yubico

JULY 2020

YubiHSM 2:

A secure keystore for the manufacturing industry

Executive summary

As a result of having worked alongside several partners within the manufacturing sector responsible for large scale production of electronic components, Yubico has now designed a standard solution centred on the YubiHSM 2 to safeguard corporate secrets. Manufacturing companies are turning to Yubico to protect their supply chain and intellectual property, ranging from sensors and medical devices to automotive and other products with electronic components, as the integrity of the supply chain is becoming increasingly paramount in a globalised world.

In all instances, the YubiHSM 2 is being used at the heart of the solution, to either protect the manufacturing process by ensuring only certified programming stations can interface with the components, or to write digital signatures onto each component to ensure authenticity. In both scenarios, the added security of the YubiHSM 2 helps to maintain a company's reputation and gives them peace of mind that their products will work as expected even after they have left the manufacturing floor.

This document will examine two common approaches within the manufacturing industry today– the first involving Joint Test Action Group (JTAG) and an Electronic Control Unit (ECU), commonly found in the automotive industry, and the second involving a X.509 certificate or Card Verifiable Certificate (CVC), commonly found in the production of smaller electronic components, smart cards and devices with contactless interfaces such as IoT. It will later become apparent that the sub-processes of key management, signing and verification are often comparable across both approaches, but do diverge with regards to output. The thesis of this proposal, therefore, is an attempt to simply generalise this output, and arrive at a singular overall design that does not stipulate specific formats. It should be highlighted however, that this general approach can be extended to any manufacturing operation outside of the listed scenarios, and applies to any process where secrets and the authenticity of electronic components needs to be managed.

Figure 1 – A high level overview of the proposed solution

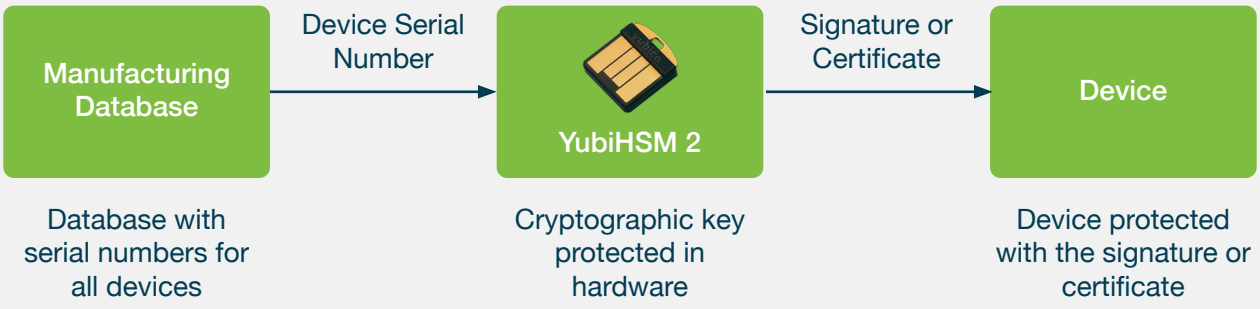


Figure 1 (above) provides a high level process summary of the proposed solution to be implemented, as either a replacement or a brand new process in manufacturing plants and/or production lines, with the YubiHSM 2 at the core. The foremost point to highlight is that the cryptographic key used to sign and/or certify components is never exposed *outside* of the YubiHSM 2 hardware, thus ensuring a high level of security.

Introduction

Background

In the manufacturing industry, it is crucial to ensure that all components involved in an end-to-end process are authentic to avoid unsolicited replication and theft, but also for quality assurance, since an assembly line should only consist of genuinely sourced products, as the whole is always a sum of its parts. As a result, there must always be a solution in place to protect the integrity and intellectual property of all components, in any context under manufacturing, from production and assembly, to repair and replacement.

It is therefore recommended that manufacturers currently employing solutions where sensitive cryptographic information is stored in software, should consider migrating to a YubiHSM 2-centred solution on the basis that Master Keys are less likely to be compromised. This is a direct result of having both the cryptographic information as well as the cryptographic operations themselves being performed directly in hardware, which innately shields the export of private information and thereby minimises the threat of Private Key leakage. To illustrate this point with an example, because confidential data residing on a YubiHSM 2 is *never* exposed, even if a remote attacker is able to compromise a network or the computer connected to it, there are still no obvious attack vectors. On the other hand, if the same attacker is able to gain full underlying access to a software equivalent, they might be able to at least run analysis on the memory or local files for potential weaknesses or patterns.

The same reasoning holds even for manufacturers looking to implement a solution from scratch, and are weighing the benefits of a software based approach versus a hardware based approach. Moreover, the hardware based YubiHSM 2 provides additional benefits by providing flexibility in the manufacturing design afforded by its dimunitive size and low price point, which allows security controls to be more localised, thus reducing the reliance on Internet conductivity and shrinking the scope of the security boundary.

Purpose of this document

The purpose of this document is to propose a generic solution directed at manufacturers and other companies within the industry, centred on the YubiHSM 2 to process and manage cryptographic information in hardware.

Other industries beside manufacturing

Even though the primary audience of this white paper is intended to be the manufacturing industry, by no means is the YubiHSM 2 itself limited to *just* manufacturing use cases. In fact, *any* industry or company looking to protect its intellectual property or brand with hardware based cryptography, stands to benefit from the proposal in this document, whether they are in the public sector, healthcare, financial services or other vertical entirely. The design itself is standard and can easily be scaled and adapted to a wide range of circumstances, so long as protecting corporate data is the objective.

Abbreviations used in this document

API	Application Programming Interface
CA	Certification Authority
CMC	Certificate Management over CMS
CMP	Certificate Management Protocol
CMS	Cryptographic Message Syntax
CNG	Cryptographic API Next Generation
CVC	Card Verifiable Certificate
DER	Distinguished Encoding Rules
EAC	Extended Access Control
ECDSA	Elliptic Curve Digital Signature Algorithm
ECU	Electronic Control Unit
HMAC	Hashed Message Authentication Code
HSM	Hardware Security Module
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
ITU-T	Telecommunications Union's Telecommunication Standardization Sector
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extensions
JTAG	Joint Test Action Group
KSP	Key Storage Provider
PBKDF2	Password-Based Key Derivation Function 2
PCB	Printed Circuit Board
PKCS #5	Public Key Cryptography Standard #5 (Password-Based Cryptography)
PKCS #10	Public Key Cryptography Standard #10 (Certification Request Syntax)
PKCS #11	Public Key Cryptography Standard #11 (Cryptographic Token Interface)
PKI	Public Key Infrastructure
RA	Registration Authority
RFID	Radio-frequency identification
RSA	Rivest Shamir Adleman
SCEP	Simple Certificate Enrollment Protocol
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SN	Serial Number
TLS	Transport Layer Security
TLV	Tag-Length-Value
TPM	Trusted Platform Module
TRNG	True Random Number Generator

Why choose the YubiHSM 2?

In an age of increased demand for telemetric services, distributed or outsourced production and remote working environments across *every* industry and sector, comes an even greater demand for the protection of *corporate* digital assets created by those opportunities. Companies which have not prepared for the shift in focus towards a data security-centric mode of operation, may struggle or overextend themselves, in order to meet the modern challenges of today's society.

While the YubiKey was designed with the objective of protecting an individual's identification and data through superior hardware, the YubiHSM 2 was designed specifically with the corporation top of mind. Yubico's legacy of excellence and reputation as an ambassador of high-grade but easy to use hardware-based security, continues with the YubiHSM 2, which provides a superior hardware-based cryptographic device to rival the traditional Hardware Security Module (HSM). As will be outlined below, the YubiHSM 2 affords a number of advantages over traditional solutions, and has been adopted by several of Yubico's partners within the manufacturing space to solve a variety of data security problems.

The unique benefits

Some of the key and unique benefits of implementing a solution centred around the YubiHSM 2, and not just any HSM, can be enumerated as follows:

- The YubiHSM 2 is the most cost *effective* HSM on the market, at a fraction of the price of its competitors
- The YubiHSM 2 comes in one of the smallest form factors available, roughly the size of a human fingernail. Additionally, it comes with a standard USB-A port making it interoperable with a wide variety of computing devices. It should be noted that the form factor may be customised to resemble any YubiKey if so desired, *depending* on order volumes
- The combination of the two aforementioned factors makes it quick and easy to deploy the YubiHSM 2 across multiple locations (e.g. factories, production lines) for less than the equivalent cost of a single HSM from other manufacturers
- All cryptographic information can be generated and stored within the YubiHSM 2, and since the keys never leave the secure element in clear text, they are therefore never exposed to the outside world
- The YubiHSM 2 authentication password used for key and device management is separate and discrete to any underlying cryptographic keys. This implies that even if the authentication password is ever compromised, the hash values derived from Private Keys in addition to the keys themselves, will remain secure
- It is possible to replicate key material on a YubiHSM 2 using export Wrap functionality, which allows for identical or backup deployments across multiple sites
- The YubiHSM 2 is flexible enough to be retrofitted into many existing environments and solutions, depending on the particular use case, because it supports open and accepted standards such as PKCS #11 and Microsoft CNG
- The YubiHSM 2 supports symmetric schemes such as AES and HMAC-SHA but *also* asymmetric schemes such as RSA and ECDSA. This makes it flexible and useful in a variety of scenarios, as will be described in the coming sections for JTAG and certificates, which use symmetric and asymmetric schemes respectively.

Use case #1: Protecting JTAG with symmetric schemes

Introduction to JTAG

JTAG is a common hardware interface that provides computers and devices with a way to communicate directly with the pins and chips soldered onto a Printed Circuit Board (PCB). It was originally developed by a consortium (namely the Joint Test Action Group) in the mid-80s to address the increasing difficulty of testing at the integrated circuit level as PCBs became more complicated. JTAG has been in widespread use ever since, and even today, JTAG has expanded its footprint to include debugging, programming and testing on virtually *all* embedded devices.

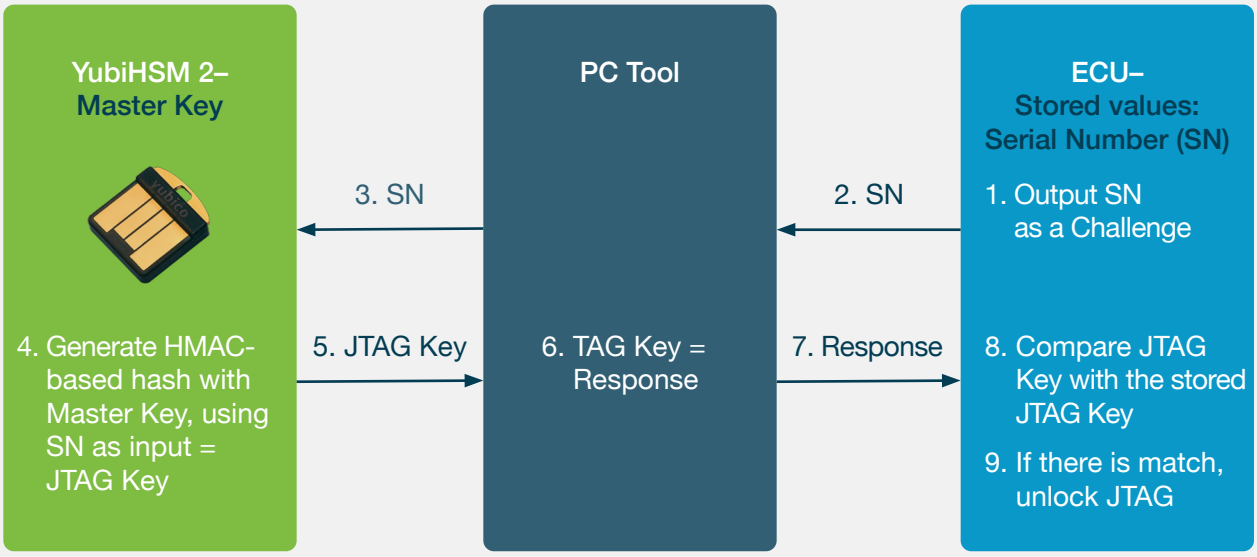
It is important to note that the use of JTAG to "debug" a PCB, in electrical engineering circles, is very different to the same term in reference to traditional software–with regards to JTAG, it actually refers to the process of making sure pin A on chip A, for example, is physically connected to pin B on chip B, and that generally, all the pins are functioning correctly. The term "boundary scan" is another common term used to describe this process, and is synonymous with JTAG.

Although JTAGs are incredibly useful to chip manufacturers during the design, testing, and even production phases, they also represent a huge potential attack vector since they provide direct access to the underlying components. Fortunately, manufacturers are aware of this risk, and will often take steps to prevent access to JTAG interfaces, such as obfuscating JTAG traces on the board, cut them entirely, or even blow fuses in the JTAG wiring as part of the manufacturing process. These methods are somewhat effective, although a determined attacker talented with a soldering iron can almost always repair the damage. A less physically invasive and arguably sounder option is to secure the interface using cryptography, such as with a Challenge-Response mechanism, but even a simple static password might suffice in many cases.

Static password and Challenge-Response

Using this approach, the manufacturer encrypts or hashes the serial number of each PCB to generate a unique *JTAG Key*, and then deploys it to the corresponding ECU as part of the production process. The JTAG Key would serve the sole purpose of protecting the associated JTAG interface, acting as a gate of sorts, before a user can interface and perform a debug. So, upon attempting to interface via JTAG, the user is presented with a 'Challenge' from the ECU based on the JTAG Key, which must then be answered with the 'Response' that *exactly* corresponds to the either the expected static output, or the output of an algorithmic operation using the 'Challenge' as the input for that iteration. *Figure 2* and *Figure 3* (below) depict the step by step flow of the static password and Challenge-Response mechanisms respectively, enumerated from steps 1 through 9.

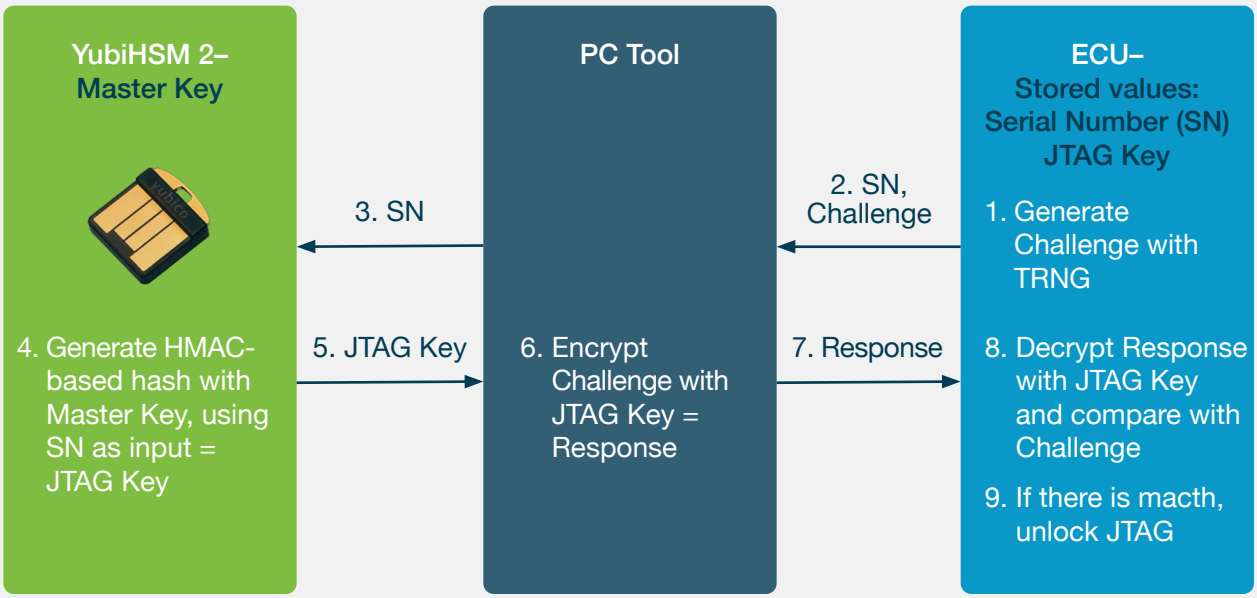
Figure 2 – Static password as a means to authorize or unlock JTAG



Challenge-Response over Static

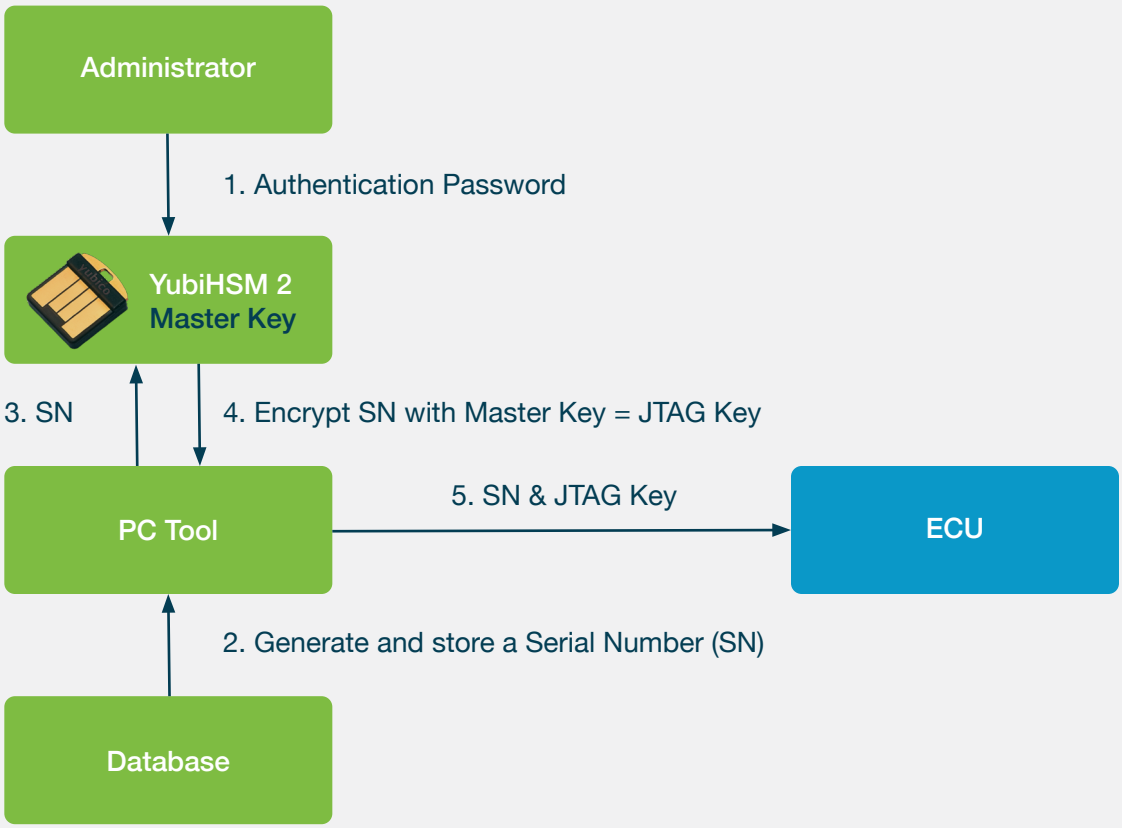
The question of *why* a manufacturer might opt for a Challenge-Response mechanism over a static password implementation can be answered by the fact that it offers greater security because both the Challenge and Response are *dynamically* generated on each new attempt, in contrast to a static password which is susceptible to a replay attack if the output is ever intercepted. The obvious downside, however, is that a Challenge-Response may be more difficult or costly to enforce, depending on the tools and processes afforded by the manufacturer at both the point of implementation and response.

Figure 3 – Challenge-Response as a means to authorize or unlock JTAG



Now, before a symmetric mechanism can even be employed, manufacturers need to generate both the Serial Numbers and JTAG Keys, record the issued Serial Numbers, and finally, deploy the JTAG Keys to the relevant ECUs. *Figure 4* (below) summarises the process of how manufacturers might generally be able to accomplish this, using a HMAC-based cryptographic algorithm–or a YubiHSM 2, as the case may be.

Figure 4 – The process of generating a JTAG Key and writing it to an ECU



To elaborate on the enumerated elements from the diagram:

1. The administrator's authentication password is used to access the cryptographic algorithm, and is *different* to the underlying HMAC Master Key (to prevent key leakage)
2. Each successful Serial Number (SN) generation should be recorded in a (secure) database for both audit and future tracking purposes
3. The use of an intermediate tool running on a connected PC is likely necessary, in order to orchestrate where SNs and JTAG Keys are sent and received
4. For each ECU, the SN is used as the 'salt' to the cryptographic algorithm, and the Master Key is used to encrypt; the output will always be a unique JTAG key
5. The JTAG key is then written to the ECU, together with the corresponding SN

In instances where this process is already in place albeit with any form of software capable of running the cryptographic algorithm, it should be possible to replace it with a YubiHSM 2, as inputs and outputs are generally standardised (see this upcoming section for more information). Overall, the proposal is not only simple, but is secure and affords elegance backed up by anecdotal evidence that it has been *successfully* deployed by Yubico customers.

Use case #2: Creating certificates to assert authenticity

Public Key Infrastructure (PKI)

Before diving into the specifics of how a YubiHSM 2 might support a public key certificate deployment, it is important to first understand the concept of a *Root Authority* and its role within a framework of trust, otherwise known as Public Key Infrastructure (PKI). A Root Authority (or sometimes referred to as Root Certification Authority), is the f*irst or primary node* to be established in an eventual network of nodes, and is considered the source of truth within a PKI. At first, only the Root Authority itself is able to add to the network by explicitly "trusting" new nodes (or in technical terms, self-signing each new certificate), but as the network expands, additional nodes can also be trusted by nodes that have already been trusted by the Root Authority. A node within the first layer of additional nodes is sometimes called a Certificate Authority (CA) within the PKI, and these nodes, in turn, are able to increase the reach of the network exponentially as a proxy to the Root Authority. It should be clear that each outward development of trust results in a "chain of trust" which can ultimately (and always) be traced all the way back to the Root Authority. To illustrate using an example, if Root Authority A trusts node B, and node B subsequently trusts node C, then to an outside entity observing those three nodes, there is a chain of trust from A to B to C, and it can be concluded that both nodes B and C are trustworthy if A is a *trusted* Root Authority.

The concept of a PKI is borne out of the tenets of *asymmetric cryptography*, whereby the Root Authority, any CAs and all other nodes are in sole possession of their own Private Key and only communicate to other nodes via messages that can be decrypted by their Public Key. To elaborate, in a secure PKI, Private Keys cannot be guessed, replicated or brute forced due to their length and complexity, and therefore prove, without a shadow of a doubt, that the *bearer* is who they claim to be. Due to the mathematical properties of asymmetric cryptography (which will not be explained because they are beyond the scope of this document), any certificate signed by a root CA's Private Key can only be verified by the corresponding Public Key and vice versa. Put another way, if an encrypted message can be decrypted by a Public Key, the only logical conclusion is that only the bearer of the corresponding Private Key must have written or constructed the original message.

The purpose of the CA is ultimately to issue public key *certificates*, which are essentially structured and pre-defined public documents that contain details about the bearer and of course, the Public Key itself. There are two types of certificates that will be described in this white paper: X.509 certificates and Card Verifiable Certificates (CVC).

Finally, the Registration Authorities (RA) is a node in the PKI which verifies and forwards certificate signing requests to a suitable CA in the network, but is also responsible for other certificate lifecycle management functions themselves, such as revocation.

X.509 certificates

An X.509 certificate contains information about the identity of the bearer of that certificate, the public key belonging to the bearer, and finally, the CA that issued it. The first X.509 certificates were issued as part of the International Telecommunications Union's Telecommunication Standardization Sector (ITU-T) and the X.500 Directory Services Standard. Later, IETF created an Internet profile of the X.509 certificate in the PKIX working group.

From a technical perspective, the X.509 certificate is encoded according to DER (Distinguished Encoding Rules). There are several additional formats and protocols in the PKIX working group that cater for issuance, revocation and life-cycle management of X.509 certificates, but will not be discussed here as they are outside the scope of this document. There are billions of issued X.509 certificates in use today, predominantly used within TLS (Transport Layer Security) over the internet, electronic signatures, encryption, and several other security protocols.

Card Verifiable Certificates (CVC)

CVCs are digital certificates that are designed to be processed by devices with limited computing power, and can often be found embedded in devices or objects such as credit cards, smart IoT devices and ePassports among other things. Vital information is written into each CVC as "fields", such as a unique identifier (or serial number), product specific details, information about the manufacturer and, as will be discussed in detail shortly, security information or certification.

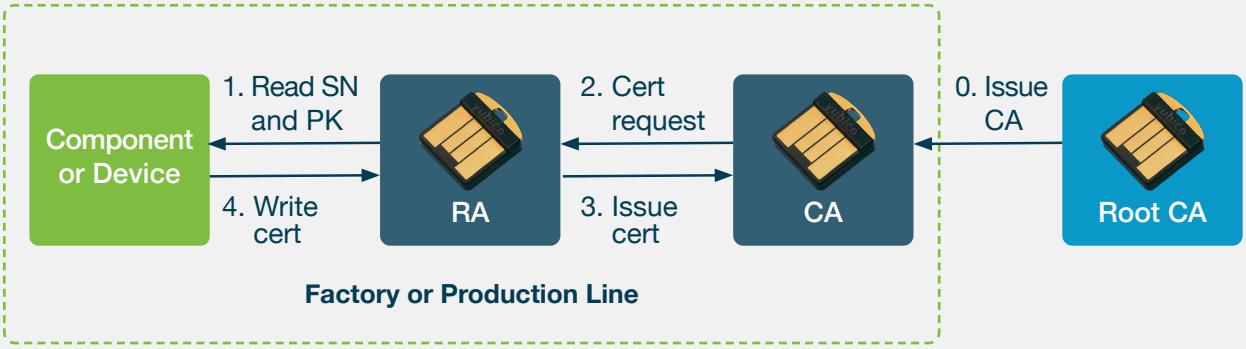
From a technical perspective, CVCs use a concept known as Tag-Length-Value (TLV) encoding, which means that each field in the underlying certificate is of fixed length (values are padded to reach maximum length if they fall short) and each field comes in a predefined order. This makes parsing the respective values extremely simple, in contrast to other forms of variable input that might require more processing power or memory registers to temporarily store field values before being consumed.

The YubiHSM 2, PKI and the Issuance process

A YubiHSM 2 can be deployed to protect a PKI and its network of cryptographic keys, namely those belonging to the Root CA, and any CAs and RAs.

The issuance process of certificates using a PKI based on the YubiHSM 2, as it pertains to a factory, production line or other deployment site, is illustrated in *Figure 5* (below).

Figure 5 – Overview of PKI and issuance process of a certificate



Under the illustrated design, the Root CA uses a YubiHSM 2 to house its key-pair (i.e. both its Private and Public Keys) in addition to its certificate. Due to the diminutive size of the YubiHSM 2, the Root CA server can practically be located anywhere, although traditionally, the Root CA is hosted in an offsite data centre for security reasons, but also due to the tremendous physical size of server farms and a typical HSM.

At the factory or production line, a CA can be deployed and configured with a certificate that has been signed by the Root CA. This process has been denoted as Step 0 in *Figure 5*, and is typically carried out offline and only once, as the factory comes online for the first time. The CA's key-pair and certificate are also protected by a YubiHSM 2, as are the credentials of the downstream RA.

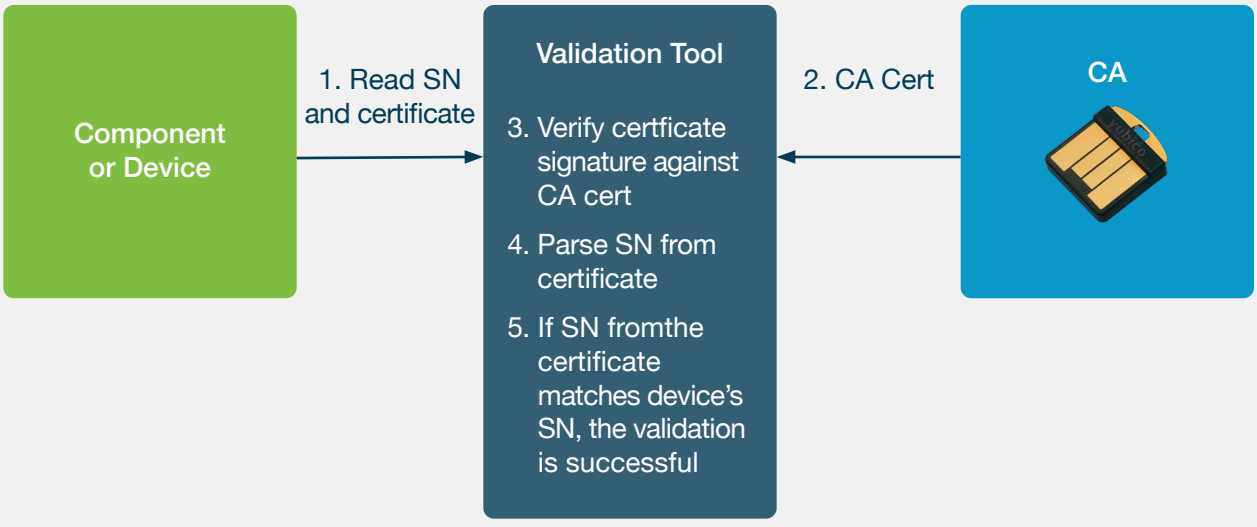
The issuance process of certificates follows the below process:

1. The RA reads the serial number (SN) or an equivalent unique identifier from the component or device being manufactured. If the device includes a TPM with the capability of generating an asymmetric key-pair, the Public Key is also read.
2. The RA creates a certificate request (for example in PKCS #10 format), which includes the serial number and the Public Key of the component or device in question. Note that if no Public Key was previously read, one is generated on behalf of the component or device. The RA too, uses a YubiHSM 2 for signing the certificate request with its Private Key, and forwards the certificate to the CA (using a protocol such as CMC, CMP or SCEP).
3. The CA signs the Certificate with its Private Key. The issued Certificate is returned to the RA.
4 The RA writes the Certificate to the device, where it may also be associated with the Private Key residing in the TPM if applicable. If the device does not have a TPM, the Certificate along with the Private Key are written to the component or device.

Validation process of a certificate to determine authenticity

In order to check the authenticity of a manufactured device, the reader must check the validity of a component or device's certificate. The principal validation process is illustrated in *Figure 6* (below).

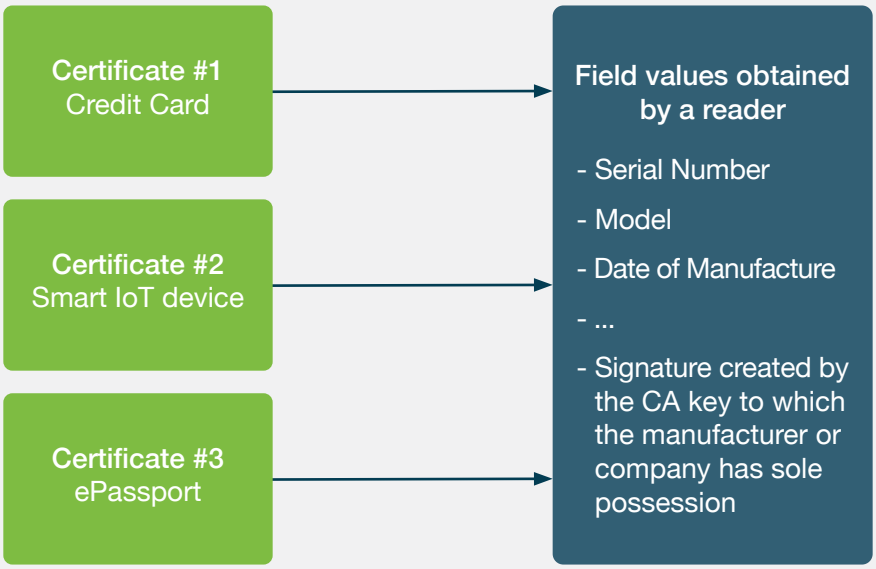
Figure 6 – The certificate validation process



To provide further clarity, the five steps are explained below:

1. The validation tool reads out the serial number (SN) and certificate from the device to be validated
2. Based on CA information in the certificate, the corresponding CA certificate is downloaded directly from the issuing CA. Note that the CA's certificate may also, in turn, be validated using the Root CA's Public Key to ensure the CA's authenticity. In some instances, the validated CA certificate may be cached by the validation tool to speed up future validations
3. The certificate's signature is validated by decrypting it with the CA's Public Key, and compared against the thumbprint stored in the certificate
4. The serial number (or similar unique identifier) is parsed from the certificate. This parsing process applies to all of the different devices, as illustrated in *Figure 7* (below)
5. If the serial number that is parsed from the valid certificate matches the serial number of the component or device, it should thusly be considered authentic

Figure 7 – Reading field values from certificates and obtaining proof of authenticity



Dynamic Challenge-Response validation protocol

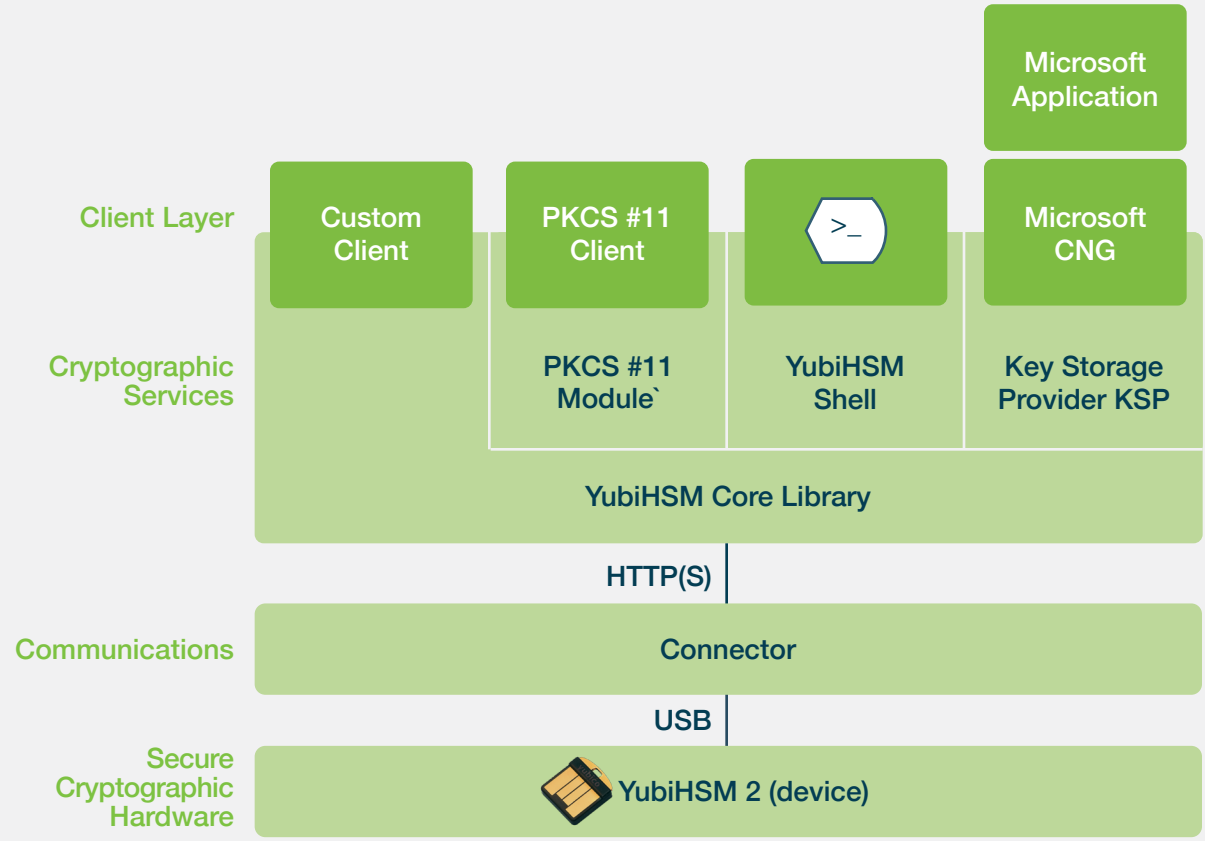
If the component or device also supports a TPM which includes the device's Private Key, it is possible to design a dynamic Challenge-Response protocol similar to the example covered in the JTAG use case. In this scenario specifically, the validation tool generates a random challenge that is signed by the device's TPM Private Key. The signed challenge can then be checked by the validation tool using the Certificate's Public Key, before the rest of the certificate validation process described above initiates.

Communication stack from hardware to software

The YubiHSM 2 communication stack is pictured in *Figure 8* (below), and illustrates how the client layer is ultimately connected to the physical device. As a brief summary (from bottom up):

1. A YubiHSM 2 device is physically inserted into a USB-A port, which in turn is connected to a computer and/or network
2. The Operating System residing on the connected computer and/or network communicates with the device using HTTP, via a deployed binary provided by Yubico known as the Connector (or *yubihsm-connector*), which behaves similarly to a driver
3. The cryptographic services layer, specifically the YubiHSM Core Library (or *libyubihsm*), provides an API which "translates" messages between the client layer above it and the Connector directly below it on the stack
4. Finally, the client layer includes a variety of methods such as PKCS #11 (a widely used cryptographic standard), a command line interface provided by Yubico known as the YubiHSM Shell (or *yubihsm-shell*), Microsoft Cryptographic API Next Generation (CNG) for Microsoft specific applications, plus any number of custom implementations, to control and manage cryptographic functionality

Figure 8 – Overview of the YubiHSM 2 communication stack



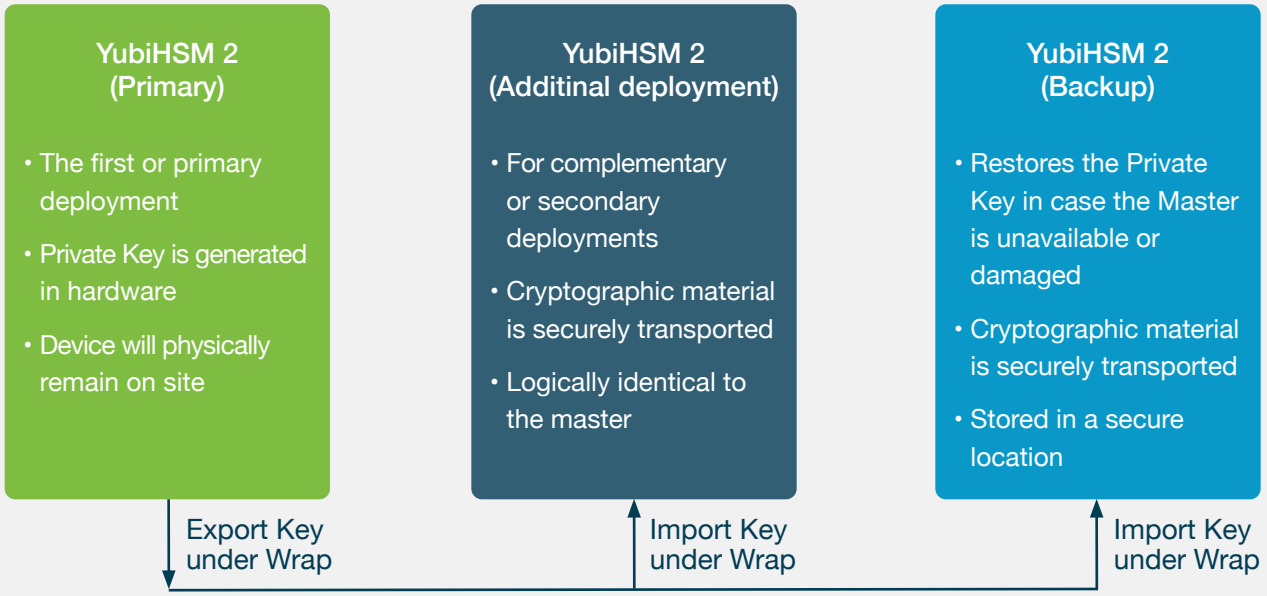
More information about the YubiHSM Shell, PKCS #11 and Microsoft CNG is available in the upcoming section, titled Integrating with the YubiHSM 2.

Replicating the contents of the YubiHSM 2

All YubiHSM 2 Private Keys can be replicated to other YubiHSM 2 devices by first exporting the material under Wrap from the source device, and then importing it onto destination devices. Without getting too much into the technical details, Key Wrapping is the process of encrypting one key using another key, in order to securely store it or transmit it over an *untrusted* channel. The untrusted channel in the context of the YubiHSM 2, to be clear, is any place *outside* the secure element.

In this manner, a YubiHSM 2 containing a Private Key may be replicated to create a logically identical Authentication Key securely, leading to a quick and scalable solution across multiple sites without material leakage. *Figure 9* (below) provides an overview of this, and summarises the benefits that replication provides. It should be noted that the Authentication Password for each device does not necessarily need to be the same, and can (should) be updated according to specific deployment preferences.

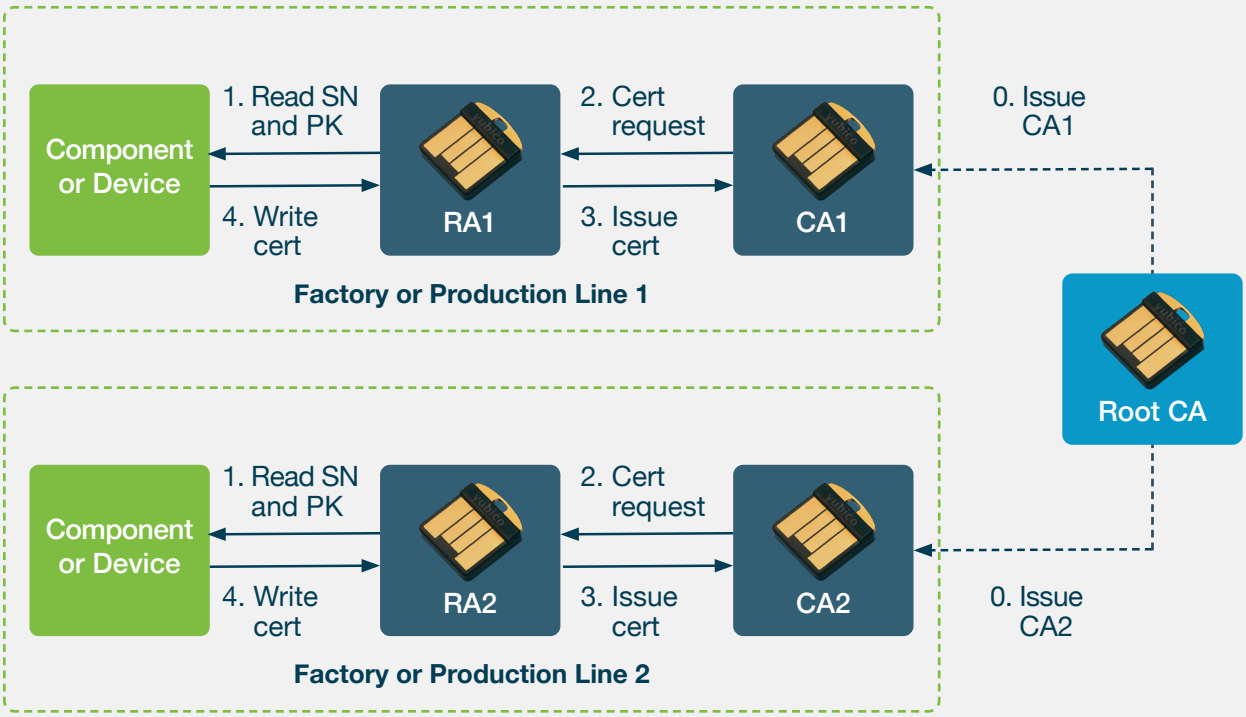
Figure 9 – Overview of device deployment across multiple sites



Relating the replication concept back to the JTAG use case covered previously in this paper, one YubiHSM 2 device could be deployed directly to a production line for writing encrypted serial numbers to the ECUs, while another device could be delivered to a trusted customer for unlocking the corresponding JTAGs for testing. Another device still, could be stored in a safe as backup, in the event the primary device is lost, stolen or damaged.

Doing the same, but with a focus on the certificate use case, multiple factories could be brought online using a replicated CA to RA to component/device configuration. *Figure 10* (below) extends the original design from *Figure 5*, by adding additional sites. Although only two are illustrated, there is no ceiling to the actual number in practice.

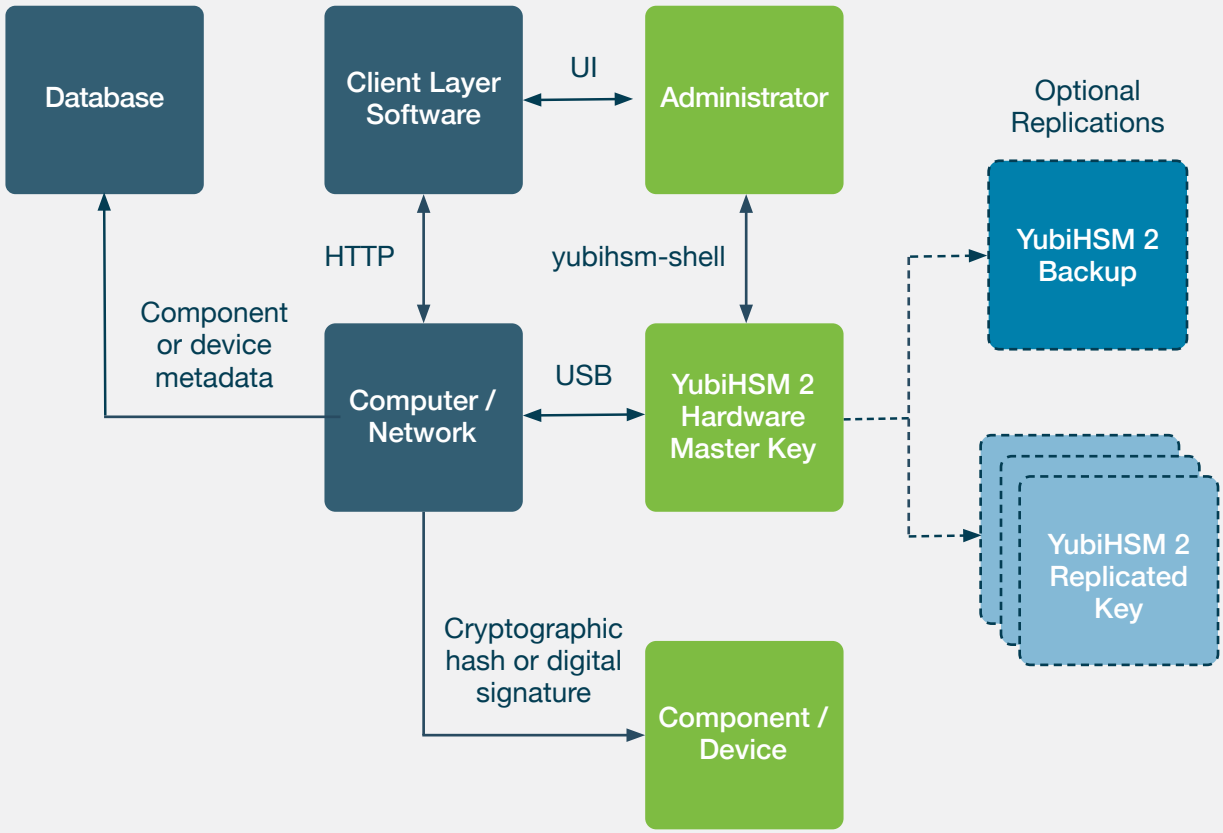
Figure 10 – Extension of the original YubiHSM 2 certificate issuance process



The unified YubiHSM 2 design proposal

By collating all of the information to this point, from the use cases, the communication stack and the option of replication, it is possible to generalise how the YubiHSM 2 can be fitted (or retrofitted) into a manufacturer's production process. *Figure 11* (below) illustrates the amalgamation of all of the factors into one coherent, unified design proposal.

Figure 11 – Design of the proposed unified solution



Although every deployment plan and manufacturing circumstance is unique and may not necessarily require all components depicted in *Figure 10*, the overall goal should always be to keep all Private Keys securely stored inside a YubiHSM 2 to minimise threats to the cryptographic sub-process. The benefits of incorporating a design around the YubiHSM 2 have already been stated, and the design can easily be scaled by increasing the number of replicated devices to suit manufacturers both large and small, local and dispersed.

Integrating with the YubiHSM 2

There are several different APIs and tools available for integration with YubiHSM 2:

- YubiHSM Shell
- PKCS #11
- Microsoft CNG
- Java libraries
- Python

YubiHSM Shell

The YubiHSM Shell is a command line tool that can be used for configuring the YubiHSM 2 and to interact with the underlying cryptographic functions. The typical use case for YubiHSM Shell is the initial configuration of the YubiHSM 2 with the master keys, but also to replicate these keys to multiple YubiHSM 2 devices.

The YubiHSM Shell can also be used for creating JTAG-Keys, but those operations require manual interaction by an administrator, and are not recommended for large scale deployments. For a programmatic integration with the YubiHSM 2, the APIs PKCS #11, Microsoft CNG, Python, or the Java libraries are instead recommended.

PKCS #11

A potentially more user-friendly and forward thinking solution (as compared to command line operation of the yubihsm-shell) is to integrate applications to the YubiHSM 2 using the client layer YubiHSM 2 PKCS #11 library API. Such an application or tool can then present users with a graphical navigation menu or interface that may perform YubiHSM 2 operations under the hood. An programmatic integration with the PKCS #11 API will also allow for an automated process of the creation of JTAG-keys or certificates. The PKCS #11 library provides low level cryptographic functions, which are useful when enumerating and selecting the keys that have been generated in the YubiHSM 2.

Microsoft CNG

If the environment in question is running on Microsoft Windows, Microsoft Cryptography API Next Generation (CNG) is a viable option. The YubiHSM 2 Key Storage Provider (KSP) can be plugged into the CNG layer, and accessed from any application that is developed on top of Microsoft CNG. Microsoft CNG offers more functionality in terms of the encoding of cryptographic messages and certificates, and could be an alternative for certificate issuance. PKCS #11 is however the recommended choice for HMAC-signing of JTAG-keys.

Java libraries

For a Java environment, there are two options when integrating with YubiHSM 2–either the native JCE PKCS #11 provider or the YubiHSM 2 Java library.

The JCE PKCS #11 provider is implemented and supported by Oracle (previously Sun) for the Java Cryptographic Architecture (JCA). The JCE provider loads the PKCS #11 library through a Java native interface, and is plugged into the JCA framework. With this architecture, developers can implement Java applications on top of JCA, and get access to the YubiHSM 2 cryptographic keys and functions.

As an alternative to the JCE/JCA architecture, Yubico provides the YubiHSM 2 Java library, which connects over HTTP(S) to the YubiHSM Connector. The YubiHSM 2 Java library is available as a GitHub project, and is not yet officially released as a Yubico SDK.

Python

The last alternative is to integrate with the YubiHSM 2 by using the YubiHSM Python library. The YubiHSM Python library allows developers to interface with a YubiHSM 2 through the Connector service using the Python programming language. Naturally, the YubiHSM Python library is a viable option when developing applications in Python.

yubico

About Yubico

Yubico sets new global standards for simple and secure access to computers, mobile devices, servers, and internet accounts.

The company's core invention, the YubiKey, delivers strong hardware protection, with a simple touch, across any number of IT systems and online services. The YubiHSM, Yubico's ultra-portable hardware security module, protects sensitive data stored in servers.

Yubico is a leading contributor to the FIDO2, WebAuthn, and FIDO Universal 2nd Factor open authentication standards, and the company's technology is deployed and loved by 9 of the top 10 internet brands and by millions of users in 160 countries.

Founded in 2007, Yubico is privately held, with offices in Sweden, UK, Germany, USA, Australia, and Singapore. For more information: www.yubico.com.