YubiHSM 2 Concepts

Yubico

May 11, 2022

CONTENTS

1 Introduction **1**

2 YubiHSM Auth **3**
- 2.1 Overview . . . 3
- 2.2 YubiHSM Introduction . . . 3
- 2.3 Credentials and PIN Codes . . . 3
- 2.4 YubiHSM 2 Secure Channel . . . 4
- 2.5 Architecture Overview . . . 4
- 2.6 YubiHSM Auth Flowchart . . . 5
- 2.7 Software and Tools . . . 7

3 Concept: Object and Object Types **9**
- 3.1 Objects . . . 9
- 3.2 Object Type . . . 9
- 3.3 Protocol Details . . . 10

4 Concept: Algorithms **11**
- 4.1 Algorithms . . . 11

5 Concept: Attestation **13**
- 5.1 Attestation . . . 13

6 Concept: Capability **15**
- 6.1 Capability . . . 15
- 6.2 Protocol Details . . . 15

7 Concept: Domain **23**
- 7.1 Domain . . . 23
- 7.2 Protocol Details . . . 23

8 Concept: Effective Capabilities **25**
- 8.1 Effective Capabilities (Tying It All Together) . . . 25
- 8.2 Workflow . . . 26

9 Concept: Errors **27**
- 9.1 Errors . . . 27

10 Concept: Label **29**
- 10.1 Label . . . 29
- 10.2 Protocol Details . . . 29

11 Concept: Logs **31**
11.1 Logs . . . 31

12 Concept: Object ID **33**
12.1 Object ID . . . 33
12.2 Protocol Details . . . 33

13 Concept: Options **35**
13.1 Options . . . 35
13.2 Force Audit . . . 35
13.3 Command Audit . . . 35

14 Concept: Sequence **37**
14.1 Sequence . . . 37
14.2 Protocol Details . . . 37

15 Concept: Session **39**
15.1 Session . . . 39

16 Copyright **41**

CHAPTER

ONE

INTRODUCTION

This document contains and explains the main concepts required to understand and use the YubiHSM 2 correctly.

CHAPTER

TWO

YUBIHSM AUTH

2.1 Overview

YubiHSM Auth is a new YubiKey module that serves as a key storage for authenticating against a YubiHSM 2 with a YubiKey instead of just using a session password alone. To leverage this functionality, use the latest release of YubiHSM 2 SDK.

YubiHSM Auth is a YubiKey CCID application that stores the long-lived credentials used to establish secure sessions to a YubiHSM 2. The secure session protocol is based on Secure Channel Protocol 3 (SCP03). YubiHSM Auth is supported by YubiKey v5.4.0 and higher.

2.2 YubiHSM Introduction

YubiHSM Auth uses hardware to protect the long-lived credentials for accessing a YubiHSM 2. This increases the security of the authentication credentials, as compared to the authentication solution for the YubiHSM 2 based on software credentials derived from the Password-Based Key Derivation Function 2 (PBKDF2) algorithm with a password as input.

2.3 Credentials and PIN Codes

Each YubiHSM Auth credential is comprised of two AES-128 keys which are used to derive the three session-specific AES-128 keys. The YubiHSM Auth application can store up to 32 YubiHSM Auth credentials in the YubiKey.

Each YubiHSM Auth credential is protected by a 16-byte user access code provided to the YubiKey for each YubiHSM Auth operation. The access code is used to access the YubiHSM Auth Credential to derive the session-specific AES-128 keys.

Storing or deleting YubiHSM Auth credentials requires a separate 16-byte admin access code.

Each access code has a limit of eight retries and optionally, verification of user presence (touch).

2.4 YubiHSM 2 Secure Channel

Use the YubiKey YubiHSM Auth application to establish an encrypted and authenticated session to a YubiHSM 2. Although the YubiHSM 2 secure channel is based on the protocol Global Platform Secure Channel Protocol '03' (SCP03), there are two important differences:

- The YubiHSM 2 secure channel protocol does not use APDUs, so the commands and possible options are not those of the complete SCP03 specification.
- SCP03 uses key sets with three long-lived AES keys, while the YubiHSM 2 secure channel uses key sets with two long-lived AES keys.

The YubiHSM 2 authentication protocol uses a set of static credentials called a long-lived key set. This consists of two AES-128 keys:

- ENC: Used for deriving keys for command and response encryption, as specified in SCP03.
- MAC: Used for deriving keys for command and response authentication, as specified in SCP03.

The identical long-lived keyset is protected in the YubiHSM 2 and in the YubiKey YubiHSM Auth application.

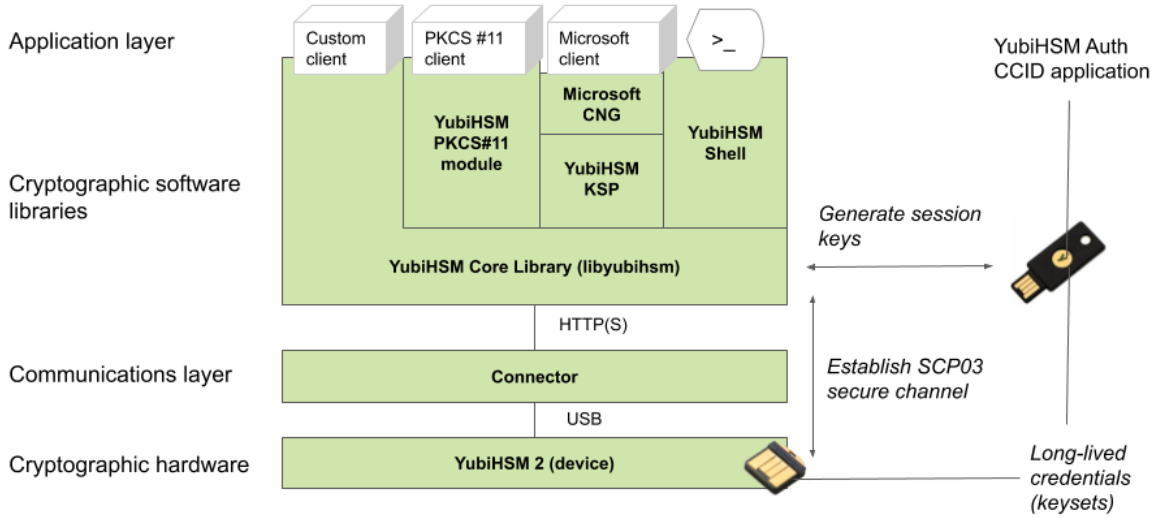
Those long-lived key sets are used by the YubiHSM Auth application to derive a set of three session-specific AES-128 keys using the challenge-response protocol as defined in SCP03:

- Session Secure Channel Encryption Key (S-ENC): Used for data confidentiality.
- Secure Channel Message Authentication Code Key for Command (S-MAC): Used for data and protocol integrity.
- Secure Channel Message Authentication Code Key for Response (S-RMAC): Used for data and protocol integrity.

The YubiHSM Auth session-specific keys are output from the YubiKey to the calling library, which uses the session keys to encrypt and authenticate commands and responses during a single session. The session keys are discarded afterwards.

2.5 Architecture Overview

The figure below shows how the YubiHSM Auth application fits in to the YubiHSM 2 architecture.



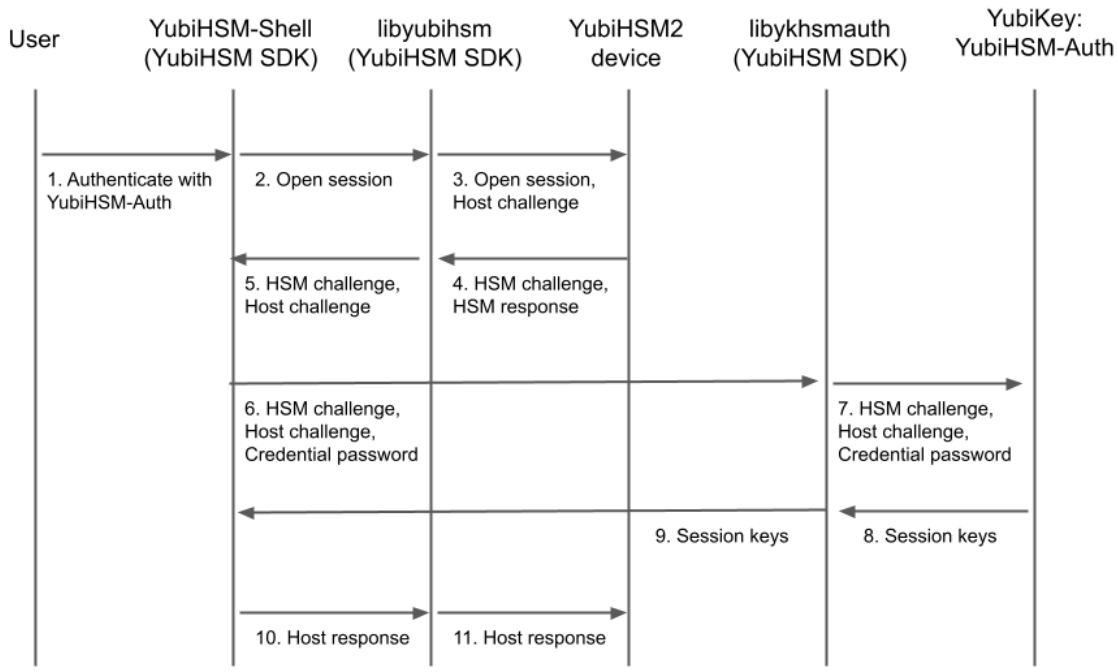
The identical long-lived credentials (key sets) are protected in both the YubiKey YubiHSM Auth application and in the YubiHSM 2. The YubiHSM-Shell software tool can be used for generating the key sets in the YubiHSM 2, and the YubiHSM-Auth software tool can be used for importing the same key sets to the YubiKey YubiHSM Auth application.

At the client, the YubiHSM authentication protocol is implemented in the `libykhsmauth` library, which derives the three session AES-keys by calling the YubiKey YubiHSM Auth CCID application. The session objects that are created can be used by the `libyubihsm` in the communication with YubiHSM.

The YubiHSM session keys are therefore generated on the basis of the long-lived credentials that are protected in the YubiHSM 2 and YubiKey YubiHSM Auth in conjunction with the SCP03 derivation scheme.

2.6 YubiHSM Auth Flowchart

The flowchart below illustrates the authentication protocol communication with YubiHSM using the static keys on YubiHSM Auth. It is assumed that the YubiHSM and YubiHSM Auth application share the same static keyset. The steps are explained below.



1. The user launches YubiHSM-Shell and enters the commands `connect` and `session open`, with the flag `ykopen` that indicates that the YubiKey with YubiHSM Auth shall be used.
2. The YubiHSM-Shell invokes the `libyubihsm` library, with a request to open a session to the YubiHSM 2.
3. The `libyubihsm` library generates a host challenge, and opens a session to the YubiHSM 2 device.
4. The YubiHSM 2 device generates an HSM challenge, and generates the session keys based on the HSM challenge, the host challenge, and the static key set in the YubiHSM 2 device. The YubiHSM 2 returns the HSM challenge in an HSM response to the `libyubihsm` library.
5. The `libyubihsm` library propagates the host challenge and HSM challenge to the YubiHSM Shell.
6. The user enters the Credential password for unlocking the static keyset in the YubiHSM Auth application in the YubiKey. The YubiHSM Shell invokes the `libykhsmauth` library, with a request to generate session keys.
7. The `libykhsmauth` library invokes the YubiHSM Auth application in the YubiKey with the Credential password, the HSM challenge and host challenge are used as input parameters.
8. The Credential password unlocks the static keyset in the YubiHSM Auth application, and the YubiHSM Auth application generates the session keys based on the static keys, HSM challenge, and host challenge.
9. The `libykhsmauth` library returns the session keys to YubiHSM Shell.
10. The YubiHSM Shell acknowledges the protocol handshake to `libyubihsm`.
11. The `libyubihsm` sends the host response to the YubiHSM 2 device. The session keys can now be used for secure channel communication between `YubiHSM-Shell/libyubihsm` in the host and the YubiHSM device.

2.7 Software and Tools

2.7.1 YubiHSM-Auth Software Tool

The YubiHSM-Auth software tool is part of the YubiHSM Shell, which is installed with the YubiHSM SDK. YubiHSM-Auth tool can be used for:

- Storing the YubiHSM Auth credentials on a YubiKey
- Deleting the YubiHSM Auth credentials on a YubiKey
- Listing the YubiHSM Auth credentials on a YubiKey
- Changing the YubiHSM Auth management key on a YubiKey
- Checking the number of retries of the YubiHSM Auth credential password
- Checking the version of the YubiHSM Auth application
- Calculating session keys, mainly for debugging and test purposes
- Resetting the YubiHSM Auth application on a YubiKey

First, the YubiHSM 2 device needs to be configured with an authentication key. The default authentication key password on KeyID=1 is set to "password", and this should be changed or replaced with other authentication keys. For the examples in this section, however, it is assumed that the default authentication key is still present on the YubiHSM 2.

In order to generate and store the equivalent YubiHSM Auth credentials on the YubiKey, the `yubihsm-auth` command line tool can be used. To invoke YubiHSM-Auth simply run `yubihsm-auth` with the required commands and parameters.

To get a list of available commands, parameters and their syntax, run: `yubihsm-auth --help`.

An example of how to use `yubihsm-auth` for storing YubiHSM Auth credentials on a YubiKey is shown below:

```
$ yubihsm-auth -a put --label="default key" --derivation-password="password" --credpwd=
↪"MyPassword" --touch=on --mgmkey="00000000000000000000000000000000" --verbose=5
Credential successfully stored
```

Where:

- `-a put` is the action to insert a YubiHSM Auth credential on the YubiKey
- `--label` is the label of the YubiHSM Auth credential on the YubiKey
- `--derivation-password` is used as input to the PBKDF2 algorithm, which is used for generating the two AES-128 keys that constitute the YubiHSM Auth credentials to be stored on the YubiKey
- `--credpwd` is the password protecting the YubiHSM Auth credentials on the YubiKey
- `--touch` is set to 'on', which requires the user to touch the YubiKey when accessing the YubiHSM Auth credential
- `--mgmkey` is the management key that is needed for writing the YubiHSM Auth credentials on the YubiKey
- `--verbose` is used to print more information as output

Note: We recommend using an offline air-gapped computer when storing the YubiHSM Auth credentials on the YubiKey.

Now the YubiKey YubiHSM Auth application can be used with YubiHSM Shell for authentication to the YubiHSM 2.

2.7.2 Using YubiHSM-Auth with YubiHSM Shell

It is now possible to authenticate to the YubiHSM 2 device with static credentials that are protected in the YubiKey application called YubiHSM Auth. For more information on this YubiKey feature and how to configure it, see Using YubiHSM Auth.

The YubiHSM Shell tool supports authentication with YubiHSM Auth credentials in both interactive mode and command line mode.

In order to use yubihsm-shell with the YubiHSM Auth-enabled YubiKey in interactive mode, open a session by executing the following yubihsm-shell command: `yubihsm> session ykopen <authkey> <label> <password>` where, in the context of using YubiHSM-Shell with the YubiHSM Auth application, the following parameters are used:

- `authkey` is the identifier of the authentication key in the YubiHSM 2
- `label` is the label of the YubiHSM-Auth credentials stored in the YubiKey
- `password` is the password that protects the YubiHSM-Auth credentials stored in the YubiKey.

Below is an example of an interactive command with YubiHSM Shell:

```
yubihsm> session ykopen 1 "default key" "MyPassword"
trying to connect to reader 'Yubico YubiKey OTP+FIDO+CCID 0'
Created session 0
```

To use yubihsm-shell with YubiHSM Auth in command line mode, add the parameter `--ykhsmauth-label` that implicitly invokes the YubiHSM Auth application at the YubiKey. Below is an example of how to use YubiHSM Shell in command line mode:

```
$ yubihsm-shell --ykhsmauth-label "default key" -p "MyPassword" -a generate-asymmetric -
↪A rsa2048 -i 11 -c sign-pss -l Signature_Key
```

If the YubiKey is configured to require touch when accessing the YubiHSM-Auth credentials, the user needs to touch the YubiKey sensor in addition to entering the credential password.

Once the user is authenticated with YubiHSM Auth, all YubiHSM-Shell commands can be used.

CHAPTER

THREE

CONCEPT: OBJECT AND OBJECT TYPES

3.1 Objects

The first concept that we will present is the Object. Any persistently stored and self-contained piece of information present in a YubiHSM 2 is an Object. This is intentionally a very generic and broad definition which can be easily rephrased as *everything is an Object*. Objects have associated properties that characterize them and give them different meanings. Regardless of the kind and the specific properties, any YubiHSM 2 device can store up to 256 Objects. Their combined size cannot exceed 126 KB.

3.2 Object Type

To identify what an Object can and cannot do, we define an attribute called Object Type, or simply Type. A Type is not enough to *uniquely* identify an Object, but it defines the set of operations that can be performed with or on it. The following types are defined:

3.2.1 Authentication Key

An Authentication Key is one of the most fundamental Objects there are. Authentication Keys can be used to establish a session with a device. An Authentication Key is basically two long-lived AES keys: an encryption key and a MAC key. When establishing a Session, the long-lived keys are used to generate three session keys:

- An encryption key used to encrypt the messages exchanged with the device
- A MAC key used to create an authentication tag for each message sent to the device
- A response MAC key used to create an authentication tag for each response message sent by the device

The session keys are temporary and are destroyed when the Session is no longer in use.

3.2.2 Asymmetric Key

An Asymmetric Key Object is what the YubiHSM 2 uses to represent an asymmetric key-pair where only the private key can be used to perform cryptographic operations.

3.2.3 HMAC Key

An HMAC Key is a secret key used when computing and verifying HMAC signatures.

3.2.4 Opaque

An Opaque Object is an unchecked kind of Object, normally used to store raw data in the device. No specific restrictions (besides size limitations) are imposed to this type of Object.

3.2.5 OTP AEAD Key

An OTP AEAD Key Object is a secret key used to decrypt Yubico OTP values for further verification by a validation process.

3.2.6 Template

A Template Object is a binary template used for example to validate SSH certificate requests.

3.2.7 Wrap Key

A Wrap Key Object is a secret key used to wrap and unwrap Objects during the export and import process.

3.3 Protocol Details

Object Types are encoded as an 8-bit value.

CHAPTER

FOUR

CONCEPT: ALGORITHMS

4.1 Algorithms

Name	Value	yubihsm-shell name	Comment
RSA PKCS1 SHA1	1	rsa-pkcs1-sha1	
RSA PKCS1 SHA256	2	rsa-pkcs1-sha256	
RSA PKCS1 SHA384	3	rsa-pkcs1-sha384	
RSA PKCS1 SHA512	4	rsa-pkcs1-sha512	
RSA PSS SHA1	5	rsa-pss-sha1	
RSA PSS SHA256	6	rsa-pss-sha256	
RSA PSS SHA384	7	rsa-pss-sha384	
RSA PSS SHA512	8	rsa-pss-sha512	
RSA 2048	9	rsa2048	
RSA 3072	10	rsa3072	
RSA 4096	11	rsa4096	
EC P256	12	ecp256	secp256r1
EC P384	13	ecp384	secp384r1
EC P521	14	ecp521	secp521r1
EC K256	15	eck256	secp256k1
EC BP256	16	ecbp256	brainpool256r1
EC BP384	17	ecbp384	brainpool384r1
EC BP512	18	ecbp512	brainpool512r1
HMAC SHA1	19	hmac-sha1	
HMAC SHA256	20	hmac-sha256	
HMAC SHA384	21	hmac-sha384	
HMAC SHA512	22	hmac-sha512	
ECDSA SHA1	23	ecdsa-sha1	
EC ECDH	24	ecdh	
RSA OAEP SHA1	25	rsa-oaep-sha1	
RSA OAEP SHA256	26	rsa-oaep-sha256	
RSA OAEP SHA384	27	rsa-oaep-sha384	
RSA OAEP SHA512	28	rsa-oaep-sha512	
AES128 CCM WRAP	29	aes128-ccm-wrap	
Opaque Data	30	opaque-data	
Opaque X509 Certificate	31	opaque-x509-certificate	

continues on next page

Table 1 – continued from previous page

Name	Value	yubihsm-shell name	Comment
MGF1 SHA1	32	mgf1-sha1	
MGF1 SHA256	33	mgf1-sha256	
MGF1 SHA384	34	mgf1-sha384	
MGF1 SHA512	35	mgf1-sha512	
SSH Template	36	template-ssh	
Yubico OTP AES128	37	aes128-yubico-otp	
Yubico AES Authentication	38	aes128-yubico-authentication	
Yubico OTP AES192	39	aes192-yubico-otp	
Yubico OTP AES256	40	aes256-yubico-otp	
AES192 CCM WRAP	41	aes192-ccm-wrap	
AES256 CCM WRAP	42	aes256-ccm-wrap	
ECDSA SHA256	43	ecdsa-sha256	
ECDSA SHA384	44	ecdsa-sha384	
ECDSA SHA512	45	ecdsa-sha512	
ED25519	46	ed25519	
EC P224	47	ecp224	secp224r1

CHAPTER

FIVE

CONCEPT: ATTESTATION

5.1 Attestation

Asymmetric keys in the YubiHSM can be attested by another Asymmetric key. The attestation process creates a new x509 certificate for the attested key.

The device comes pre-loaded with an attestation key and certificate referenced by ID `0`. It is possible to use your own key and certificate for attestation, these then have to have the same ID and the key has to have the `sign-attestation-certificate` Capability set.

5.1.1 Details

- Serial will be a random 16 byte integer
- Issuer will be the subject of the attesting certificate
- Dates will be copied from the attesting certificate
- Subject will be the string `YubiHSM Attestation id 0x` with the attested ID appended
- If the attesting key is RSA the signature will be SHA256-PKCS#1v1.5
- If the attesting key is EC the signature will be ECDSA-SHA256

5.1.2 Certificate Extensions

Some certificate extensions are added in the generated certificate and the pre-loaded certificate:

OID	Description	Data Type
1.3.6.1.4.1.41482.4.1	Firmware version	Octet String
1.3.6.1.4.1.41482.4.2	Serial number	Integer
1.3.6.1.4.1.41482.4.3	Origin	Bit String
1.3.6.1.4.1.41482.4.4	*Concept: Domain*	Bit String
1.3.6.1.4.1.41482.4.5	*Concept: Capability*	Bit String
1.3.6.1.4.1.41482.4.6	*Concept: Object ID*	Integer
1.3.6.1.4.1.41482.4.9	*Concept: Label*	Utf8String

5.1.3 Pre-Loaded Certificates

The pre-loaded certificate can be fetched as an opaque object with ID 0. This will in turn be signed by an intermediate CA which is signed by a Yubico root CA.

5.1.4 Intermediates:

E45DA5F361B091B30D8F2C6FA040DB6FEF57918E.pem

CHAPTER

SIX

CONCEPT: CAPABILITY

6.1 Capability

A Capability is an attribute that can be given to an *Concept: Object and Object Types* allowing specific operations to be performed on or with it. Commands like digital signature generation and data decryption require (and check) for a predetermined set of Capabilities to be present on an Object. Further below is the list of existing Capabilities.

It is important to know that there are no restrictions on which Capabilities can be set on an Object. Specifically, this means that it is possible to assign meaningless Capabilities to Objects that will never be able to use them, for example it is possible to have an Asymmetric Object with the Capability `verify-hmac`. Such a Capability only makes sense for HMAC Key objects, but the device will allow defining a superset. Lack of Capabilities required for a specific operation will cause a command requiring that Capability to fail.

6.1.1 Delegated Capabilities

Every Object stored on the device has an associated set of Capabilities. There is a second set of so-called Delegated Capabilities that only Authentication Keys and Wrap Keys have. This is used to capture the indirection that Authentication Keys and Wrap Keys can be used as a means of storing more Objects on a device. In both cases Delegated Capabilities are used as a filter.

For Authentication Keys, Delegated Capabilities define the set of Capabilities that can be set or "bestowed" onto an Object created by the Authentication Key. Any operation attempting to create Objects with a Capability outside of this set will fail.

For Wrap Keys, Delegated Capabilities define the set of Capabilities that an Object can have when imported or exported using the Wrap Key. A larger set of Capabilities will cause the import operation to fail.

6.2 Protocol Details

A Set of Capabilities is an 8-byte value. Each Capability is identified by a specific bit, as shown in the `Hex Mask` column below.

Name	Hex Mask	Applicable Objects	Description
Asymmetric Keys			

continues on next page

Table 1 – continued from previous page

Name	Hex Mask	Applicable Objects	Description
delete-asymmetric -key	0x0000020000000000	authentication -key	Delete Delete Asymmetric Key Objects
generate-asymmetric -key	0x0000000000000010	authentication -key	Generate Asymmetric Key Objects
put-asymmetric-key	0x0000000000000008	authentication -key	Write Asymmetric Key Objects
——————————**Authentication Keys**——————————-			
delete-authen- tication-key	0x0000010000000000	authentication -key	Delete Authentication Key Objects
put-authentication -key	0x0000000000000004	authentication -key	Write Authentication Key Objects
change- authentication-key	0x0000400000000000	authentication -key	Replace Authentication Key Objects
——————————**Certificate**——————————-			
sign-attestation- certificate	0x0000000400000000	authentication -key, asymmetric-key	Attest properties of Asymmetric Key Objects
sign-ssh-certificate	0x0000000002000000	authentication -key, asymmetric-key	Sign SSH certificates
——————————**Data**——————————			

continues on next page

Table 1 – continued from previous page

Name	Hex Mask	Applicable Objects	Description
decrypt-oaep	0x0000000000000400	authentication -key, asymmetric-key	Decrypt data using RSA-OAEP
decrypt-pkcs	0x0000000000000200	authentication -key, asymmetric-key	Decrypt data using RSA-PKCS1v1.5
——————————**ECDH**——————————			
derive-ecdh	0x0000000000000800	authentication -key, asymmetric-key	Perform ECDH
——————————**Global**——————————			
get-option	0x0000000000040000	authentication -key	Read device-global options
set-option	0x0000000000020000	authentication -key	Write device-global options
——————————**HMAC**——————————			
delete-hmac-key	0x0000080000000000	authentication -key	Delete HMAC Key Objects
generate-hmac-key	0x0000000000200000	authentication -key	Generate HMAC Key Objects
put-mac-key	0x0000000000100000	authentication -key	Write HMAC Key Objects
sign-hmac	0x0000000000400000	authentication -key, hmac-key	Compute HMAC of data

continues on next page

Table 1 – continued from previous page

Name	Hex Mask	Applicable Objects	Description
verify-hmac	0x0000000000800000	authentication -key, hmac-key	Verify HMAC of data
Log			
get-log-entries	0x0000000001000000	authentication -key	Read the Log Store
Opaque			
delete-opaque	0x0000008000000000	authentication -key	Delete Opaque Objects
get-opaque	0x0000000000000001	authentication -key	Read Opaque Objects
put-opaque	0x0000000000000002	authentication -key	Write Opaque Objects
OTP			
create-otp-aead	0x0000000040000000	authentication -key, otp-aead-key	Create OTP AEAD
decrypt-otp	0x0000000020000000	authentication -key, otp-aead-key	Decrypt OTP
delete-otp-aead-key	0x0000200000000000	authentication -key	Delete OTP AEAD Key Objects
generate-otp-aead -key	0x0000001000000000	authentication -key	Generate OTP AEAD Key Objects

continues on next page

Table 1 – continued from previous page

Name	Hex Mask	Applicable Objects	Description
put-otp-aead-key certificate	0x0000000800000000	authentication -key	Write OTP AEAD Key Objects
randomize-otp-aead	0x0000000080000000	authentication -key, otp-aead-key	Create OTP AEAD from random data
rewrap-from-otp-aead-key	0x0000000100000000	authentication -key, otp-aead-key	Rewrap AEADs from one OTP AEAD Key Object to another
rewrap-to-otp-aead-key	0x0000000200000000	authentication -key, otp-aead-key	Rewrap AEADs to one OTP AEAD Key Object from another
Random			
get-pseudo-random	0x0000000000080000	authentication -key	Extract random bytes
Reset			
reset-device	0x0000000010000000	authentication -key	Perform a factory reset on the device
Signatures			
sign-ecdsa	0x0000000000000080	authentication -key, asymmetric-key	Compute digital signatures using ECDSA

continues on next page

Table 1 – continued from previous page

Name	Hex Mask	Applicable Objects	Description
sign-eddsa	0x0000000000000100	authentication -key, asymmetric-key	Compute digital signatures using EDDSA
sign-pkcs	0x0000000000000020	authentication -key, asymmetric-key	Compute signatures using RSA-PKCS1v1.5
sign-pss	0x0000000000000040	authentication -key, asymmetric-key	Compute digital signatures using using RSA-PSS
——————————**Template**——————————			
delete-template	0x0000100000000000	authentication -key	Delete Template Objects
get-template	0x0000000004000000	authentication -key	Read Template Objects
put-template	0x0000000008000000	authentication -key	Write Template Objects
——————————**Wrap**——————————			
delete-wrap-key	0x0000040000000000	authentication -key	Delete Delete Wrap Key Objects
export-wrapped	0x0000000000001000	authentication -key, wrap-key	Export other Objects under wrap

continues on next page

Table 1 – continued from previous page

Name	Hex Mask	Applicable Objects	Description
exportable-under -wrap	0x0000000000010000	all	Mark an Object as exportable under wrap
generate-wrap-key	0x0000000000008000	authentication -key	Generate Wrap Key Objects
import-wrapped	0x0000000000002000	authentication -key, wrap-key	Import wrapped Objects
put-wrap-key	0x0000000000004000	authentication -key	Write Wrap Key Objects
unwrap-data	0x0000004000000000	authentication -key, wrap-key	Unwrap user-provided data
wrap-data	0x0000002000000000	authentication -key, wrap-key	Wrap user-provided data

CHAPTER

SEVEN

CONCEPT: DOMAIN

7.1 Domain

A Domain is a logical partition that can be conceptually mapped to a container. In a YubiHSM 2 there are 16 independent Domains; an Object can belong to one or more Domains.

Note: Authentication Keys are Objects and thus can belong to multiple Domains.

Domains serve as a means to secure Objects so that they cannot be addressed by independent applications running on the same device. This is achieved by specifying the Object's Domain. Only users or applications that belong to the same Domain as an Object can access it or use it.

The details involved in accessing an Object are explained in the *Concept: Effective Capabilities* page.

7.2 Protocol Details

Domains are encoded as 16-bit values, where each Domain is represented by a bit

Domain Number	Hex Mask
1	0x0001
2	0x0002
3	0x0004
4	0x0008
5	0x0010
6	0x0020
7	0x0040
8	0x0080
9	0x0100
10	0x0200
11	0x0400
12	0x0800
13	0x1000
14	0x2000
15	0x4000
16	0x8000

CHAPTER

EIGHT

CONCEPT: EFFECTIVE CAPABILITIES

8.1 Effective Capabilities (Tying It All Together)

This document describes how Object-related concepts interact with each another.

Let us assume that we are establishing a Session with Authentication Key `0xabcd` so that the Session can use the Asymmetric Key `0x1234` to sign some data. We are assuming that Asymmetric Key `0x1234` is an RSA 2048-bit key and that we would like to generate a signature using RSASSA-PSS.

8.1.1 Create and Authenticate a Session

Creating and authenticating a Session requires knowledge of what the long-lived keys are (or what the associated derivation password is).

When a valid Session is established, certain properties of the Authentication Key used to create the Session are inherited by the Session itself. These are:

- The Domain(s) to which the Authentication Key belongs (for more information, see *Concept: Domain*),
- The Capabilities of the Authentication Key (see *Concept: Capability*) and
- The Delegated Capabilities (see *Concept: Capability*) associated with Authentication Key `0xabcd` .

The Session's inherited properties serve to ensure that the only Objects stored in the HSM 2 that we can see and access are those that belong to the same Domain(s) as Authentication Key `0xabcd`.

8.1.2 Generate a Signature

The required capability must be set on both the Authentication Key used to establish the Session (Authentication Key `0xabcd`) and the target Object used to perform the operation (Asymmetric Key `0x1234`).

Assuming that Asymmetric Key `0x1234` is in one such Domain, we can now continue and ask the HSM 2 to generate a signature. To do so we will send the `Sign Data` command over the Session. It will not execute successfully unless the arguments of the command are valid, i.e., no malformed data can be sent to the device or an error will occur.

Both Authentication Key `0xabcd` and Asymmetric Key `0x1234` must have the Capability `sign-pss` set.

8.1.3 Effective Capabilities and Role Definition

The overlap between

- The Capabilities of the Authentication Key used to establish the Session and
- The Capabilities of the target Object involved in the operation

defines the **Effective Capabilities**. An operation on a given target Object over a given Session can succeed only if the Capabilities required by the operation are included in the Effective Capabilities.

The interaction between Domains and Effective Capabilities enables flexible setup and role definition. For example,

- It is possible to assign a set of Capabilities to an Object, and then distribute those Capabilities across different Authentication Keys so that each key is enabled to perform only a single operation on the target Object, and no key performs the same operation as any other key.
- Similarly, it is possible to disable specified operations by not assigning the requisite Capabilities to an Authentication Key. For example, an "Administrator" Authentication Key could be enabled only to create keys while a "User" Authentication Key could be enabled only to use those same keys.

8.2 Workflow

1. Determine which Objects will have operations performed on them
2. Determine which Authentication Keys you will use
3. Determine which operations will be performed
4. Use a spreadsheet (if necessary) to map out the interaction between the first three items
5. With the aid of the spreadsheet, create domains to enable the interaction.

   Note: Authentication Keys are Objects and thus can belong to multiple Domains.

6. You could construct your domains:
   - per operation - put an Object and an Authentication Key into each domain, or
   - per Object - put the Authentication Key(s) for all the operations to be performed on each Object into a single domain
   - per Authentication Key - put the requisite Object(s) into each Domain.

   For example, if you wanted Jan to do the signing and Ola to do the importing, you could adopt any of the above options, but the Effective Capabilities enable you to assign far more complex webs of responsibilities.
7. Use the spreadsheet to set the Capabilities and Delegated Capabilities appropriately, "appropriateness" being determined by the Objects and operations to be performed on them.

CHAPTER

NINE

CONCEPT: ERRORS

9.1 Errors

Below are error codes returned by a YubiHSM device.

Name	Value	Description
OK	0x00	Success
INVALID COMMAND	0x01	Unknown command
INVALID DATA	0x02	Malformed data for the command
INVALID SESSION	0x03	The session has expired or does not exist
AUTHENTICATION FAILED	0x04	Wrong Authentication Key
SESSIONS FULL	0x05	No more available sessions
SESSION FAILED	0x06	Session setup failed
STORAGE FAILED	0x07	Storage full
WRONG LENGTH	0x08	Wrong data length for the command
INSUFFICIENT PERMISSIONS	0x09	Insufficient permissions for the command
LOG FULL	0x0a	The log is full and force audit is enabled
OBJECT NOT FOUND	0x0b	No object found matching given ID and Type
INVALID ID	0x0c	Invalid ID
SSH CA CONSTRAINT VIOLATION	0x0e	Constraints in SSH Template not met
INVALID OTP	0x0f	OTP decryption failed
DEMO MODE	0x10	Demo device must be power-cycled
OBJECT EXISTS	0x11	Unable to overwrite object

CHAPTER

TEN

CONCEPT: LABEL

10.1 Label

A Label is a sequence of bytes that can be used to add a mnemonic reference to Objects.

10.2 Protocol Details

Labels are `40` bytes long. As far as the YubiHSM is concerned, the label is only a string of raw bytes and are not restricted to printable characters or valid UTF-8 glyphs.

CHAPTER

ELEVEN

CONCEPT: LOGS

11.1 Logs

A YubiHSM 2 device maintains a list of recently executed commands in a portion of non-volatile memory known as the Log Store. This allows to log commands across different power cycles. Specific commands are used to extract logs from the device. Since the Log Store uses non-volatile memory, it can only store up to `62` different entries. When the Log Store is full, it is used as a circular buffer, meaning that the least recently used entry is overwritten.

It is possible to set the device in Force Audit mode. When this is done entries from the Log Store must be retrieved or commands that cannot be logged will fail. Together with individual commands, also power-on and reboot events are logged.

Establishing a session is logged like any other operation, however those commands are always allowed, independent of the current status of the Log Store. This is so that it is always possible to retrieve logs and free up the Log Store, even when the device is in Force Audit mode and the Log Store is full. However, the number of unlogged authentication and power-up events is stored in a counter that is retrieved as part of the log retrieval.

Entries in the Log Store are organized to form a chain of hashes. This allows auditors to verify that a given set of entries has not been tampered with after extraction, and that all entries are present. More details on the format of log entries can be found in the protocol description document for extracting logs entries.

CHAPTER

TWELVE

CONCEPT: OBJECT ID

12.1 Object ID

The ID property is used to identify an Object of a given Type. This means that to **uniquely** identify an Object stored on a YubiHSM 2, the couple `(Type, ID)` is required. There can be more than one Object with a given ID and more than one Object with a given Type, but only one Object with a specific ID and Type. This is so that logical connections between Objects can be established by giving a set of connected Objects of different Types the same ID.

An Object ID can have values in the range `[0-65535]` or `[0x0000-0xffff]` in hexadecimal. Note that this range is larger than the maximum number of Objects that can be stored in the device (256). Regardless of the type, ID `0x0000` and `0xffff` are reserved for internal Objects.

12.2 Protocol Details

Object IDs are encoded as 16-bit values.

CHAPTER

THIRTEEN

CONCEPT: OPTIONS

13.1 Options

Options are device-global settings. The following Options are defined:

Option Name	Hex Value
force-audit	0x01
command-audit	0x03

The data payload is Option-specific.

13.2 Force Audit

This Option is used to enable Force Audit mode which prevents the device from performing additional operations whilst the *Concept: Logs* is full.

The Option accepts three different values:

- 0x00: Option disabled
- 0x01: Option enabled
- 0x02: Option permanently enabled (only possible to turn off through factory reset)

13.3 Command Audit

This Option is used to enable or disable logging of specific commands. Logging commands has a noticeable impact on performance. By default logging is enabled for all operations.

The Option accepts three different values:

- 0x00: Option disabled
- 0x01: Option enabled
- 0x02: Option permanently enabled (only possible to turn off through factory reset)

Multiple commands can be specified at once with the syntax `C1 V1, C2 V2, ..., Cn Vn` where `Ci` is the Command Code and `Vi` is the Option Value. An example of this syntax can be found at the Set Option Command description.

CHAPTER

FOURTEEN

CONCEPT: SEQUENCE

14.1 Sequence

Sequence is a one-byte value that is part of the metadata associated with an Object. The Sequence describes how many times an Object with a given ID and Type has been written. This is mostly useful for caching to determine if new data needs to be fetched from the device.

14.2 Protocol Details

Sequence is 8 bits long and will wrap.

CHAPTER

FIFTEEN

CONCEPT: SESSION

15.1 Session

A Session is not a property of a specific Object, but rather it is used to describe a logical connection between an application and a device. Sessions are end-to-end encrypted and authenticated using Session Keys. Those keys are derived from long-lived, pre-shared Authentication Key Objects as part of the sessions authentication process. The Session creation and authentication protocol is based on Global Platform SCP03.

On a single YubiHSM 2 it is possible to establish up to `16` independent and concurrent Sessions. Note that while multiple concurrent Sessions can be active at a given time, the device still serves as a rendezvous point. This means that time-consuming operations, like generating a long RSA key, will block commands in other Sessions. Sessions are addressed with a number in the range `[0-15]`.

Sessions have an expiration period of 30 seconds of inactivity in order to prevent resource starvation. After such a period, the device will consider a Session inactive and will move it to the pool of re-usable Sessions. Whenever a command is executed on a given Session, the inactivity timer is reset, meaning that if a Session is being constantly used then it will not expire.

Some of the operations that can be performed on a YubiHSM 2 do *not* require a Session. The implications are that the command and its response will travel unencrypted to and from the device. These commands are only generic status commands, making Sessions effectively required for any meaningful operation.

The long-lived keys required to derive Sessions can be explicitly used in the relevant commands. There are however built-in functionalities to derive those keys from a password using `10,000` iterations of `PBKDF2` with the salt `Yubico`, making the process more human-friendly. **Every new or factory-reset YubiHSM 2 has a default Authentication Key with ID 1 and all Capabilities and all Domains set.** This is equivalent to a superuser or an administrator. The long-lived keys for this Object are derived using the process described before with the password password.

Warning: It is crucial to delete this well-known Authentication Key before deployment.

CHAPTER

SIXTEEN

COPYRIGHT

Trademarks

Yubico and YubiKey are registered trademarks of Yubico AB. All other trademarks are the property of their respective owners.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design, and manufacturing. Yubico shall have no liability for any error or damages of any kind resulting from the use of this document.

The Yubico Software referenced in this document is licensed to you under the terms and conditions accompanying the software or as otherwise agreed between you or the company that you are representing.

Contact Information

Yubico Inc.
530 Lytton Street
Suite 301
Palo Alto, CA 94301
USA

Click the links to:

- Submit a support request
- Send a Contact Me request
- See additional contact options for getting touch with us

Document Updated

2022-05-11 22:40:36 UTC