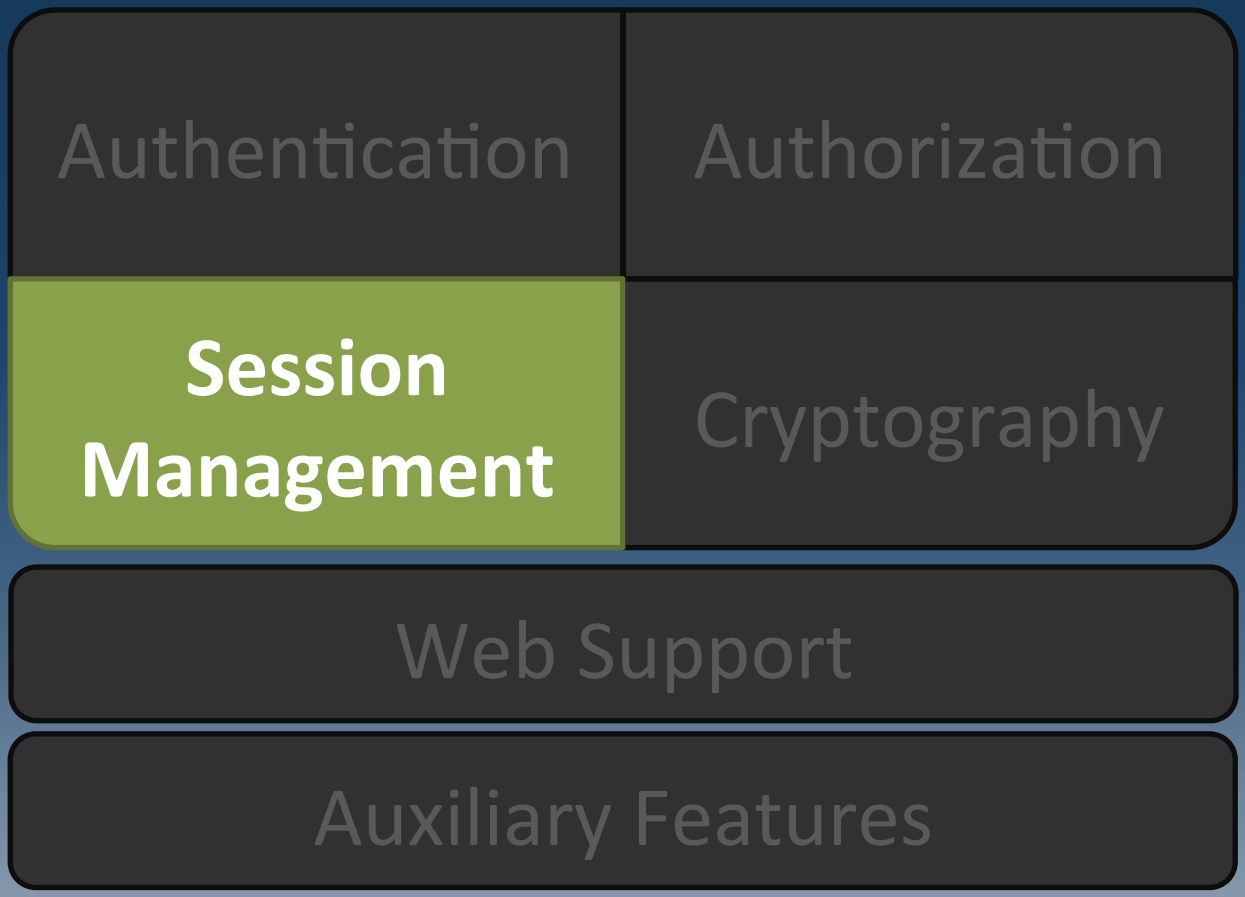
```
@RequiresRoles( “teller” )
public void openAccount(Account a) {
    //do something in here that
    //only a ‘teller’ should do
}
```

Annotation Authorization

Permission Check

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

Enterprise Session Management



SHIRO

Stormpath

Session Management Defined

Managing the lifecycle of Subject-specific temporal data context

Session Management Features

- Heterogeneous client access
- POJO/J2SE based (IoC friendly)
- Event listeners
- Host address retention
- Inactivity/expiration support (touch())
- Transparent web use - HttpSession
- Container-Independent Clustering!

Acquiring and Creating Sessions

```
Subject currentUser =
    SecurityUtils.getSubject()

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


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

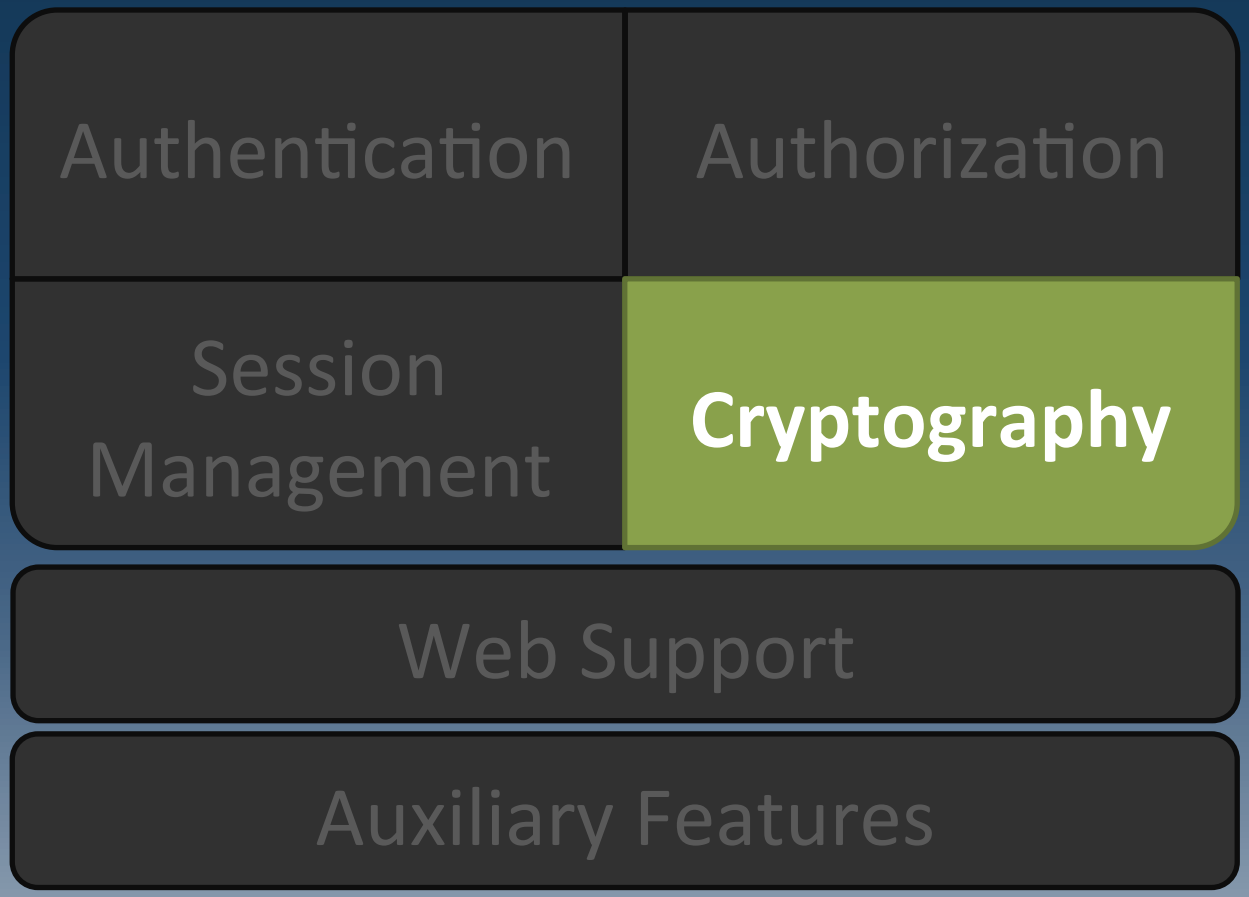
Session API

```
getStartTimestamp()

getLastAccessTime()

getAttribute(key)

setAttribute(key, value)

get/setTimeout(long)

touch()

...
```

SHIRO

Stormpath

Cryptography



SHIRO

Stormpath

Cryptography Defined

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

Elements of Cryptography

- Ciphers
- Hashes

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

A one-way, irreversible conversion of an input source (a.k.a. Message Digest)

Used for:

- Credentials transformation, Checksum
- Data with underlying byte array

  Files, Streams, etc

Cryptography Features

Simplicity

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

Cipher Features

- OO Hierarchy

  JcaCipherService, AbstractSymmetricCipherService, DefaultBlockCipherService, etc

- Just instantiate a class

  No “Transformation String”/Factory methods

- More secure default settings than JDK!

  Cipher Modes, Initialization Vectors, et. al.

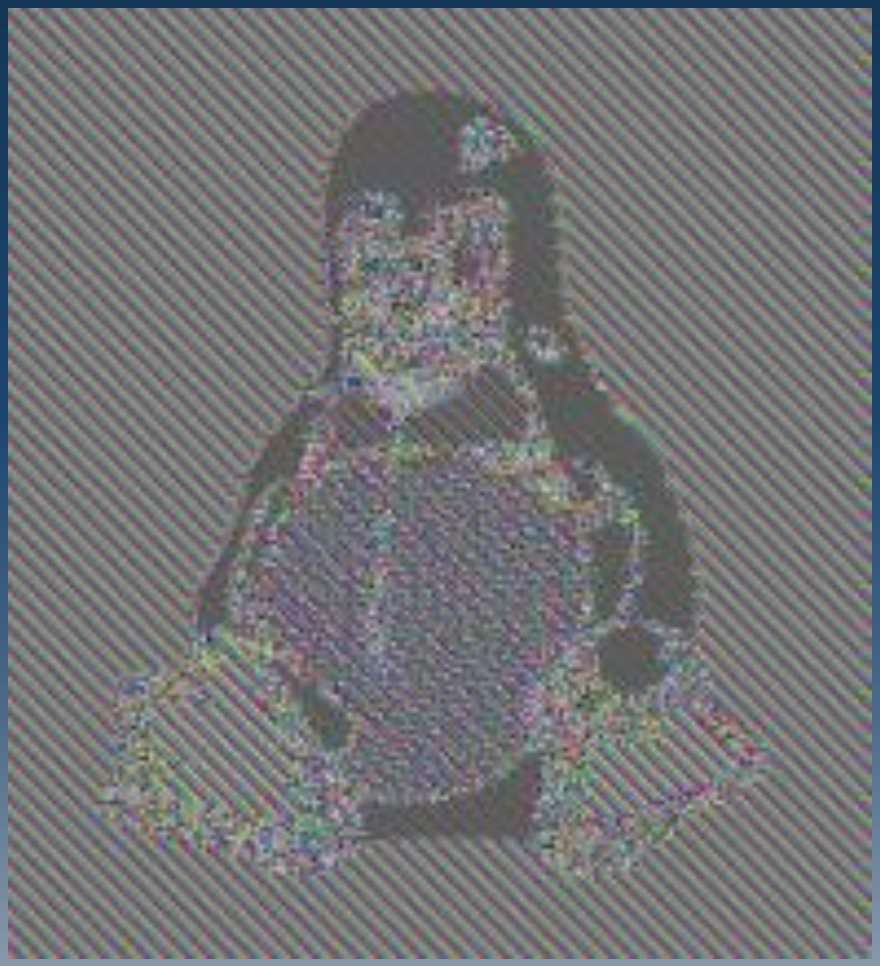
Example: Plaintext

(image courtesy WikiPedia)

SHIRO

Stormpath

Example: ECB Mode (JDK Default!)

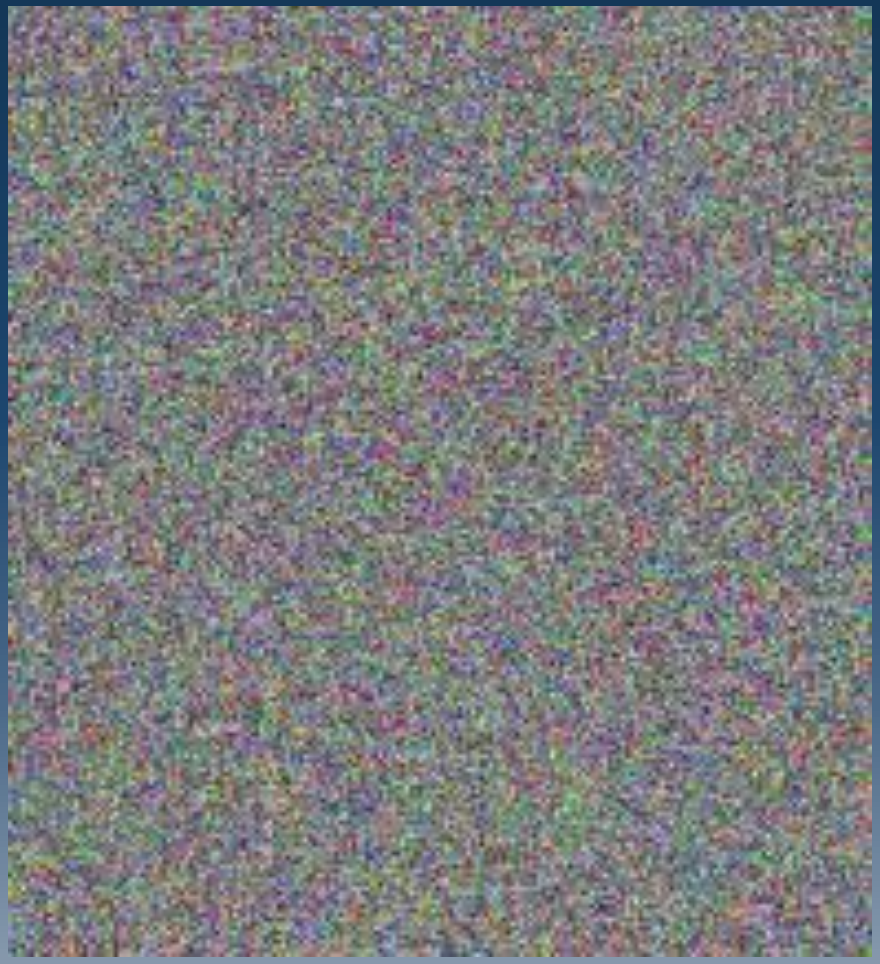


(image courtesy WikiPedia)

SHIRO

Stormpath

Example: Shiro Defaults



(image courtesy WikiPedia)

SHIRO

Stormpath

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, et. al.
- 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();

}
```

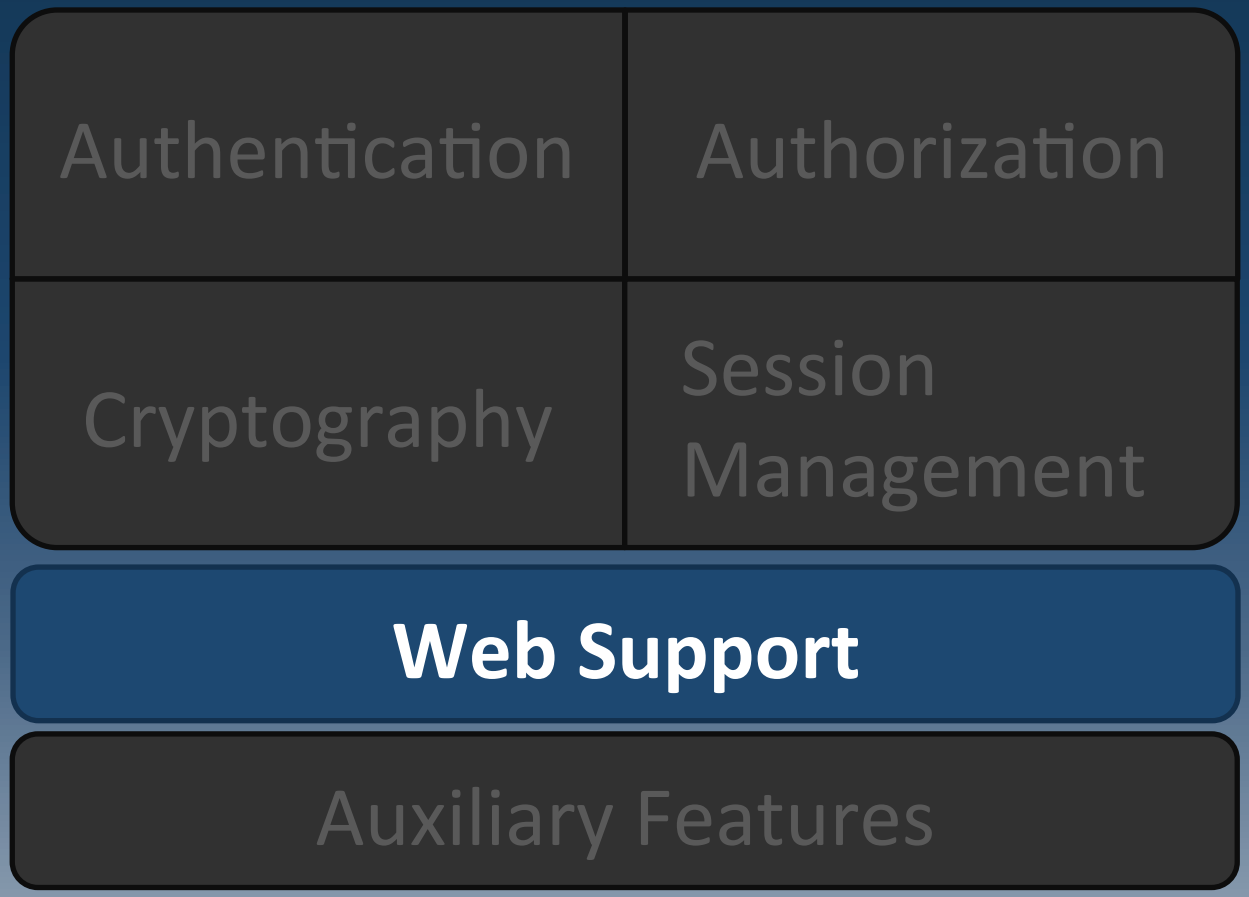
Intuitive OO Hash API

```
//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();
```

SHIRO

Stormpath

Web Support



SHIRO

Stormpath

Web Support Features

- Simple ShiroFilter web.xml definition
- Protects all URLs
- Innovative Filtering (URL-specific chains)
- JSP Tag support
- Transparent HttpSession support

web.xml

```
<filter>
  <filter-name>ShiroFilter</filter-name>
  <filter-class>
 org.apache.shiro.web.servlet.IniShiroFilter
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>ShiroFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

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], …
```

SHIRO

Stormpath

JSP TagLib Authorization

```
<%@ taglib prefix="shiro"
           uri="http://shiro.apache.org/tags" %>
<html>
<body>
    <shiro:hasRole name="administrator">
        <a href="manageUsers.jsp">
            Click here to manage users
        </a>
    </shiro:hasRole>
    <shiro:lacksRole name="administrator">
        No user admin for you!
    </shiro:hasRole>
</body>
</html>
```

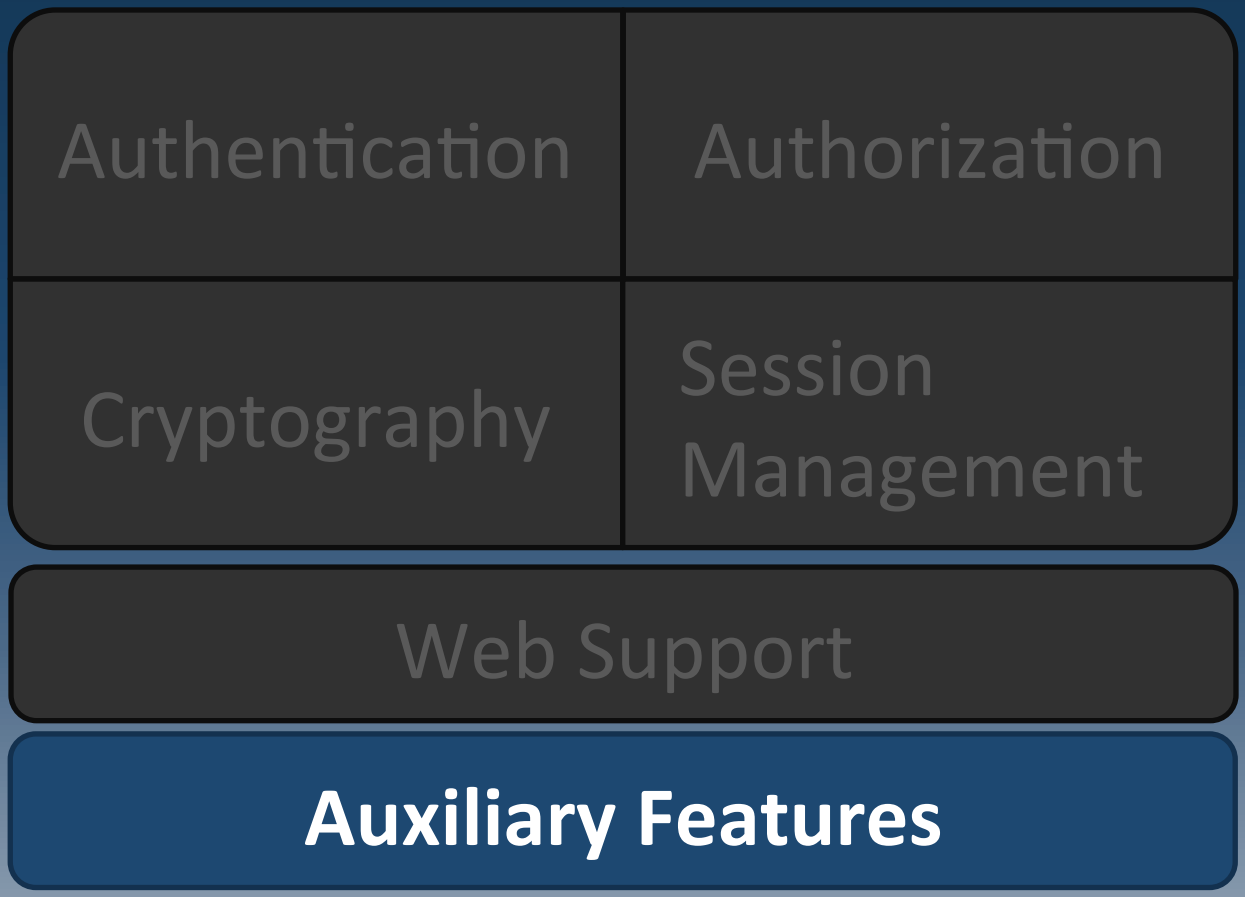
JSP TagLibs

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

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

SHIRO

Stormpath

Auxiliary Features



SHIRO

Stormpath

Auxiliary Features

- Threading & Concurrency

  Callable/Runnable & Executor/ExecutorService

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

Logging Out

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

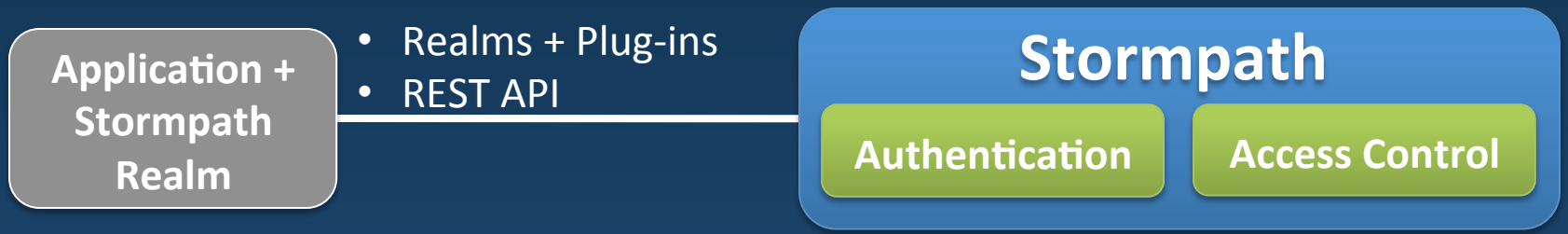
App-specific log-out logic:

- Before/After the call
- Listen for Authentication or StoppedSession events.

Coming in 1.3, 2.0

- Typesafe EventBus
- OOTB Hazelcast Session clustering
- Lower coupling in components
  Composition over Inheritance
- Stronger JEE (CDI, JSF) support
- Default Realm
  Pluggable authc lookup, authz lookup
- Default Authentication Filter
  (multiple HTTP schemes + UI fallback)

Stormpath: User Management API Service



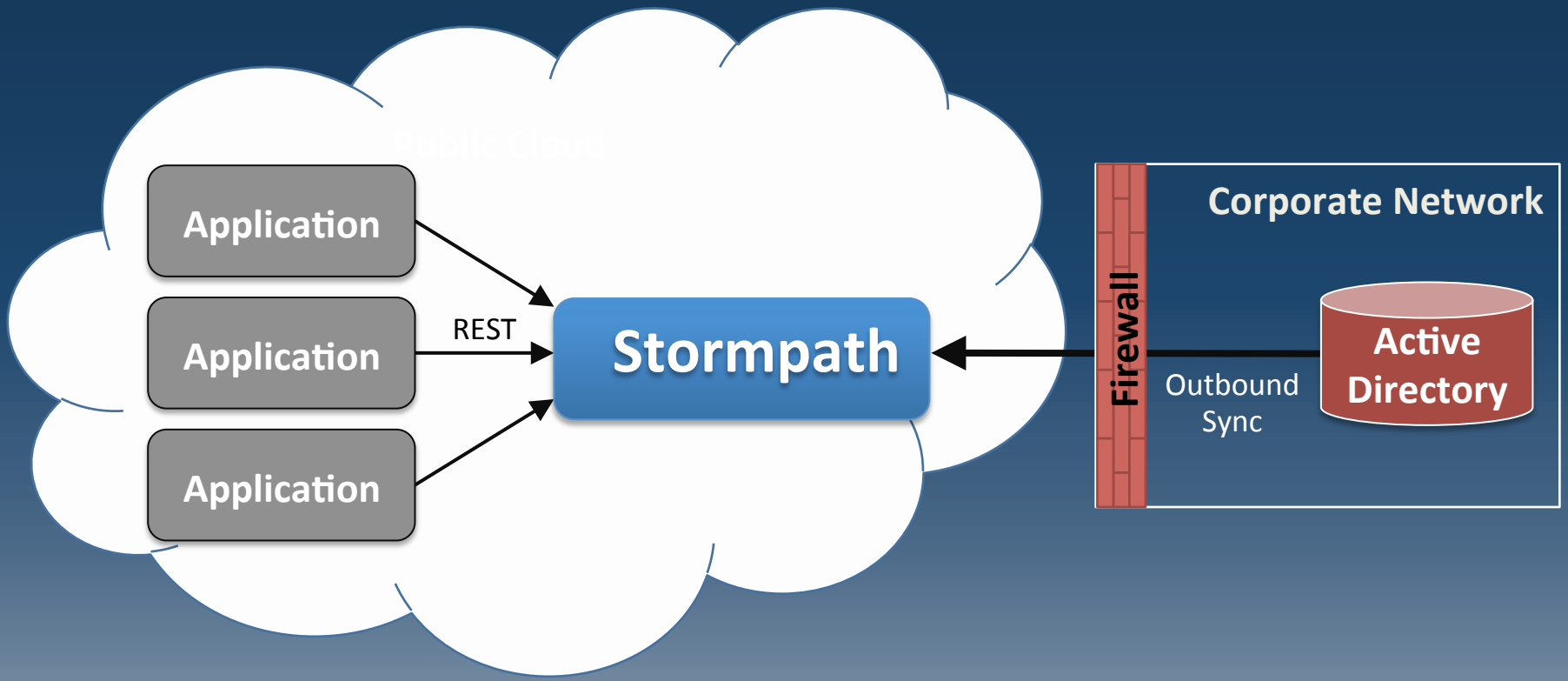
Out-of-the-box Features

- Managed security data model
- Secure credential storage
- Password self-service
- Management GUI

SHIRO

Stormpath

Stormpath: Cloud Deployment



SHIRO

Stormpath

Thank You!

- les@stormpath.com
- Twitter: @lhazlewood
- http://www.stormpath.com