Intel® SGX Token for SoftHSM
System Architecture and API Specification

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Without limiting the foregoing, Intel does not represent or warrant that the technology or Intel® product described herein is immune to hacking, code-breaking, piracy or other circumvention efforts.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel® sales office or your distributor to obtain the latest specifications and before placing your product order.

Terminology

AES	Advanced Encryption Standard
Authentication	The act of verifying that digital data (binary code or data) has been created by an authorized source. Authorized sources can include silicon manufacturer, device manufacturer, service provider (SP), and others based on the trust model.
Chipset Fuse Key	Unique chipset fuse key burned into the chipset.
Crypto HW	Specialized security functions implemented in HW, such as SHA-1, RSA, and a true random number generator (TRNG).
CSME	Converged Security Management Engine
DH	DiffieHellman
EPID	Enhanced Privacy ID
Independent Software Vendor (ISV[1])	The vendor who makes and sells software products that run on one or more computer hardware or operating system platforms.
Intel® Management Engine service	Intel® firmware module that executes in the Intel® Management Engine (Intel® ME) or Secure Processor[2].
MEI (HECI)	Management Engine Interface. BIOS and OS each include a MEI (HECI) driver for communication with the secure processor. MEI (HECI) hardware is a set of registers and command queues for enabling secure processor to host communication as a device driver.
OEM	Original Equipment Manufacturer
PKCS	Public Key Cryptographic Standards
Root of Trust	A technical measure embedded in a device that is the sole basis for the authentication or integrity checking or security of binary code and data operating on that device. Each of the basic properties (authentication, integrity, and secrecy) must have a clearly defined root of trust.
RSA	An algorithm for public-key cryptography
Service Provider	An entity that provides services to computers and computer users.
SPI Flash	The motherboard SPI flash part. The flash part has partitions for the Secure Processor, the BIOS, and the integrated gigabit Ethernet.
PCH	Platform Controller Hub
SGX	Software Guard Extensions
ECC	Elliptic Curve Cryptography
SDK	Software Development Kit
SP	Service Provider
TPM	Trusted Platform Module

[1] In this document the terms SP and ISV are used interchangeably

TRS	Tamper Resistant Software

[2] The term secure processor will be used to refer to the ME in this document.

Contents

Product Description

Intel® Software Guard Extensions (SGX) Token for SoftHSM is a library exposing hardware-based crypto functionality for the PKCS#11 compliant library, SoftHSM. The PKCS#11 API standard defines a set of functions that provide the capability for operations such as encryption, decryption, MAC algorithms, Hashing, and key generation. The SGX Token for SoftHSM provides a set of these functions through an interface defined by the SoftHSM "Token API". The provided functionality is secured through the SGX Trusted Execution Environment (TEE), which performs operations within a secured environment inside hardware, called an SGX enclave. The SGX Token for SoftHSM must be configured via the CMakeLists.txt and Make Files to statically compile into the SoftHSM library. It provide consumers (e.g. SoftHSM library) with APIs that may be combined to build a PKCS#11 compliant library for use cases such as:

- HW-based private key generation and management for credential management and platform attestation
- HW-based secure crypto operations, such as encryption/decryption, MAC, and Hash

External Documents

Documentation about SGX in general may be obtained from its respective locations in Intel's github repository at:

- https://github.com/intel/intel-sgx-ssl
- https://github.com/intel/intel-sgx-ssl/blob/master/Linux/package/docs/Intel(R)%20Software%20Guard%20Extensions%20SSL%20Library%20Linux%20Developer%20Guide.pdf

The SGX Token for SoftHSM utilizes the SGX Crypto API Toolkit for crypto operations, and is available through its github repository located at:

https://github.com/intel/crypto-api-toolkit

Information about SoftHSM may be obtained from its maintained site at https://www.opendnssec.org/softhsm/

The PKCS#11 specification may be referred to at its public website:

- http://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/os/pkcs11-base-v2.40-os.html
- http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/pkcs11-profiles-v2.40.html
- http://docs.oasis-open.org/pkcs11/pkcs11-profiles/v2.40/os/pkcs11-profiles-v2.40-os.pdf

Library Architecture

The architecture of SoftHSM is such that a token module ("token") may be compiled into the SoftHSM library and output a library (libsofthsm2.so) with a public PKCS#11 interface with the integrated token providing the crypto operations for this PKCS#11 implementation. SoftHSM provides a management layer that coordinates Slots, Tokens, Object Handles, and Sessions, which along with the crypto operations provided by the token, facilitate compliance to the PKCS#11 standard. The SoftHSM library contains a "Token API" interfaces that defines the contract for a token to interface with for supplying the cryptographic operations consistent with PKCS#11.

A token library implements the token API and associated crypto operations using its desired method, either in software, e.g. backed by a corresponding implementation like Botan or OpenSSL as in the default software configuration available through the soft_token, or in hardware with a comparable crypto library. In the SGX Token for SoftHSM, the *token* library doesn't perform any crypto operations itself but offloads them to SGX via the SGX Crypto API Toolkit, who, per the standards SGX model, has untrusted and trusted counterparts, p11Provider and p11Enclave, respectively. The token is mostly a pass-through interface, to the SGX Crypto API Toolkit but also manages the structures required by SoftHSM, namely the object and session mapping between the two components.

Error! Reference source not found., shows the high-level architecture of the SoftHSM library and a generic token (TokenModule) that is statically linked and provides the crypto operations required by the SoftHSM token. Generally, the token may be substituted with another Token library that fulfills the API requirements of the token interface and provides sufficient crypto operations to satisfy the PKCS#11 APIs.

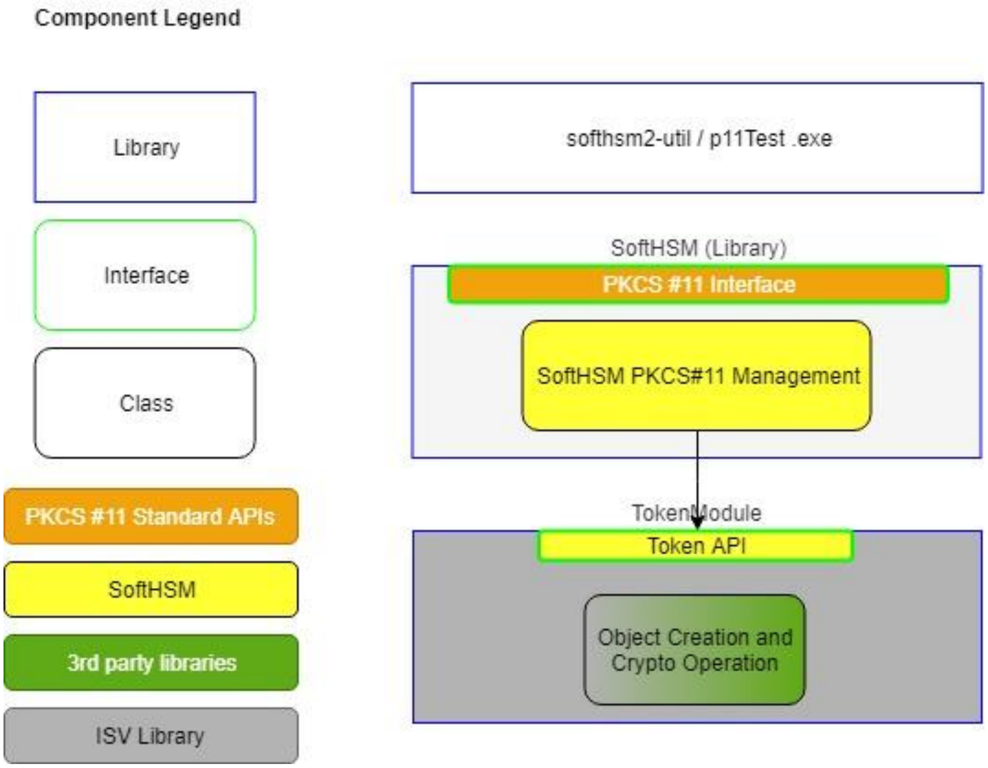

Figure 1: SoftHSM with pluggable Token diagram

Table 1 SoftHSM and Token Library Architecture Description, describes the high-level components and their relationships.

Table 1 SoftHSM and Token Library Architecture Description

Component	**Function**
SoftHSM2-util / p11Test.exe	A calling application or library such as the SoftHSM2-util or p11Test utility or test application, respectively, provide an executable interface for calling the SoftHSM shared library with loadable Token.
SoftHSM	Library with PKCS#11 public interface that manages Sessions, Slots, Handles, and Tokens. Contains a default SW-based Token (soft_token) that optionally uses either Botan, OpenSSL, or another crypto library.
TokenModule	Library compliant with the SoftHSM Token API. Modular library that may be swapped with another implementation based in SW or HW.

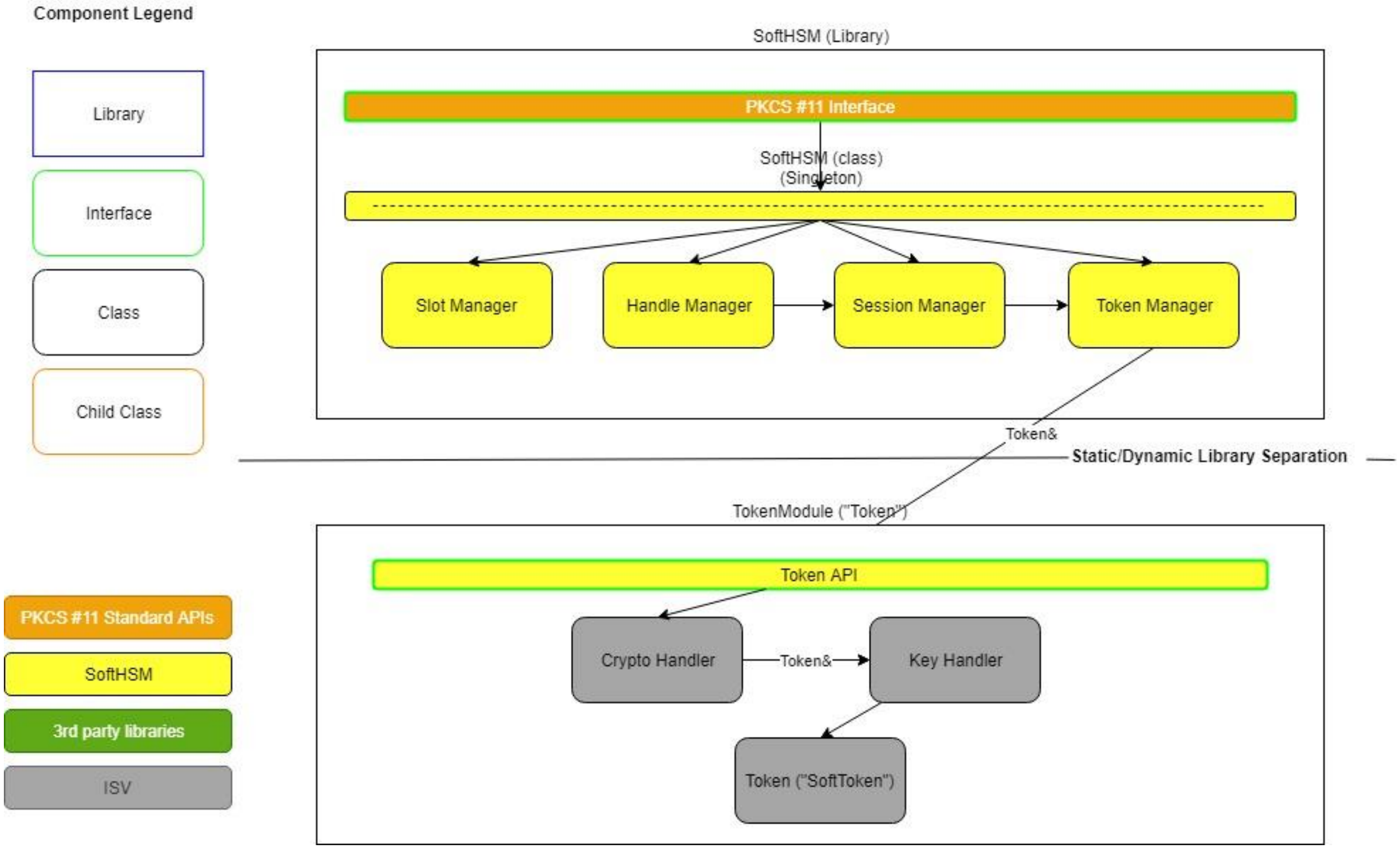

Figure 2: SoftHSM Management Library and Token Module

A more detailed view of the architectures of SoftHSM and a generic token library is shown in Figure 2: SoftHSM Management Library and Token Module. The SoftHSM library is comprised of a singleton parent object that coordinates work within and between four major classes. The default SW-based token library (soft_token) consists of three primary components that manage operation flow and crypto operations, namely the Crypto Handler, Key Handler, and [Soft]Token classes. An ISV is free to develop their own token library that has sufficient crypto operations for the token interface and define their own internal architecture as long as the Token API contract is fulfilled. An ISV could replace the crypto library (Key Handler), or the entire token library, to support a different implementation.

Table 2 SoftHSM and Token Class Description

Component	**Function**
PKCS#11 Interface	PKCS#11-compliant API Interface
Slot Manager	Class that manages PKCS#11 Slots, which may contain a Token object
Handle Manager	Class that manages object Handles, such as session objects, or objects created by Tokens, such as keys
Session Manager	Class that manages PKCS#11 Sessions, which are associated with a specific Token object
Token Manager	Class that manages Token (TokenModule) objects
Token API	SoftHSM API specification for a Token (TokenModule) library. This is common across all Tokens wanting to utilize SoftHSM's management layer
Crypto Handler	Static Class that disseminates call arguments and directs according calls through to Key Handler
Key Handler	Static Class backed by a software or hardware crypto implementation where crypto operations are performed
[Soft]Token	Class that consolidates secure object storage and crypto operations for a Token library

The Intel® SGX Token for SoftHSM implementation with SoftHSM library is shown in Figure 3: SGX Token for SoftHSM Implementation.

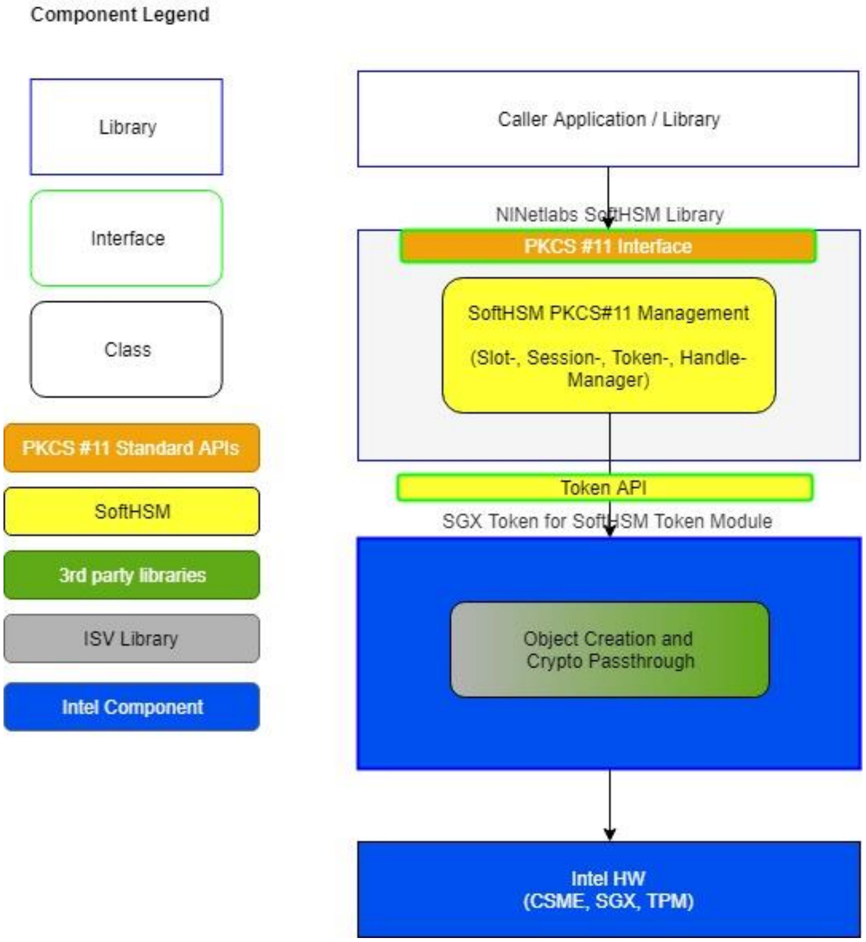
Component Legend
Library
Interface
Class
PKCS #11 Standard APIs
SoftHSM
3rd party libraries
ISV Library
Intel Component
Caller Application / Library
NINetlabs SoftHSM Library
PKCS #11 Interface
SoftHSM PKCS#11 Management
(Slot-, Session-, Token-, Handle-Manager)
Token API
SGX Token for SoftHSM Token Module
Object Creation and Crypto Passthrough
Intel HW
(CSME, SGX, TPM)

Figure 3: SGX Token for SoftHSM Implementation

The SGX Token for SoftHSM implementation uses the SGX Crypto API Toolkit as the crypto provider, and using the SGX HW TEE for managing keys and performing crypto operations, rather than being done in software, thus securing those sensitive operations and materials within the hardware trusted execution environment. No objects are able to be extracted from the enclave without first wrapping (encrypting) them with an appropriate wrapping key. All secret objects that are ever in a state to be manipulated are only usable from within protected memory of the enclave.

Table 3: Detailed SGX Token for SoftHSM with SoftHSM Description

Component	**Function**
Intel® SGX Token for SoftHSM Library	SoftHSM "Token API" compliant library that manages a token's session context and calls to SGX Toolkit (Provider and Enclave)
Intel® SGX [Crypto API Toolkit] p11 Provider	This library is the Untrusted portion of the SGX Toolkit (p11Provider) and manages the SGX Toolkit Enclave library (p11Enclave). It loads the p11Enclave and makes calls into it for crypto operations.
Intel® SGX [Crypto API Toolkit] p11 Enclave	This library is the Trusted portion of the SGX Toolkit (p11Enclave), is loaded by SGX, and calls the SGX OpenSSL crypto APIs for operations required by the SGX Token.

Component	Function
Intel® SGX OpenSSL Library	Intel® SGX-based OpenSSL library. This is an SGX supported version of OpenSSL and provides crypto operations to Intel® SGX enclaves.

A more detailed view of a generic token library is shown in Figure 4: Token library structure. This is the view of the current soft_token token. It is possible for a token developer to utilize the CryptoFactory interface which currently loads either Botan or OpenSSL for crypto operations as the secondary interface for plugging in implementation-specific crypto operations.

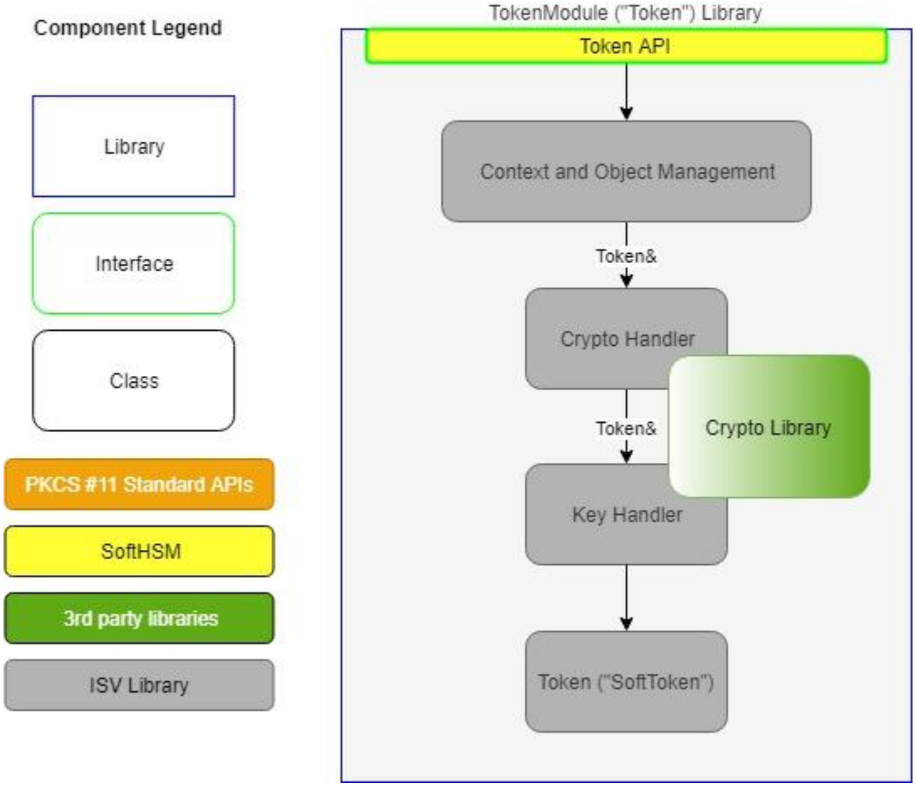

Figure 4: Token library structure

This also shows the general call structure, whereby a [Soft]Token class object reference is passed by the SoftHSM Token Manager that is associated with a Token library. All calls from SoftHSM to the token pass through the Token API and call the Crypto Handler and Key Handler where crypto operations are passed to the token's implementation crypto provider (Botan, OpenSSL, other). In Crypto Handler and Key Handler, PKCS#11 attributes are validated and a specific crypto operation is called. In the SGX Token, no crypto operations are performed in the token itself but are passed to the SGX Crypto API Toolkit instead. The Crypto Handler and Key Handler are used for initial attribute validation, as well as creation of SoftHSM-specific structures, such as an OSObject (Object Store Object). Only object mappings are managed within the token to maintain context and redirect object references between the caller (SoftHSM) and crypto delegate (SGX Crypto API Toolkit). References to secure objects are managed in both the enclave with randomly generated ids, and within the token library to be used for multi-part operations. No secret objects are ever exported from the enclave without first being encrypted.

The token's subordinate [Soft]Token class is shown in more detail in Figure 5: [Soft]Token class, showing the collection of classes and pointers to the crypto APIs (within Crypto Handler and Key Handler)

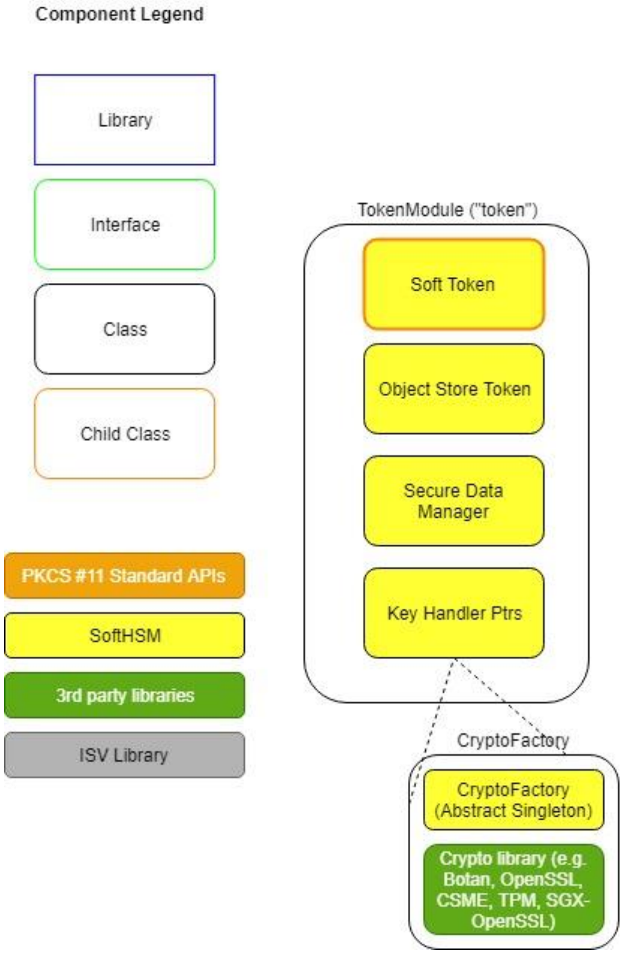

Figure 5: [Soft]Token class

The SoftToken object is used for PIN management and additionally wrapping and PKCS#11 Token or Session objects, as well as for managing login state. The PKCS#11 PIN that is provided to the SoftToken class generates a private key associated with the respective session. Thus all key generation, wrapping, unwrapping, and other crypto operations are only performed securely from within the enclave, and object references/handles are the "sensitive" material instead of the key material itself.

APIs

The Intel® SGX Token for SoftHSM is comprised of three different libraries, and as such, each provides its own set of APIs. Described below are APIs from each library.

Intel® SGX Token for SoftHSM

The Intel® SGX Token for SoftHSM implements the APIs specified by the SoftHSM "Token API" interface, required by any token seeking to plug into the SoftHSM extensibility framework. Aside from mapping code and object management for SoftHSM, all calls in the SGX Token library are pass through to the SGX Crypto API Toolkit. In order to reference objects created in the SGX enclave, object handles or IDs are returned to the Token who manages the mapping to SoftHSM, which has its own object handle management logic. These only contain indirect reference ids of objects stored within the enclave and not the objects themselves.

Crypto Mechanisms and Modes

Note that some crypto algorithms required by either PKCS#11 or FIPS have been deemed insecure by security experts and are unsupported by either the SGX Token for SoftHSM, or the underlying crypto library, SGX Crypto API Toolkit. These include, but are not limited to, SHA-1, MD5, DES, GOSTR3410, and GOSTR3411. As such, these algorithms/modes have been disabled in code within the Token library (CryptoHandler.cpp) and are denoted as disallowed. Although these mechanisms may be allowed within the Token library, they are currently unsupported by the Crypto API Toolkit (v1.3), and if desired for use will require modifications to both the Token library and the Crypto API Toolkit. It is recommended for users of either the SGX Token library or Crypto API Toolkit to ensure that secure versions of crypto algorithms are used to satisfy individual security requirements.

Token API

The following table lists all of the APIs required by the Token API interface that each token library must implement. For the SGX Token, a mapping is provided to the SGX Crypto API Toolkit on which APIs it uses for corresponding crypto calls, if applicable.

Table 4: Intel® SGX Token for SoftHSMLibrary APIs

API Name	Description
token_createtoken	Create new SoftToken and sets mechanisms
token_encryptupdate	Calls p11Provider C_EncryptUpdate
token_encryptfinal	Calls p11Provider C_EncryptFinal
token_decryptupdate	Calls p11Provider C_DecryptUpdate
token_decryptfinal	Calls p11Provider C_DecryptFinal
token_digestupdate	Calls p11Provider C_DigestUpdate
token_digestkey	Calls p11Provider C_DigestKey
token_digestfinal	Calls p11Provider C_DigestFinal
token_signupdate	Calls p11Provider C_SignUpdate
token_signfinal	Calls p11Provider C_SignFinal
token_verifyupdate	Calls p11Provider C_VerifyUpdate

API Name	Description
token_verifyfinal	Calls p11Provider C_VerifyFinal
token_load	Calls p11Provider C_Initialize and C_InitToken
token_close	No-Op
token_validate	Verifies SoftToken is valid
token_getinfo	Calls getTokenInfo on SoftToken
token_cleartoken	Calls createToken on SoftToken
token_adduser	Calls initUserPIN on SoftToken
token_changeuser	Sets PIN on either User or SO via SoftToken
token_createsession	Calls p11Provider C_OpenSession
token_endsession	Calls p11Provider C_CloseSession
token_authorize	Verifies login for User or SO via SoftToken
token_revokeauthorization	Logs out User or SO via SoftToken
token_createobject	Calls ObjectHandler createObject
token_copyobject	Calls ObjectHandler copyObject
token_destroyobject	Calls ObjectHandler copyObject, and p11Provider C_DestroyObject
token_getobjectsize	Gets object size from SoftToken and OSObject
token_getattributevalue	Calls ObjectHandler::getAttributeValue
token_setattributevalue	Calls ObjectHandler::setAttributeValue
token_searchobjectsstart	Starts search of objects in Object Store
token_searchobjectsmore	Continues search of objects in Object Store
token_searchobjectsstop	Stops search of objects in Object Store
token_encryptInit	Calls p11Provider C_EncryptInit
token_encrypt	Calls p11Provider C_Encrypt
token_decryptInit	Calls p11Provider C_DecryptInit
token_decrypt	Calls p11Provider C_Decrypt
token_digestInit	Calls p11Provider C_DigestInit
token_digest	Calls p11Provider C_Digest
token_signInit	Calls p11Provider C_SignInit
token_sign	Calls p11Provider C_Sign
token_verifyInit	Calls p11Provider C_VerifyInit
token_verify	Calls p11Provider C_Verify
token_generatekey	Calls p11Provider C_GenerateKey
token_generatekeypair	Calls p11Provider C_GenerateKeyPair
token_wrapkey	Calls p11Provider C_WrapKey
token_unwrapkey	Calls p11Provider C_UnwrapKey
token_derivekey	Calls p11Provider C_DeriveKey
token_disposeobject	No-op
token_seedrandom	Calls p11Provider C_SeedRandom
token_generaterandom	Calls p11Provider C_GenerateRandom

These APIs are described further in the NLnetLabs SoftHSM token interface header file, located at https://gitlab.nlnetlabs.nl/NLnetLabs/SoftHSM/tags/3.0.0-pre1/src/lib/pkcs11/interface.h

Error Codes

Most of the API descriptions that follow list some of the typical errors that can be returned from the function.

Error codes are defined by the PKCS#11 standard and listed within the source code located at: https://gitlab.nlnetlabs.nl/NLnetLabs/SoftHSM/tags/3.0.0-pre1/src/lib/pkcs11/pkcs11.h

Intel® SGX Crypto API Toolkit p11Provider

The P11Provider is the untrusted library counterpart to the SGX enclave library (P11Enclave) that provides APIs consistent with the PKCS#11 standard. Not all APIs are supported internally and may just return an error code if not supported, but all APIs are provided by the interface to be consistent with the PKCS#11 standard whether an implementation is supported or not.

Table 5: SGX toolkit P11Provider APIs

API Name	Description
C_Initialize	Initializes the Cryptoki library.
C_Finalize	Indicates that an application is done with the Cryptoki library.
C_GetInfo	Returns general information about Cryptoki.
C_GetFunctionList	Returns the function list.
C_GetSlotList	Obtains a list of slots in the system.
C_GetSlotInfo	Obtains information about a particular slot in the system.
C_GetTokenInfo	Obtains information about a particular token in the system.
C_GetMechanismList	Obtains a list of mechanism types supported by a token.
C_GetMechanismInfo	Obtains information about a particular mechanism possibly supported by a token.
C_InitToken	Initializes a token.
C_InitPIN	Initializes the normal user's PIN.
C_SetPIN	Modifies the PIN of the user who is logged in.
C_OpenSession	Opens a session between an application and a token.
C_CloseSession	Closes a session between an application and a token.
C_CloseAllSessions	Closes all sessions with a token.
C_GetSessionInfo	Obtains information about the session.
C_GetOperationState	Obtains the state of the cryptographic operation in a session.
C_SetOperationState	Restores the state of the cryptographic operation in a session.
C_Login	Logs a user into a token.
C_Logout	Logs a user out from a token.
C_CreateObject	Creates a new object.
C_CopyObject	Copies an object, creating a new object for the copy.
C_DestroyObject	Destroys and object.
C_GetObjectSize	Gets the size of an object in bytes.
C_GetAttributeValue	Obtains the value of one or more object attributes.
C_SetAttributeValue	Modifies the value of one or more object attributes.

API Name	Description
C_FindObjectsInit	Initializes a search for token and session objects that match a template.
C_FindObjects	Continues a search for token and session objects that match a template, obtaining additional object handles.
C_FindObjectsFinal	Finishes a search for token and session objects.
C_EncryptInit	Initializes an encryption operation.
C_Encrypt	Encrypts single-part data.
C_EncryptUpdate	Continues a multiple-part encryption.
C_EncryptFinal	Finishes a multiple-part encryption.
C_DecryptInit	Initializes a decryption operation.
C_Decrypt	Decrypts encrypted data in a single part.
C_DecryptUpdate	Continues a multiple-part decryption.
C_DecryptFinal	Finishes a multiple-part decryption.
C_DigestInit	Initializes a message-digesting operation.
C_Digest	Digests data in a single part.
C_DigestUpdate	Continues a multiple-part message-digesting operation.
C_DigestKey	Continues a multi-part message-digesting operation, by digesting the value of a secret key as part of the data already digested.
C_DigestFinal	Finishes a multiple-part message-digesting operation.
C_SignInit	Initializes a signature (private key encryption) operation, where the signature is (will be) an appendix to the data, and plaintext cannot be recovered from the signature.
C_Sign	Signs (encrypts with private key) data in a single part, where the signature is (will be) an appendix to the data, and plaintext cannot be recovered from the signature.
C_SignUpdate	Continues a multiple-part signature operation, where the signature is (will be) an appendix to the data, and plaintext cannot be recovered from the signature.
C_SignFinal	Finishes a multiple-part signature operation, returning the signature.
C_SignRecoverInit	Initializes a signature operation, where the data can be recovered from the signature.
C_SignRecover	Signs data in a single operation, where the data can be recovered from the signature.
C_VerifyInit	Initializes a verification operation, where the signature is an appendix to the data, and plaintext cannot be recovered from the signature (e.g. DSA).
C_Verify	Verifies a signature in a single-part operation, where the signature is an appendix to the data, and plaintext cannot be recovered from the signature.
C_VerifyUpdate	Continues a multiple-part verification operation, where the signature is an appendix to the data, and plaintext cannot be recovered from the signature.
C_VerifyFinal	Finishes a multiple-part verification operation, checking the signature.
C_VerifyRecoverInit	Initializes a signature verification operation, where the data is recovered from the signature.
C_VerifyRecover	Verifies a signature in a single-part operation, where the data is recovered from the signature.
C_DigestEncryptUpdate	Continues a multiple-part digesting and encryption operation.

API Name	Description
C_DecryptDigestUpdate	Continues a multiple-part decryption and digesting operation.
C_SignEncryptUpdate	Continues a multiple-part signing and encryption operation.
C_DecryptVerifyUpdate	Continues a multiple-part decryption and verify operation.
C_GenerateKey	Generates a secret key, creating a new key object.
C_GenerateKeyPair	Generates a public-key private-key pair, creating new key objects.
C_WrapKey	Wraps (i.e., encrypts) a key.
C_UnwrapKey	Unwraps (decrypts) a wrapped key, creating a new key object.
C_DeriveKey	Derives a key from a base key, creating a new key object.
C_SeedRandom	Mixes additional seed material into the token's random number generator.
C_GenerateRandom	Generates random data.
C_GetFunctionStatus	Legacy function; it obtains an updated status of a function running in parallel with an application.
C_CancelFunction	Legacy function; it cancels a function running in parallel.
C_WaitForSlotEvent	Waits for a slot event (token insertion, removal, etc.) to occur.

Additional information about these APIs are provided, including their parameters, data structures, return types and return codes, in the SGX Toolkit within the External Documents section.

Intel® SGX Crypto API Toolkit p11Enclave

Table 6: SGX toolkit P11Enclave APIs

API Name	Description
initCryptoEnclave	Initializes enclave, clears enclave state and objects
deinitCryptoEnclave	Deinitializes enclave (calls initCryptoEnclave)
digestInit	Initializes Hash function
digestUpdate	Updates Hash function
digestFinal	Finalizes Hash function
destroyHashState	Destroys Hash operation state
generateSymmetricKey	Generates Symmetric Key via OpenSSL EVP
importSymmetricKey	Imports Symmetric Key via OpenSSL EVP into enclave
symmetricEncryptInit	Initializes Symmetric Encryption via OpenSSL EVP
symmetricEncryptUpdate	Updates Symmetric Encryption via OpenSSL EVP
symmetricEncrypt	Performs Symmetric Encryption via OpenSSL EVP
symmetricEncryptFinal	Finalizes Symmetric Encryption via OpenSSL EVP
symmetricDecryptInit	Initializes Symmetric Decryption via OpenSSL EVP
symmetricDecryptUpdate	Updates Symmetric Decryption via OpenSSL EVP
symmetricDecrypt	Performs Symmetric Decryption via OpenSSL EVP
symmetricDecryptFinal	Finalizes Symmetric Decryption via OpenSSL EVP
destroySymmetricKey	Removes Symmetric key from enclave
generateAsymmetricKey	Generates Asymmetric via OpenSSL EVP
destroyAsymmetricKey	Removes Asymmetric key from enclave
asymmetricEncrypt	Performs Asymmetric encryption via OpenSSL EVP
asymmetricDecrypt	Performs Asymmetric decryption via OpenSSL EVP
asymmetricSign	Performs Asymmetric Sign via OpenSSL EVP
asymmetricVerify	Performs Asymmetric Verify via OpenSSL EVP
generateId	Generates a random ID (object handle) via sgx_read_rand
wrapSymmetricKeyWithSymmetricKey	Encrypts a Symmetric key with a Symmetric key specified by an ID
unwrapSymmetricKeyWithSymmetricKey	Decrypts a Symmetric key with a Symmetric key specified by an ID

API Name	Description
platformBindSymmetricKey	Seals (writes) a Symmetric Key encrypted via SGX to platform
unwrapAndImportPlatformBoundSymmetricKey	Unseals (decrypts) a Symmetric Key that was bound to the platform and stores in enclave
platformBindAsymmetricKey	Seals (writes) a Asymmetric Key encrypted via SGX to platform
unwrapAndImportPlatformBoundAsymmetricKey	Unseals (decrypts) an Asymmetric Key that was bound to the platform and stores in enclave
wrapWithAsymmetricKey	Encrypts an Asymmetric key with a Symmetric key specified by an ID
unwrapWithAsymmetricKey	Decrypts an Asymmetric key with a Symmetric key specified by an ID
asymmetricExportKey	Exports public portion of Asymmetric key from enclave
asymmetricImportKey	Imports public portion of Asymmetric key to enclave
createReportForKeyHandle	Returns a hash of the specified public key

For more detailed descriptions of the enclave APIs including their parameters, data structures, return types and return codes, refer to External Documents SGX Toolkit.