## DSA

|  | **Functions** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Mechanism** | **Encrypt**  **&**  **Decrypt** | **Sign**  **&**  **Verify** | **SR**  **&**  **VR**1 | **Digest** | **Gen.**  **Key/**  **Key**  **Pair** | **Wrap**  **&**  **Unwrap** | **Derive** |
| CKM\_DSA\_KEY\_PAIR\_GEN |  |  |  |  |  |  |  |
| CKM\_DSA\_PARAMETER\_GEN |  |  |  |  |  |  |  |
| CKM\_DSA\_PROBALISTIC\_PARAMETER\_GEN |  |  |  |  |  |  |  |
| CKM\_DSA\_SHAWE\_TAYLOR\_PARAMETER\_GEN |  |  |  |  |  |  |  |
| CKM\_DSA\_FIPS\_G\_GEN |  |  |  |  |  |  |  |
| CKM\_DSA |  | 2 |  |  |  |  |  |
| CKM\_DSA\_SHA1 |  |  |  |  |  |  |  |
| CKM\_DSA\_SHA224 |  |  |  |  |  |  |  |
| CKM\_DSA\_SHA256 |  |  |  |  |  |  |  |
| CKM\_DSA\_SHA384 |  |  |  |  |  |  |  |
| CKM\_DSA\_SHA512 |  |  |  |  |  |  |  |

### Definitions

This section defines the key type “CKK\_DSA” for type CK\_KEY\_TYPE as used in the CKA\_KEY\_TYPE attribute of DSA key objects.

Mechanisms:

CKM\_DSA\_KEY\_PAIR\_GEN

CKM\_DSA

CKM\_DSA\_SHA1

CKM\_DSA\_SHA224

CKM\_DSA\_SHA256

CKM\_DSA\_SHA384

CKM\_DSA\_SHA512

CKM\_DSA\_PARAMETER\_GEN

CKM\_DSA\_PROBABLISTIC\_PARAMETER\_GEN

CKM\_DSA\_SHAWE\_TAYLOR\_PARAMETER\_GEN

CKM\_DSA\_FIPS\_G\_GEN

CKM\_FORTEZZA\_TIMESTAMP

* CK\_DSA\_PARAMETER\_GEN\_PARAM

CK\_DSA\_PARAMETER\_GEN\_PARAM is a structure , which provides and returns parameters for the NIST FIPS 186-4 parameter generating algorithms.

typedef struct CK\_DSA\_PARAMETER\_GEN\_PARAM {

CK\_MECHANISM\_TYPE hash;

CK\_BYTE\_PTR pSeed;

CK\_ULONG ulSeedLen;

CK\_ULONG ulIndex;

};

The fields of the structure has the following meanings:

hash Mechanism value for the base hash used in PQG generation, Valid values are CKM\_SHA1, CKM\_SHA244, CKM\_SHA256, CKM\_SHA384, CKM\_SHA512.

pSeed Seed value used to generate PQ and G. This value is returned by CKM\_DSA\_PROBABLISTIC\_PARAMETER\_GEN, CKM\_SHAWE\_TAYLOR\_PARAMETER\_GEN, and passed into CKM\_DSA\_FIPS\_G\_GEN.

ulSeedLen Length of seed value.

ulIndex Index value for generating G. Input for CKM\_DSA\_FIPS\_G\_GEN. Ignored by CKM\_DSA\_PROBALISTIC\_PARAMETER\_GEN and CKM\_SHAWE\_TAYLOR\_PARAMETER\_GEN.

### DSA public key objects

DSA public key objects (object class **CKO\_PUBLIC\_KEY,** key type **CKK\_DSA**) hold DSA public keys. The following table defines the DSA public key object attributes, in addition to the common attributes defined for this object class:

Table 1, DSA Public Key Object Attributes

| **Attribute** | **Data type** | **Meaning** |
| --- | --- | --- |
| CKA\_PRIME1,3 | Big integer | Prime *p* (512 to 1024 bits, in steps of 64 bits, 2048 bits, or 3072 bits) |
| CKA\_SUBPRIME1,3 | Big integer | Subprime *q* (160 bits, 224 bits, or 256 bits) |
| CKA\_BASE1,3 | Big integer | Base *g* |
| CKA\_VALUE1,4 | Big integer | Public value *y* |

- Refer to [PKCS #11-B] table 15 for footnotes

The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the “DSA domain parameters”. See FIPS PUB 186-4 for more information on DSA keys.

The following is a sample template for creating a DSA public key object:

CK\_OBJECT\_CLASS class = CKO\_PUBLIC\_KEY;

CK\_KEY\_TYPE keyType = CKK\_DSA;

CK\_UTF8CHAR label[] = “A DSA public key object”;

CK\_BYTE prime[] = {...};

CK\_BYTE subprime[] = {...};

CK\_BYTE base[] = {...};

CK\_BYTE value[] = {...};

CK\_BBOOL true = CK\_TRUE;

CK\_ATTRIBUTE template[] = {

{CKA\_CLASS, &class, sizeof(class)},

{CKA\_KEY\_TYPE, &keyType, sizeof(keyType)},

{CKA\_TOKEN, &true, sizeof(true)},

{CKA\_LABEL, label, sizeof(label)-1},

{CKA\_PRIME, prime, sizeof(prime)},

{CKA\_SUBPRIME, subprime, sizeof(subprime)},

{CKA\_BASE, base, sizeof(base)},

{CKA\_VALUE, value, sizeof(value)}

};

### DSA Key Restrictions

FIPS PUB 186-4 specifies permitted combinations of prime and sub-prime lengths. They are:

* Prime: 1024 bits, Subprime: 160
* Prime: 2048 bits, Subprime: 224
* Prime: 2048 bits, Subprime: 256
* Prime: 3072 bits, Subprime: 256

Earlier versions of FIPS 186 permitted smaller prime lengths, and those are included here for backwards compatibility. A implementation that is compliant to FIPS 186-4 does not permit the use of primes of any length less than 1024 bits.

### DSA private key objects

DSA private key objects (object class **CKO\_PRIVATE\_KEY,** key type **CKK\_DSA**) hold DSA private keys. The following table defines the DSA private key object attributes, in addition to the common attributes defined for this object class:

Table 2, DSA Private Key Object Attributes

| **Attribute** | **Data type** | **Meaning** |
| --- | --- | --- |
| CKA\_PRIME1,4,6 | Big integer | Prime *p* (512 to 1024 bits, in steps of 64 bits, 2048 bits, or 3072 bits) |
| CKA\_SUBPRIME1,4,6 | Big integer | Subprime *q* (160 bits, 224 bits, or 256 bits) |
| CKA\_BASE1,4,6 | Big integer | Base *g* |
| CKA\_VALUE1,4,6,7 | Big integer | Private value *x* |

- Refer to [PKCS #11-B] table 15 for footnotes

The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the “DSA domain parameters”. See FIPS PUB 186-4 for more information on DSA keys.

Note that when generating a DSA private key, the DSA domain parameters are *not* specified in the key’s template. This is because DSA private keys are only generated as part of a DSA key *pair*, and the DSA domain parameters for the pair are specified in the template for the DSA public key.

The following is a sample template for creating a DSA private key object:

CK\_OBJECT\_CLASS class = CKO\_PRIVATE\_KEY;

CK\_KEY\_TYPE keyType = CKK\_DSA;

CK\_UTF8CHAR label[] = “A DSA private key object”;

CK\_BYTE subject[] = {...};

CK\_BYTE id[] = {123};

CK\_BYTE prime[] = {...};

CK\_BYTE subprime[] = {...};

CK\_BYTE base[] = {...};

CK\_BYTE value[] = {...};

CK\_BBOOL true = CK\_TRUE;

CK\_ATTRIBUTE template[] = {

{CKA\_CLASS, &class, sizeof(class)},

{CKA\_KEY\_TYPE, &keyType, sizeof(keyType)},

{CKA\_TOKEN, &true, sizeof(true)},

{CKA\_LABEL, label, sizeof(label)-1},

{CKA\_SUBJECT, subject, sizeof(subject)},

{CKA\_ID, id, sizeof(id)},

{CKA\_SENSITIVE, &true, sizeof(true)},

{CKA\_SIGN, &true, sizeof(true)},

{CKA\_PRIME, prime, sizeof(prime)},

{CKA\_SUBPRIME, subprime, sizeof(subprime)},

{CKA\_BASE, base, sizeof(base)},

{CKA\_VALUE, value, sizeof(value)}

};

### DSA domain parameter objects

DSA domain parameter objects (object class **CKO\_DOMAIN\_PARAMETERS,** key type **CKK\_DSA**) hold DSA domain parameters. The following table defines the DSA domain parameter object attributes, in addition to the common attributes defined for this object class:

Table 3, DSA Domain Parameter Object Attributes

| **Attribute** | **Data type** | **Meaning** |
| --- | --- | --- |
| CKA\_PRIME1,4 | Big integer | Prime *p* (512 to 1024 bits, in steps of 64 bits, 2048 bits, or 3072 bits) |
| CKA\_SUBPRIME1,4 | Big integer | Subprime *q* (160 bits, 224 bits, or 256 bits) |
| CKA\_BASE1,4 | Big integer | Base *g* |
| CKA\_PRIME\_BITS2,3 | CK\_ULONG | Length of the prime value. |
| CKA\_SUBPRIME\_BITS2 | CK\_ULONG | Length of the subprime value. |

- Refer to [PKCS #11-B] table 15 for footnotes

The **CKA\_PRIME**, **CKA\_SUBPRIME** and **CKA\_BASE** attribute values are collectively the “DSA domain parameters”. See FIPS PUB 186-4 for more information on DSA domain parameters.

To ensure backwards compatibility, if **CKA\_SUBPRIME\_BITS** is not specified for a call to **C\_GenerateKey**, it takes on a default based on the value of **CKA\_PRIME\_BITS** as follows:

* If **CKA\_PRIME\_BITS** is less than or equal to 1024 then CKA\_SUBPRIME\_BITS shall be 160 bits
* If **CKA\_PRIME\_BITS** equals 2048 then CKA\_SUBPRIME\_BITS shall be 224 bits
* If **CKA\_PRIME\_BITS** equals 3072 then CKA\_SUBPRIME\_BITS shall be 256 bits

The following is a sample template for creating a DSA domain parameter object:

CK\_OBJECT\_CLASS class = CKO\_DOMAIN\_PARAMETERS;

CK\_KEY\_TYPE keyType = CKK\_DSA;

CK\_UTF8CHAR label[] = “A DSA domain parameter object”;

CK\_BYTE prime[] = {...};

CK\_BYTE subprime[] = {...};

CK\_BYTE base[] = {...};

CK\_BBOOL true = CK\_TRUE;

CK\_ATTRIBUTE template[] = {

{CKA\_CLASS, &class, sizeof(class)},

{CKA\_KEY\_TYPE, &keyType, sizeof(keyType)},

{CKA\_TOKEN, &true, sizeof(true)},

{CKA\_LABEL, label, sizeof(label)-1},

{CKA\_PRIME, prime, sizeof(prime)},

{CKA\_SUBPRIME, subprime, sizeof(subprime)},

{CKA\_BASE, base, sizeof(base)},

};

### DSA key pair generation

The DSA key pair generation mechanism, denoted **CKM\_DSA\_KEY\_PAIR\_GEN**, is a key pair generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-2.

This mechanism does not have a parameter.

The mechanism generates DSA public/private key pairs with a particular prime, subprime and base, as specified in the **CKA\_PRIME**, **CKA\_SUBPRIME**, and **CKA\_BASE** attributes of the template for the public key.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_VALUE** attributes to the new public key and the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_SUBPRIME**, **CKA\_BASE**, and **CKA\_VALUE** attributes to the new private key. Other attributes supported by the DSA public and private key types (specifically, the flags indicating which functions the keys support) may also be specified in the templates for the keys, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA domain parameter generation

Note: This is a FIPS 186-2 legacy mechanism and may not necessarily be present in implementations which are compliant with FIPS 186-4.

The DSA domain parameter generation mechanism, denoted **CKM\_DSA\_PARAMETER\_GEN**, is a domain parameter generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-2.

This mechanism does not have a parameter.

The mechanism generates DSA domain parameters with a particular prime length in bits, as specified in the **CKA\_PRIME\_BITS** attribute of the template.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_SUBPRIME**, **CKA\_BASE** and **CKA\_PRIME\_BITS** attributes to the new object. Other attributes supported by the DSA domain parameter types may also be specified in the template, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA probabilistic domain parameter generation

The DSA probabilistic domain parameter generation mechanism, denoted **CKM\_DSA\_PROBABLISTIC\_PARAMETER\_GEN**, is a domain parameter generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-4, section Appendix A.1.1 Generation and Validation of Probable Primes..

This mechanism takes a **CK\_DSA\_PARAMETER\_GEN\_PARAM** which supplies the base hash and returns the seed (pSeed) and the length (ulSeedLen).

The mechanism generates DSA the prime and subprime domain parameters with a particular prime length in bits, as specified in the **CKA\_PRIME\_BITS** attribute of the template and the subprime length as specified in the **CKA\_SUBPRIME\_BITS** attribute of the template.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_SUBPRIME**, **CKA\_PRIME\_BITS, and CKA\_SUBPRIME\_BITS** attributes to the new object. **CKA\_BASE** is not set by this call. Other attributes supported by the DSA domain parameter types may also be specified in the template, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA Shawe-Taylor domain parameter generation

The DSA Shawe-Taylor domain parameter generation mechanism, denoted **CKM\_DSA\_SHAWE\_TAYLOR\_PARAMETER\_GEN**, is a domain parameter generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-4, section Appendix A.1.2 Construction and Validation of Provable Primes p and q.

This mechanism takes a **CK\_DSA\_PARAMETER\_GEN\_PARAM** which supplies the base hash and returns the seed (pSeed) and the length (ulSeedLen).

The mechanism generates DSA the prime and subprime domain parameters with a particular prime length in bits, as specified in the CKA\_PRIME\_BITS attribute of the template and the subprime length as specified in the **CKA\_SUBPRIME\_BITS** attribute of the template.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, **CKA\_PRIME**, **CKA\_SUBPRIME**, **CKA\_PRIME\_BITS, and CKA\_SUBPRIME\_BITS** attributes to the new object. **CKA\_BASE** is not set by this call. Other attributes supported by the DSA domain parameter types may also be specified in the template, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA base domain parameter generation

The DSA base domain parameter generation mechanism, denoted **CKM\_DSA\_FIPS\_G\_GEN**, is a base parameter generation mechanism based on the Digital Signature Algorithm defined in FIPS PUB 186-4, section Appendix A.2 Generation of Generator G.

This mechanism takes a **CK\_DSA\_PARAMETER\_GEN\_PARAM** which supplies the base hash the seed (pSeed) and the length (ulSeedLen) and the index value.

The mechanism generates DSA the base domain parameter with as specified in the **CKA\_PRIME** and **CKA\_SUBPRIME** attributes of the template.

The mechanism contributes the **CKA\_CLASS**, **CKA\_KEY\_TYPE**, and **CKA\_BASE** attributes to the new object.Other attributes supported by the DSA domain parameter types may also be specified in the template, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA without hashing

The DSA without hashing mechanism, denoted **CKM\_DSA**, is a mechanism for single-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4. (This mechanism corresponds only to the part of DSA that processes the 20-byte hash value; it does not compute the hash value.)

For the purposes of this mechanism, a DSA signature is a 40-byte string, corresponding to the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

It does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 4, DSA: Key And Data Length

| **Function** | **Key type** | **Input length** | **Output length** |
| --- | --- | --- | --- |
| C\_Sign1 | DSA private key | 20, 28, 32, 48, or 64 bits | 2\*length of subprime |
| C\_Verify1 | DSA public key | (20, 28, 32, 48, or 64 bits), (2\*length of subprime)2 | N/A |

1 Single-part operations only.

2 Data length, signature length.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA with SHA-1

The DSA with SHA-1 mechanism, denoted **CKM\_DSA\_SHA1**, is a mechanism for single- and multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA-1.

For the purposes of this mechanism, a DSA signature is a string of length 2\*subprime, corresponding to the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

This mechanism does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 5, DSA with SHA-1: Key And Data Length

| **Function** | **Key type** | **Input length** | **Output length** |
| --- | --- | --- | --- |
| C\_Sign | DSA private key | any2\*subprime length |
| C\_Verify | DSA public key | any, 2\*subprime length2 | N/A |

2 Data length, signature length.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA with SHA-224

The DSA with SHA-1 mechanism, denoted **CKM\_DSA\_SHA224**, is a mechanism for single- and multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA-224.

For the purposes of this mechanism, a DSA signature is a string of length 2\*subprime, corresponding to the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

This mechanism does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 6, DSA with SHA-244: Key And Data Length

| **Function** | **Key type** | **Input length** | **Output length** |
| --- | --- | --- | --- |
| C\_Sign | DSA private key | any | 2\*subprime length |
| C\_Verify | DSA public key | any, 2\*subprime length2 | N/A |

2 Data length, signature length.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK\_MECHANISM\_INFO** structure specify the supported range of DSA prime sizes, in bits.

### DSA with SHA-256

The DSA with SHA-1 mechanism, denoted **CKM\_DSA\_SHA256**, is a mechanism for single- and multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA-256.

For the purposes of this mechanism, a DSA signature is a string of length 2\*subprime, corresponding to the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

This mechanism does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 7, DSA with SHA-256: Key And Data Length

| **Function** | **Key type** | **Input length** | **Output length** |
| --- | --- | --- | --- |
| C\_Sign | DSA private key | any | 2\*subprime length |
| C\_Verify | DSA public key | any, 2\*subprime length2 | N/A |

2 Data length, signature length.

### DSA with SHA-384

The DSA with SHA-1 mechanism, denoted **CKM\_DSA\_SHA384**, is a mechanism for single- and multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA-384.

For the purposes of this mechanism, a DSA signature is a string of length 2\*subprime, corresponding to the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

This mechanism does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 8, DSA with SHA-384: Key And Data Length

| **Function** | **Key type** | **Input length** | **Output length** |
| --- | --- | --- | --- |
| C\_Sign | DSA private key | any | 2\*subprime length |
| C\_Verify | DSA public key | any, 2\*subprime length2 | N/A |

2 Data length, signature length.

### DSA with SHA-512

The DSA with SHA-1 mechanism, denoted **CKM\_DSA\_SHA512**, is a mechanism for single- and multiple-part signatures and verification based on the Digital Signature Algorithm defined in FIPS PUB 186-4. This mechanism computes the entire DSA specification, including the hashing with SHA-512.

For the purposes of this mechanism, a DSA signature is a string of length 2\*subprime, corresponding to the concatenation of the DSA values *r* and *s*, each represented most-significant byte first.

This mechanism does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 9, DSA with SHA-512: Key And Data Length

| **Function** | **Key type** | **Input length** | **Output length** |
| --- | --- | --- | --- |
| C\_Sign | DSA private key | any | 2\*subprime length |
| C\_Verify | DSA public key | any, 2\*subprime length2 | N/A |

2 Data length, signature length.

# Manifest Constants

The following definitions can be found in the appropriate header file.

Also, refer [PKCS #11-B] for additional definitions.

#define CKK\_RSA 0x00000000

#define CKK\_DSA 0x00000001

#define CKK\_DH 0x00000002

#define CKK\_ECDSA 0x00000003

#define CKK\_EC 0x00000003

#define CKK\_X9\_42\_DH 0x00000004

#define CKK\_GENERIC\_SECRET 0x00000010

#define CKK\_RC2 0x00000011

#define CKK\_RC4 0x00000012

#define CKK\_DES 0x00000013

#define CKK\_DES2 0x00000014

#define CKK\_DES3 0x00000015

#define CKK\_CDMF 0x0000001E

#define CKK\_AES 0x0000001F

#define CKK\_BLOWFISH 0x00000020

#define CKK\_TWOFISH 0x00000021

#define CKK\_ARIA 0x00000024

#define CKK\_CAMELLIA 0x00000025

#define CKK\_SEED 0x00000026

#define CKK\_MD5\_HMAC 0x00000027

#define CKK\_SHA\_1\_HMAC 0x00000028

#define CKK\_RIPEMD128\_HMAC 0x00000029

#define CKK\_RIPEMD160\_HMAC 0x0000002A

#define CKK\_SHA256\_HMAC 0x0000002B

#define CKK\_SHA384\_HMAC 0x0000002C

#define CKK\_SHA512\_HMAC 0x0000002D

#define CKK\_SHA224\_HMAC 0x0000002E

#define CKK\_GOSTR3410 0x00000030

#define CKK\_GOSTR3411 0x00000031

#define CKK\_GOST28147 0x00000032

#define CKK\_VENDOR\_DEFINED 0x80000000

#define CKC\_X\_509 0x00000000

#define CKC\_X\_509\_ATTR\_CERT 0x00000001

#define CKC\_WTLS 0x00000002

#define CKC\_VENDOR\_DEFINED 0x80000000

#define CKD\_NULL                       0x00000001

#define CKD\_SHA1\_KDF                     0x00000002

#define CKD\_SHA1\_KDF\_ASN1               0x00000003

#define CKD\_SHA1\_KDF\_CONCATENATE 0x00000004

#define CKD\_SHA224\_KDF 0x00000005

#define CKD\_SHA256\_KDF 0x00000006

#define CKD\_SHA384\_KDF 0x00000007

#define CKD\_SHA512\_KDF 0x00000008

#define CKD\_CPDIVERSIFY\_KDF 0x00000009

#define CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN 0x00000000

#define CKM\_RSA\_PKCS 0x00000001

#define CKM\_RSA\_9796 0x00000002

#define CKM\_RSA\_X\_509 0x00000003

#define CKM\_SHA1\_RSA\_PKCS 0x00000006

#define CKM\_RSA\_PKCS\_OAEP 0x00000009

#define CKM\_RSA\_X9\_31\_KEY\_PAIR\_GEN 0x0000000A

#define CKM\_RSA\_X9\_31 0x0000000B

#define CKM\_SHA1\_RSA\_X9\_31 0x0000000C

#define CKM\_RSA\_PKCS\_PSS 0x0000000D

#define CKM\_SHA1\_RSA\_PKCS\_PSS 0x0000000E

#define CKM\_DSA\_KEY\_PAIR\_GEN 0x00000010

#define CKM\_DSA 0x00000011

#define CKM\_DSA\_SHA1 0x00000012

#define CKM\_DSA\_SHA244 0x00000013

#define CKM\_DSA\_SHA256 0x00000014

#define CKM\_DSA\_SHA384 0x00000015

#define CKM\_DSA\_SHA512 0x00000016

#define CKM\_DH\_PKCS\_KEY\_PAIR\_GEN 0x00000020

#define CKM\_DH\_PKCS\_DERIVE 0x00000021

#define CKM\_X9\_42\_DH\_KEY\_PAIR\_GEN 0x00000030

#define CKM\_X9\_42\_DH\_DERIVE 0x00000031

#define CKM\_X9\_42\_DH\_HYBRID\_DERIVE 0x00000032

#define CKM\_X9\_42\_MQV\_DERIVE 0x00000033

#define CKM\_SHA256\_RSA\_PKCS 0x00000040

#define CKM\_SHA384\_RSA\_PKCS 0x00000041

#define CKM\_SHA512\_RSA\_PKCS 0x00000042

#define CKM\_SHA256\_RSA\_PKCS\_PSS 0x00000043

#define CKM\_SHA384\_RSA\_PKCS\_PSS 0x00000044

#define CKM\_SHA512\_RSA\_PKCS\_PSS 0x00000045

#define CKM\_RC2\_KEY\_GEN 0x00000100

#define CKM\_DES2\_KEY\_GEN 0x00000130

#define CKM\_DES3\_KEY\_GEN 0x00000131

#define CKM\_DES3\_ECB 0x00000132

#define CKM\_DES3\_CBC 0x00000133

#define CKM\_DES3\_MAC 0x00000134

#define CKM\_DES3\_MAC\_GENERAL 0x00000135

#define CKM\_DES3\_CBC\_PAD 0x00000136

#define CKM\_DES3\_CMAC\_GENERAL 0x00000137

#define CKM\_DES3\_CMAC 0x00000138

#define CKM\_CDMF\_KEY\_GEN 0x00000140

#define CKM\_CDMF\_ECB 0x00000141

#define CKM\_CDMF\_CBC 0x00000142

#define CKM\_CDMF\_MAC 0x00000143

#define CKM\_CDMF\_MAC\_GENERAL 0x00000144

#define CKM\_CDMF\_CBC\_PAD 0x00000145

#define CKM\_DES\_OFB64 0x00000150

#define CKM\_DES\_OFB8 0x00000151

#define CKM\_DES\_CFB64 0x00000152

#define CKM\_DES\_CFB8 0x00000153

#define CKM\_SHA\_1 0x00000220

#define CKM\_SHA\_1\_HMAC 0x00000221

#define CKM\_SHA\_1\_HMAC\_GENERAL 0x00000222

#define CKM\_SHA256 0x00000250

#define CKM\_SHA256\_HMAC 0x00000251

#define CKM\_SHA256\_HMAC\_GENERAL 0x00000252

#define CKM\_SHA384 0x00000260

#define CKM\_SHA384\_HMAC 0x00000261

#define CKM\_SHA384\_HMAC\_GENERAL 0x00000262

#define CKM\_SHA512 0x00000270

#define CKM\_SHA512\_HMAC 0x00000271

#define CKM\_SHA512\_HMAC\_GENERAL 0x00000272

#define CKM\_GENERIC\_SECRET\_KEY\_GEN 0x00000350

#define CKM\_CONCATENATE\_BASE\_AND\_KEY 0x00000360

#define CKM\_CONCATENATE\_BASE\_AND\_DATA 0x00000362

#define CKM\_CONCATENATE\_DATA\_AND\_BASE 0x00000363

#define CKM\_XOR\_BASE\_AND\_DATA 0x00000364

#define CKM\_EXTRACT\_KEY\_FROM\_KEY 0x00000365

#define CKM\_SSL3\_PRE\_MASTER\_KEY\_GEN 0x00000370

#define CKM\_SSL3\_MASTER\_KEY\_DERIVE 0x00000371

#define CKM\_SSL3\_KEY\_AND\_MAC\_DERIVE 0x00000372

#define CKM\_SSL3\_MASTER\_KEY\_DERIVE\_DH 0x00000373

#define CKM\_TLS\_PRE\_MASTER\_KEY\_GEN 0x00000374

#define CKM\_TLS\_MASTER\_KEY\_DERIVE 0x00000375

#define CKM\_TLS\_KEY\_AND\_MAC\_DERIVE 0x00000376

#define CKM\_TLS\_MASTER\_KEY\_DERIVE\_DH 0x00000377

#define CKM\_TLS\_PRF 0x00000378

#define CKM\_SSL3\_MD5\_MAC 0x00000380

#define CKM\_SSL3\_SHA1\_MAC 0x00000381

#define CKM\_MD5\_KEY\_DERIVATION 0x00000390

#define CKM\_MD2\_KEY\_DERIVATION 0x00000391

#define CKM\_SHA1\_KEY\_DERIVATION 0x00000392

#define CKM\_SHA256\_KEY\_DERIVATION 0x00000393

#define CKM\_SHA384\_KEY\_DERIVATION 0x00000394

#define CKM\_SHA512\_KEY\_DERIVATION 0x00000395

#define CKM\_PBE\_SHA1\_DES3\_EDE\_CBC 0x000003A8

#define CKM\_PBE\_SHA1\_DES2\_EDE\_CBC 0x000003A9

#define CKM\_PBE\_SHA1\_RC2\_128\_CBC 0x000003AA

#define CKM\_PBE\_SHA1\_RC2\_40\_CBC 0x000003AB

#define CKM\_PKCS5\_PBKD2 0x000003B0

#define CKM\_PBA\_SHA1\_WITH\_SHA1\_HMAC 0x000003C0

#define CKM\_WTLS\_PRE\_MASTER\_KEY\_GEN 0x000003D0

#define CKM\_WTLS\_MASTER\_KEY\_DERIVE 0x000003D1

#define CKM\_WTLS\_MASTER\_KEY\_DERVIE\_DH\_ECC 0x000003D2

#define CKM\_WTLS\_PRF 0x000003D3

#define CKM\_WTLS\_SERVER\_KEY\_AND\_MAC\_DERIVE 0x000003D4

#define CKM\_WTLS\_CLIENT\_KEY\_AND\_MAC\_DERIVE 0x000003D5

#define CKM\_KEY\_WRAP\_LYNKS 0x00000400

#define CKM\_KEY\_WRAP\_SET\_OAEP 0x00000401

#define CKM\_CMS\_SIG 0x00000500

#define CKM\_ECDSA\_KEY\_PAIR\_GEN 0x00001040

#define CKM\_EC\_KEY\_PAIR\_GEN 0x00001040

#define CKM\_ECDSA 0x00001041

#define CKM\_ECDSA\_SHA1 0x00001042

#define CKM\_ECDH1\_DERIVE 0x00001050

#define CKM\_ECDH1\_COFACTOR\_DERIVE 0x00001051

#define CKM\_ECMQV\_DERIVE 0x00001052

#define CKM\_AES\_KEY\_GEN 0x00001080

#define CKM\_AES\_ECB 0x00001081

#define CKM\_AES\_CBC 0x00001082

#define CKM\_AES\_MAC 0x00001083

#define CKM\_AES\_MAC\_GENERAL 0x00001084

#define CKM\_AES\_CBC\_PAD 0x00001085

#define CKM\_AES\_CMAC\_GENERAL 0x00001089

#define CKM\_AES\_CMAC 0x0000108A

#define CKM\_BLOWFISH\_KEY\_GEN 0x00001090

#define CKM\_BLOWFISH\_CBC 0x00001091

#define CKM\_TWOFISH\_KEY\_GEN 0x00001092

#define CKM\_TWOFISH\_CBC 0x00001093

#define CKM\_DES\_ECB\_ENCRYPT\_DATA 0x00001100

#define CKM\_DES\_CBC\_ENCRYPT\_DATA 0x00001101

#define CKM\_DES3\_ECB\_ENCRYPT\_DATA 0x00001102

#define CKM\_DES3\_CBC\_ENCRYPT\_DATA 0x00001103

#define CKM\_AES\_ECB\_ENCRYPT\_DATA 0x00001104

#define CKM\_AES\_CBC\_ENCRYPT\_DATA 0x00001105

#define CKM\_DSA\_PARAMETER\_GEN 0x00002000

#define CKM\_DSA\_PROBALISTIC\_PARAMETER\_GEN 0x00002010

#define CKM\_DSA\_SHAWE\_TAYLOR\_PARAMETER\_GEN 0x00002011

#define CKM\_DH\_PKCS\_PARAMETER\_GEN 0x00002001

#define CKM\_X9\_42\_DH\_PARAMETER\_GEN 0x00002002

#define CKM\_SHA224 0x00000255

#define CKM\_SHA224\_HMAC 0x00000256

#define CKM\_SHA224\_HMAC\_GENERAL 0x00000257

#define CKM\_SHA224\_RSA\_PKCS 0x00000046

#define CKM\_SHA224\_RSA\_PKCS\_PSS 0x00000047

#define CKM\_SHA224\_KEY\_DERIVATION 0x00000396

#define CKG\_MGF1\_SHA224 0x00000005

#define CKM\_AES\_CTR 0x00001086

#define CKM\_AES\_CTS 0x00001089

#define CKM\_KIP\_DERIVE 0x00000510

#define CKM\_KIP\_WRAP 0x00000511

#define CKM\_KIP\_MAC 0x00000512

#define CKM\_CAMELLIA\_KEY\_GEN 0x00000550

#define CKM\_CAMELLIA\_ECB 0x00000551

#define CKM\_CAMELLIA\_CBC 0x00000552

#define CKM\_CAMELLIA\_MAC 0x00000553

#define CKM\_CAMELLIA\_MAC\_GENERAL 0x00000554

#define CKM\_CAMELLIA\_CBC\_PAD 0x00000555

#define CKM\_CAMELLIA\_ECB\_ENCRYPT\_DATA 0x00000556

#define CKM\_CAMELLIA\_CBC\_ENCRYPT\_DATA 0x00000557

#define CKM\_ARIA\_KEY\_GEN 0x00000560

#define CKM\_ARIA\_ECB 0x00000561

#define CKM\_ARIA\_CBC 0x00000562

#define CKM\_ARIA\_MAC 0x00000563

#define CKM\_ARIA\_MAC\_GENERAL 0x00000564

#define CKM\_ARIA\_CBC\_PAD 0x00000565

#define CKM\_ARIA\_ECB\_ENCRYPT\_DATA 0x00000566

#define CKM\_ARIA\_CBC\_ENCRYPT\_DATA 0x00000567

#define CKM\_SEED\_KEY\_GEN 0x00000650

#define CKM\_SEED\_ECB 0x00000651

#define CKM\_SEED\_CBC 0x00000652

#define CKM\_SEED\_MAC 0x00000653

#define CKM\_SEED\_MAC\_GENERAL 0x00000654

#define CKM\_SEED\_CBC\_PAD 0x00000655

#define CKM\_SEED\_ECB\_ENCRYPT\_DATA 0x00000656

#define CKM\_SEED\_CBC\_ENCRYPT\_DATA 0x00000657

#define CKM\_AES\_GCM 0x00001087

#define CKM\_AES\_CCM 0x00001088

#define CKM\_AES\_OFB 0x00002104

#define CKM\_AES\_CFB64 0x00002105

#define CKM\_AES\_CFB8 0x00002106

#define CKM\_AES\_CFB128 0x00002107

#define CKM\_BLOWFISH\_CBC\_PAD 0x00001094

#define CKM\_TWOFISH\_CBC\_PAD 0x00001095

#define CKM\_AES\_KEY\_WRAP 0x00001090

#define CKM\_AES\_KEY\_WRAP\_PAD 0x00001091

#define CKM\_RSA\_PKCS\_TPM\_1\_1 0x00004001

#define CKM\_RSA\_PKCS\_OAEP\_TPM\_1\_1 0x00004002

#define CKM\_GOSTR3410\_KEY\_PAIR\_GEN 0x00001200

#define CKM\_GOSTR3410 0x00001201

#define CKM\_GOSTR3410\_WITH\_GOSTR3411 0x00001202

#define CKM\_GOSTR3410\_KEY\_WRAP 0x00001203

#define CKM\_GOSTR3410\_DERIVE 0x00001204

#define CKM\_GOSTR3411 0x00001210

#define CKM\_GOSTR3411\_HMAC 0x00001211

#define CKM\_GOST28147\_KEY\_GEN 0x00001220

#define CKM\_GOST28147\_ECB 0x00001221

#define CKM\_GOST28147 0x00001222

#define CKM\_GOST28147\_MAC 0x00001223

#define CKM\_GOST28147\_KEY\_WRAP 0x00001224

#define CKA\_GOSTR3410\_PARAMS 0x00000250

#define CKA\_GOSTR3411\_PARAMS 0x00000251

#define CKA\_GOST28147\_PARAMS 0x00000252

#define CKM\_VENDOR\_DEFINED 0x80000000