NAME

EVP_PKEY_Q_keygen, EVP_PKEY_keygen_init, EVP_PKEY_paramgen_init, EVP_PKEY_generate, EVP_PKEY_CTX_set_cb, EVP_PKEY_CTX_get_cb, EVP_PKEY_CTX_get_keygen_info, EVP_PKEY_CTX_set_app_data, EVP_PKEY_CTX_get_app_data, EVP_PKEY_gen_cb, EVP_PKEY_paramgen, EVP_PKEY_keygen - key and parameter generation and check functions

SYNOPSIS

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

EVP_PKEY *EVP_PKEY_Q_keygen(OSSL_LIB_CTX *libctx, const char *propq,
                            const char *type, ...);

int EVP_PKEY_keygen_init(EVP_PKEY_CTX *ctx);
int EVP_PKEY_paramgen_init(EVP_PKEY_CTX *ctx);
int EVP_PKEY_generate(EVP_PKEY_CTX *ctx, EVP_PKEY **ppkey);
int EVP_PKEY_paramgen(EVP_PKEY_CTX *ctx, EVP_PKEY **ppkey);
int EVP_PKEY_keygen(EVP_PKEY_CTX *ctx, EVP_PKEY **ppkey);

typedef int EVP_PKEY_gen_cb(EVP_PKEY_CTX *ctx);

void EVP_PKEY_CTX_set_cb(EVP_PKEY_CTX *ctx, EVP_PKEY_gen_cb *cb);
EVP_PKEY_gen_cb *EVP_PKEY_CTX_get_cb(EVP_PKEY_CTX *ctx);

int EVP_PKEY_CTX_get_keygen_info(EVP_PKEY_CTX *ctx, int idx);

void EVP_PKEY_CTX_set_app_data(EVP_PKEY_CTX *ctx, void *data);
void *EVP_PKEY_CTX_get_app_data(EVP_PKEY_CTX *ctx);
```

DESCRIPTION

Generating keys is sometimes straight forward, just generate the key's numbers and be done with it. However, there are certain key types that need key parameters, often called domain parameters but not necessarily limited to that, that also need to be generated. In addition to this, the caller may want to set user provided generation parameters that further affect key parameter or key generation, such as the desired key size.

To flexibly allow all that's just been described, key parameter and key generation is divided into an initialization of a key algorithm context, functions to set user provided parameters, and finally the key parameter or key generation function itself.

The key algorithm context must be created using **EVP_PKEY_CTX_new**(3) or variants thereof, see that manual for details.

EVP_PKEY_keygen_init() initializes a public key algorithm context *ctx* for a key generation operation.

EVP_PKEY_paramgen_init() is similar to **EVP_PKEY_keygen_init()** except key parameters are generated.

After initialization, generation parameters may be provided with **EVP_PKEY_CTX_ctrl**(3) or **EVP_PKEY_CTX_set_params**(3), or any other function described in those manuals.

EVP_PKEY_generate() performs the generation operation, the resulting key parameters or key are written to **ppkey*. If **ppkey* is NULL when this function is called, it will be allocated, and should be freed by the caller when no longer useful, using **EVP_PKEY_free**(3).

EVP_PKEY_paramgen() and **EVP_PKEY_keygen()** do exactly the same thing as **EVP_PKEY_generate()**, after checking that the corresponding **EVP_PKEY_paramgen_init()** or **EVP_PKEY_keygen_init()** was used to initialize *ctx*. These are older functions that are kept for backward compatibility. It is safe to use **EVP_PKEY_generate()** instead.

The function **EVP_PKEY_set_cb()** sets the key or parameter generation callback to *cb*. The function **EVP_PKEY_CTX_get_cb()** returns the key or parameter generation callback.

The function **EVP_PKEY_CTX_get_keygen_info()** returns parameters associated with the generation operation. If *idx* is -1 the total number of parameters available is returned. Any non negative value returns the value of that parameter. **EVP_PKEY_CTX_gen_keygen_info()** with a nonnegative value for *idx* should only be called within the generation callback.

If the callback returns 0 then the key generation operation is aborted and an error occurs. This might occur during a time consuming operation where a user clicks on a "cancel" button.

The functions **EVP_PKEY_CTX_set_app_data()** and **EVP_PKEY_CTX_get_app_data()** set and retrieve an opaque pointer. This can be used to set some application defined value which can be retrieved in the callback: for example a handle which is used to update a "progress dialog".

EVP_PKEY_Q_keygen() abstracts from the explicit use of **EVP_PKEY_CTX** while providing a ’quick’ but limited way of generating a new asymmetric key pair. It provides shorthands for simple and common cases of key generation. As usual, the library context *libctx* and property query *propq* can be given for fetching algorithms from providers. If *type* is "RSA", a **size_t** parameter must be given to specify the size of the RSA key. If *type* is "EC", a string parameter must be given to specify the name

of the EC curve. If *type* is "X25519", "X448", "ED25519", "ED448", or "SM2" no further parameter is needed.

RETURN VALUES

EVP_PKEY_keygen_init(), **EVP_PKEY_paramgen_init()**, **EVP_PKEY_keygen()** and **EVP_PKEY_paramgen()** return 1 for success and 0 or a negative value for failure. In particular a return value of -2 indicates the operation is not supported by the public key algorithm.

EVP_PKEY_Q_keygen() returns an **EVP_PKEY**, or NULL on failure.

NOTES

After the call to **EVP_PKEY_keygen_init()** or **EVP_PKEY_paramgen_init()** algorithm specific control operations can be performed to set any appropriate parameters for the operation.

The functions **EVP_PKEY_keygen()** and **EVP_PKEY_paramgen()** can be called more than once on the same context if several operations are performed using the same parameters.

The meaning of the parameters passed to the callback will depend on the algorithm and the specific implementation of the algorithm. Some might not give any useful information at all during key or parameter generation. Others might not even call the callback.

The operation performed by key or parameter generation depends on the algorithm used. In some cases (e.g. EC with a supplied named curve) the "generation" option merely sets the appropriate fields in an EVP_PKEY structure.

In OpenSSL an EVP_PKEY structure containing a private key also contains the public key components and parameters (if any). An OpenSSL private key is equivalent to what some libraries call a "key pair". A private key can be used in functions which require the use of a public key or parameters.

EXAMPLES

Generate a 2048 bit RSA key:

```
#include <openssl/evp.h>
#include <openssl/rsa.h>

EVP_PKEY_CTX *ctx;
EVP_PKEY *pkey = NULL;

ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, NULL);
if (!ctx)
```

```
    /* Error occurred */
if (EVP_PKEY_keygen_init(ctx) <= 0)
    /* Error */
if (EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, 2048) <= 0)
    /* Error */

/* Generate key */
if (EVP_PKEY_keygen(ctx, &pkey) <= 0)
    /* Error */
```

Generate a key from a set of parameters:

```
#include <openssl/evp.h>
#include <openssl/rsa.h>

EVP_PKEY_CTX *ctx;
ENGINE *eng;
EVP_PKEY *pkey = NULL, *param;

/* Assumed param, eng are set up already */
ctx = EVP_PKEY_CTX_new(param, eng);
if (!ctx)
    /* Error occurred */
if (EVP_PKEY_keygen_init(ctx) <= 0)
    /* Error */

/* Generate key */
if (EVP_PKEY_keygen(ctx, &pkey) <= 0)
    /* Error */
```

Example of generation callback for OpenSSL public key implementations:

```
/* Application data is a BIO to output status to */

EVP_PKEY_CTX_set_app_data(ctx, status_bio);

static int genpkey_cb(EVP_PKEY_CTX *ctx)
{
    char c = '*';
    BIO *b = EVP_PKEY_CTX_get_app_data(ctx);
```

```
    int p = EVP_PKEY_CTX_get_keygen_info(ctx, 0);

    if (p == 0)
        c = '.';
    if (p == 1)
        c = '+';
    if (p == 2)
        c = '*';
    if (p == 3)
        c = '\n';
    BIO_write(b, &c, 1);
    (void)BIO_flush(b);
    return 1;
}
```

SEE ALSO

EVP_RSA_gen(3), **EVP_EC_gen**(3), **EVP_PKEY_CTX_new**(3), **EVP_PKEY_encrypt**(3), **EVP_PKEY_decrypt**(3), **EVP_PKEY_sign**(3), **EVP_PKEY_verify**(3), **EVP_PKEY_verify_recover**(3), **EVP_PKEY_derive**(3)

HISTORY

EVP_PKEY_keygen_init(), int **EVP_PKEY_paramgen_init()**, **EVP_PKEY_keygen()**, **EVP_PKEY_paramgen()**, **EVP_PKEY_gen_cb()**, **EVP_PKEY_CTX_set_cb()**, **EVP_PKEY_CTX_get_cb()**, **EVP_PKEY_CTX_get_keygen_info()**, **EVP_PKEY_CTX_set_app_data()** and **EVP_PKEY_CTX_get_app_data()** were added in OpenSSL 1.0.0.

EVP_PKEY_Q_keygen() and **EVP_PKEY_generate()** were added in OpenSSL 3.0.

COPYRIGHT