Full credit is given to the above companies including the OS that this PDF file was generated!

ed Hat Enterprise Linux Release 9.2 Manual Pages on 'EVP_PKEY_CTX_get_rsa_pss_saltlen.3ossl' comman

$ man EVP_PKEY_CTX_get_rsa_pss_saltlen.3ossl

EVP_PKEY_CTX_CTRL(3ossl) OpenSSL EVP_PKEY_CTX_CTRL(3ossl)

NAME

EVP_PKEY_CTX_ctrl, EVP_PKEY_CTX_ctrl_str, EVP_PKEY_CTX_ctrl_uint64,

EVP_PKEY_CTX_md, EVP_PKEY_CTX_set_signature_md,

EVP_PKEY_CTX_get_signature_md, EVP_PKEY_CTX_set_mac_key,

EVP_PKEY_CTX_set_group_name, EVP_PKEY_CTX_get_group_name,

EVP_PKEY_CTX_set_rsa_padding, EVP_PKEY_CTX_get_rsa_padding,

EVP_PKEY_CTX_set_rsa_pss_saltlen, EVP_PKEY_CTX_get_rsa_pss_saltlen,

EVP_PKEY_CTX_set_rsa_keygen_bits, EVP_PKEY_CTX_set_rsa_keygen_pubexp,

EVP_PKEY_CTX_set1_rsa_keygen_pubexp,

EVP_PKEY_CTX_set_rsa_keygen_primes, EVP_PKEY_CTX_set_rsa_mgf1_md_name,

EVP_PKEY_CTX_set_rsa_mgf1_md, EVP_PKEY_CTX_get_rsa_mgf1_md,

EVP_PKEY_CTX_get_rsa_mgf1_md_name, EVP_PKEY_CTX_set_rsa_oaep_md_name,

EVP_PKEY_CTX_set_rsa_oaep_md, EVP_PKEY_CTX_get_rsa_oaep_md,

EVP_PKEY_CTX_get_rsa_oaep_md_name, EVP_PKEY_CTX_set0_rsa_oaep_label,

EVP_PKEY_CTX_get0_rsa_oaep_label, EVP_PKEY_CTX_set_dsa_paramgen_bits,

EVP_PKEY_CTX_set_dsa_paramgen_q_bits, EVP_PKEY_CTX_set_dsa_paramgen_md,

EVP_PKEY_CTX_set_dsa_paramgen_md_props,

EVP_PKEY_CTX_set_dsa_paramgen_gindex,

EVP_PKEY_CTX_set_dsa_paramgen_type, EVP_PKEY_CTX_set_dsa_paramgen_seed,

EVP_PKEY_CTX_set_dh_paramgen_prime_len,

EVP_PKEY_CTX_set_dh_paramgen_subprime_len,

EVP_PKEY_CTX_set_dh_paramgen_generator,

EVP_PKEY_CTX_set_dh_paramgen_type, EVP_PKEY_CTX_set_dh_paramgen_gindex,

EVP_PKEY_CTX_set_dh_paramgen_seed, EVP_PKEY_CTX_set_dh_rfc5114,

EVP_PKEY_CTX_set_dhx_rfc5114, EVP_PKEY_CTX_set_dh_pad,

EVP_PKEY_CTX_set_dh_nid, EVP_PKEY_CTX_set_dh_kdf_type,

EVP_PKEY_CTX_get_dh_kdf_type, EVP_PKEY_CTX_set0_dh_kdf_oid,

EVP_PKEY_CTX_get0_dh_kdf_oid, EVP_PKEY_CTX_set_dh_kdf_md,

EVP_PKEY_CTX_get_dh_kdf_md, EVP_PKEY_CTX_set_dh_kdf_outlen,

EVP_PKEY_CTX_get_dh_kdf_outlen, EVP_PKEY_CTX_set0_dh_kdf_ukm,

EVP_PKEY_CTX_get0_dh_kdf_ukm, EVP_PKEY_CTX_set_ec_paramgen_curve_nid,

EVP_PKEY_CTX_set_ec_param_enc, EVP_PKEY_CTX_set_ecdh_cofactor_mode,

EVP_PKEY_CTX_get_ecdh_cofactor_mode, EVP_PKEY_CTX_set_ecdh_kdf_type,

EVP_PKEY_CTX_get_ecdh_kdf_type, EVP_PKEY_CTX_set_ecdh_kdf_md,

EVP_PKEY_CTX_get_ecdh_kdf_md, EVP_PKEY_CTX_set_ecdh_kdf_outlen,

EVP_PKEY_CTX_get_ecdh_kdf_outlen, EVP_PKEY_CTX_set0_ecdh_kdf_ukm,

EVP_PKEY_CTX_get0_ecdh_kdf_ukm, EVP_PKEY_CTX_set1_id,

EVP_PKEY_CTX_get1_id, EVP_PKEY_CTX_get1_id_len, EVP_PKEY_CTX_set_kem_op

- algorithm specific control operations

SYNOPSIS

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

int EVP_PKEY_CTX_ctrl(EVP_PKEY_CTX *ctx, int keytype, int optype,
                      int cmd, int p1, void *p2);
int EVP_PKEY_CTX_ctrl_uint64(EVP_PKEY_CTX *ctx, int keytype, int optype,
                             int cmd, uint64_t value);
int EVP_PKEY_CTX_ctrl_str(EVP_PKEY_CTX *ctx, const char *type,
                          const char *value);
int EVP_PKEY_CTX_md(EVP_PKEY_CTX *ctx, int optype, int cmd, const char *md);
int EVP_PKEY_CTX_set_signature_md(EVP_PKEY_CTX *ctx, const EVP_MD *md);
int EVP_PKEY_CTX_get_signature_md(EVP_PKEY_CTX *ctx, const EVP_MD **pmd);
int EVP_PKEY_CTX_set_mac_key(EVP_PKEY_CTX *ctx, const unsigned char *key,
                             int len);
int EVP_PKEY_CTX_set_group_name(EVP_PKEY_CTX *ctx, const char *name);
int EVP_PKEY_CTX_get_group_name(EVP_PKEY_CTX *ctx, char *name, size_t namelen);
int EVP_PKEY_CTX_set_kem_op(EVP_PKEY_CTX *ctx, const char *op);
```

```
#include <openssl/rsa.h>
int EVP_PKEY_CTX_set_rsa_padding(EVP_PKEY_CTX *ctx, int pad);
int EVP_PKEY_CTX_get_rsa_padding(EVP_PKEY_CTX *ctx, int *pad);
int EVP_PKEY_CTX_set_rsa_pss_saltlen(EVP_PKEY_CTX *ctx, int saltlen);
int EVP_PKEY_CTX_get_rsa_pss_saltlen(EVP_PKEY_CTX *ctx, int *saltlen);
int EVP_PKEY_CTX_set_rsa_keygen_bits(EVP_PKEY_CTX *ctx, int mbits);
int EVP_PKEY_CTX_set1_rsa_keygen_pubexp(EVP_PKEY_CTX *ctx, BIGNUM *pubexp);
int EVP_PKEY_CTX_set_rsa_keygen_primes(EVP_PKEY_CTX *ctx, int primes);
int EVP_PKEY_CTX_set_rsa_mgf1_md_name(EVP_PKEY_CTX *ctx, const char *mdname,
                                      const char *mdprops);
int EVP_PKEY_CTX_set_rsa_mgf1_md(EVP_PKEY_CTX *ctx, const EVP_MD *md);
int EVP_PKEY_CTX_get_rsa_mgf1_md(EVP_PKEY_CTX *ctx, const EVP_MD **md);
int EVP_PKEY_CTX_get_rsa_mgf1_md_name(EVP_PKEY_CTX *ctx, char *name,
                                      size_t namelen);
int EVP_PKEY_CTX_set_rsa_oaep_md_name(EVP_PKEY_CTX *ctx, const char *mdname,
                                      const char *mdprops);
int EVP_PKEY_CTX_set_rsa_oaep_md(EVP_PKEY_CTX *ctx, const EVP_MD *md);
int EVP_PKEY_CTX_get_rsa_oaep_md(EVP_PKEY_CTX *ctx, const EVP_MD **md);
int EVP_PKEY_CTX_get_rsa_oaep_md_name(EVP_PKEY_CTX *ctx, char *name,
                                      size_t namelen);
int EVP_PKEY_CTX_set0_rsa_oaep_label(EVP_PKEY_CTX *ctx, void *label,
                                     int len);
int EVP_PKEY_CTX_get0_rsa_oaep_label(EVP_PKEY_CTX *ctx, unsigned char **label);
#include <openssl/dsa.h>
int EVP_PKEY_CTX_set_dsa_paramgen_bits(EVP_PKEY_CTX *ctx, int nbits);
int EVP_PKEY_CTX_set_dsa_paramgen_q_bits(EVP_PKEY_CTX *ctx, int qbits);
int EVP_PKEY_CTX_set_dsa_paramgen_md(EVP_PKEY_CTX *ctx, const EVP_MD *md);
int EVP_PKEY_CTX_set_dsa_paramgen_md_props(EVP_PKEY_CTX *ctx,
                                           const char *md_name,
                                           const char *md_properties);
int EVP_PKEY_CTX_set_dsa_paramgen_type(EVP_PKEY_CTX *ctx, const char *name);
int EVP_PKEY_CTX_set_dsa_paramgen_gindex(EVP_PKEY_CTX *ctx, int gindex);
int EVP_PKEY_CTX_set_dsa_paramgen_seed(EVP_PKEY_CTX *ctx,
```

```
                                const unsigned char *seed,
                                size_t seedlen);
#include <openssl/dh.h>
int EVP_PKEY_CTX_set_dh_paramgen_prime_len(EVP_PKEY_CTX *ctx, int len);
int EVP_PKEY_CTX_set_dh_paramgen_subprime_len(EVP_PKEY_CTX *ctx, int len);
int EVP_PKEY_CTX_set_dh_paramgen_generator(EVP_PKEY_CTX *ctx, int gen);
int EVP_PKEY_CTX_set_dh_paramgen_type(EVP_PKEY_CTX *ctx, int type);
int EVP_PKEY_CTX_set_dh_pad(EVP_PKEY_CTX *ctx, int pad);
int EVP_PKEY_CTX_set_dh_nid(EVP_PKEY_CTX *ctx, int nid);
int EVP_PKEY_CTX_set_dh_rfc5114(EVP_PKEY_CTX *ctx, int rfc5114);
int EVP_PKEY_CTX_set_dhx_rfc5114(EVP_PKEY_CTX *ctx, int rfc5114);
int EVP_PKEY_CTX_set_dh_paramgen_gindex(EVP_PKEY_CTX *ctx, int gindex);
int EVP_PKEY_CTX_set_dh_paramgen_seed(EVP_PKEY_CTX *ctx,
                                const unsigned char *seed,
                                size_t seedlen);
int EVP_PKEY_CTX_set_dh_kdf_type(EVP_PKEY_CTX *ctx, int kdf);
int EVP_PKEY_CTX_get_dh_kdf_type(EVP_PKEY_CTX *ctx);
int EVP_PKEY_CTX_set0_dh_kdf_oid(EVP_PKEY_CTX *ctx, ASN1_OBJECT *oid);
int EVP_PKEY_CTX_get0_dh_kdf_oid(EVP_PKEY_CTX *ctx, ASN1_OBJECT **oid);
int EVP_PKEY_CTX_set_dh_kdf_md(EVP_PKEY_CTX *ctx, const EVP_MD *md);
int EVP_PKEY_CTX_get_dh_kdf_md(EVP_PKEY_CTX *ctx, const EVP_MD **md);
int EVP_PKEY_CTX_set_dh_kdf_outlen(EVP_PKEY_CTX *ctx, int len);
int EVP_PKEY_CTX_get_dh_kdf_outlen(EVP_PKEY_CTX *ctx, int *len);
int EVP_PKEY_CTX_set0_dh_kdf_ukm(EVP_PKEY_CTX *ctx, unsigned char *ukm, int len);
#include <openssl/ec.h>
int EVP_PKEY_CTX_set_ec_paramgen_curve_nid(EVP_PKEY_CTX *ctx, int nid);
int EVP_PKEY_CTX_set_ec_param_enc(EVP_PKEY_CTX *ctx, int param_enc);
int EVP_PKEY_CTX_set_ecdh_cofactor_mode(EVP_PKEY_CTX *ctx, int cofactor_mode);
int EVP_PKEY_CTX_get_ecdh_cofactor_mode(EVP_PKEY_CTX *ctx);
int EVP_PKEY_CTX_set_ecdh_kdf_type(EVP_PKEY_CTX *ctx, int kdf);
int EVP_PKEY_CTX_get_ecdh_kdf_type(EVP_PKEY_CTX *ctx);
int EVP_PKEY_CTX_set_ecdh_kdf_md(EVP_PKEY_CTX *ctx, const EVP_MD *md);
int EVP_PKEY_CTX_get_ecdh_kdf_md(EVP_PKEY_CTX *ctx, const EVP_MD **md);
```

```
int EVP_PKEY_CTX_set_ecdh_kdf_outlen(EVP_PKEY_CTX *ctx, int len);
int EVP_PKEY_CTX_get_ecdh_kdf_outlen(EVP_PKEY_CTX *ctx, int *len);
int EVP_PKEY_CTX_set0_ecdh_kdf_ukm(EVP_PKEY_CTX *ctx, unsigned char *ukm, int len);
int EVP_PKEY_CTX_set1_id(EVP_PKEY_CTX *ctx, void *id, size_t id_len);
int EVP_PKEY_CTX_get1_id(EVP_PKEY_CTX *ctx, void *id);
int EVP_PKEY_CTX_get1_id_len(EVP_PKEY_CTX *ctx, size_t *id_len);
```

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):

```
#include <openssl/rsa.h>
int EVP_PKEY_CTX_set_rsa_keygen_pubexp(EVP_PKEY_CTX *ctx, BIGNUM *pubexp);
#include <openssl/dh.h>
int EVP_PKEY_CTX_get0_dh_kdf_ukm(EVP_PKEY_CTX *ctx, unsigned char **ukm);
#include <openssl/ec.h>
int EVP_PKEY_CTX_get0_ecdh_kdf_ukm(EVP_PKEY_CTX *ctx, unsigned char **ukm);
```

DESCRIPTION

EVP_PKEY_CTX_ctrl() sends a control operation to the context ctx. The key type used must match keytype if it is not -1. The parameter optype is a mask indicating which operations the control can be applied to. The control command is indicated in cmd and any additional arguments in p1 and p2.

For cmd = EVP_PKEY_CTRL_SET_MAC_KEY, p1 is the length of the MAC key, and p2 is the MAC key. This is used by Poly1305, SipHash, HMAC and CMAC.

Applications will not normally call EVP_PKEY_CTX_ctrl() directly but will instead call one of the algorithm specific functions below.

EVP_PKEY_CTX_ctrl_uint64() is a wrapper that directly passes a uint64 value as p2 to EVP_PKEY_CTX_ctrl().

EVP_PKEY_CTX_ctrl_str() allows an application to send an algorithm specific control operation to a context ctx in string form. This is intended to be used for options specified on the command line or in text files. The commands supported are documented in the openssl utility command line pages for the option -pkeyopt which is supported

by the pkeyutl, genpkey and req commands.

EVP_PKEY_CTX_md() sends a message digest control operation to the context ctx. The message digest is specified by its name md.

EVP_PKEY_CTX_set_signature_md() sets the message digest type used in a signature. It can be used in the RSA, DSA and ECDSA algorithms.

EVP_PKEY_CTX_get_signature_md()gets the message digest type used in a signature. It can be used in the RSA, DSA and ECDSA algorithms.

Key generation typically involves setting up parameters to be used and generating the private and public key data. Some algorithm implementations allow private key data to be set explicitly using EVP_PKEY_CTX_set_mac_key(). In this case key generation is simply the process of setting up the parameters for the key and then setting the raw key data to the value explicitly. Normally applications would call EVP_PKEY_new_raw_private_key(3) or similar functions instead. EVP_PKEY_CTX_set_mac_key() can be used with any of the algorithms supported by the EVP_PKEY_new_raw_private_key(3) function.

EVP_PKEY_CTX_set_group_name() sets the group name to name for parameter and key generation. For example for EC keys this will set the curve name and for DH keys it will set the name of the finite field group.

EVP_PKEY_CTX_get_group_name() finds the group name that's currently set with ctx, and writes it to the location that name points at, as long as its size namelen is large enough to store that name, including a terminating NUL byte.

RSA parameters

EVP_PKEY_CTX_set_rsa_padding() sets the RSA padding mode for ctx. The pad parameter can take the value RSA_PKCS1_PADDING for PKCS#1 padding, RSA_NO_PADDING for no padding, RSA_PKCS1_OAEP_PADDING for OAEP padding (encrypt and decrypt only), RSA_X931_PADDING for X9.31 padding (signature operations only), RSA_PKCS1_PSS_PADDING (sign and verify only) and RSA_PKCS1_WITH_TLS_PADDING for TLS RSA ClientKeyExchange message padding (decryption only).

Two RSA padding modes behave differently if EVP_PKEY_CTX_set_signature_md() is used. If this function is called for

PKCS#1 padding the plaintext buffer is an actual digest value and is encapsulated in a DigestInfo structure according to PKCS#1 when signing and this structure is expected (and stripped off) when verifying. If this control is not used with RSA and PKCS#1 padding then the supplied data is used directly and not encapsulated. In the case of X9.31 padding for RSA the algorithm identifier byte is added or checked and removed if this control is called. If it is not called then the first byte of the plaintext buffer is expected to be the algorithm identifier byte.

EVP_PKEY_CTX_get_rsa_padding() gets the RSA padding mode for ctx.

EVP_PKEY_CTX_set_rsa_pss_saltlen() sets the RSA PSS salt length to saltlen. As its name implies it is only supported for PSS padding. If this function is not called then the salt length is maximized up to the digest length when signing and auto detection when verifying. Four special values are supported:

RSA_PSS_SALTLEN_DIGEST

sets the salt length to the digest length.

RSA_PSS_SALTLEN_MAX

sets the salt length to the maximum permissible value.

RSA_PSS_SALTLEN_AUTO

causes the salt length to be automatically determined based on the PSS block structure when verifying. When signing, it has the same meaning as RSA_PSS_SALTLEN_MAX.

RSA_PSS_SALTLEN_AUTO_DIGEST_MAX

causes the salt length to be automatically determined based on the PSS block structure when verifying, like RSA_PSS_SALTLEN_AUTO. When signing, the salt length is maximized up to a maximum of the digest length to comply with FIPS 186-4 section 5.5.

EVP_PKEY_CTX_get_rsa_pss_saltlen() gets the RSA PSS salt length for ctx. The padding mode must already have been set to RSA_PKCS1_PSS_PADDING.

EVP_PKEY_CTX_set_rsa_keygen_bits() sets the RSA key length for RSA key generation to bits. If not specified 2048 bits is used.

EVP_PKEY_CTX_set1_rsa_keygen_pubexp() sets the public exponent value for RSA key generation to the value stored in pubexp. Currently it should be an odd integer. In accordance with the OpenSSL naming convention, the pubexp pointer must be freed independently of the EVP_PKEY_CTX (ie, it is internally copied). If not specified 65537 is used.

EVP_PKEY_CTX_set_rsa_keygen_pubexp() does the same as EVP_PKEY_CTX_set1_rsa_keygen_pubexp() except that there is no internal copy and therefore pubexp should not be modified or freed after the call.

EVP_PKEY_CTX_set_rsa_keygen_primes() sets the number of primes for RSA key generation to primes. If not specified 2 is used.

EVP_PKEY_CTX_set_rsa_mgf1_md_name() sets the MGF1 digest for RSA padding schemes to the digest named mdname. If the RSA algorithm implementation for the selected provider supports it then the digest will be fetched using the properties mdprops. If not explicitly set the signing digest is used. The padding mode must have been set to RSA_PKCS1_OAEP_PADDING or RSA_PKCS1_PSS_PADDING.

EVP_PKEY_CTX_set_rsa_mgf1_md() does the same as EVP_PKEY_CTX_set_rsa_mgf1_md_name() except that the name of the digest is inferred from the supplied md and it is not possible to specify any properties.

EVP_PKEY_CTX_get_rsa_mgf1_md_name() gets the name of the MGF1 digest algorithm for ctx. If not explicitly set the signing digest is used. The padding mode must have been set to RSA_PKCS1_OAEP_PADDING or RSA_PKCS1_PSS_PADDING.

EVP_PKEY_CTX_get_rsa_mgf1_md() does the same as EVP_PKEY_CTX_get_rsa_mgf1_md_name() except that it returns a pointer to an EVP_MD object instead. Note that only known, built-in EVP_MD objects will be returned. The EVP_MD object may be NULL if the digest is not one of these (such as a digest only implemented in a third party provider).

EVP_PKEY_CTX_set_rsa_oaep_md_name() sets the message digest type used

in RSA OAEP to the digest named mdname. If the RSA algorithm implementation for the selected provider supports it then the digest will be fetched using the properties mdprops. The padding mode must have been set to RSA_PKCS1_OAEP_PADDING.

EVP_PKEY_CTX_set_rsa_oaep_md() does the same as EVP_PKEY_CTX_set_rsa_oaep_md_name() except that the name of the digest is inferred from the supplied md and it is not possible to specify any properties.

EVP_PKEY_CTX_get_rsa_oaep_md_name() gets the message digest algorithm name used in RSA OAEP and stores it in the buffer name which is of size namelen. The padding mode must have been set to RSA_PKCS1_OAEP_PADDING. The buffer should be sufficiently large for any expected digest algorithm names or the function will fail.

EVP_PKEY_CTX_get_rsa_oaep_md() does the same as EVP_PKEY_CTX_get_rsa_oaep_md_name() except that it returns a pointer to an EVP_MD object instead. Note that only known, built-in EVP_MD objects will be returned. The EVP_MD object may be NULL if the digest is not one of these (such as a digest only implemented in a third party provider).

EVP_PKEY_CTX_set0_rsa_oaep_label() sets the RSA OAEP label to binary data label and its length in bytes to len. If label is NULL or len is 0, the label is cleared. The library takes ownership of the label so the caller should not free the original memory pointed to by label. The padding mode must have been set to RSA_PKCS1_OAEP_PADDING.

EVP_PKEY_CTX_get0_rsa_oaep_label() gets the RSA OAEP label to label. The return value is the label length. The padding mode must have been set to RSA_PKCS1_OAEP_PADDING. The resulting pointer is owned by the library and should not be freed by the caller.

RSA_PKCS1_WITH_TLS_PADDING is used when decrypting an RSA encrypted TLS pre-master secret in a TLS ClientKeyExchange message. It is the same as RSA_PKCS1_PADDING except that it additionally verifies that the result is the correct length and the first two bytes are the protocol version initially requested by the client. If the encrypted content is publicly

invalid then the decryption will fail. However, if the padding checks fail then decryption will still appear to succeed but a random TLS premaster secret will be returned instead. This padding mode accepts two parameters which can be set using the EVP_PKEY_CTX_set_params(3) function. These are OSSL_ASYM_CIPHER_PARAM_TLS_CLIENT_VERSION and OSSL_ASYM_CIPHER_PARAM_TLS_NEGOTIATED_VERSION, both of which are expected to be unsigned integers. Normally only the first of these will be set and represents the TLS protocol version that was first requested by the client (e.g. 0x0303 for TLSv1.2, 0x0302 for TLSv1.1 etc). Historically some buggy clients would use the negotiated protocol version instead of the protocol version first requested. If this behaviour should be tolerated then OSSL_ASYM_CIPHER_PARAM_TLS_NEGOTIATED_VERSION should be set to the actual negotiated protocol version. Otherwise it should be left unset.

DSA parameters

EVP_PKEY_CTX_set_dsa_paramgen_bits() sets the number of bits used for DSA parameter generation to nbits. If not specified, 2048 is used.

EVP_PKEY_CTX_set_dsa_paramgen_q_bits() sets the number of bits in the subprime parameter q for DSA parameter generation to qbits. If not specified, 224 is used. If a digest function is specified below, this parameter is ignored and instead, the number of bits in q matches the size of the digest.

EVP_PKEY_CTX_set_dsa_paramgen_md() sets the digest function used for DSA parameter generation to md. If not specified, one of SHA-1, SHA-224, or SHA-256 is selected to match the bit length of q above.

EVP_PKEY_CTX_set_dsa_paramgen_md_props() sets the digest function used for DSA parameter generation using md_name and md_properties to retrieve the digest from a provider. If not specified, md_name will be set to one of SHA-1, SHA-224, or SHA-256 depending on the bit length of q above. md_properties is a property query string that has a default value of '' if not specified.

EVP_PKEY_CTX_set_dsa_paramgen_gindex() sets the gindex used by the generator G. The default value is -1 which uses unverifiable g,

otherwise a positive value uses verifiable g. This value must be saved if key validation of g is required, since it is not part of a persisted key.

EVP_PKEY_CTX_set_dsa_paramgen_seed() sets the seed to use for generation rather than using a randomly generated value for the seed. This is useful for testing purposes only and can fail if the seed does not produce primes for both p & q on its first iteration. This value must be saved if key validation of p, q, and verifiable g are required, since it is not part of a persisted key.

EVP_PKEY_CTX_set_dsa_paramgen_type() sets the generation type to use FIPS186-4 generation if name is "fips186_4", or FIPS186-2 generation if name is "fips186_2". The default value for the default provider is "fips186_2". The default value for the FIPS provider is "fips186_4".

DH parameters

EVP_PKEY_CTX_set_dh_paramgen_prime_len() sets the length of the DH prime parameter p for DH parameter generation. If this function is not called then 2048 is used. Only accepts lengths greater than or equal to 256.

EVP_PKEY_CTX_set_dh_paramgen_subprime_len() sets the length of the DH optional subprime parameter q for DH parameter generation. The default is 256 if the prime is at least 2048 bits long or 160 otherwise. The DH paramgen type must have been set to "fips186_4".

EVP_PKEY_CTX_set_dh_paramgen_generator() sets DH generator to gen for DH parameter generation. If not specified 2 is used.

EVP_PKEY_CTX_set_dh_paramgen_type() sets the key type for DH parameter generation. The supported parameters are:

DH_PARAMGEN_TYPE_GROUP

Use a named group. If only the safe prime parameter p is set this can be used to select a ffdhe safe prime group of the correct size.

DH_PARAMGEN_TYPE_FIPS_186_4

FIPS186-4 FFC parameter generator.

DH_PARAMGEN_TYPE_FIPS_186_2

FIPS186-2 FFC parameter generator (X9.42 DH).

DH_PARAMGEN_TYPE_GENERATOR

Uses a safe prime generator g (PKCS#3 format).

The default in the default provider is DH_PARAMGEN_TYPE_GENERATOR for the "DH" keytype, and DH_PARAMGEN_TYPE_FIPS_186_2 for the "DHX" keytype. In the FIPS provider the default value is DH_PARAMGEN_TYPE_GROUP for the "DH" keytype and <DH_PARAMGEN_TYPE_FIPS_186_4 for the "DHX" keytype.

EVP_PKEY_CTX_set_dh_paramgen_gindex() sets the gindex used by the generator G. The default value is -1 which uses unverifiable g, otherwise a positive value uses verifiable g. This value must be saved if key validation of g is required, since it is not part of a persisted key.

EVP_PKEY_CTX_set_dh_paramgen_seed() sets the seed to use for generation rather than using a randomly generated value for the seed. This is useful for testing purposes only and can fail if the seed does not produce primes for both p & q on its first iteration. This value must be saved if key validation of p, q, and verifiable g are required, since it is not part of a persisted key.

EVP_PKEY_CTX_set_dh_pad() sets the DH padding mode. If pad is 1 the shared secret is padded with zeros up to the size of the DH prime p. If pad is zero (the default) then no padding is performed.

EVP_PKEY_CTX_set_dh_nid() sets the DH parameters to values corresponding to nid as defined in RFC7919 or RFC3526. The nid parameter must be NID_ffdhe2048, NID_ffdhe3072, NID_ffdhe4096, NID_ffdhe6144, NID_ffdhe8192, NID_modp_1536, NID_modp_2048, NID_modp_3072, NID_modp_4096, NID_modp_6144, NID_modp_8192 or NID_undef to clear the stored value. This function can be called during parameter or key generation. The nid parameter and the rfc5114 parameter are mutually exclusive.

EVP_PKEY_CTX_set_dh_rfc5114() and EVP_PKEY_CTX_set_dhx_rfc5114() both set the DH parameters to the values defined in RFC5114. The rfc5114 parameter must be 1, 2 or 3 corresponding to RFC5114 sections 2.1, 2.2 and 2.3. or 0 to clear the stored value. This macro can be called

during parameter generation. The ctx must have a key type of EVP_PKEY_DHX. The rfc5114 parameter and the nid parameter are mutually exclusive.

DH key derivation function parameters

Note that all of the following functions require that the ctx parameter has a private key type of EVP_PKEY_DHX. When using key derivation, the output of EVP_PKEY_derive() is the output of the KDF instead of the DH shared secret. The KDF output is typically used as a Key Encryption Key (KEK) that in turn encrypts a Content Encryption Key (CEK).

EVP_PKEY_CTX_set_dh_kdf_type() sets the key derivation function type to kdf for DH key derivation. Possible values are EVP_PKEY_DH_KDF_NONE and EVP_PKEY_DH_KDF_X9_42 which uses the key derivation specified in RFC2631 (based on the keying algorithm described in X9.42). When using key derivation, the kdf_oid, kdf_md and kdf_outlen parameters must also be specified.

EVP_PKEY_CTX_get_dh_kdf_type() gets the key derivation function type for ctx used for DH key derivation. Possible values are EVP_PKEY_DH_KDF_NONE and EVP_PKEY_DH_KDF_X9_42.

EVP_PKEY_CTX_set0_dh_kdf_oid() sets the key derivation function object identifier to oid for DH key derivation. This OID should identify the algorithm to be used with the Content Encryption Key. The library takes ownership of the object identifier so the caller should not free the original memory pointed to by oid.

EVP_PKEY_CTX_get0_dh_kdf_oid() gets the key derivation function oid for ctx used for DH key derivation. The resulting pointer is owned by the library and should not be freed by the caller.

EVP_PKEY_CTX_set_dh_kdf_md() sets the key derivation function message digest to md for DH key derivation. Note that RFC2631 specifies that this digest should be SHA1 but OpenSSL tolerates other digests.

EVP_PKEY_CTX_get_dh_kdf_md() gets the key derivation function message digest for ctx used for DH key derivation.

EVP_PKEY_CTX_set_dh_kdf_outlen() sets the key derivation function output length to len for DH key derivation.

EVP_PKEY_CTX_get_dh_kdf_outlen() gets the key derivation function output length for ctx used for DH key derivation.

EVP_PKEY_CTX_set0_dh_kdf_ukm() sets the user key material to ukm and its length to len for DH key derivation. This parameter is optional and corresponds to the partyAInfo field in RFC2631 terms. The specification requires that it is 512 bits long but this is not enforced by OpenSSL.
The library takes ownership of the user key material so the caller should not free the original memory pointed to by ukm.

EVP_PKEY_CTX_get0_dh_kdf_ukm() gets the user key material for ctx. The return value is the user key material length. The resulting pointer is owned by the library and should not be freed by the caller.

EC parameters

Use EVP_PKEY_CTX_set_group_name() (described above) to set the curve name to name for parameter and key generation.

EVP_PKEY_CTX_set_ec_paramgen_curve_nid() does the same as EVP_PKEY_CTX_set_group_name(), but is specific to EC and uses a nid rather than a name string.

For EC parameter generation, one of EVP_PKEY_CTX_set_group_name() or EVP_PKEY_CTX_set_ec_paramgen_curve_nid() must be called or an error occurs because there is no default curve. These function can also be called to set the curve explicitly when generating an EC key.

EVP_PKEY_CTX_get_group_name() (described above) can be used to obtain the curve name that's currently set with ctx.

EVP_PKEY_CTX_set_ec_param_enc() sets the EC parameter encoding to param_enc when generating EC parameters or an EC key. The encoding can be OPENSSL_EC_EXPLICIT_CURVE for explicit parameters (the default in versions of OpenSSL before 1.1.0) or OPENSSL_EC_NAMED_CURVE to use named curve form. For maximum compatibility the named curve form should be used. Note: the OPENSSL_EC_NAMED_CURVE value was added in OpenSSL 1.1.0; previous versions should use 0 instead.

ECDH parameters

EVP_PKEY_CTX_set_ecdh_cofactor_mode() sets the cofactor mode to cofactor_mode for ECDH key derivation. Possible values are 1 to enable

cofactor key derivation, 0 to disable it and -1 to clear the stored

cofactor mode and fallback to the private key cofactor mode.

EVP_PKEY_CTX_get_ecdh_cofactor_mode() returns the cofactor mode for ctx

used for ECDH key derivation. Possible values are 1 when cofactor key

derivation is enabled and 0 otherwise.

ECDH key derivation function parameters

EVP_PKEY_CTX_set_ecdh_kdf_type() sets the key derivation function type

to kdf for ECDH key derivation. Possible values are

EVP_PKEY_ECDH_KDF_NONE and EVP_PKEY_ECDH_KDF_X9_63 which uses the key

derivation specified in X9.63. When using key derivation, the kdf_md

and kdf_outlen parameters must also be specified.

EVP_PKEY_CTX_get_ecdh_kdf_type() returns the key derivation function

type for ctx used for ECDH key derivation. Possible values are

EVP_PKEY_ECDH_KDF_NONE and EVP_PKEY_ECDH_KDF_X9_63.

EVP_PKEY_CTX_set_ecdh_kdf_md() sets the key derivation function message

digest to md for ECDH key derivation. Note that X9.63 specifies that

this digest should be SHA1 but OpenSSL tolerates other digests.

EVP_PKEY_CTX_get_ecdh_kdf_md() gets the key derivation function message

digest for ctx used for ECDH key derivation.

EVP_PKEY_CTX_set_ecdh_kdf_outlen() sets the key derivation function

output length to len for ECDH key derivation.

EVP_PKEY_CTX_get_ecdh_kdf_outlen() gets the key derivation function

output length for ctx used for ECDH key derivation.

EVP_PKEY_CTX_set0_ecdh_kdf_ukm() sets the user key material to ukm for

ECDH key derivation. This parameter is optional and corresponds to the

shared info in X9.63 terms. The library takes ownership of the user key

material so the caller should not free the original memory pointed to

by ukm.

EVP_PKEY_CTX_get0_ecdh_kdf_ukm() gets the user key material for ctx.

The return value is the user key material length. The resulting pointer

is owned by the library and should not be freed by the caller.

Other parameters

EVP_PKEY_CTX_set1_id(), EVP_PKEY_CTX_get1_id() and

EVP_PKEY_CTX_get1_id_len() are used to manipulate the special identifier field for specific signature algorithms such as SM2. The EVP_PKEY_CTX_set1_id() sets an ID pointed by id with the length id_len to the library. The library takes a copy of the id so that the caller can safely free the original memory pointed to by id.

EVP_PKEY_CTX_get1_id_len() returns the length of the ID set via a previous call to EVP_PKEY_CTX_set1_id(). The length is usually used to allocate adequate memory for further calls to EVP_PKEY_CTX_get1_id().

EVP_PKEY_CTX_get1_id() returns the previously set ID value to caller in id. The caller should allocate adequate memory space for the id before calling EVP_PKEY_CTX_get1_id().

EVP_PKEY_CTX_set_kem_op() sets the KEM operation to run. This can be set after EVP_PKEY_encapsulate_init() or EVP_PKEY_decapsulate_init() to select the kem operation. RSA is the only key type that supports encapsulation currently, and as there is no default operation for the RSA type, this function must be called before EVP_PKEY_encapsulate() or EVP_PKEY_decapsulate().

RETURN VALUES

All other functions described on this page return a positive value 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.

SEE ALSO

EVP_PKEY_CTX_set_params(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), EVP_PKEY_keygen(3) EVP_PKEY_encapsulate(3) EVP_PKEY_decapsulate(3)

HISTORY

EVP_PKEY_CTX_get_rsa_oaep_md_name(), EVP_PKEY_CTX_get_rsa_mgf1_md_name(), EVP_PKEY_CTX_set_rsa_mgf1_md_name(), EVP_PKEY_CTX_set_rsa_oaep_md_name(), EVP_PKEY_CTX_set_dsa_paramgen_md_props(),

EVP_PKEY_CTX_set_dsa_paramgen_gindex(),

EVP_PKEY_CTX_set_dsa_paramgen_type(),

EVP_PKEY_CTX_set_dsa_paramgen_seed(), EVP_PKEY_CTX_set_group_name() and

EVP_PKEY_CTX_get_group_name() were added in OpenSSL 3.0.

The EVP_PKEY_CTX_set1_id(), EVP_PKEY_CTX_get1_id() and

EVP_PKEY_CTX_get1_id_len() macros were added in 1.1.1, other functions

were added in OpenSSL 1.0.0.

In OpenSSL 1.1.1 and below the functions were mostly macros. From

OpenSSL 3.0 they are all functions.

EVP_PKEY_CTX_set_rsa_keygen_pubexp(), EVP_PKEY_CTX_get0_dh_kdf_ukm(),

and EVP_PKEY_CTX_get0_ecdh_kdf_ukm() were deprecated in OpenSSL 3.0.

COPYRIGHT