NAME

OSSL_PROVIDER−FIPS − OpenSSL FIPS provider

DESCRIPTION

The OpenSSL FIPS provider is a special provider that conforms to the Federal Information Processing Standards (FIPS) specified in FIPS 140−2. This 'module' contains an approved set of cryptographic algorithms that is validated by an accredited testing laboratory.

Properties

The implementations in this provider specifically have these properties defined:

''provider=fips''
''fips=yes''

It may be used in a property query string with fetching functions such as **EVP_MD_fetch**(3) or **EVP_CIPHER_fetch**(3), as well as with other functions that take a property query string, such as **EVP_PKEY_CTX_new_from_name**(3).

It isn't mandatory to query for any of these properties, except to make sure to get implementations of this provider and none other.

The ''fips=yes'' property can be use to make sure only FIPS approved implementations are used for crypto operations. This may also include other non-crypto support operations that are not in the fips provider, such as asymmetric key encoders, see ''Asymmetric Key Management'' in **OSSL_PROVIDER−default**(7).

OPERATIONS AND ALGORITHMS

The OpenSSL FIPS provider supports these operations and algorithms:

Hashing Algorithms / Message Digests

SHA1, see **EVP_MD−SHA1**(7)
SHA2, see **EVP_MD−SHA2**(7)
SHA3, see **EVP_MD−SHA3**(7)
KECCAK-KMAC, see **EVP_MD−KECCAK−KMAC**(7)

Symmetric Ciphers

AES, see **EVP_CIPHER−AES**(7)
DES−EDE3 (TripleDES), see **EVP_CIPHER−DES**(7)

Message Authentication Code (MAC)

CMAC, see **EVP_MAC−CMAC**(7)
GMAC, see **EVP_MAC−GMAC**(7)
HMAC, see **EVP_MAC−HMAC**(7)
KMAC, see **EVP_MAC−KMAC**(7)

Key Derivation Function (KDF)

HKDF, see **EVP_KDF−HKDF**(7)
TLS13−KDF, see **EVP_KDF−TLS13_KDF**(7)
SSKDF, see **EVP_KDF−SSKDF**(7)
PBKDF2, see **EVP_KDF−PBKDF2**(7)
SSHKDF, see **EVP_KDF−SSHKDF**(7)
TLS1−PRF, see **EVP_KDF−TLS1_PRF**(7)
KBKDF, see **EVP_KDF−KBKDF**(7)
X942KDF−ASN1, see **EVP_KDF−X942−ASN1**(7)
X942KDF−CONCAT, see **EVP_KDF−X942−CONCAT**(7)
X963KDF, see **EVP_KDF−X963**(7)

Key Exchange

DH, see **EVP_KEYEXCH−DH**(7)
ECDH, see **EVP_KEYEXCH−ECDH**(7)
X25519, see **EVP_KEYEXCH−X25519**(7)

X448, see **EVP_KEYEXCH-X448**(7)

Asymmetric Signature

DSA, see **EVP_KEYEXCH-DSA**(7)
RSA, see **EVP_SIGNATURE-RSA**(7)
X25519, see **EVP_SIGNATURE-ED25519**(7)
X448, see **EVP_SIGNATURE-ED448**(7)
HMAC, see **EVP_SIGNATURE-HMAC**(7)
CMAC, see **EVP_SIGNATURE-CMAC**(7)

Asymmetric Cipher

RSA, see **EVP_KEYEXCH-RSA**(7)

Asymmetric Key Encapsulation

RSA, see **EVP_KEM-RSA**(7)

Asymmetric Key Management

DH, see **EVP_KEYMGMT-DH**(7)
DSA, see **EVP_KEYMGMT-DSA**(7)
RSA, see **EVP_KEYMGMT-RSA**(7)

SELF TESTING

One of the requirements for the FIPS module is self testing. An optional callback mechanism is available to return information to the user using **OSSL_SELF_TEST_set_callback**(3).

The parameters passed to the callback are described in **OSSL_SELF_TEST_new**(3)

The OpenSSL FIPS module uses the following mechanism to provide information about the self tests as they run. This is useful for debugging if a self test is failing. The callback also allows forcing any self test to fail, in order to check that it operates correctly on failure. Note that all self tests run even if a self test failure occurs.

The FIPS module passes the following type(s) to **OSSL_SELF_TEST_onbegin()**.

''Module_Integrity'' (**OSSL_SELF_TEST_TYPE_MODULE_INTEGRITY**)
Uses HMAC SHA256 on the module file to validate that the module has not been modified. The integrity value is compared to a value written to a configuration file during installation.

''Install_Integrity'' (**OSSL_SELF_TEST_TYPE_INSTALL_INTEGRITY**)
Uses HMAC SHA256 on a fixed string to validate that the installation process has already been performed and the self test KATS have already been tested, The integrity value is compared to a value written to a configuration file after successfully running the self tests during installation.

''KAT_Cipher'' (**OSSL_SELF_TEST_TYPE_KAT_CIPHER**)
Known answer test for a symmetric cipher.

''KAT_AsymmetricCipher'' (**OSSL_SELF_TEST_TYPE_KAT_ASYM_CIPHER**)
Known answer test for a asymmetric cipher.

''KAT_Digest'' (**OSSL_SELF_TEST_TYPE_KAT_DIGEST**)
Known answer test for a digest.

''KAT_Signature'' (**OSSL_SELF_TEST_TYPE_KAT_SIGNATURE**)
Known answer test for a signature.

''PCT_Signature'' (**OSSL_SELF_TEST_TYPE_PCT_SIGNATURE**)
Pairwise Consistency check for a signature.

''KAT_KDF'' (**OSSL_SELF_TEST_TYPE_KAT_KDF**)
Known answer test for a key derivation function.

''KAT_KA'' (**OSSL_SELF_TEST_TYPE_KAT_KA**)
Known answer test for key agreement.

‘‘DRBG’’ (**OSSL_SELF_TEST_TYPE_DRBG**)
Known answer test for a Deterministic Random Bit Generator.

‘‘Conditional_PCT’’ (**OSSL_SELF_TEST_TYPE_PCT**)
Conditional test that is run during the generation of key pairs.

‘‘Continuous_RNG_Test’’ (**OSSL_SELF_TEST_TYPE_CRNG**)
Continuous random number generator test.

The ‘‘Module_Integrity’’ self test is always run at startup. The ‘‘Install_Integrity’’ self test is used to check if the self tests have already been run at installation time. If they have already run then the self tests are not run on subsequent startups. All other self test categories are run once at installation time, except for the ‘‘Pairwise_Consistency_Test’’.

There is only one instance of the ‘‘Module_Integrity’’ and ‘‘Install_Integrity’’ self tests. All other self tests may have multiple instances.

The FIPS module passes the following descriptions(s) to **OSSL_SELF_TEST_onbegin()**.

‘‘HMAC’’ (**OSSL_SELF_TEST_DESC_INTEGRITY_HMAC**)
‘‘Module_Integrity’’ and ‘‘Install_Integrity’’ use this.

‘‘RSA’’ (**OSSL_SELF_TEST_DESC_PCT_RSA_PKCS1**)
‘‘ECDSA’’ (**OSSL_SELF_TEST_DESC_PCT_ECDSA**)
‘‘DSA’’ (**OSSL_SELF_TEST_DESC_PCT_DSA**)
Key generation tests used with the ‘‘Pairwise_Consistency_Test’’ type.

‘‘RSA_Encrypt’’ (**OSSL_SELF_TEST_DESC_ASYM_RSA_ENC**)
‘‘RSA_Decrypt’’ (**OSSL_SELF_TEST_DESC_ASYM_RSA_DEC**)
‘‘KAT_AsymmetricCipher’’ uses this to indicate an encrypt or decrypt KAT.

‘‘AES_GCM’’ (**OSSL_SELF_TEST_DESC_CIPHER_AES_GCM**)
‘‘AES_ECB_Decrypt’’ (**OSSL_SELF_TEST_DESC_CIPHER_AES_ECB**)
‘‘TDES’’ (**OSSL_SELF_TEST_DESC_CIPHER_TDES**)
Symmetric cipher tests used with the ‘‘KAT_Cipher’’ type.

‘‘SHA1’’ (**OSSL_SELF_TEST_DESC_MD_SHA1**)
‘‘SHA2’’ (**OSSL_SELF_TEST_DESC_MD_SHA2**)
‘‘SHA3’’ (**OSSL_SELF_TEST_DESC_MD_SHA3**)
Digest tests used with the ‘‘KAT_Digest’’ type.

‘‘DSA’’ (**OSSL_SELF_TEST_DESC_SIGN_DSA**)
‘‘RSA’’ (**OSSL_SELF_TEST_DESC_SIGN_RSA**)
‘‘ECDSA’’ (**OSSL_SELF_TEST_DESC_SIGN_ECDSA**)
Signature tests used with the ‘‘KAT_Signature’’ type.

‘‘ECDH’’ (**OSSL_SELF_TEST_DESC_KA_ECDH**)
‘‘DH’’ (**OSSL_SELF_TEST_DESC_KA_DH**)
Key agreement tests used with the ‘‘KAT_KA’’ type.

‘‘HKDF’’ (**OSSL_SELF_TEST_DESC_KDF_HKDF**)
‘‘TLS13_KDF_EXTRACT’’ (**OSSL_SELF_TEST_DESC_KDF_TLS13_EXTRACT**)
‘‘TLS13_KDF_EXPAND’’ (**OSSL_SELF_TEST_DESC_KDF_TLS13_EXPAND**)
‘‘SSKDF’’ (**OSSL_SELF_TEST_DESC_KDF_SSKDF**)
‘‘X963KDF’’ (**OSSL_SELF_TEST_DESC_KDF_X963KDF**)
‘‘X942KDF’’ (**OSSL_SELF_TEST_DESC_KDF_X942KDF**)
‘‘PBKDF2’’ (**OSSL_SELF_TEST_DESC_KDF_PBKDF2**)
‘‘SSHKDF’’ (**OSSL_SELF_TEST_DESC_KDF_SSHKDF**)
‘‘TLS12_PRF’’ (**OSSL_SELF_TEST_DESC_KDF_TLS12_PRF**)
‘‘KBKDF’’ (**OSSL_SELF_TEST_DESC_KDF_KBKDF**)
Key Derivation Function tests used with the ‘‘KAT_KDF’’ type.

“CTR” (**OSSL_SELF_TEST_DESC_DRBG_CTR**)
“HASH” (**OSSL_SELF_TEST_DESC_DRBG_HASH**)
“HMAC” (**OSSL_SELF_TEST_DESC_DRBG_HMAC**)
DRBG tests used with the “DRBG” type.

= item “RNG” (**OSSL_SELF_TEST_DESC_RNG**)

“Continuous_RNG_Test” uses this.

EXAMPLES

A simple self test callback is shown below for illustrative purposes.

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

static OSSL_CALLBACK self_test_cb;

static int self_test_cb(const OSSL_PARAM params[], void *arg)
{
  int ret = 0;
  const OSSL_PARAM *p = NULL;
  const char *phase = NULL, *type = NULL, *desc = NULL;

  p = OSSL_PARAM_locate_const(params, OSSL_PROV_PARAM_SELF_TEST_PHASE);
  if (p == NULL || p->data_type != OSSL_PARAM_UTF8_STRING)
      goto err;
  phase = (const char *)p->data;

  p = OSSL_PARAM_locate_const(params, OSSL_PROV_PARAM_SELF_TEST_DESC);
  if (p == NULL || p->data_type != OSSL_PARAM_UTF8_STRING)
      goto err;
  desc = (const char *)p->data;

  p = OSSL_PARAM_locate_const(params, OSSL_PROV_PARAM_SELF_TEST_TYPE);
  if (p == NULL || p->data_type != OSSL_PARAM_UTF8_STRING)
      goto err;
  type = (const char *)p->data;

  /* Do some logging */
  if (strcmp(phase, OSSL_SELF_TEST_PHASE_START) == 0)
      BIO_printf(bio_out, "%s : (%s) : ", desc, type);
  if (strcmp(phase, OSSL_SELF_TEST_PHASE_PASS) == 0
          || strcmp(phase, OSSL_SELF_TEST_PHASE_FAIL) == 0)
      BIO_printf(bio_out, "%s\n", phase);

  /* Corrupt the SHA1 self test during the 'corrupt' phase by returning 0 */
  if (strcmp(phase, OSSL_SELF_TEST_PHASE_CORRUPT) == 0
          && strcmp(desc, OSSL_SELF_TEST_DESC_MD_SHA1) == 0) {
      BIO_printf(bio_out, "%s %s", phase, desc);
      return 0;
  }
  ret = 1;
err:
  return ret;
}
```

SEE ALSO

openssl−fipsinstall (1), **fips_config** (5), **OSSL_SELF_TEST_set_callback** (3), **OSSL_SELF_TEST_new** (3), **OSSL_PARAM** (3), **openssl−core.h** (7), **openssl−core_dispatch.h** (7), **provider** (7)

HISTORY

The type and functions described here were added in OpenSSL 3.0.

COPYRIGHT