# WEBD 236

## Web Information Systems Programming

## Week 10


Todd Whittaker and Scott Sharkey
(sharkesc@franklin.edu)

FRANKLIN UNIVERSITY

# Agenda

- This week's expected outcomes
- This week's topics
- This week's homework
- Upcoming deadlines
- Questions and answers

# Week 10 Outcomes

- Explain how a browser uses a certificate to establish an encrypted connection to a server
- Compare and contrast ACL and RBAC approaches to authorization.
- Implement authentication/authorization.

# Securing Web Sites

- Recall the issues with sessions:
  - Use opaque session keys (why?)
  - Store session data on the server (why?)
  - Always use HTTPS (why)?

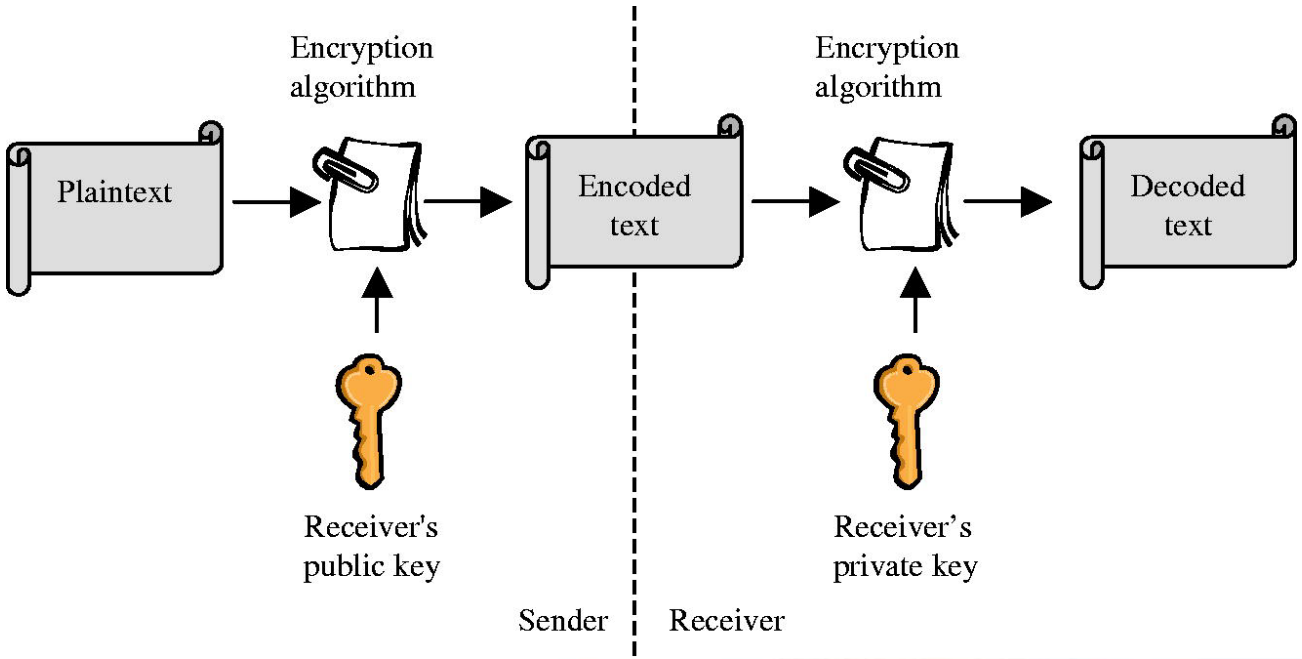
HTTPS requires certificates that are signed by a *signing authority* recognized by the browser.

# Securing Web Sites

- Encryption crash course
  - Public key / private key encryption
    - Mathematically based on the difficulty of factoring very large numbers into two primes
    - Using the primes, construct a public/private key pair (this is an involved process).
    - Messages encrypted with the public key can be decrypted by the private key and vice versa.
    - Keep private key secret, distribute the public key

# Securing Web Sites

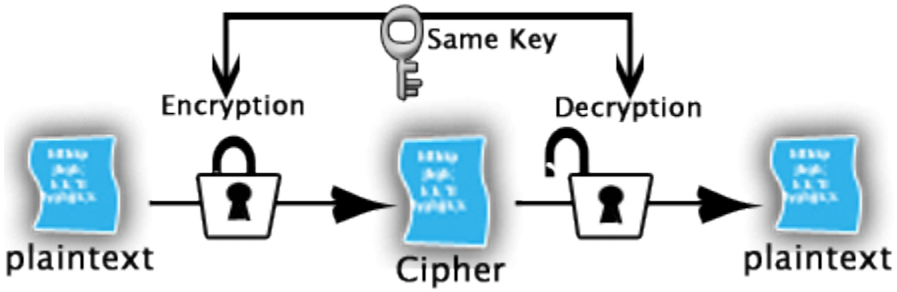
- Encryption crash course
  - Public key / private key encryption




Source http://www.emeraldinsight.com/content_images/fig/0670370804002.png

# Securing Web Sites

- Encryption crash course
  - Symmetric key encryption
    - The same key is used by both parties (DES, 3DES, etc.)
    - But, how do you securely exchange keys?




Source http://www.pwinfotech.com/2010/10/cryptographic-algorithm.html

# Securing Web Sites

- Private keys can also be used for *signing*
  - Signing is about authenticity (you are who you say you are)
  - If I use my private key to sign your public key, then I am vouching for your identity.
    - How? People can get my public key, decrypt what I encrypted, and compare it against your original public key.
    - Thus, anyone who trusts me can trust you.

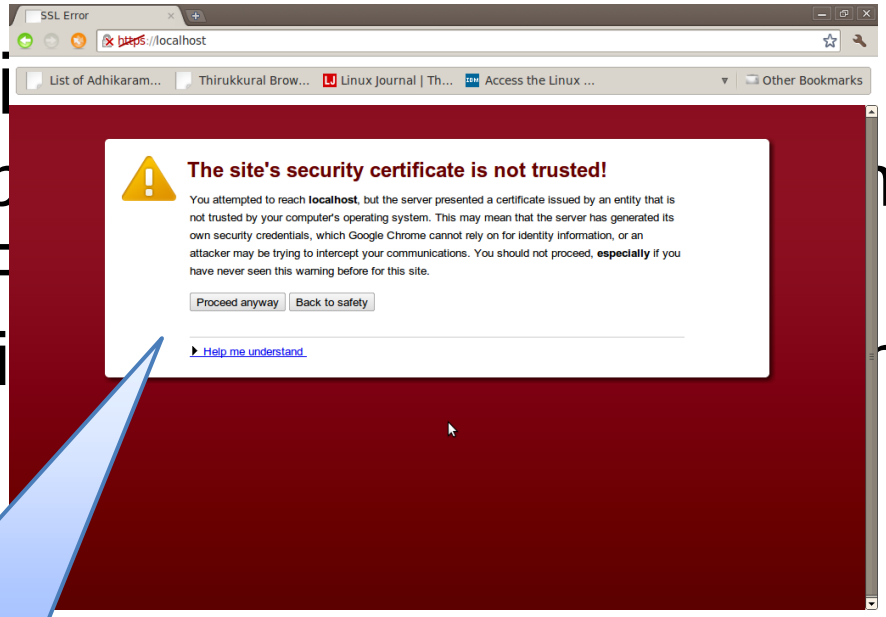
# Securing Web Sites

- X.509 certificates
  - Have the *public key* of the site wanting a valid HTTPS connection
  - Signed using the *private key* of a signing authority

HTTPS requires certificates that are signed by a *signing authority* recognized by the browser.

# Securing Web Sites

- X.509 certifi
  - Have the *p* ng a valid HTTP
  - Signed usi ning authority




If the signing authority isn't recognized, you get this!

# Securing Web Sites

- X.509 certifi
  - Have the *p*
    valid HTTP
    Signed usi
    y

Public key of HTTPS site

Hash of the whole certificate by signing authority

```
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number: 7829 (0x1e95)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
        OU=Certification Services Division,
        CN=Thawte Server CA/emailAddress=server-certs@thawte.com
    Validity
      Not Before: Jul  9 16:04:02 1998 GMT
      Not After : Jul  9 16:04:02 1999 GMT
    Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
        OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
          [illegible]:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
          66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
          70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
          16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
          c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
          8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
          d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
          e8:35:1c:9e:27:52:7e:41:8f
        Exponent: 65537 (0x10001)
  Signature Algorithm: md5WithRSAEncryption
    93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
    [illegible]2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
    ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
    d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
    0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
    5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
    8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
    68:9f
```

Source http://en.wikipedia.org/wiki/X.509

# Securing Web Sites

- Problem
  - Public/private key encryption is *expensive* computationally whereas symmetric key encryption is relatively *inexpensive*.
  - But, we would need to securely exchange symmetric keys.
  - Solution: encrypt a symmetric key using a public key!

# Securing Web Sites

- HTTPS
  - Get and verify the site's certificate
  - Use the public key in the certificate to encrypt a random secret key (generated by the browser) used for symmetric encryption.
  - Send the symmetric key (encrypted) to the server.
  - Use that key for the duration of the conversation.

# Securing Web Sites

- HTTPS
  - We used Apache to force HTTPS via the .htaccess file rewrite rules.

```
Options +FollowSymLinks
IndexIgnore */*
# Turn on the RewriteEngine
RewriteEngine On
#  Force HTTPS for security of cookies
RewriteCond %{HTTPS} !on
RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI} [L]
# Handle URL routing
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . urlrouter.php
```

# Securing Web Sites

- HTTPS
  - We used Apache to force HTTPS via the .htaccess file rewrite rules.
  - Can also do this with straight PHP in the master controller (see page 685).

# Securing Web Sites

- Authentication
  - “I am who I say I am.”
  - How can we make this assertion?
    - Two-factor authentication: something you have and something you know (i.e. card and PIN, thumbprint and password)
    - Username/password are *weak* authentication mechanisms, but workable.

# Securing Web Sites

- Authentication
  - Web authentication
    - Basic – browser feature (ugly dialog boxes)
    - Digest – same as above
    - Form-based – what almost everyone uses
      - Forms are not encrypted
      - Must use HTTPS!

# Securing Web Sites

- Authentication
  - Passwords
    - Should ***never*** be stored in the DB in plain text (why?)
    - Instead, *hash* the password and store the hash.
  - Cryptographic hashing
    - A one-way function that given text, outputs a fixed size bit string (usually as hex digits)
    - Can't take the bit string and figure out the original text
    - SHA, MD5, etc.

# Securing Web Sites

- Authentication
  - Passwords
    - Should ***never*** be stored in the DB in plain text (why?)
    - Instead, *hash* the password and store the hash.
  - Cryptographic hashing
    - A one-way function that takes plain text, outputs a fixed size bit string (usually 32 digits)
    - Can't take the hash and figure out the original text
    - SHA, MD5, etc.

Use sha1() function to do this in PHP.

# Securing Web Sites

- Authentication
  - Passwords
    - Should ***never*** be s… t (why?)
    - Instead, *hash* the … sh.
  - Cryptographic has…
    - A one-way function that given text, …s a fixed size bit string (usually as hex digits)
    - Can't take the bit string and figure out the original text
    - SHA, MD5, etc.

This is why you must "reset" your password rather than have your original password mailed to you.

# Securing Web Sites

- Encryption
  - Sensitive data should always be stored encrypted.
    - E.g. credit card numbers, social security numbers, etc.
    - Be aware of privacy laws where you operate!
  - Why not hash sensitive data?
  - Encrypt with mcrypt_* functions
    - See page 701
    - Keep your key secret and safe!

# Securing Web Sites

- Authorization (i.e. “access control”)
  - “Can I do this?” – many ways to authorize actions
  - Most involve “subjects” and “objects”
    - Subject initiates an action (normally a user)
    - Objects are the targets of an action (normally a resource or URL in our context)
    - Examples: Access control lists (ACLs), Discretionary Access Control (DAC), Mandatory Access Control (MAC), RBAC (Role-based Access Control)

# Securing Web Sites

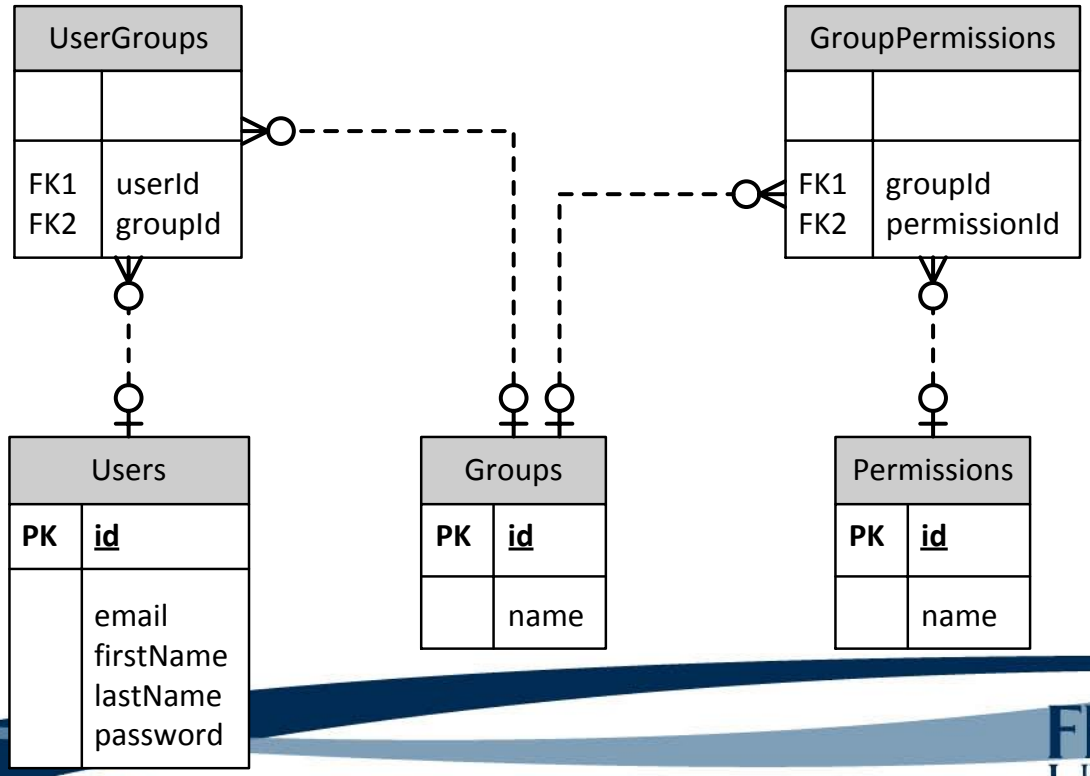
- Authorization (i.e. “access control”)
  - ACLs (access control lists)
    - Objects keep a list of subjects and actions that those subjects are permitted to do.
    - E.g. salaries.txt □ {(Bob: R), (Sally: R,W)}

# Securing Web Sites

- Authorization (i.e. “access control”)
  - RBAC (role-based access control)
    - Objects have a required permission
    - Permissions are granted to roles (groups)
    - Subjects are assigned one or more roles
  - Example:
    - Viewing salaries.txt has permission ‘view_salary’
    - ‘view_salary’ is granted to the group ‘Managers’
    - Sally is in the group Managers

# Securing Web Sites

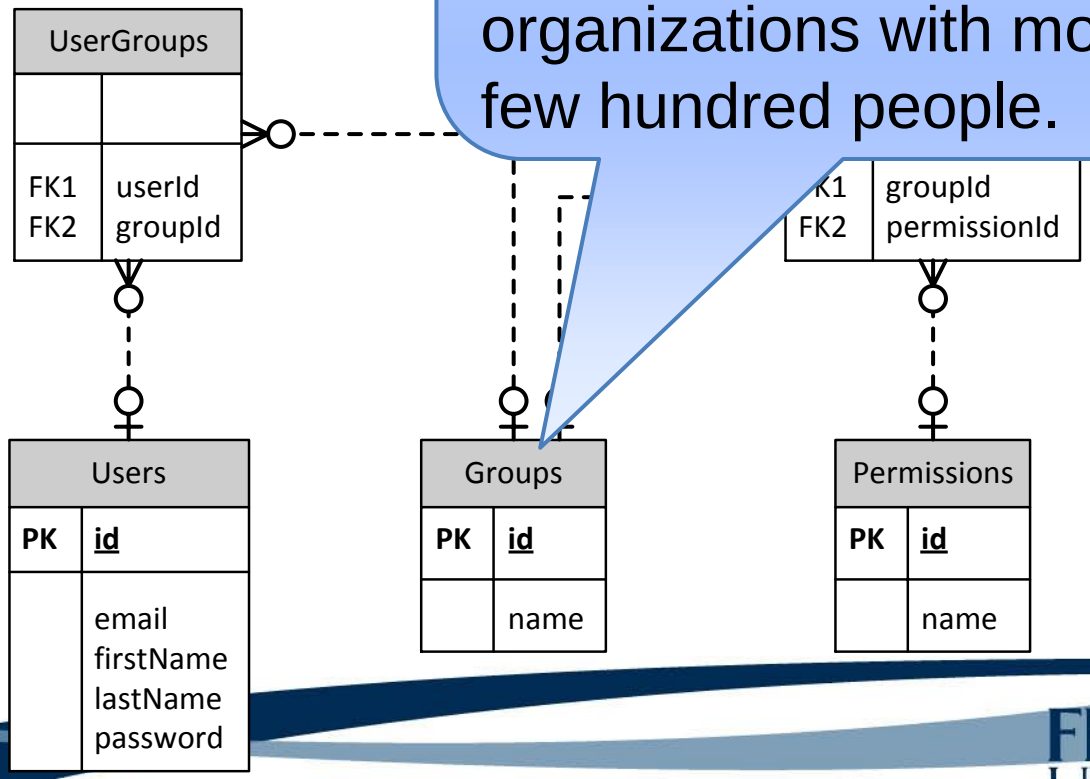
- Authorization (i.e. “access control”)
  - RBAC (role-based access control)



# Securing Web Sites

- Authorization (i.
  - RBAC (role-ba

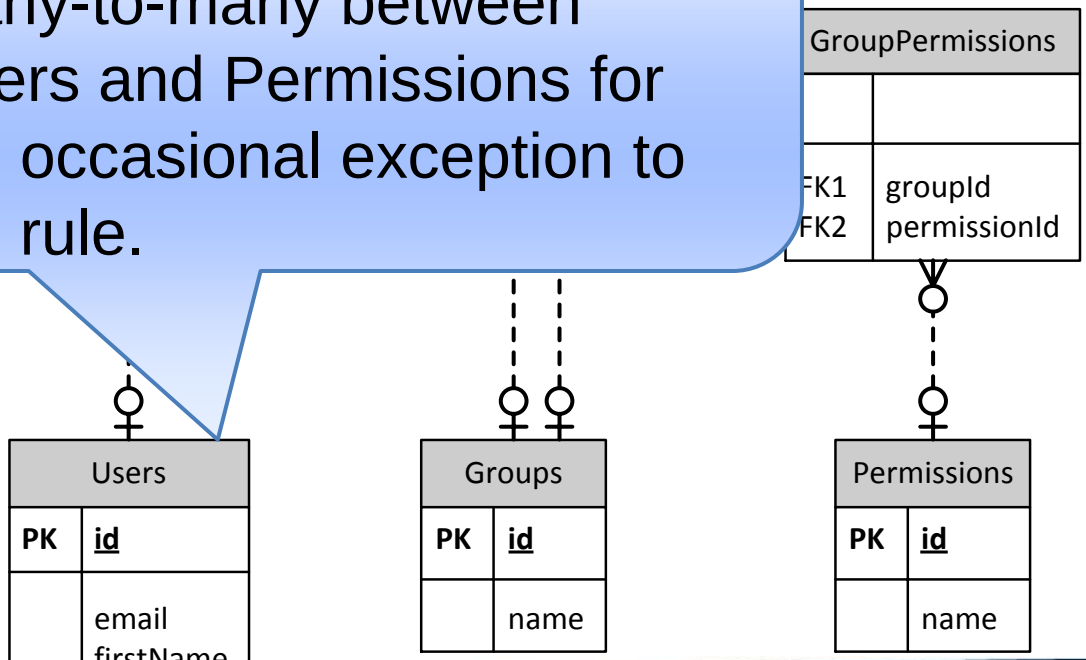
Most permissions can be handled through roles (Groups). RBAC is almost universal for authorization in organizations with more than a few hundred people.



# Securing Web Sites

- Authorization (i.e. “access control”)
  - control)

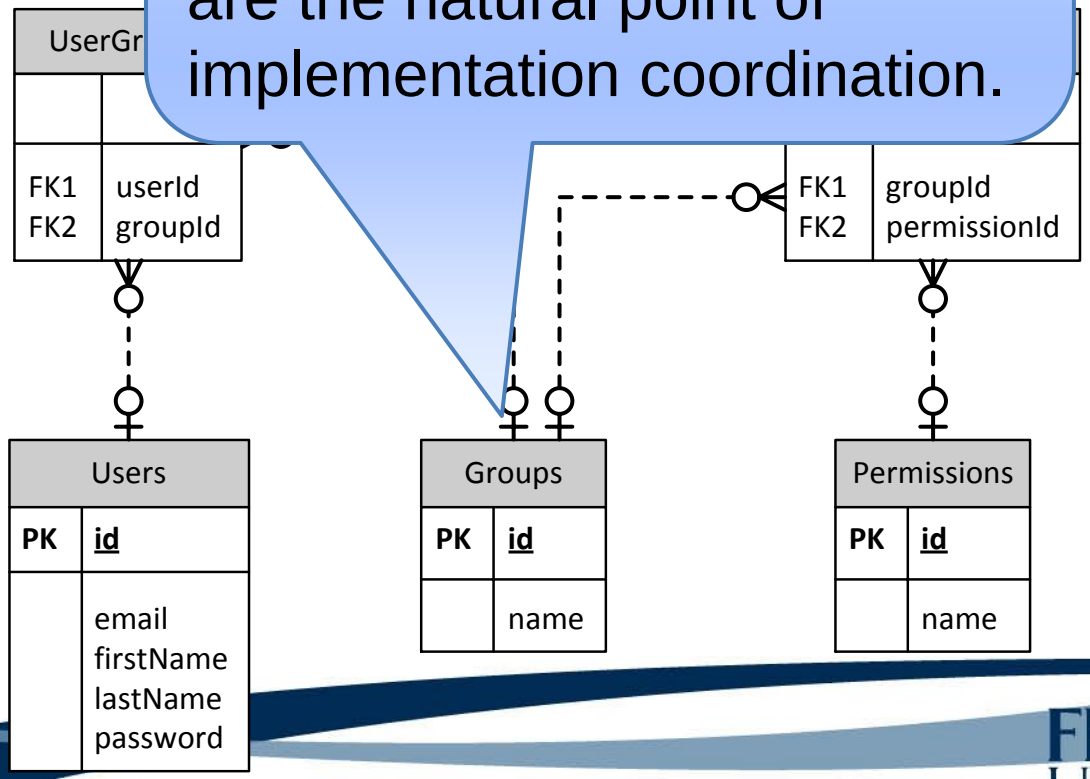
Sometimes there is also a many-to-many between Users and Permissions for the occasional exception to the rule.



# Securing Web Sites

- Authoriz…l")
  - RBAC…

Since Permissions are granted to groups, and Users are added to Groups, Groups are the natural point of implementation coordination.



# Implementing RBAC

- Selected code from models/Group.inc

```
class Group extends Model {

    protected $name;

    // ... stuff skipped...

    public function addUser($user) {
        $db = Db::getDb();
        $statement = $db -> prepare(
            "INSERT INTO usergroups (groupId, userId)
             VALUES (:groupId, :userId)");
        $statement -> bindValue(':groupId', $this -> getId());
        $statement -> bindValue(':userId', $user -> getId());
        $statement -> execute();
    }
```

# Implementing RBAC

- Selected code from models/Group.inc

```
class Group extends Model {

   // ... stuff skipped...

   public function addPermission($permission) {
      $db = Db::getDb();
      $statement = $db -> prepare(
         "INSERT INTO grouppermissions (groupId, permissionId)
          VALUES (:groupId, :permissionId)");
      $statement -> bindValue(':groupId', $this -> getId());
      $statement -> bindValue(':permissionId',
          $permission -> getId());
      $statement -> execute();
   }
```

# Implementing RBAC

- Selected code from models/Group.inc

```
class Group extends Model {

   // ... stuff skipp

   public function add
      $db = Db::getDb(
      $statement = $db
         "INSERT INTO
          VALUES (:group
      $statement -> bindValue(':groupId', $this -> getId());
      $statement -> bindValue(':permissionId',
          $permission -> getId());
      $statement -> execute();
   }
```

There would be other code for removing members, removing positions, getting lists of members, getting lists of permissions, etc.

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    private $cache;
    private static $instance;

    private function __construct() {
        $cache = array();
    }

    public static function instance() {
        if (!isset(self::$instance)) {
            self::$instance = new Authenticator();
        }
        return self::$instance;
    }
```

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    private $cache;
    private static $instance;

    private function __construct() {
        $cache = array();
    }

    public static function instance() {
        if (!isset(self::$instance)) {
            self::$instance = new Authenticator();
        }
        return self::$instance;
    }
```

The Singleton pattern

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    public function can($permissionKey, $userId = false) {
        $userId = $this -> realUserId($userId);
        $permissions = $this -> permissionsFor($userId);
        foreach ($permissions as $permission) {
            if ($permission -> getName() === $permissionKey) {
                return true;
            }
        }
        return false;
    }
```

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    private function permissionsFor($userId) {
        if (!isset($this -> cache[$userId])) {
            $db = Db::getDb();
            $st = $db -> prepare(self::PERMS_QUERY);
            $st -> bindParam(':userId', $userId);
            $st -> execute();
            $this -> cache[$userId] = Permission::fromRows(
                $st -> fetchAll(PDO::FETCH_ASSOC));
        }
        return $this -> cache[$userId];
    }
```

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    const PERMS_QUERY =
        "SELECT DISTINCT permissions.id as id,
            permissions.name as name
         FROM
            users, usergroups, groups, grouppermissions,
            permissions
         WHERE
            users.id = :userId AND
            users.id = usergroups.userId AND
            usergroups.groupId = groups.id AND
            groups.id = grouppermissions.groupId AND
            grouppermissions.permissionId = permissions.id";
```

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    public function ensure($permissionKey, $userId = false) {
        if (!$this -> can($permissionKey, $userId)) {
            $userId = $this -> realUserId($userId);
            Logger::instance() -> warn(
                "User $userId attempted unauthorized " .
                "operation $permissionKey");
            die("You do not have permission to access this " .
                "resource.  This attempt has been logged.");
        }
    }
}
```

# Implementing RBAC

- Code from Lib/Authenticator.inc

```
class Authenticator {

    public function ensure($permissionKey, $userId = false) {
        if (!$this -> can($permissionKey, $userId)) {
            $userId = $this -> realUserId($userId);
            Logger::instance() -> warn(
                "User $us rId attempted unauthorized " .
                "operation $p mi
            die("You do not h
                "resource.  This a
        }
    }
}
```

Logging is an important aspect of security, providing an audit trail. Even successful operations may need to be logged.

# Using RBAC

- Inside controllers/todo.inc

```
function post_add($params) {
    Authenticator::instance() -> ensure('create_todo');

    $todo = safeParam($_REQUEST, 'todo', false);
    $todo = new Todo($todo);
    $validator = $todo -> validate();

    if (!$validator -> hasErrors()) {
        $todo -> insert();
    }
    redirectRelative("index");
}
```

# Using RBAC

- Inside controllers/todo.inc

```
function post_add($params) {
    Authenticator::instance() -> ensure('create_todo');

    $todo = safeParam($_REQUEST, 'todo', false);
    $todo = new Todo($todo);
    $validator = $todo -> validate();

    if (!$validator -> hasErrors()) {
        $todo -> insert();
    }
    redirectRelative("index");
}
```

ensure() method dies if the user doesn't have permission.

# Using RBAC

- Inside views/index.inc

```
[[ include_once ('include/Authenticator.inc'); ]]
%% views/header.html %%
<h1>{{$title}}</h1>

[[if (Authenticator::instance() -> can('create_todo')) : ]]
<form action="@@todo/add@@" method="post">
    <label for="description">Description:</label>
    <input type="text" id="description“
        name="todo[description]" />
    <input type="submit" value="Add" />
</form>
[[ endif; ]]

<h2>Current To Do:</h2>
<!-- remainder removed -->
```

# Using RBAC

- Inside views/index.inc

```
[[ include_once ('include/Authenticator.inc'); ]]
%% views/header.html %%
<h1>{{$title}}</h1>

[[if (Authenticator::instance() -> can('create_todo')) : ]]
<form action="@@todo/add@@" method="post">
    <label for="description">Description:</label>
    <input type="text" id="description“
        name="todo[description]" />
    <input type="submit" value="Add" />
</form>
[[ endif; ]]

<h2>Current To Do:</h2>
<!-- remainder removed -->
```

can() method just checks permissions (use for optional GUI element display)

# Using RBAC

- Some checks need more logic than the Authenticator provides.
  - Controllers/user.inc

```
function post_edit($params) {
    $user = safeParam($_REQUEST, 'user', false);
    $user = new User($user);

    if (!Authenticator::instance() -> can('edit_user')) {
        ensureLoggedInUserIs($user->getId());
    }

    // ...snip...
```

# Using RBAC

- More logic needed to bootstrap your application (e.g. creating groups/permissions)

```
class Db {
    public static function getDb() {
        try {
            $fileName = "ToDoList.db3";

            // see if we need to create tables
            $makeDb = !file_exists($fileName);
            $db = new PDO("sqlite:${fileName}");

            // force exceptions for better debugging.
            $db -> setAttribute(PDO::ATTR_ERRMODE,
                PDO::ERRMODE_EXCEPTION);
            // ...continued...
```

# Using RBAC

- Bootstrapping the DB

```
        // force cascading deletes on foreign keys
        $st = $db -> prepare("PRAGMA foreign_keys = ON");
        $st -> execute();

        if ($makeDb) {
            self::makeTables($db);
            self::populateTables($db);
        }

    } catch (PDOException $e) {
        die("Could not open database. " . $e ->
            getMessage());
    }
    return $db;
}
```

# Using RBAC

- Bootstrapping the DB

```
private static function makeTables(&$db) {
    Logger::instance() -> info("Creating tables");
    $statements = array(
        "CREATE TABLE users (
            id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
            email TEXT NOT NULL,
            password TEXT NOT NULL,
            firstName TEXT NOT NULL,
            lastName TEXT NOT NULL)"); // more DDL cut...
    $db -> beginTransaction();
    foreach ($statements as $statement) {
        $st = $db -> prepare($statement);
        $st -> execute();
    }
    $db -> commit();
}
```

# Using RBAC

- Bootstrapping the DB

```
private static function populateTables(&$db) {
    // create the permissions
    $userperms = array('create_todo', 'edit_todo',
        'delete_todo', 'view_todo');
    $adminperms = array('admin_page', 'edit_user',
        'delete_user', 'view_user');
    $permissions = array();
    $allperms = array($adminperms, $userperms);
    foreach ($allperms as $perms) {
        foreach ($perms as $name) {
            $p = new Permission( array('name' => $name));
            $p -> insert();
            $permissions[$name] = $p;
        }
    } // ...etc.
```

# Using RBAC

- Bootstrapping the DB
  - Must also
    - Create the group “Users”
    - Assign permissions to Users
    - Create the group “Administrators”
    - Assign permissions to Administrators
    - Create a super-user
    - Add the super-user to Administrators

# Using RBAC

- Bootstrapping the DB
  - Must also
    - Create the group “Users”
    - Assign permissions to Users
    - Create the group “Administrators”
    - Assign permissions to Administrators
    - Create a super-user
    - Add the super-user to Administrators

Why must we do all this?

# Using RBAC

- Bootstrapping the DB
  - Must also
    - Create the
    - Assign permi
    - Create the grou
    - Assign permiss
    - Create a super
    - Add the super-

A professional application has a mini-application built in just for walking the installing user through the bootstrap process.

# Show me the code!

- Mini-markdown and the full source code for authentication/authorization are available at http://cs.franklin.edu/~sharkesc/webd236/

# Upcoming Deadlines

Readings for next week

- Chapters 22 and 23 in *PHP and MySQL*

Assignments

- Homework 8 due end of week 10
- Lab 4 due end of week 12

Next week:

- File Uploads

# General Q & A

- Questions?
- Comments?
- Concerns?

FRANKLIN UNIVERSITY