NAME

SSL_CTX_set_tmp_dh_callback, SSL_CTX_set_tmp_dh, SSL_set_tmp_dh_callback, SSL_set_tmp_dh - handle DH keys for ephemeral key exchange

SYNOPSIS

```
#include <openssl/ssl.h>

void SSL_CTX_set_tmp_dh_callback(SSL_CTX *ctx,
DH *(*tmp_dh_callback)(SSL *ssl, int is_export, int keylength));
long SSL_CTX_set_tmp_dh(SSL_CTX *ctx, DH *dh);

void SSL_set_tmp_dh_callback(SSL *ctx,
DH *(*tmp_dh_callback)(SSL *ssl, int is_export, int keylength));
long SSL_set_tmp_dh(SSL *ssl, DH *dh)
```

DESCRIPTION

SSL_CTX_set_tmp_dh_callback() sets the callback function for **ctx** to be used when a DH parameters are required to **tmp_dh_callback**. The callback is inherited by all**ssl** objects created from **ctx**.

SSL_CTX_set_tmp_dh() sets DH parameters to be used to be **dh**. The key is inherited by all **ssl** objects created from **ctx**.

SSL_set_tmp_dh_callback() sets the callback only for **ssl**.

SSL_set_tmp_dh() sets the parameters only for **ssl**.

These functions apply to SSL/TLS servers only.

NOTES

When using a cipher with RSA authentication, an ephemeral DH key exchange can take place. Ciphers with DSA keys always use ephemeral DH keys as well. In these cases, the session data are negotiated using the ephemeral/temporary DH key and the key supplied and certified by the certificate chain is only used for signing. Anonymous ciphers (without a permanent server key) also use ephemeral DH keys.

Using ephemeral DH key exchange yields forward secrecy, as the connection can only be decrypted, when the DH key is known. By generating a temporary DH key inside the server application that is lost when the application is left, it becomes impossible for an attacker to decrypt past sessions, even if he gets hold of the normal (certified) key, as this key was only used for signing.

In order to perform a DH key exchange the server must use a DH group (DH parameters) and generate a DH key. The server will always generate a new DH key during the negotiation.

As generating DH parameters is extremely time consuming, an application should not generate the parameters on the fly but supply the parameters. DH parameters can be reused, as the actual key is newly generated during the negotiation. The risk in reusing DH parameters is that an attacker may specialize on a very often used DH group. Applications should therefore generate their own DH parameters during the installation process using the openssl *dhparam(1)* application. This application guarantees that ‘‘strong’’ primes are used.

Files dh2048.pem, and dh4096.pem in the ’apps’ directory of the current version of the OpenSSL distribution contain the ’SKIP’ DH parameters, which use safe primes and were generated verifiably pseudo-randomly. These files can be converted into C code using the **-C** option of the *dhparam(1)* application. Generation of custom DH parameters during installation should still be preferred to stop an attacker from specializing on a commonly used group. File dh1024.pem contains old parameters that must not be used by applications.

An application may either directly specify the DH parameters or can supply the DH parameters via a callback function.

Previous versions of the callback used **is_export** and **keylength** parameters to control parameter generation for export and non-export cipher suites. Modern servers that do not support export ciphersuites are advised to either use *SSL_CTX_set_tmp_dh()* or alternatively, use the callback but ignore **keylength** and **is_export**

and simply supply at least 2048-bit parameters in the callback.

EXAMPLES

Setup DH parameters with a key length of 2048 bits. (Error handling partly left out.)

```
Command-line parameter generation:
$ openssl dhparam -out dh_param_2048.pem 2048

Code for setting up parameters during server initialization:

...
SSL_CTX ctx = SSL_CTX_new();
...

/* Set up ephemeral DH parameters. */
DH *dh_2048 = NULL;
FILE *paramfile;
paramfile = fopen("dh_param_2048.pem", "r");
if (paramfile) {
dh_2048 = PEM_read_DHparams(paramfile, NULL, NULL, NULL);
fclose(paramfile);
} else {
/* Error. */
}
if (dh_2048 == NULL) {
/* Error. */
}
if (SSL_CTX_set_tmp_dh(ctx, dh_2048) != 1) {
/* Error. */
}
...
```

RETURN VALUES

SSL_CTX_set_tmp_dh_callback() and *SSL_set_tmp_dh_callback()* do not return diagnostic output.

SSL_CTX_set_tmp_dh() and *SSL_set_tmp_dh()* do return 1 on success and 0 on failure. Check the error queue to find out the reason of failure.

SEE ALSO

ssl(3), *SSL_CTX_set_cipher_list(3)*, *SSL_CTX_set_options(3)*, *ciphers(1)*, *dhparam(1)*

COPYRIGHT