Pro PHP: Patterns, Frameworks, Testing and More

ISBN-13 (pbk): 978-1-59059-819-1

ISBN-10 (pbk): 1-59059-819-9

ISBN-13 (electronic): 978-1-4302-0279-0

ISBN-10 (electronic): 1-4302-0279-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Jason Gilmore, Tom Welsh
Technical Reviewer: Jeffrey Sambells
Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Kevin Goff, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Beth Christmas
Copy Editor: Marilyn Smith
Associate Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Susan Glinert
Proofreader: Lisa Hamilton
Indexer: Broccoli Information Management
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at `http://www.apress.com/info/bulksales`.

The source code for this book is available to readers at `http://www.apress.com`.

CHAPTER 21

Certificate Authentication

Certificate authentication is one of the most powerful methods for securing your web applications and services. It is also one of the most underutilized authentication mechanisms available to PHP developers, probably because it is significantly more complex to set up and manage than any of the other authentication mechanisms. The APIs can be picky, the certificate-generation process can be complicated, and certificates must be deployed to clients. Yet certificate authentication is definitely worth the effort, as your site security will be markedly improved.

SSL certificates are typically used in a one-way validation mechanism, allowing a client to determine that a server is who it says it is. With certificate authentication, this process is expanded, with the server verifying the client's identity.

In this chapter, you will learn how to set up certificate authentication from start to finish, using your very own certificate authority.

Public Key Infrastructure Security

Certificate authentication is a public key infrastructure (PKI) security mechanism. The basic concepts in PKI security for web applications and services include certificate authorities (CAs), web server certificates, client-side certificates, and the root CA certificate.

Certificate Authority

A CA is an organization that is responsible for issuing and revoking certificates for third parties. It is the responsibility of the CA to verify the identity of certificate users. Thus, it is exclusively the CA that is enabled to communicate this authority to the web servers and clients that operate with your certificates.

Most modern web browsers have a list of trusted CAs that are enabled by default. These are the guys you need to pay money to when you want to set up an SSL-enabled web site for public consumption. When interacting with client-side certificates, however, you are the CA, as it would be foolish to trust a public CA to control access to your web application.

This presents a few challenges:

- You must set up the infrastructure required to be a CA.
- You must configure all your software to use your CA to verify client certificates.
- You must handle issuing the client certificate.

Technically, your CA is just another self-signed certificate, which is not inherently trusted by any browser or client. This means that, unlike a commercial certificate, which is already trusted, you will need to distribute your CA. Fortunately, you can embed your CA certificate within the client certificate itself before it is installed into the web browser.

Web Server Certificate

To set up client authentication, you will first need an SSL-enabled web site. Luckily, you don't need a commercial SSL certificate for this server.

A commercial certificate is not required because, when you send your client a certificate for use in authentication, you also provide it with the public CA certificate that was used to sign your server. By exchanging keys in this way, you and your client can verify that each other's computers are actually talking to each other, and not an intermediary web site or client. This process is called *peer verification*.

■ **Caution** Verifying the peer certificate using the shared CA is a critically important step in the trust relationship between client and server. Many tutorials do not explain how to properly enable peer verification. Improper peer verification will greatly reduce your security effectiveness.

Client Certificate

A client certificate consists of three parts: a private key, the client certificate, and the CA's certificate. To be imported into a browser, these three parts are rolled into one `.p12` file, which will be password-protected. Files with the `.p12` extension are in the PKCS 12 format, which is the Public Key Cryptography Standard 12, Personal Information Exchange Syntax Standard.

You have two main options for creating this archive:

- You can create all three parts yourself and send your client the final file.
- You can have your client generate a certificate-signing request (CSR) file, which you will sign and give back to the client, along with your public CA certificate. The client would then be responsible for building the `.p12` file.

The benefit of having the client generate a CSR file is that it will not share its private key with you, and you will not need to transmit an archive containing a private key between the client and server.

The entire security model of SSL depends on the private keys remaining private, and as such, the fewer locations in which a private key exists, the better. Ideally, a private key should exist on only one machine and be known to a single party; however, this may not always be possible.

Root CA Certificate

The pinnacle of the CA is the *root CA certificate*. This certificate is extremely important, and you must be very careful with it.

If your root CA certificate is lost, you will not be able to generate any new certificates without redeploying every existing certificate you have ever deployed with the CA. Worse, if your root

CA private key is compromised, you have the worst possible scenario: your entire application is now insecure, and any user can be impersonated.

■ **Caution** If I've not made it crystal clear, do not put your root CA private key on your web server! The only time you need the CA key is when signing a new certificate.

Setting Up Client Certificate Authentication

This chapter's example uses OpenSSL, an open source implementation of SSL (tested on a Debian "etch" Linux release with the default packages). The nature of OpenSSL is that things are usually in the right place, but please take a minute to confirm that your `CA.pl` and `openssl.cnf` files are the following locations. Also, check that your version is at least 0.9.8:

```
> locate CA.pl

/usr/lib/ssl/misc/CA.pl

> locate openssl.cnf

/usr/lib/ssl/openssl.cnf

> openssl version

OpenSSL 0.9.8g 19 Oct 2007
```

If these values check out, the next step is to proceed to create the CA.

Creating Your Own Certificate Authority

Create your very own CA by executing the following commands:

```
> cd /usr/lib/ssl/misc
> ./CA.pl -newca
```

```
CA certificate filename (or enter to create)

Making CA certificate ...
Generating a 1024 bit RSA private key
.....................................................++++++
.........++++++
writing new private key to './demoCA/private/cakey.pem'
Enter PEM pass phrase: <Enter a _strong_ password>
Verifying - Enter PEM pass phrase: <confirm password>

-----
```

```
You are about to be asked to enter information that will be incorporated
into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

-----

Country Name (2 letter code): CA
State or Province Name (full name): Alberta
Locality Name (eg, city):
Organization Name (eg, company): Kevin McArthur
Organizational Unit Name (eg, section):
Common Name (eg, YOUR name): Kevin McArthur Root CA
Email Address:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem: <type pass again>

Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            c8:ff:56:7b:d8:e3:18:64
        Validity
            Not Before: May 13 01:52:18 2007 GMT
            Not After : May 12 01:52:18 2010 GMT
        Subject:
            countryName               = CA
            stateOrProvinceName       = Alberta
            organizationName          = Kevin McArthur
            commonName                = Kevin McArthur Root CA
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                91:FB:90:02:A0:76:3E:21:02:FE:B6:97:4C:3C:99:B5:79:63:90:1D
            X509v3 Authority Key Identifier:
                keyid:91:FB:90:02:A0:76:3E:21:02:FE:B6:97:4C:3C:99:B5:79:63:90:1D
                DirName:/C=CA/ST=Alberta/O=Kevin McArthur/CN=Kevin McArthur Root CA
                serial:C8:FF:56:7B:D8:E3:18:64
```

```
            X509v3 Basic Constraints:
                CA:TRUE <Confirm this reads TRUE!>

Certificate is to be certified until May 12 01:52:18 2010 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
```

■ **Caution** Make sure `X509V3 Basic Constraints: CA:TRUE` is set. If it is not, your entire CA will not work as expected for peer verification! Some versions of OpenSSL ship with a `CA.sh` script that appears to do the same thing as the `CA.pl script`, but it will *not* set this attribute.

This operation creates a new directory structure `demoCA` and generates your root CA certificate. This certificate is contained in `demoCA/cacert.pem`. The private key, which must be protected, is placed in `demoCA/private/cakey.pem`. The `cacert.pem` file may be shared, as it contains only public-key information.

Next, you will use this certificate to set up an Apache 2.*x* web server for SSL operation using a self-signed certificate.

Create a Self-Signed Web Server Certificate

Creating a web server certificate using the `CA.pl` script and your new CA certificate is trivial. Execute the following command:

```
> ./CA.pl -newreq
```

```
Generating a 1024 bit RSA private key
..................++++++
.............++++++
writing new private key to 'newkey.pem'
Enter PEM pass phrase: <password>
Verifying - Enter PEM pass phrase: <password>

-----

You are about to be asked to enter information that will be incorporated into
your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

-----
```

```
Country Name (2 letter code): CA
State or Province Name (full name): Alberta
Locality Name (eg, city):
Organization Name (eg, company): Kevin McArthur
Organizational Unit Name (eg, section): Web Server
Common Name (eg, YOUR name): localhost
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Request is in newreq.pem, private key is in newkey.pem
```

■ **Caution** The `Common Name` entry must match your web server domain exactly. I've used `localhost`, but you should use your domain name if you are setting up SSL for non-`localhost` operation.

At this point, you have a certificate request and a new private key for the server. You have not yet signed this certificate as being authentic according to your CA. To sign this certificate, execute the following command:

```
> ./CA.pl -sign
```

```
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem: <enter the CA pass>
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            a7:8f:54:aa:74:66:29:4f
        Validity
            Not Before: May 15 02:45:28 2007 GMT
            Not After : May 14 02:45:28 2008 GMT
        Subject:
            countryName               = CA
            stateOrProvinceName       = Alberta
            organizationName          = Kevin McArthur
            organizationalUnitName    = Web Server
            commonName                = localhost
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

```
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                46:40:6A:B4:56:B6:73:3A:5B:F8:0F:89:C2:89:AD:3D:07:99:52:2A
            X509v3 Authority Key Identifier:
                keyid:0F:3C:EF:06:9D:10:7B:17:81:A9:E5:74:4F:B4:72:1D:C4:4E:22:E2

Certificate is to be certified until May 14 02:45:28 2008 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
Signed certificate is in newcert.pem
```

Now you have several files, including one that contains a signed certificate. Since `CA.pl` overwrites the files it creates when it is rerun, you need to move these files to files with unique names. Execute the following commands:

```
> mv newcert.pem server.pem
> mv newkey.pem server.key
```

Currently, you have your web server certificate in `server.pem` and the private key in `server.key`. The only problem here is that your `server.key` file is encrypted with a pass phrase, and you will need to enter the password every time the server starts. If you don't want this to happen, you can decrypt the key and write it out without a password, so you do not need to type a password to start Apache. To decrypt the key, execute the following command:

```
> openssl rsa < server.key > serverkey.pem
```

```
Enter pass phrase: <password>
writing RSA key
```

Now you need to set up Apache with your site.

Configuring Apache for SSL

Start your Apache setup by making a new configuration directory for your SSL certificates:

```
> mkdir /etc/apache2/ssl
```

Then move your server certificates in there.

```
> mv server.* /etc/apache2/ssl
```

■ **Note** This step assumes a Debian layout. Your server layout may be slightly different.

Later on, your web server will need to know about your CA certificate, so create a symbolic link to that certificate:

```
> ln -s demoCA/cacert.pem /etc/apache2/ssl/cacert.pem
```

To set up Apache for SSL operation, you need to edit the configuration file. On some systems, this is `httpd.conf`; on others, it has a different name. On Debian systems, you will find the configuration file in `/etc/apache2/sites-available/default`.

In your Apache configuration file, you need to enable the `SSLEngine`, set the Cipher Suite, and point Apache to your new certificate files. You'll do this in the main virtual host section, which will probably look like this:

```
<VirtualHost *>
. . .
</VirtualHost>
```

Add the code shown in Listing 21-1 to your `VirtualHost` section in your Apache configuration file.

Listing 21-1. *SSL Configuration in Apache Configuration File*

```
SSLEngine on
SSLCipherSuite
ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:!eNULL

SSLCertificateFile /etc/apache2/ssl/server.pem
SSLCertificateKeyFile /etc/apache2/ssl/serverkey.pem
SSLCACertificateFile /etc/apache2/ssl/cacert.pem
```

After you've saved your modified configuration file, restart Apache and point your web browser at `https://localhost` (or the site for which you have configured SSL). If everything is working, you should get an SSL warning saying that the certificate is not trusted. Accept the certificate *temporarily*; do not accept it permanently.

This certificate should be untrusted because it is not signed by a CA that your browser recognizes. Usually, you would pay an SSL company like Go Daddy or VeriSign to sign your certificate, but because you are setting up this site for client-side certificates, that step is redundant and unnecessary.

Creating the Client-Side Certificates

Creating the client-side certificates is slightly more complicated because they must be encoded in the PKCS 12 format. It is also trickier because you have a choice to make. You must decide if you will generate your clients' private keys for them, or if you will ask them to create their own private keys and send you only a CSR file.

First, you or your client will need to determine if the `openssl.cnf` file you located earlier is configured correctly to create a client-side certificate. This requires opening the file, locating a section, and confirming that it looks like Listing 21-2. If it doesn't, you should add the code in Listing 21-2 to the `openssl.cnf` file.

Listing 21-2. *Client-Side Certificate Creation Configuration (in openssl.cnf)*

```
[ ssl_client ]
basicConstraints        = CA:FALSE
nsCertType              = client
keyUsage                = digitalSignature, keyEncipherment
extendedKeyUsage        = clientAuth
nsComment               = "OpenSSL Certificate for SSL Client"
```

Creating client-side certificates requires working with the raw `openssl` commands, which `CA.pl` was previously doing for you. The first step is to create a new CSR and private key for your client. If your client is generating the CSR, it should execute this command and send you the resulting `client.pem`, while keeping the `client.key` private.

```
> openssl req -new -sha1 -newkey rsa:1024 -keyout client.key -out client.pem \
> -subj '/O=Kevin McArthur/OU=Kevin McArthur Web Services/CN=Joe Smith'
```

The `subj` parameters can be slightly confusing. The `O` parameter stands for Organization, `OU` is for Organizational Unit, and `CN` is for Common Name. The `O` and `OU` parameters must be the same for every client, and the `CN` must be distinct. This is because the `O` and `OU` fields will be used by Apache, along with the authority given by your CA certificate, to determine who may access a resource. Changing the `OU` field will allow you to create different "zones" of access.

Once you have the `client.pem`, either generated by this command or received from your client, you need to sign it with your CA. This signature is what your web server will use to trust that the `O` and `OU` fields are actually valid. In this step, you are certifying these values, so be sure that they are correct!

To sign the certificate, execute the following command:

```
> openssl ca -config /usr/lib/ssl/openssl.cnf -policy policy_anything \
> -extensions ssl_client -out client.signed.pem -infiles client.pem
```

```
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem: <enter CA password>
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number:
            a7:8f:54:aa:74:66:29:50
        Validity
            Not Before: May 15 03:19:04 2007 GMT
            Not After : May 14 03:19:04 2008 GMT
```

```
        Subject:
            organizationName          = Kevin McArthur
            organizationalUnitName    = Kevin McArthur Web Services
            commonName                = Joe Smith
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Cert Type:
                SSL Client
            X509v3 Key Usage:
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
            Netscape Comment:
                OpenSSL Certificate for SSL Client
Certificate is to be certified until May 14 03:19:04 2008 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

■ **Caution** It is essential in this step to verify the O and OU information. Do not blindly sign the certificate.

You will now have a `client.signed.pem` certificate. Now, if you are doing this for a client, you will need to send this certificate and your `cacert.pem` file back to the client. In any case, these two certificates will be combined with the private key into a PKCS 12 certificate.

To create the PKCS 12 certificate, execute the following command:

```
> openssl pkcs12 -export -clcerts -in client.signed.pem -inkey client.key \
> -out client.p12 -certfile demoCA/cacert.pem -name "Joe Smith"
```

```
Enter pass phrase for client.key: <client's key pass>
Enter Export Password: <a new pass for the p12 archive>
Verifying - Enter Export Password: <confirm>
```

You now have the final `.p12` file. The trick now is exchanging this file. If your client created the `.p12` file, you don't need to exchange it; however, if you created it, a security issue presents itself.

The file contains a private key, which is really important to keep secure. At this point, it is password-protected, and presumably you used a password that is hard to crack, so it would be difficult to use the file if it were intercepted. Still, if intercepted, this archive might be opened by brute force.

My advice is to communicate the file to your client securely, either via an encoded Secure/Multipurpose Internet Mail Extensions (S/MIME) e-mail message, Secure Copy/Secure FTP (SCP/SFTP), or on physical media of some sort. Then, separately, via a different medium, such

as a phone call or letter, communicate the password that was used to encrypt the archive. For extra security, consider deploying smart cards or tokens, so that private keys are not left on client machines while not in use.

■ **Caution** The absolute worst thing you could do with certificate authentication is to send the `.p12` file along with the archive password via standard e-mail. Not only can this defeat the mechanism if the data is intercepted, but it may give you a false sense of security. Under no circumstances should you transfer the file and communicate the password via the same medium.

The `.p12` file will work with web browsers and Subversion clients, but it will not work with PHP. To use SSL authentication from a PHP script, such as a web service for opening remote files, you will need to use the PEM-encoded certificates. To get that working, you will need to merge the client private key with the client certificate file. Execute the following commands:

```
> mkdir /var/www/ssl
> mv /usr/lib/ssl/misc/client.* /var/www/ssl/
> cd /var/www/ssl
> cat client.signed.pem > services.pem
> cat client.key >> services.pem
```

You will now have a `services.pem` with both the signed client certificate and the client's private key in it.

The next step in creating your authentication mechanism is to allow access only to SSL-authenticated clients.

Permitting Only Certificate Authentication

Allowing access only to SSL-authenticated clients requires editing your Apache configuration files again. Place the code shown in Listing 21-3 in your configuration file, adjusting the `location` tag as necessary, and then reload Apache.

Listing 21-3. *SSL Restrictions*

```
SSLVerifyClient require
SSLVerifyDepth 1

SSLOptions +StrictRequire +StdEnvVars

<Location />
  SSLRequireSSL
  SSLRequire %{SSL_CLIENT_VERIFY} eq "SUCCESS"
  SSLRequire %{SSL_CLIENT_S_DN_O} eq "Kevin McArthur"
  SSLRequire %{SSL_CLIENT_S_DN_OU} eq "Kevin McArthur Web Services"
</Location>
```

This code tells Apache to require the client to present an SSL certificate. It then says it must be verified by an immediate CA certificate. The `1` represents one degree of separation, which means that the certificate must be directly signed by the `SSLCACertificateFile` that you specified earlier.

Next, the `SSLOptions` `StrictRequire` option overrides any of the Apache `Satisfy` rules, and the `StdEnvVars` creates some environment variables you will use with `SSLRequire`.

The `SSLRequire` statements state that SSL is required, that the client must verify successfully against the CA, and that the `O` and `OU` properties must match the provided values. If any of these properties are not satisfied, access will be denied to the location. By varying the values for the `O` and `OU` fields in this configuration and in your client certificates, you can create different areas of access.

Reload your web server, and proceed to testing.

Testing the Certificate

Now it's time to test the certificates. Web browsers will require the `.p12` certificate.

For Internet Explorer, you will need to execute the `.p12` file by double-clicking it and following along with the on-screen instructions, accepting the default options.

In Firefox, you will need to configure the certificate, as follows:

1. Select Edit ➤ Preferences (Linux) or Tools ➤ Options (Windows) from the menu bar.
2. Select the Advanced tab, and then the Encryption subtab. On this tab, click View Certificates.
3. Click Import. Locate your p12 file, and then click Import. You will be prompted for the export password you specified earlier.
4. After the certificate is imported, click the Authorities tab. Find and select your root CA certificate, and then click Edit.
5. Check the box that says "This certificate can identify web sites" and click OK.

You have configured your client to provide the client certification when requested and that your root CA can sign certificates to identify web sites. It is this latter step that makes a commercial CA redundant for SSL sites with client certificates.

Visit `https://localhost`. You may be prompted for a password for your certificate store if one is set and it is your first login this session. You should not receive any warnings about the site, as you did during your initial visit. Your web server now knows that the client it is talking to has the certificate, and the client knows that the server is who it says it is because its SSL certificate matches the one contained in the `.p12` file.

Everything is now very secure, but everything has occurred at the web server-level, and it has verified only the `O` and `OU` fields, and not the `CN` field. Now you probably will want to implement some sort of PHP authentication control that inspects the `CN` field.

PHP Authentication Control

You will find a complete list of `SSL_CLIENT_*` variables in your `$_SERVER` variable. You can use the information from these variables to identify clients by name, and since you certified the certificate, you can be sure that the user has access to that certificate and does not just know a username and password.

That said, certificate authentication guarantees only that the user has access to the `.p12` file; it does not confirm identity. Therefore, you will want to use multiple authentication controls, such as adding a username and password login, for sensitive operations.

Binding PHP to a Certificate

Once you are communicating with your server via certificate authentication, a script like the one shown in Listing 21-4 can bind PHP to the presented certificate. In this case, you will access the Common Name field from the SSL certificate. This data can be trusted because Apache has already verified the certificate with your CA certificate.

Listing 21-4. *PHP and SSL Interaction*

```
<?php

//The user's name is stored in the certificate's Common Name field.
echo "Hello, ". $_SERVER['SSL_CLIENT_S_DN_CN'];
```

```
Hello, Joe Smith
```

Setting Up Web Service Authentication

PHP web services authenticate using client certificates by setting transport options with the HTTPS protocol wrapper. This is commonly known as a *stream context*, and allows you to provide advanced options to stream-interacting calls like `fopen` and the `SoapClient` class. Stream contexts have a lot of other options; however, for this purpose, you will need to use only the SSL subset of options.

Continuing from the examples in the previous chapter, you will need to change your `PhoneCompany.wsdl` file to use HTTPS instead of HTTP for `soap:address`, as shown in Listing 21-5. You don't need to change the namespace or any other options.

Listing 21-5. *Switching the WSDL Port to Bind with HTTPS (in PhoneCompany.wsdl)*

```
  </binding>

  <service name="PhoneCompanyService">
    <port
      name="PhoneCompanyPort"
      binding="PhoneCompanyBinding"
```

```
    >
      <soap:address location="https://localhost/PhoneCompany.php"/>
    </port>
  </service>
</definitions>
```

Next, add the code in Listing 21-6 to your `PhoneClient.php` script, replacing the `$client` assignment.

Listing 21-6. *Using a SoapClient with an SSL Stream Context (in PhoneClient.php)*

```
$contextDetails = array(
  'ssl'=> array(
    'local_cert'=>'/var/www/ssl/services.pem',
    'cafile'=>'/usr/lib/ssl/misc/demoCA/cacert.pem',
    'verify_peer'=>true,
    'allow_self_signed'=>false,
    'CN_match'=>'localhost',
    'passphrase'=>'password'
  )
);

$streamContext = stream_context_create($contextDetails);

$client = new SoapClient(
                'PhoneCompany.wsdl',
                array(
                  'classmap'=>$classmap,
                  'stream_context'=>$streamContext,
                )
              );
```

This code tells `SoapClient` to use the `services.pem` certificate you created earlier and also enables peer verification. As in the browser, this ensures that the remote server is not being impersonated and that the server is who it says it is. This can prevent many DNS poisoning and man-in-the-middle attacks, so it's very important that you use peer verification in any deployed application.

Without peer verification, the SOAP client would happily talk to any web server that presented any SSL certificate; it would not know, or care, who it is talking to and would proceed to send the SOAP envelopes without any verification. This would make your application's security entirely dependent on DNS and the network infrastructure between your client and server. With peer verification, only the server that has a certificate signed by the CA will be considered acceptable to the SOAP client.

The `CN_match` field is part of this peer verification, and ensures that the certificate presented by the remote server contains a Common Name (`CN`) field that matches the value you expect. This value will typically be the same as the web server address.

The `passphrase` option is the password for the `client.key` file you created earlier. This is because the `services.pem` file incorporates the encrypted private key.

With this code in place and the WSDL updated, you should now be able to call the web service through a secure tunnel. The `PhoneCompany.php` file does not need to be modified, as it performs no additional authentication. If you do want to authenticate the client's SSL parameters, you can use the `$_SERVER` approach presented in the previous section.

Just the Facts

The techniques presented in this chapter aren't for the faint of heart, but if mastered, they will provide your applications and clients with an added level of security.

The basic concepts in PKI security include CAs, web server certificates, and client-side certificates. Verifying the peer certificate using your shared CA is a critically important step in the trust relationship between client and server. Peer verification is important to preventing DNS poisoning and man-in-the-middle attacks from affecting your web applications.

In setting up an application for client authentication, you create your own CA. You use this CA to generate a self-signed web server certificate, and configure Apache to use this certificate.

You can issue a client certificate, in multiple formats, and deploy these certificates in your web browser and web services clients.

At the administration level, you limit access to Apache locations by inspecting the properties in client-side certificates.