NAME

SSL_CTX_set_srp_username, SSL_CTX_set_srp_password, SSL_CTX_set_srp_strength, SSL_CTX_set_srp_cb_arg, SSL_CTX_set_srp_username_callback, SSL_CTX_set_srp_client_pwd_callback, SSL_CTX_set_srp_verify_param_callback, SSL_set_srp_server_param, SSL_set_srp_server_param_pw, SSL_get_srp_g, SSL_get_srp_N, SSL_get_srp_username, SSL_get_srp_userinfo - SRP control operations

SYNOPSIS

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

The following functions have been deprecated since OpenSSL 3.0, and can be hidden entirely by defining **OPENSSL_API_COMPAT** with a suitable version value, see **openssl_user_macros**(7):

```
int SSL_CTX_set_srp_username(SSL_CTX *ctx, char *name);
int SSL_CTX_set_srp_password(SSL_CTX *ctx, char *password);
int SSL_CTX_set_srp_strength(SSL_CTX *ctx, int strength);
int SSL_CTX_set_srp_cb_arg(SSL_CTX *ctx, void *arg);
int SSL_CTX_set_srp_username_callback(SSL_CTX *ctx,
                                      int (*cb) (SSL *s, int *ad, void *arg));
int SSL_CTX_set_srp_client_pwd_callback(SSL_CTX *ctx,
                                        char *(*cb) (SSL *s, void *arg));
int SSL_CTX_set_srp_verify_param_callback(SSL_CTX *ctx,
                                          int (*cb) (SSL *s, void *arg));

int SSL_set_srp_server_param(SSL *s, const BIGNUM *N, const BIGNUM *g,
                             BIGNUM *sa, BIGNUM *v, char *info);
int SSL_set_srp_server_param_pw(SSL *s, const char *user, const char *pass,
                                const char *grp);

BIGNUM *SSL_get_srp_g(SSL *s);
BIGNUM *SSL_get_srp_N(SSL *s);

char *SSL_get_srp_username(SSL *s);
char *SSL_get_srp_userinfo(SSL *s);
```

DESCRIPTION

All of the functions described on this page are deprecated. There are no available replacement functions at this time.

These functions provide access to SRP (Secure Remote Password) parameters, an alternate

authentication mechanism for TLS. SRP allows the use of usernames and passwords over unencrypted channels without revealing the password to an eavesdropper. SRP also supplies a shared secret at the end of the authentication sequence that can be used to generate encryption keys.

The SRP protocol, version 3 is specified in RFC 2945. SRP version 6 is described in RFC 5054 with applications to TLS authentication.

The **SSL_CTX_set_srp_username()** function sets the SRP username for **ctx**. This should be called on the client prior to creating a connection to the server. The length of **name** must be shorter or equal to 255 characters.

The **SSL_CTX_set_srp_password()** function sets the SRP password for **ctx**. This may be called on the client prior to creating a connection to the server. This overrides the effect of **SSL_CTX_set_srp_client_pwd_callback()**.

The **SSL_CTX_set_srp_strength()** function sets the SRP strength for **ctx**. This is the minimal length of the SRP prime in bits. If not specified 1024 is used. If not satisfied by the server key exchange the connection will be rejected.

The **SSL_CTX_set_srp_cb_arg()** function sets an extra parameter that will be passed to all following callbacks as **arg**.

The **SSL_CTX_set_srp_username_callback()** function sets the server side callback that is invoked when an SRP username is found in a ClientHello. The callback parameters are the SSL connection **s**, a writable error flag **ad** and the extra argument **arg** set by **SSL_CTX_set_srp_cb_arg()**. This callback should setup the server for the key exchange by calling **SSL_set_srp_server_param()** with the appropriate parameters for the received username. The username can be obtained by calling **SSL_get_srp_username()**. See **SRP_VBASE_init**(3) to parse the verifier file created by **openssl-srp**(1) or **SRP_create_verifier**(3) to generate it. The callback should return **SSL_ERROR_NONE** to proceed with the server key exchange, **SSL3_AL_FATAL** for a fatal error or any value < 0 for a retryable error. In the event of a **SSL3_AL_FATAL** the alert flag given by ***al** will be sent back. By default this will be **SSL_AD_UNKNOWN_PSK_IDENTITY**.

The **SSL_CTX_set_srp_client_pwd_callback()** function sets the client password callback on the client. The callback parameters are the SSL connection **s** and the extra argument **arg** set by **SSL_CTX_set_srp_cb_arg()**. The callback will be called as part of the generation of the client secrets. It should return the client password in text form or NULL to abort the connection. The resulting memory will be freed by the library as part of the callback resolution. This overrides the effect of **SSL_CTX_set_srp_password()**.

The **SSL_CTX_set_srp_verify_param_callback()** sets the SRP gN parameter verification callback on the client. This allows the client to perform custom verification when receiving the server SRP proposed parameters. The callback parameters are the SSL connection **s** and the extra argument **arg** set by **SSL_CTX_set_srp_cb_arg()**. The callback should return a positive value to accept the server parameters. Returning 0 or a negative value will abort the connection. The server parameters can be obtained by calling **SSL_get_srp_N()** and **SSL_get_srp_g()**. Sanity checks are already performed by the library after the handshake (B % N non zero, check against the strength parameter) and are not necessary. If no callback is set the g and N parameters will be checked against known RFC 5054 values.

The **SSL_set_srp_server_param()** function sets all SRP parameters for the connection **s**. **N** and **g** are the SRP group parameters, **sa** is the user salt, **v** the password verifier and **info** is the optional user info.

The **SSL_set_srp_server_param_pw()** function sets all SRP parameters for the connection **s** by generating a random salt and a password verifier. **user** is the username, **pass** the password and **grp** the SRP group parameters identifier for **SRP_get_default_gN**(3).

The **SSL_get_srp_g()** function returns the SRP group generator for **s**, or from the underlying SSL_CTX if it is NULL.

The **SSL_get_srp_N()** function returns the SRP prime for **s**, or from the underlying SSL_CTX if it is NULL.

The **SSL_get_srp_username()** function returns the SRP username for **s**, or from the underlying SSL_CTX if it is NULL.

The **SSL_get_srp_userinfo()** function returns the SRP user info for **s**, or from the underlying SSL_CTX if it is NULL.

RETURN VALUES

All SSL_CTX_set_* functions return 1 on success and 0 on failure.

SSL_set_srp_server_param() returns 1 on success and -1 on failure.

The SSL_get_SRP_* functions return a pointer to the requested data, the memory is owned by the library and should not be freed by the caller.

EXAMPLES

Setup SRP parameters on the client:

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

const char *username = "username";
const char *password = "password";

SSL_CTX *ctx = SSL_CTX_new(TLS_client_method());
if (!ctx)
    /* Error */
if (!SSL_CTX_set_srp_username(ctx, username))
    /* Error */
if (!SSL_CTX_set_srp_password(ctx, password))
    /* Error */
```

Setup SRP server with verifier file:

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

const char *srpvfile = "password.srpv";

int srpServerCallback(SSL *s, int *ad, void *arg)
{
    SRP_VBASE *srpData = (SRP_VBASE*) arg;
    char *username = SSL_get_srp_username(s);

    SRP_user_pwd *user_pwd = SRP_VBASE_get1_by_user(srpData, username);
    if (!user_pwd)
        /* Error */
        return SSL3_AL_FATAL;

    if (SSL_set_srp_server_param(s, user_pwd->N, user_pwd->g,
        user_pwd->s, user_pwd->v, user_pwd->info) < 0)
        /* Error */

    SRP_user_pwd_free(user_pwd);
    return SSL_ERROR_NONE;
}

SSL_CTX *ctx = SSL_CTX_new(TLS_server_method());
if (!ctx)
```

```
    /* Error */

/*
 * seedKey should contain a NUL terminated sequence
 * of random non NUL bytes
 */
const char *seedKey;

SRP_VBASE *srpData = SRP_VBASE_new(seedKey);
if (SRP_VBASE_init(srpData, (char*) srpvfile) != SRP_NO_ERROR)
  /* Error */

SSL_CTX_set_srp_cb_arg(ctx, srpData);
SSL_CTX_set_srp_username_callback(ctx, srpServerCallback);
```

SEE ALSO

ssl(7), **openssl-srp**(1), **SRP_VBASE_new**(3), **SRP_create_verifier**(3)

HISTORY

These functions were added in OpenSSL 1.0.1 and deprecated in OpenSSL 3.0.

COPYRIGHT