

**NAME**

`EVP_PKEY_fromdata_init`, `EVP_PKEY_fromdata`, `EVP_PKEY_fromdata_settable` - functions to create keys and key parameters from user data

**SYNOPSIS**

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

```
int EVP_PKEY_fromdata_init(EVP_PKEY_CTX *ctx);
int EVP_PKEY_fromdata(EVP_PKEY_CTX *ctx, EVP_PKEY **ppkey, int selection,
                      OSSL_PARAM params[]);
const OSSL_PARAM *EVP_PKEY_fromdata_settable(EVP_PKEY_CTX *ctx, int selection);
```

**DESCRIPTION**

The functions described here are used to create new keys from user provided key data, such as  $n$ ,  $e$  and  $d$  for a minimal RSA keypair.

These functions use an `EVP_PKEY_CTX` context, which should primarily be created with `EVP_PKEY_CTX_new_from_name(3)` or `EVP_PKEY_CTX_new_id(3)`.

The exact key data that the user can pass depends on the key type. These are passed as an `OSSL_PARAM(3)` array.

`EVP_PKEY_fromdata_init()` initializes a public key algorithm context for creating a key or key parameters from user data.

`EVP_PKEY_fromdata()` creates the structure to store a key or key parameters, given data from *params*, *selection* and a context that's been initialized with `EVP_PKEY_fromdata_init()`. The result is written to *\*ppkey*. *selection* is described in "Selections". The parameters that can be used for various types of key are as described by the diverse "Common parameters" sections of the `EVP_PKEY-RSA(7)`, `EVP_PKEY-DSA(7)`, `EVP_PKEY-DH(7)`, `EVP_PKEY-EC(7)`, `EVP_PKEY-ED448(7)`, `EVP_PKEY-X25519(7)`, `EVP_PKEY-X448(7)`, and `EVP_PKEY-ED25519(7)` pages.

`EVP_PKEY_fromdata_settable()` gets a constant `OSSL_PARAM(3)` array that describes the settable parameters that can be used with `EVP_PKEY_fromdata()`. *selection* is described in "Selections".

Parameters in the *params* array that are not among the settable parameters for the given *selection* are ignored.

**Selections**

The following constants can be used for *selection*:

**EVP\_PKEY\_KEY\_PARAMETERS**

Only key parameters will be selected.

**EVP\_PKEY\_PUBLIC\_KEY**

Only public key components will be selected. This includes optional key parameters.

**EVP\_PKEY\_KEYPAIR**

Any keypair components will be selected. This includes the private key, public key and key parameters.

**NOTES**

These functions only work with key management methods coming from a provider. This is the mirror function to **EVP\_PKEY\_todata(3)**.

**RETURN VALUES**

**EVP\_PKEY\_fromdata\_init()** and **EVP\_PKEY\_fromdata()** 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.

**EXAMPLES**

These examples are very terse for the sake of staying on topic, which is the **EVP\_PKEY\_fromdata()** set of functions. In real applications, BIGNUMs would be handled and converted to byte arrays with **BN\_bn2nativepad()**, but that's off topic here.

**Creating an RSA keypair using raw key data**

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

/*
 * These are extremely small to make this example simple. A real
 * and secure application will not use such small numbers. A real
 * and secure application is expected to use BIGNUMs, and to build
 * this array dynamically.
 */
unsigned long rsa_n = 0xbc747fc5;
unsigned long rsa_e = 0x10001;
unsigned long rsa_d = 0x7b133399;
OSSL_PARAM params[] = {
    OSSL_PARAM_ulong("n", &rsa_n),
    OSSL_PARAM_ulong("e", &rsa_e),
    OSSL_PARAM_ulong("d", &rsa_d),
```

```

    OSSL_PARAM_END
};

int main()
{
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_from_name(NULL, "RSA", NULL);
    EVP_PKEY *pkey = NULL;

    if (ctx == NULL
        || EVP_PKEY_fromdata_init(ctx) <= 0
        || EVP_PKEY_fromdata(ctx, &pkey, EVP_PKEY_KEYPAIR, params) <= 0)
        exit(1);

    /* Do what you want with |pkey| */
}

```

### Creating an ECC keypair using raw key data

```

#include <openssl/evp.h>
#include <openssl/param_build.h>
#include <openssl/ec.h>

/*
 * Fixed data to represent the private and public key.
 */
const unsigned char priv_data[] = {
    0xb9, 0x2f, 0x3c, 0xe6, 0x2f, 0xfb, 0x45, 0x68,
    0x39, 0x96, 0xf0, 0x2a, 0xaf, 0x6c, 0xda, 0xf2,
    0x89, 0x8a, 0x27, 0xbf, 0x39, 0x9b, 0x7e, 0x54,
    0x21, 0xc2, 0xa1, 0xe5, 0x36, 0x12, 0x48, 0x5d
};

/* UNCOMPRESSED FORMAT */
const unsigned char pub_data[] = {
    POINT_CONVERSION_UNCOMPRESSED,
    0xcf, 0x20, 0xfb, 0x9a, 0x1d, 0x11, 0x6c, 0x5e,
    0x9f, 0xec, 0x38, 0x87, 0x6c, 0x1d, 0x2f, 0x58,
    0x47, 0xab, 0xa3, 0x9b, 0x79, 0x23, 0xe6, 0xeb,
    0x94, 0x6f, 0x97, 0xdb, 0xa3, 0x7d, 0xbd, 0xe5,
    0x26, 0xca, 0x07, 0x17, 0x8d, 0x26, 0x75, 0xff,
    0xcb, 0x8e, 0xb6, 0x84, 0xd0, 0x24, 0x02, 0x25,
    0x8f, 0xb9, 0x33, 0x6e, 0xcf, 0x12, 0x16, 0x2f,

```

```
    0x5c, 0xcd, 0x86, 0x71, 0xa8, 0xbf, 0x1a, 0x47
};

int main()
{
    EVP_PKEY_CTX *ctx;
    EVP_PKEY *pkey = NULL;
    BIGNUM *priv;
    OSSL_PARAM_BLD *param_bld;
    OSSL_PARAM *params = NULL;
    int exitcode = 0;

    priv = BN_bin2bn(priv_data, sizeof(priv_data), NULL);

    param_bld = OSSL_PARAM_BLD_new();
    if (priv != NULL && param_bld != NULL
        && OSSL_PARAM_BLD_push_utf8_string(param_bld, "group",
                                           "prime256v1", 0)
        && OSSL_PARAM_BLD_push_BN(param_bld, "priv", priv)
        && OSSL_PARAM_BLD_push_octet_string(param_bld, "pub",
                                           pub_data, sizeof(pub_data)))
        params = OSSL_PARAM_BLD_to_param(param_bld);

    ctx = EVP_PKEY_CTX_new_from_name(NULL, "EC", NULL);
    if (ctx == NULL
        || params == NULL
        || EVP_PKEY_fromdata_init(ctx) <= 0
        || EVP_PKEY_fromdata(ctx, &pkey, EVP_PKEY_KEYPAIR, params) <= 0) {
        exitcode = 1;
    } else {
        /* Do what you want with |pkey| */
    }

    EVP_PKEY_free(pkey);
    EVP_PKEY_CTX_free(ctx);
    OSSL_PARAM_free(params);
    OSSL_PARAM_BLD_free(param_bld);
    BN_free(priv);

    exit(exitcode);
}
```

```
}
```

### Finding out params for an unknown key type

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

/* Program expects a key type as first argument */
int main(int argc, char *argv[])
{
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new_from_name(NULL, argv[1], NULL);
    const OSSL_PARAM *settable_params = NULL;

    if (ctx == NULL)
        exit(1);
    settable_params = EVP_PKEY_fromdata_settable(ctx, EVP_PKEY_KEYPAIR);
    if (settable_params == NULL)
        exit(1);

    for (; settable_params->key != NULL; settable_params++) {
        const char *datatype = NULL;

        switch (settable_params->data_type) {
        case OSSL_PARAM_INTEGER:
            datatype = "integer";
            break;
        case OSSL_PARAM_UNSIGNED_INTEGER:
            datatype = "unsigned integer";
            break;
        case OSSL_PARAM_UTF8_STRING:
            datatype = "printable string (utf-8 encoding expected)";
            break;
        case OSSL_PARAM_UTF8_PTR:
            datatype = "printable string pointer (utf-8 encoding expected)";
            break;
        case OSSL_PARAM_OCTET_STRING:
            datatype = "octet string";
            break;
        case OSSL_PARAM_OCTET_PTR:
            datatype = "octet string pointer";
            break;
        }
```

```
    }
    printf("%s : %s ", settable_params->key, datatype);
    if (settable_params->data_size == 0)
        printf("(unlimited size)\n");
    else
        printf("(maximum size %zu)\n", settable_params->data_size);
    }
}
```

The descriptor **OSSL\_PARAM(3)** returned by **EVP\_PKEY\_fromdata\_settable()** may also be used programmatically, for example with **OSSL\_PARAM\_allocate\_from\_text(3)**.

### SEE ALSO

**EVP\_PKEY\_CTX\_new(3)**, **provider(7)**, **EVP\_PKEY\_gettable\_params(3)**, **OSSL\_PARAM(3)**, **EVP\_PKEY\_todata(3)**, **EVP\_PKEY-RSA(7)**, **EVP\_PKEY-DSA(7)**, **EVP\_PKEY-DH(7)**, **EVP\_PKEY-EC(7)**, **EVP\_PKEY-ED448(7)**, **EVP\_PKEY-X25519(7)**, **EVP\_PKEY-X448(7)**, **EVP\_PKEY-ED25519(7)**

### HISTORY

These functions were added in OpenSSL 3.0.

### COPYRIGHT

Copyright 2019-2022 The OpenSSL Project Authors. All Rights Reserved.

Licensed under the Apache License 2.0 (the "License"). You may not use this file except in compliance with the License. You can obtain a copy in the file LICENSE in the source distribution or at <https://www.openssl.org/source/license.html>.