Encrypting/Decrypting data

The Yii2 framework contains a special security component that provides a set of methods for handling  
common security-related tasks. The \yii\base\security class requires the openssL PHP extension  
instead of mcrypt.

Подготовка

1. Создайте новое приложение с помощью диспетчера пакетов Composer, как описано в официальном руководстве по адресу  
   <http://www.yiiframework.com/doc-2.0/guide-start-installation.html>.   
   По русски <http://yiiframework.domain-na.me/doc/guide/2.0/ru/start-installation>.
2. Set up the database connection and create a table named order, as follows:

DROP TABLE IF EXISTS 'order';

CREATE TABLE IF NOT EXISTS 'order' (

'id' INT(10) UNSIGNED NOT NULL AUTO\_INCREMENT,

'client' VARCHAR(255) NOT NULL,

'total' FLOAT NOT NULL,

'encrypted\_field' BLOB NOT NULL,

PRIMARY KEY ('id')

);

1. Generate an Order model using Gii.

How to do it...

1. Add an additional key parameter to config/params. php, as follows:

<?php  
return [

'adminEmail' => 'admin@example.com',

'key' => 'mysecretkey'

];

1. Add the behaviors and helper properties to the Order model as follows:

public $encrypted\_field\_temp;

public function behaviors()

{

return [

[

'class' => AttributeBehavior::className(),

'attributes' => [

ActiveRecord::EVENT\_BEFORE\_INSERT => 'encrypted\_field',  
ActiveRecord::EVENT\_BEFORE\_UPDATE => 'encrypted\_field',

],

'value' => function ($event) {

$event->sender->encrypted\_field\_temp = $event->sender-  
>encrypted\_field;

return Yii::$app->security->encryptByKey(  
$event->sender->encrypted\_field,

Yii::$app->params['key']

);

},

],

[

'class' => AttributeBehavior::className(),

'attributes' => [

ActiveRecord::EVENT\_AFTER\_INSERT => 'encrypted\_field',  
ActiveRecord::EVENT\_AFTER\_UPDATE => 'encrypted\_field',

],

'value' => function ($event) {

return $event->sender->encrypted\_field\_temp;

},

],

[

'class' => AttributeBehavior::className(),

'attributes' => [

ActiveRecord::EVENT\_AFTER\_FIND => 'encrypted\_field',

],

'value' => function ($event) {

return Yii::$app->security->decryptByKey(  
$event->sender->encrypted\_field,

Yii::$app->params['key']

);

},

],

];

}

3. Add controllers/CryptoController.php:

<?php

namespace app\controllers;

use app\models\Order;  
use Yii;

use yii\db\Query;

use yii\helpers\ArrayHelper;

use yii\helpers\Html;

use yii\helpers\VarDumper;

use yii\web\Controller;

/\*\*

* Class CryptoController.
* @package app\controllers  
  \*/

class CryptoController extends Controller  
{

public function actionTest()

{

$newOrder = new Order();

$newOrder->client = "Alex";

$newOrder->total = 100;

$newOrder->encrypted\_field = 'very-secret-info';

$newOrder->save();

$findOrder = Order::findOne($newOrder->id);

return $this->renderContent(Html::ul([

'New model: ' . VarDumper::dumpAsString($newOrder->attributes),

'Find model: ' . VarDumper::dumpAsString($findOrder->attributes)

]));

}

public function actionRaw()

{

$row = (new Query())->from('order')

->where(['client' => 'Alex'])

->one();

return $this->renderContent(Html::ul(  
$row

));

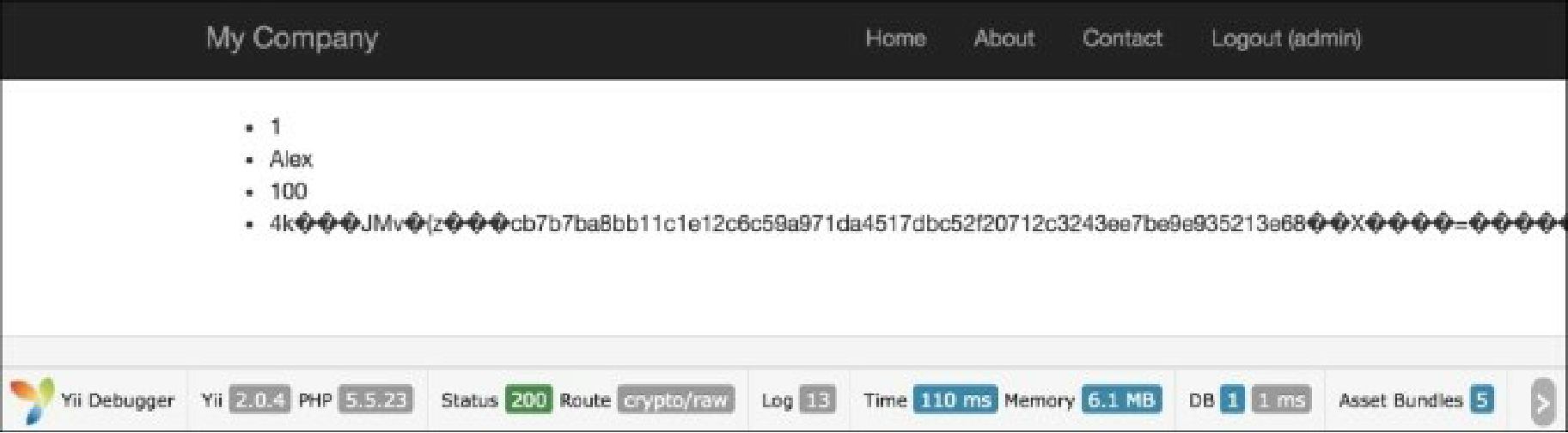
}

}

4. Run crypto/test and you will get the following:



5. To view raw data, run crypto/raw:



How it works...

Firstly, we have added the AttributeBehavior, which automatically processes our data when certain  
events happen. Our certain events are ActiveRecord: : event\_after\_insert,

ActiveRecord::EVENT\_AFTER\_UPDATE and ActiveRecord::EVENT\_AFTER\_FIND.

During insert and update events, we decrypt our data with a special method: Yii: : $app->security-  
>encryptByKey();. This method uses HKDF and a random salt to decrypt our data before saving it to the  
database. After getting data from the database, we can also use the ActiveRecord: : event\_after\_find  
method to decrypt our data. In this case, we also use the special Yii2 method Yii: :$app->security-  
>encryptByKey(); .This method accepts two params: encrypted data and key.

There’s more...

Besides data encryption and data decryption, a secure component also provides key derivation using  
standard algorithms, data tampering prevention, and password validation.

Working with passwords

Verifying a password:

if (Yii::$app->getSecurity()->validatePassword($password, $hash)) {

// all good, logging user in  
} else {

// wrong password

}

See also

In order to learn more about SQL injections and working with databases through Yii, refer to  
<http://www.yiiframework.com/doc-2.0/guide-security-passwords.html>