# Client Module with Multifactor Authentication for *Remote* Electronic Signature Generation Using Cryptography API: Next Generation

Ionuț Cătălin DUMITRU and Mihai TOGAN

*Abstract*—**In classic PKI systems, users resorted to cryptographic devices like smartcard or electronic-token to perform the required cryptographic operations, therefore all cryptographic operations will be done in a safe environment. Although a widely-accepted and highly-efficient method, along with the evolution of technology, it is also desirable to simplify user experience with the applications and at the same time to reduce costs. Thus, there is the problem of cryptographic devices and passwords needing to be retained and secret. To overcome all these impediments, we introduce the concept of remote signing, a concept that will simplify the signature generation process, and also increase security to the whole process while lowering costs for equipment. Therefore, by the fact that the signature generation process will be delegated to a remote service, the possibility of generating errors due to the client system is minimized, the need for a cryptographic device for users is eliminated. At the same time, the entire computational effort is also delegated to the *remote* service, which means an increased computing speed, and by the fact that the cryptographic devices that generate the signature are located at distance, there is also the possibility of physical securing and limiting unauthorized access.**

*Index Terms*—**Cryptography API Next Generation, Cryptographic Service Provider, digital signature, Key Storage Provider, remote signature.**

## I. Introduction

After the real success of the public key infrastructures (PKI) in the process of digitalization the flow of documents, technological evolution comes to the aid of the user through the concept of cloud computing.

PKI systems run based on public-private key pairs and digital certificates issued by an accredited certificate authority (CA).

In order to protect private keys, the PKI system requires the use of cryptographic devices (e-Token, smart card) for storing the private key for each user, which significantly raises the costs.

Since the effort of maintaining a PKI architecture is quite expensive, the certification authorities consider the alternative of issuing digital certificates compatible with remote signatures generation. The user private keys that were originally stored by each user can now be stored and managed in a controlled environment such as Hardware Secure Modules (HSM). To restrict access to critical resources, users will go through an authentication process by specialized authentication methods, according to each certificate authority.

This paper suggests a way to migrate the existing signing applications to remote signatures generation, without changing the user experience or the flow of the application.

As organizations have an existing public key infrastructure using digital certificates through an integrated signing application, we can consider modifying the application so that the signatures will be generated within a remote server. In this way, several problems can be solved simultaneously. The user experience within the signing application will not be modified; the application flow will be managed in the same manner. The companies will not have to change their entire infrastructure, will reduce the costs required for cryptographic devices, and by isolating the private keys, their integrity and security can be increased by physical means.

The rest of the paper is organized as follows: section two presents the architecture of Cryptography API: Next Generation and gives a brief introduction to Key storage providers. Section three presents the standards that define a remote signature generation, the state of the art in cloud signature and a description of remote cryptography. Section four describes the implementation of the cloud key storage provider. The last section deals with some conclusions and future work directions.

## II. Cryptography API: Next Generation Key Storage Providers

Cryptography API: Next Generation (CNG) [1] is the long-term replacement for the CryptoAPI. CNG is designed to be extensible at many levels and cryptography agnostic in behavior.

With this new concept, support for the old CryptoAPI is also kept, so that the applications compatible with this standard are not affected.

Microsoft CNG is integrated and used with high priority in signing solutions such as Adobe Acrobat Reader or Microsoft Outlook. Given the spread of these solutions, the modification of the CNG API for generating remote digital signatures can be considered.

The past CAPI (CryptoAPI) had to manage upgrading schemes in a self-organized separate way when new functions were required or the algorithms and environmental

I.C. DUMITRU is with the Computer and Cyber Security Department, "Ferdinand I" Military Technical Academy, Bucharest, Romania (e-mail: catalin.dumitru@mta.ro).

M. TOGAN is with the Computer and Cyber Security Department, "Ferdinand I" Military Technical Academy, Bucharest, Romania (e-mail: mihai.togan@mta.ro).

Digital Object Identifier 10.32754/JMT.2020.1.01

facilities needed to be revised. In this approach, in the development of new algorithms or for the upgrading of the existing ones, there is a problem of restructuring the providers, which will eventually lead to an unsustainable solution and reduce the focus on developing and streamlining algorithms.

According to CNG Features [2], Microsoft CNG New Generation comes up with cryptographic agility. It was necessary to provide substitution and discoverability for all the algorithm types (symmetric, asymmetric, hash functions), random number generation, and other utility functions. The protocol-level changes are more significant because in many cases the protocol APIs needed to add algorithm selection and other flexibility options that did not previously exist.

For an increase of efficiency and security, Microsoft CNG has a finer-grained abstraction for key storage and separation of storage from algorithm operations. The API also provides a set of functions that perform basic cryptographic operations such as creating hashes, random number generator, symmetric encryption/decryption data, asymmetric encryption/decryption data, digital signature, and secret agreement.

CNG supports cryptography in kernel mode or user mode, the same APIs are used in both kernel and user mode to fully support the cryptography features [3].

Each algorithm class in CNG is represented by a primitive router. Applications using the CNG primitive functions will link to the router binary file Bcrypt.dll in user mode, or Ksecdd.sys in kernel mode before calling the functions [4].

The basic structure of the CNG API consists of the cryptographic algorithm providers (CAP), and the cryptographic key storage provider (KSP). Since the Next Generation (CNG) separates cryptographic providers from key storage providers [17], cryptographic algorithm providers implement the following algorithms: symmetric, asymmetric, Hashing and Key Exchange.

KSPs can be used to create, delete, export, import, open and store keys. Depending on the implementation, they can also be used for asymmetric encryption, secret agreement, and signing [17].

The structure of Microsoft CNG can be represented as follows:

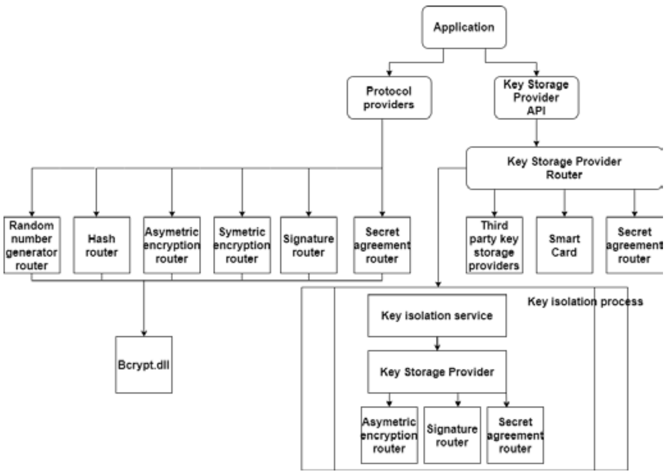

Figure 1. Microsoft CNG Architecture

CNG provides a model for private key storage that allows adapting to the current and future demands of creating applications that use cryptography features such as public or private key encryption, as well as the demands of the storage of key material. The key storage router is the central routine in this model and is implemented in Ncrypt.dll. An application accesses the key storage providers (KSPs) on the system through the key storage router, which conceals details, such as key isolation, from both the application and the storage provider itself [18]. The long-lived keys must be isolated so that they are never present in the application process, starting with Windows Server 2008 and Windows Vista, asymmetric private keys are supported by default [18].

The cryptographic primitives are part of the BCrypt family, while the key storage and recovery module is part of the NCrypt family, and the major difference between them is that the functions in BCrypt are specialized for working with ephemeral keys, while the functions in NCrypt are used to work with persistent ones.

While some public/private key operations can be performed with BCrypt functions, they can only be used with ephemeral keys and are therefore of limited use.

CNG stores the public portion of the stored key separately from the private portion. The public portion of a key pair is also maintained in the key isolation service and is accessed by using local remote procedure call; all access to private keys goes through the private key router and is audited by CNG [18].

For Microsoft CNG to be compatible with generating digital remote signatures, it is necessary at least to modify these functions:

1. <u>Open a CNG cryptographic algorithm provider</u>: The first step is executed by **BCryptOpenAlgorithm Provider** and it returns the handle address at the **phAlgorithm** pointer as the result, which is the first argument of the **BCryptOpenAlgorithmProvider** function.
2. <u>Set some algorithm properties:</u> depending on the desired, this step may be executed by functions such as BCryptGetProperty or BCryptSetProperty.
3. <u>Create or import a key, if it is necessary</u>: symmetric keys will be managed by **Bcrypt** library through functions such as **BCryptExportKey** or **BCrypt ImportKey** while asymmetric keys will be managed by **Ncrypt** library through functions such as **NCryptCreatePersistedKey** or **NCryptImportKey.**
4. <u>Perform the cryptographic operation</u>: create a handle for the cryptographic operation such as **BCryptCreate Hash** or **BCryptEncrypt.**
5. <u>Close the cryptographic algorithm provider:</u> it is possible using **BCryptCloseAlgorithmProvider** function, and it can return an error code if the algorithm handle in the **hAlgorithm** parameter is not valid.

## III. Cryptography In The Cloud

The concept of remote signing appears with the advancement of technology in the field of communications networks, it is also a measure to increase both the ease of use of applications and data security.

### *A. Standards in cloud signature generation*

Once the concept is established, it is necessary to develop a series of standards that define each component included in this new process. Thus, the European Telecommunications Standards Institute (ETSI) proposes in [5] their standardization method.

Given developments in distributed systems, cloud computing, mobile equipment, and related technologies, solutions have been emerging in the last few years where the process of digital signature creation and construction of AdES format [9] is done in a distributed way with different steps of the process carried out by different systems/services that may be controlled by different actors. TS 119 432 [10] document specifies protocols and interfaces for components providing specific functionalities as part of a process for remote digital signatures creation.

European Standardization Committee issues the required standards for the server signing system through EN 419 241-1. The protection profile for the signing process on the server is issued in EN 419 241-2 standard [6]. These two documents are the basis of the implementation of any remote digital signatures system.

Because the entire digital signature system is based on the connection between a digital certificate and a private key, the trust service provider (TSP) has the responsibility to assure the integrity of electronic identification for signatories and services through strong mechanisms for authentication, electronic signatures and digital certificates.

The Cryptographic Module for Trust Services is described by CEN in EN 419 221-5 [7] document and ETSI in TS 119 431-1 [8].

The following schema of the standards describes the interaction between the signing application and the TSP component:

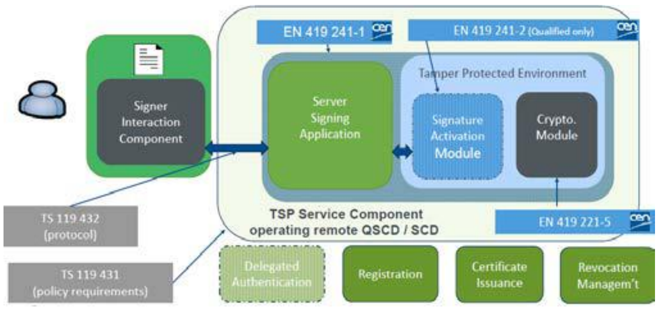
EN 419 241-1
EN 419 241-2 (Qualified only)
Tamper Protected Environment
Signer Interaction Component
Server Signing Application
Signature Activation Module
Crypto. Module
TSP Service Component operating remote QSCD / SCD
EN 419 221-5
TS 119 432 (protocol)
TS 119 431 (policy requirements)
Delegated Authentication
Registration
Certificate Issuance
Revocation Managem't

Figure 2. Structure of ETSI standards in remote signing

### *B. Cloud signature generation solutions*

On the market there are many cloud signing solutions, these include:

- SigningHub: as shown in [11], it represents a web platform that aims to dispose of cloud technologies.
- Cryptomathic: is a global provider of secure server solutions for companies across the industry, including the banking, government, technology, cloud and mobile sectors [12].

## IV. Implementation of the CNG Cloud KSP

As presented in [13], Microsoft has two totally separate dynamic libraries, namely Basecsp.dl responsible for function calls required for devices that are compatible only with the CryptoAPI standard, and Scksp.dll responsible for function calls required for devices compatible with the CNG standard. For security reasons, devices compatible with both standards will use CNG with high priority.

Key Storage Providers (KSP) is an added component with the advent of Cryptography API: Next Generation (CNG). It can be used to create, delete, export, import, save or use keys. A KSP provides many cryptographic facilities, in order to perform a digital signature, it is necessary to call the following functions: NCryptOpenStorageProvider, NCrypt OpenKey, NCryptSignHash, NCryptFreeKey, NCrypt FreeProvider.

The entire process is depicted in Fig. 3.

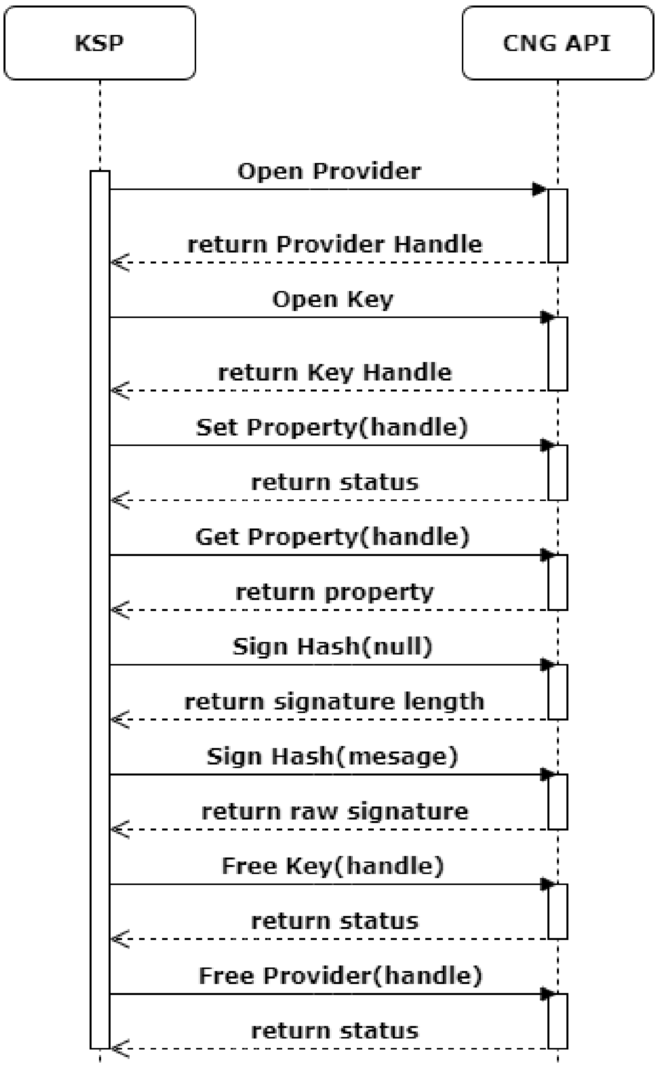

Figure 3. CNG signature generation process diagram

The process of generating a digital signature is based on the use of digital certificates, hence connected with the certificate, the required provider is chosen and opened through the function NCryptOpenStorageProvider. As a private key is required for signing, the NCryptOpenKey function is called, giving as a parameter the identifier of the private key assigned to the digital certificate used for signing. To find out the byte length of the private key used, call the NCryptGetProperty function, with the property NCRYPT_LENGTHS_ PROPERTY. It is necessary for the

NCryptSignHash function to be called twice, the first time, to request for the signature length, the second time, once we have allocated the signature buffer, the actual value of the signature will be returned. NCryptFreeKey and NCryptFreeProvider were called to free the key and provider handles.

The architecture for implementing remote digital signature using a key storage provider is composed of a customized Key Storage Provider, a cryptographic server with a database that stores digital certificates alongside the associated private key and a windows service that imports digital certificates from the database into the windows certificate store. The structure diagram is depicted in Fig. 4.

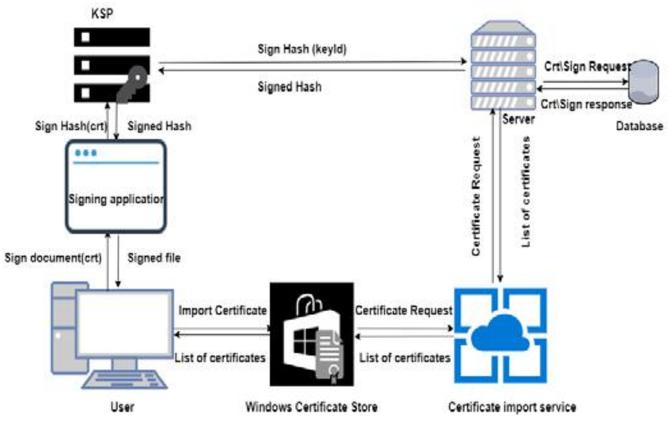

Figure 4. Structure diagram of remote signing application

The cryptographic server provides the necessary functions to generate a digital signature through a dynamic link library. The certificate import service will upload to the user's Windows certificate store all certificates present in the database, through the server. All digital certificates from the server will be registered in the custom KSP.

When a user wants to initiate the electronic signature procedure, through a signing application, of a pdf document, the application lists all the certificates that are currently in the local store, and according to the certificate will use the appropriate Key Storage Provider.

The KSP does not directly store the private keys corresponding to the digital certificates, but a handle that refers to it. Therefore, the actual signing of the pdf document will be managed by KSP, but it will be done through the server. The certificate import service will request the server at a certain interval of seconds to import all existing certificates for the user and these will be imported into the Windows certificate store.

When a user initiates a signing process of a pdf file the flow will be as follows: the user will select a digital certificate from the windows certificate store, depending on the chosen certificate the appropriate KSP will be assigned. If the selected certificate is imported from the server, it will be stored in the custom KSP. For every certificate stored in the new KSP, the signing process will be modified. The Signing application will send the hash of the selected document and a private key identifier correspondent with the digital certificate to KSP. For the signing process, the KSP will send the hash and the key identifier to the cryptographic server for the effective signing. The server depending on the identifier sent generates a signature using the private key stored on a cryptographic device and will send back the signature to the signing application. The signature storing algorithm is PKCS # 7 detached standard and is detailed in RFC 2315 [14].

In the rest of the chapter, we will focus on the implementation of the custom Key Storage Provider.

Microsoft provides a development kit for generating a custom Key Storage Provider, depending on the required application [15], this was the starting point for the development of the remote KSP.

The following functions have been modified in the customization process: GetKeyStorageInterface OpenStorageProvider, OpenKey, SetProperty, GetProperty, SignHash, FreeKey, FreeProvider.

These functions are called by the signing application in the following manner:

Depending on the Key Storage Provider assigned to the selected certificate for signing, it calls the GetKeyStorageInterface function, in order to access the specified KSP functions.

Once establishing the access to the provider functions, it is necessary to start the interaction with that provider, this is determined by calling the function NcryptOpenStorage Provider, giving as parameter the name of the provider assigned to the digital certificate used. The name of the key storage provider is also set as part of the properties of the digital certificate when it gets imported into the certificate store.

As a private key is required for signing, the NCryptOpenKey function is called, giving as a parameter the identifier of the private key assigned to the digital certificate used for signing. The key identifier is also set as part of the properties of the digital certificate.

For security reasons, presented in [16], the NCryptSetProperty function is called, with the property NCRYPT_WINDOW_HANDLE_ PROPERTY.

To find out the byte length of the private key used, call the NCryptGetProperty function, with the property NCRYPT_LENGTHS_PROPERTY. At this step, the KSP will call the server for the private key length.

The first NCryptSignHash function call, with the output buffer parameter NULL_PTR, is required to make the request to the signature length. The server will compute and send the signature length.

The second call of the NCryptSignHash function, this time having allocated the buffer where the signature returns with the byte length found before. Following this call, the server will compute the signature of the received hash, and the actual value of the signature will be returned to the signing application through KSP.

NCryptFreeKey and NCryptFreeProvider functions were called to release the handle-type objects to key and provider.

Before the signing process begins, the key storage provider asks the user for the signature password/PIN, and if they are correct, the server will perform the raw signature and will send it back to the KSP. The signing application will envelop the raw signature to a PKCS#7 structure and will attach it to the document. Finally, the signature can be successfully verified by the application if necessary. The remote signature flow is described in Fig. 5.

Due to the fact that the signature applications use the CNG standard, the user experience will not be affected. In order for the certificates to be imported into the Windows certificate store, it is necessary to register the provider. This can be done using the Microsoft samples provided through the Cryptographic Provider Development Kit [15].

The main objective of the paper was to implement a solution for generating digital remote signatures using the Microsoft Cryptography API: Next Generation, with digital certificates and private keys stored remotely.

Because the digital signing process is applied to the hash of the file and the server can use advanced cryptographic devices, there was no issue with the runtime.

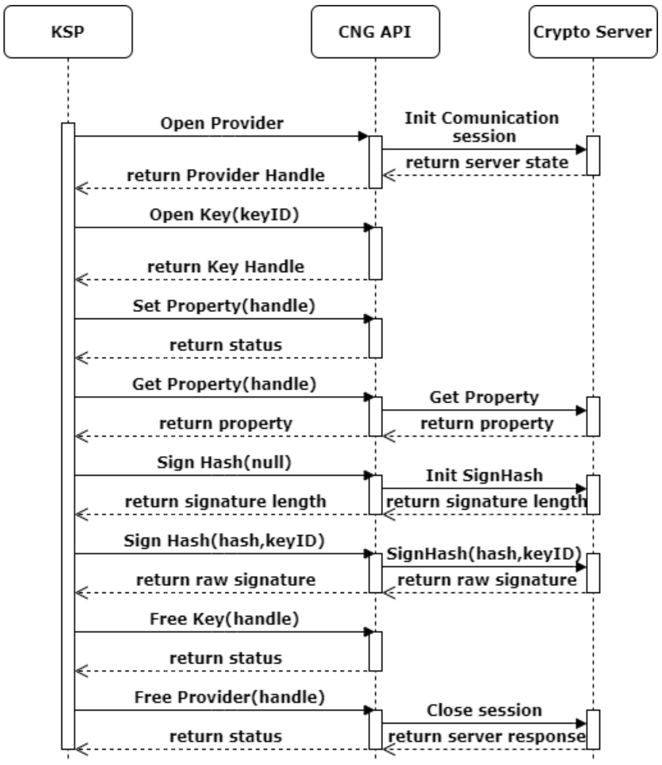

Figure 5. Remote signing flow using customized KSP

Fig. 5 is a sequence diagram of the entire cryptographic mode of generating a digital signature. The server has the role of storing the private keys of the users, but also of performing the cryptographic operations related to them.

In the process of generating a remote digital signature, KSP sends to the server the required requests for the following operations: to initiate or close a communication session with the server, taking over some private key features, such as length in bytes, or for initiating and performing the signing operation.

The presented implementation comes as an alternative to systems based on individual cryptographic devices such as e-Token or smartcards. It offers an added ease of use by eliminating the necessity of cryptographic devices for each user while reducing the overall management and maintenance costs.

In terms of security, proper user authentication is the key to remote signature generation systems. For this purpose, multiple-factor authentication can be used.

Given that the critical infrastructure is isolated at one point, the danger of private keys being compromised is less likely. In addition, physical protection mechanisms can also be applied to cryptographic devices used (HSM).

## V. Conclusion

This article illustrates a remote digital signage generation system based on the Microsoft CNG architecture adopted by applications such as Adobe Acrobat Reader or Microsoft Outlook. The idea of the project was to detach the private keys from the users and to store them remotely in a protected environment, all without changing the use of the signing applications. This concept is a new one, which represents the result of the advances in technology in the field of communications networks but also of the security of the cyberspace.

For increased security, multiple user authentication factors can be added, as well as additional security of the cryptographic server.

The module was tested on Adobe Acrobat Reader Dc for signing pdf documents and Microsoft Outlook for signing emails.

Future goals include adding a biometric user authentication factor but also developing mutual authentication functionality in Microsoft Internet Explorer trough the Microsoft Key Storage Provider.

## References

[1] Microsoft, "Cryptography API: Next Generation", Microsoft Developer, https://docs.microsoft.com/en-us/windows/win32/seccng/cng-portal, 05/31/2020.
[2] Cloud Signature Consortium Standard, Architectures, Protocols and API Specifications for Remote Signature Applications, public pre-release version 0.1.7.9, 2017, published on-line: http://www.cloudsignatureconsortium.org/.
[3] Microsoft, "Cryptography API: Next Generation", CNG Features, https://docs.microsoft.com/en-us/windows/win32/seccng/cng-features 05/31/2020
[4] Microsoft, "Cryptography API: Next Generation", Cryptographic Primitives, https://docs.microsoft.com/en-us/windows/win32/seccng/cryptographic-primitives, 05/31/2020
[5] Nick Pope, Where do we stand on standards for Remote Signature Creation https://docbox.etsi.org/Workshop/2018/201806_ETSISECURITYWEEK/REMOTE_SIGNATURE_CREATION/ETSI%20_TC_ESI_POPE.pdf, 2018.
[6] EN 419 241-2 Trustworthy Systems Supporting Server Signing Part 2: Protection Profile for QSCD for Server Signing https://www.ssi.gouv.fr/uploads/2018/09/anssi-cc-pp-2018_02fr_pp.pdf, 2020.
[7] EN 419 221-5, Protection profiles for TSP Cryptographic modules-Part-5 https://www.commoncriteriaportal.org/files/ppfiles/ANSSI-CC-PP-2016_05%20PP.pdf, 2016.
[8] TS 119 431-1, TSP service components operating a remote QSCD/SCDev https://www.etsi.org/deliver/etsi_ts/119400_119499/11943101/01.01.01_60/ts_11943101v010101p.pdf, 2018.
[9] TS 119 431-2 TSP service components supporting AdES digital signature creation https://www.etsi.org/deliver/etsi_ts/119400_119499/11943102/01.01.01_60/ts_11943102v010101p.pdf, 2018.
[10] TS 119 432 Protocols for remote digital signature creation https://www.etsi.org/deliver/etsi_ts/119400_119499/119432/01.01.01_60/ts_119432v010101p.pdf, 2019.
[11] Signing Hub https://www.signinghub.com.

[12] A guide for Banks and financial Institutions, Cryptomathic, https://www.cryptomathic.com/hubfs/Documents/White_Papers/Cryptomathic_White_Paper_-_Achieving_Real-World_Crypto-Agility:_a_Guide_for_Banks_and_Financial_Institutions.pdf.
[13] Microsoft Smart Card Architecture https://docs.microsoft.com/en-us/previous-versions/windows/desktop/secsmart/smart-card-architecture, 05/31/2020
[14] PKCS#7:Cryptographic Message Syntax, RSA Laboratories https://www.ietf.org/rfc/ RFC2315.txt, 03.1998
[15] Microsoft Cryptographic Provider Development Kit https://www.microsoft.com/en-us/download/details.aspx?id=30688.
[16] Understanding Cryptographic Providers https://docs.microsoft.com/ en-us/windows/desktop/seccertenroll/understanding-cryptographic-providers
[17] Microsoft CNG Key Storage Providers, https://docs.microsoft.com/en-us/windows/win32/seccertenroll/cng-key-storage-providers, 05/31/2020
[18] Microsoft Key Storage and Retrieval, https://docs.microsoft.com/en-us/windows/win32/seccng/key-storage-and-retrieval, 05/31/2020