Flexible Symmetric Key Derivation Mechanism – Draft 1

March 13, 2017

Author: Darren Johnson

Summary:

SafeNet currently supports a vendor defined mechanism (CKM\_NIST\_PRF\_KDF) for a key derivation function (KDF) that implements a small subset of the functionality defined in NIST SP800-108. Rather than promote the existing mechanism, this proposal introduces a new mechanism for a flexible key derivation function that is based on (and includes all the functionality defined in) NIST SP800-108, but extends it further to meet some additional customer requirements.

Proposal:

The CKM\_FLEXIBLE\_KDF mechanism is based on the KDF framework defined in NIST SP800-108 which outlines various methods of using a PRF (pseudo random function) to derive a symmetric key from another symmetric key. This mechanism extends beyond what is defined in NIST SP800-108 to allow for additional use cases. For example:

* provide multiple counters (this feature is not based on any requirement and is a side effect of the design and is unlikely to be supported by any vendor)

The mechanism parameter structure is based on the input parameters defined in NIST SP800-108 which allow a KDF type, PRF type and PRF input to be specified.

The PRFs defined in NIST SP800-108 take two input parameters; a key and an iteration variable combined with optional fixed input data. For this mechanism, the key parameter is taken from the *hBaseKey* parameter to **C\_Derive**. The PRF input data is constructed from the array of CK\_FKDF\_DATA\_PARAM structures in the mechanism parameter. All of the values defined by the CK\_FKDF\_DATA\_PARAM array are concatenated in the order they are defined and passed in to the PRF as the data parameter.

This mechanism requires that at least one CK\_FKDF\_DATA\_PARAM of type CK\_FKDF\_COUNTER must be defined. Beyond that requirement, any number of CK\_FKDF\_DATA\_PARAM structures can be defines as was as any number of duplicate types and values. Any addition limitation and restrictions on the number and type of CK\_FKDF\_DATA\_PARAM structures are vendor-specific.

The CK\_FKDF\_PARAMS structure provides the input and output parameters to the CKM\_FLEXIBLE\_KDF mechanism. It is defined as follows:

typedef struct CK\_FKDF\_PARAMS {

 CK\_FKDF\_KDF\_TYPE kdfType;

 CK\_FKDF\_PRF\_TYPE prfType;

 CK\_ULONG ulNumberOfDataParams;

 CK\_FKDF\_DATA\_PARAM\_PTR pDataParams;

 CK\_ULONG ulDerivedKeys;

 CK\_FKDF\_DERIVED\_KEYS pDerivedKeys;

} CK\_FKDF\_PARAMS;

typedef CK\_FKDF\_PARAMS CK\_PTR CK\_FKDF\_PARAMS;

The fields of the CK\_FKDF\_PARAMS structure have the following meaning:

 kdfType type of KDF (counter, feedback, pipeline)

 prfType type of PRR (HMAC, CMAC)

 ulNumberOfDataParams number of elements in the array pointed to by pDataParams

 pDataParams an array of CK\_FKDF\_DATA\_PARAM structures. The array defines input parameters that make up the “data” input to the PRF.

 ulAdditionalDerivedKeys number of additional keys that will be derived and the number of elements in the array pointed to by pDerivedKeys.

 pDerivedKeys array of CK\_FDF\_DERIVED\_KEYS structures

The supported values for CK\_FKDF\_KDF\_TYPE are taken from and defined in NIST SP800-108.

typedef CK\_ULONG CK\_FKDF\_KDF\_TYPE;

#define CK\_FKDF\_COUNTER 0x00000000

#define CK\_FKDF\_FEEDBACK 0x00000001

#define CK\_FKDF\_DOUBLE\_PIPELINE 0x00000002

The supported values for CK\_FKDF\_PRF\_TYPE are taken from and defined in SP800-108.

typedef CK\_ULONG CK\_FKDF\_PRF\_TYPE;

#define CK\_PRF\_HMAC\_SHA1 0x00000000

#define CK\_PRF\_HMAC\_SHA224 0x00000001

#define CK\_PRF\_HMAC\_SHA256 0x00000002

… (additional SHA2 and SHA3 HMACs)

#define CK\_PRF\_3DES\_CMAC 0x00000003

#define CK\_PRF\_AES\_CMAC 0x00000004

(consider using existing mechanism instead of additional types?)

The CK\_FKDF\_DATA\_PARAM structure is used to define a piece of data that makes up part of the PRF data parameter.

The following types of data can be defined by a CF\_FKDF\_DATA\_PARAM structure:

typedef CK\_ULONG CK\_FKDF\_DATA\_TYPE;

#define CK\_FKDF\_COUNTER 0x00000000

#define CK\_FKDF\_BYTE\_ARRAY 0x00000001

#define CK\_FKDF\_KEY\_HANDLE 0x00000002

 The CK\_FKDF\_DATA\_PARAM structure is defined as follows:

typedef struct CK\_FKDF\_DATA\_PARAM {

 CK\_FKDF\_DATA\_TYPE type;

 CK\_VOID\_PTR pValue;

 CK\_ULONG ulValueLen;

} CK\_FKDF\_DATA\_PARAM;

typedef CK\_FKDF\_DATA\_PARAM CK\_PTR CK\_FKDF\_DATA\_PARAM\_PTR;

The fields of the CK\_FKDF\_DATA\_PARAM structure have the following meaning:

 type defines the type of data pointed to by pValue

 pValue pointer to the data defined by type

 ulValueLen size of the data pointed to by pValue

If the *type* field of the CK\_FKDF\_DATA\_PARAM structure is set to CK\_FKDF\_COUNTER, then *pValue* must be assigned a valid CK\_FKDF\_COUNTER\_PARAM\_PTR and *ulValueLen* must be set to sizeof(CK\_FKDF\_COUNTER\_PARAM).

If the *type* field of the CK\_FKDF\_DATA\_PARAM structure is set to CK\_FKDF\_BYTE\_ARRAY, then *pValue* must be assigned a valid CK\_BYTE\_PTR value and *ulValueLen* must be set to a non-zero length.

If the *type* field of the CK\_FKDF\_DATA\_PARAM structure is set to CK\_FKDF\_KEY\_HANDLE, then *pValue* must be assigned a valid CK\_OBJECT\_HANDLE\_PTR value and *ulValueLen* must be set to sizeof(CK\_OBJECT\_HANDLE). The object handle provided must be for an object of type CKO\_SECRET and must have CKA\_DERIVE set to true.

The CK\_FKDF\_COUNTER\_PARAM structure defines information about a counter. It is defined as follows:

typedef struct CK\_FKDF\_COUNTER\_PARAM {

 CK\_ULONG ulCounterInitialValue;

 CK\_ULONG ulCounterWidthInBits;

 CK\_COUNTER\_ENDIAN counterEndian;

} CK\_FKDF\_COUNTER\_PARAM;

typedef CK\_FKDF\_COUNTER\_PARAM CK\_PTR CK\_FKDF\_COUNTER\_PARAM\_PTR;

typedef CK\_ULONG CK\_COUNTER\_ENDIAN;

#define CK\_FKDF\_COUNTER\_BIG\_ENDIAN 0x00000000

#define CK\_FKDF\_COUNTER\_LITTLE\_ENDIAN 0x00000001

The fields of the CK\_FKDF\_COUNTER\_PARAM structure have the following meaning:

 ulCounterInitialValue defines the initial counter value

 ulCounterWidthInBits defines the counter width in bits

 counterEndian defines how the counter should be represented

The **C\_Derive** function already accepts a template to define the derived key as well as a CK\_OBJECT\_HANDLE pointer to receive the handle of the derived key. This mechanism can derive multiple keys and uses the CK\_FKDF\_DERIVED\_KEYS structure to provide information about the additional keys that should be derived; a template to define the derive key and CK\_OBJECT\_HANDLE pointer to receive the handle of the derived key. The CK\_FKDF\_DERIVED\_KEYS structure is defined as follows:

typedef struct CK\_FKDF\_DERIVED\_KEYS {

 CK\_ATTRIBUTE\_PTR pTemplate;

 CK\_ULONG ulAttributeCount;

 CK\_OBJECT\_HANDLE\_PTR phKey;

} CK\_FKDF\_DERIVED\_KEYS;

typedef CK\_FKDF\_DERIVED\_KEYS CK\_PTR CK\_FKDF\_DERIVED\_KEYS\_PTR;

The fields of the structure have the following meaning:

 pTemplate pointer to a template that defines a key to derive

 ulAttributeCount number of attributes in the template pointed to by pTemplate

 phKey pointer to receive the handle for a derived key

**Examples of how to use this mechanism**:

Example 1:

An example of how to use this mechanism to perform the KDF defined in section 5.1 of SP800-108 to derive an AES-256 key.

#define DIM(a) (sizeof((a))/sizeof((a)[0]))

CK\_OBJECT\_HANDLE hBaseKey;

CK\_OBJECT\_HANDLE hDerivedKey;

CK\_ATTRIBUTE derivedKeyTemplate = { … };

CK\_FKDF\_COUNTER\_PARAM counterParam = {

 1,

 32,

 CK\_FKDF\_COUNTER\_BIG\_ENDIAN

};

/\*

Define the data parameter for the PRF

data = [32-bit counter | label | 0x00 | label | length]

\*/

CK\_FKDF\_DATA\_PARAM dataParams[] = {

 { CK\_FKDF\_COUNTER, & counterParam, sizeof(counterParam) },

 { CK\_FKDF\_BYTE\_ARRAY, {0xde, 0xad, 0xbe , 0xef }, 4 },

 { CK\_FKDF\_BYTE\_ARRAY, {0x00}, 1 },

 { CK\_FKDF\_BYTE\_ARRAY, {0xfe, 0xed, 0xbe , 0xef }, 4 },

 { CK\_FKDF\_BYTE\_ARRAY, {0x00, 0x00, 0x01, 0x00}, 4 }

};

CK\_FKDF\_PARAMS kdfParams = {

 CK\_FKDF\_COUNTER,

 CK\_PRF\_AES\_CMAC,

 DIM(dataParams),

 &dataParams,

 0,

 NULL

};

CK\_MECHANISM = mechanism {

 CKM\_FLEXIBLE\_KDF,

 &kdfParams,

 sizeof(kdfParams)

};

hBaseKey = GetBaseKeyHandle(.....);

rv = C\_DeriveKey(

 hSession,

 &kdfParams,

 hBaseKey,

 &derivedKeyTemplate,

 DIM(derivedKeyTemplate),

 &hDerivedKey);

Example 2:

An example using a SCP03 compliant KDF to derive a 16-byte key.

#define DIM(a) (sizeof((a))/sizeof((a)[0]))

CK\_OBJECT\_HANDLE hBaseKey;

CK\_OBJECT\_HANDLE hDerivedKey;

CK\_ATTRIBUTE derivedKeyTemplate = { … };

CK\_FKDF\_COUNTER\_PARAM counterParam = {

 CK\_FKDF\_COUNTER,

 1,

 16,

 CK\_FKDF\_COUNTER\_BIG\_ENDIAN

};

/\*

Define the data parameter for the PRF

data = [label | 0x00 | 16-bit length | 16-bit counter | context]

\*/

CK\_FKDF\_DATA\_PARAM dataParams[] = {

 { CK\_FDPT\_BYTE\_ARRAY, {0xde, 0xad, 0xbe , 0xef }, 4 },

 { CK\_FDPT\_BYTE\_ARRAY, {0x00}, 1 },

 { CK\_FDPT\_BYTE\_ARRAY, {0x80}, 1 },

 { CK\_FDPT\_COUNTER, &counterParam, sizeof(counterParam) },

 { CK\_FDPT\_BYTE\_ARRAY, {0xfe, 0xed, 0xbe , 0xef }, 4 },

};

CK\_FKDF\_PARAMS kdfParams = {

 CK\_FKDF\_COUNTER,

 CK\_PRF\_AES\_CMAC,

 DIM(dataParams),

 &dataParams,

 0,

 NULL

};

CK\_MECHANISM = mechanism {

 CKM\_FLEXIBLE\_KDF,

 &kdfParams,

 sizeof(kdfParams)

};

hBaseKey = GetBaseKeyHandle(.....);

rv = C\_DeriveKey(

 hSession,

 &kdfParams,

 hBaseKey,

 &derivedKeyTemplate,

 DIM(derivedKeyTemplate),

 &hDerivedKey);

Example 3:

As an example, a fixed data array element containing a key handle might appear as follows:

#define DIM(a) (sizeof((a))/sizeof((a)[0]))

CK\_OBJECT\_HANDLE hBaseKey1;

CK\_OBJECT\_HANDLE hBaseKey2;

CK\_OBJECT\_HANDLE hDerivedKey1;

CK\_OBJECT\_HANDLE hDerivedKey2;

CK\_ATTRIBUTE derivedKeyTemplate1 = { … };

CK\_ATTRIBUTE derivedKeyTemplate2 = { … };

CK\_FKDF\_COUNTER\_PARAM counterParam1 = {

 CK\_FKDF\_COUNTER,

 0,

 32,

 CK\_FKDF\_COUNTER\_BIG\_ENDIAN

};

CK\_FKDF\_COUNTER\_PARAM counterParam2 = {

 CK\_FKDF\_COUNTER,

 128,

 32,

 CK\_FKDF\_COUNTER\_BIG\_ENDIAN

};

/\*

Define the data parameter for the PRF

data = [counter1 | context| counter2 | base key 2]

\*/

CK\_FKDF\_DATA\_PARAM dataParams[] = {

 { CK\_FKDF\_COUNTER, & counterParam1, sizeof(counterParam1) },

 { CK\_FDPT\_BYTE\_ARRAY, {0xde, 0xad, 0xbe , 0xef }, 4 },

 { CK\_FKDF\_COUNTER, & counterParam2, sizeof(counterParam2) },

 { CK\_FKDF\_KEY\_HANDLE, &bBaseKey2, sizeof(CK\_OBJECT\_HANDLE) },

};

CK\_FKDF\_DERIVED\_KEYS derivedKeys = {

 { derivedKeyTemplate2 ,

 DIM(derivedKeyTemplate2),

 &hDerivedKey2 }

};

CK\_FKDF\_PARAMS kdfParams = {

 CK\_FKDF\_COUNTER,

 CK\_PRF\_AES\_CMAC,

 DIM(dataParams),

 &dataParams,

 DIM(derivedKeys),

 &derivedKeys

};

CK\_MECHANISM = mechanism {

 CKM\_FLEXIBLE\_KDF,

 &kdfParams,

 sizeof(kdfParams)

};

hBaseKey1 = GetBaseKeyHandle1(.....);

hBaseKey2 = GetBaseKeyHandle2(.....);

rv = C\_DeriveKey(

 hSession,

 &kdfParams,

 hBaseKey1,

 &derivedKeyTemplate1,

 DIM(derivedKeyTemplate1),

 &hDerivedKey1);