DOCUMENTO TÉCNICO IDNIE

CQe-Solutions

Teléfono: 646 021 741
Email: info@cqesolutions.com

Contenido

Objetivo 3
Lectura de datos públicos 4
readPassport 4
Firma electrónica 6
signTextDNIe 6
signDocumentDNIe 8
signHashDNIe 9
Autenticación 12
authenticationDNIeOpenSession 12
signChallengeDNIe 13

Objetivo

El presente documento tiene como objetivo explicar en detalle los distintos módulos incluidos en la librería iDNIe detallando las diferentes funcionalidades disponibles.
Los módulos disponibles en iDNIe son los siguientes:

- Lectura de la parte pública. Permite establecer la conexión NFC con el documento electrónico de identidad (DNIe, pasaporte, NIE…) mediante el uso del código CAN o los datos MRZ para la lectura de los grupos ICAO contenidos en el documento y comprobar la autenticidad del mismo.
- Firma electrónica. Permite utilizar los certificados de firma o autenticación incluidos en el DNIe para firmar electrónicamente documentos pdf, cadenas de texto o un hash previamente calculado.
- Autenticación en servicios externos. Permite realizar las operaciones de firma sobre el desafío enviado por servicio externo para autenticar a un usuario utilizando el certificado de autenticación incluido en el DNIe.

Lectura de datos públicos

Este módulo dentro de iDNIe permite establecer la conexión con el documento electrónico de identidad (DNIe, pasaporte, NIE…) mediante la tecnología NFC para poder recuperar los datos incluidos en los data groups públicos contenidos en el documento.
Este módulo permite realizar una primera lectura de los datos del DNIe imprescindible para realizar operaciones de firma y autenticación.
Este módulo incluye las siguientes funciones:

readPassport

Permite el establecimiento del canal seguro para recuperar los datos del documento electrónico de utilizando el código CAN o los datos MRZ.
Para generar el código MRZ, iDNIe ofrece la función getMRZKey que permite generar el código MRZ a partir de los siguientes datos: número de documento o número de soporte, fecha de nacimiento y fecha de validez.

Definición de la función:

```
public func readPassport( accessKey : String, paceKeyReference: UInt8,
tags: [DataGroupId] = [], skipSecureElements :Bool = true,
customDisplayMessage: ((NFCViewDisplayMessage) -> String?)? = nil,
leeCertificadosPublicos: Bool = true, completed: @escaping
(NFCPassportModel?, NFCPassportReaderError?)->()) {}
```

Parámetros de entrada:

Parámetro	Tipo	Observaciones
accessKey	String	Espera recibir el código can o mrz del documento.
paceKeyReference	UInt8	Indica el tipo de canal PACE que se debe montar. Este campo acepta los valores: PACEHandler.CAN_PACE_KEY_REFERENCE o PACEHandler.MRZ_PACE_KEY_REFERENCE
tags	[DataGroupId]	Recibe un array con los datagroups a leer. Si no se indica este parámetro o se envía un array vacío ([]) se realizará la lectura de todos los datagroups. Por ejemplo puede enviarse el array [.COM, .SOD, .DG1, .DG2, .DG7, .DG11, .DG12, .DG13, .DG14, .DG15]
skipSecureElements	Bool	Indica si queremos evitar la lectura de los elementos de seguridad (DG3 que contiene la huella digital y DG4 que contiene el iris) aunque se incluyan entre los Datagroups a leer. Estos datos están

		protegidos y no se pueden por apps de terceros. El valor por defecto es true.
customDisplayMessage	((NFCViewDisplayMessage) -> String?)?	Con este parámetro es posible indicar la función que utilizaremos para traducir los mensajes enviados desde la tarjeta a la app. iDNIe contiene una función que permite traducir estos mensajes al castellano.
leeCertificadosPublicos	Bool	Este parámetro indica si desea realizar la lectura de la parte pública de los certificados de autenticación y firma contenidos en el DNIe (no aplica a otros documentos electrónicos de identidad que no continen certificados digitales).

Resultado:

Parámetro	Tipo	Observaciones
passport	NFCPassportModel	Objeto que contiene la lectura de todos los datos leídos del documento electrónico de identidad. La librería incluye una utilidad que permite recuperar procesar estos datos y organizarlos en una estructura más usable.
error	NFCPassportReaderError	Objeto que contiene los diferentes tipos de error y sus textos correspondientes traducidos al castellano en la librería iDNIe.

Ejemplo de llamada a la función:

```
passportReader.readPassport(accessKey: usuarioDnie.can, paceKeyReference:
PACEHandler.CAN_PACE_KEY_REFERENCE, tags: dataGroups, skipSecureElements:
true, customDisplayMessage: { (displayMessage) in  return
NFCUtils.customDisplayMessage(displayMessage: displayMessage)
        }, completed: { (passport, error) in
            if let passport = passport {
              //Passport contiene todos los datos del DNIe
            } else {
              //Se procesa error
            }
        })
}
```

Firma electrónica

Este módulo permite enviar a la tarjeta un dato a firmar y ésta lo firme con la parte privada del certificado seleccionado.

Para poder realizar estas operaciones es imprescindible haber realizado una lectura previa del DNIe y conocer el PIN del DNIe para poder establecer los canales seguros para poder enviar las operaciones de firma a la tarjeta.

Este módulo incluye las siguientes funciones:

signTextDNIe

Firma de un texto en formato String con el certificado del DNIe que se le indique en certToUse.

Definición de la función:

```
public func signTextDNIe( accessKey : String, pin : String, datosFirma:
String, signPadding: DNIeSingPadding = .PKCS, certToUse:
DNIeCertificates?, passport: NFCPassportModel?, paceKeyReference: UInt8,
tags: [DataGroupId] = [], skipSecureElements :Bool = true,
customDisplayMessage: ((NFCViewDisplayMessage) -> String?)? = nil,
completed: @escaping (NFCPassportModel?, NFCPassportReaderError?)->()) {}
```

Parámetros de entrada:

Parámetro	Tipo	Observaciones
accessKey	String	Espera recibir el código can del DNIe.
pin	String	Espera recibir el pin del DNIe.
datosFirma	String	Contiene la cadena que se desea firmar. A esta cadena de texto se le aplica un hash 256 que será el contenido firmado.
signPadding	DNIeSingPadding	Especifica el tipo de padding que se utilizará al firmar el hash del texto especificado. Se permiten los siguientes valores: DNIeSingPadding.PKCS o DNIeSingPadding.PSS. Por defecto se utiliza el padding PKCS que es el formato nativo del DNIe.
certToUse	DNIeCertificates	Permite especificar el certificado a usar en la firma del hash generado. Se permiten los siguiente valores: DNIeCertificates.AUTENTICACION o DNIeCertificates.FIRMA
passport	NFCPassportModel	Espera recibir el objeto NFCPassportModel que se ha obtenido en la lectura del DNIe utilizando la función readPassport.

paceKeyReference	UInt8	Indica el tipo de canal PACE que se debe montar. Este campo acepta el valor: PACEHandler.CAN_PACE_KEY_REFERENCE
tags	[DataGroupId]	Recibe un array con los datagroups a leer. Si no se indica este parámetro o se envía un array vacío ([]) se realizará la lectura de todos los datagroups. Por ejemplo puede enviarse el array [.COM, .SOD, .DG1, .DG2, .DG7, .DG11, .DG12, .DG13, .DG14, .DG15]
skipSecureElements	Bool	Indica si queremos evitar la lectura de los elementos de seguridad (DG3 que contiene la huella digital y DG4 que contiene el iris) aunque se incluyan entre los Datagroups a leer. Estos datos están protegidos y no se pueden por apps de terceros. El valor por defecto es true.
customDisplayMessage	((NFCViewDisplayMessage) -> String?)?	Con este parámetro es posible indicar la función que utilizaremos para traducir los mensajes enviados desde la tarjeta a la app. iDNIe contiene una función que permite traducir estos mensajes al castellano.

Resultado:

Parámetro	Tipo	Observaciones
passport	NFCPassportModel	Objeto que contiene los datos firmados dentro del atributo signedMessage.
error	NFCPassportReaderError	Objeto que contiene los diferentes tipos de error y sus textos correspondientes traducidos al castellano en la librería iDNIe.

Ejemplo de llamada a la función:

```
passportReader.signTextDNIe(accessKey: Tramite.usuarioDNIe!.can, pin:
pinDNIe, datosFirma: (Tramite.datosDnie?.getdatosICAO()?.DG1)!, certToUse:
DNIeCertificates.AUTENTICACION, passport: passport, paceKeyReference:
PACEHandler.CAN_PACE_KEY_REFERENCE, tags: [], skipSecureElements: true,
customDisplayMessage: { (displayMessage) in
            return NFCUtils.customDisplayMessage(displayMessage:
displayMessage)
            }, completed: { (passport, error) in
            if let passport = passport {
                //passport.signedMessage contiene los datos firmados
            } else {
                //procesamos error
            }
        })
    }
```

signDocumentDNIe

Firma el hash del documento pasado como parámetro en document con el certificado del DNIe que se le indique en certToUse.

Definición de la función:

```
public func signDocumentDNIe( accessKey : String, pin : String, document: URL, signPadding: DNIeSingPadding = .PKCS, certToUse: DNIeCertificates, passport: NFCPassportModel?, paceKeyReference: UInt8, tags: [DataGroupId] = [], skipSecureElements :Bool = true, customDisplayMessage: ((NFCViewDisplayMessage) -> String?)? = nil, completed: @escaping (NFCPassportModel?, NFCPassportReaderError?)->()) {}
```

Parámetros de entrada:

Parámetro	Tipo	Observaciones
accessKey	String	Espera recibir el código can del DNIe.
pin	String	Espera recibir el pin del DNIe.
document	URL	Contiene la ruta del documento que se desea firmar. Se calcula el hash 256 del documento que será el contenido firmado.
signPadding	DNIeSingPadding	Especifica el tipo de padding que se utilizará al firmar el hash del texto especificado. Se permiten los siguientes valores: DNIeSingPadding.PKCS o DNIeSingPadding.PSS. Por defecto se utiliza el padding PKCS que es el formato nativo del DNIe.
certToUse	DNIeCertificates	Permite especificar el certificado a usar en la firma del hash generado. Se permiten los siguiente valores: DNIeCertificates.AUTENTICACION o DNIeCertificates.FIRMA
passport	NFCPassportModel	Espera recibir el objeto NFCPassportModel que se ha obtenido en la lectura del DNIe utilizando la función readPassport.
paceKeyReference	UInt8	Indica el tipo de canal PACE que se debe montar. Este campo acepta el valor: PACEHandler.CAN_PACE_KEY_REFERENCE
tags	[DataGroupId]	Recibe un array con los datagroups a leer. Si no se indica este parámetro o se envía un array vacío ([]) se realizará la lectura de todos los datagroups. Por ejemplo puede enviarse el array [.COM, .SOD,

		.DG1, .DG2, .DG7, .DG11, .DG12, .DG13, .DG14, .DG15]
skipSecureElements	Bool	Indica si queremos evitar la lectura de los elementos de seguridad (DG3 que contiene la huella digital y DG4 que contiene el iris) aunque se incluyan entre los Datagroups a leer. Estos datos están protegidos y no se pueden por apps de terceros. El valor por defecto es true.
customDisplayMessage	((NFCViewDisplayMessage) -> String?)?	Con este parámetro es posible indicar la función que utilizaremos para traducir los mensajes enviados desde la tarjeta a la app. iDNIe contiene una función que permite traducir estos mensajes al castellano.

Resultado:

Parámetro	Tipo	Observaciones
passport	NFCPassportModel	Objeto que contiene los datos firmados dentro del atributo signedMessage.
error	NFCPassportReaderError	Objeto que contiene los diferentes tipos de error y sus textos correspondientes traducidos al castellano en la librería iDNIe.

Ejemplo de llamada a la función:

```
passportReader. signDocumentDNIe (accessKey: Tramite.usuarioDNIe!.can,
pin: pinDNIe, document: documentURL, certToUse:
DNIeCertificates.AUTENTICACION, passport: passport, paceKeyReference:
PACEHandler.CAN_PACE_KEY_REFERENCE, tags: [], skipSecureElements: true,
customDisplayMessage: { (displayMessage) in
            return NFCUtils.customDisplayMessage(displayMessage:
displayMessage)
            }, completed: { (passport, error) in
            if let passport = passport {
                //passport.signedMessage contiene los datos firmados
            } else {
                //procesamos error
            }
        })
    }
```

signHashDNIe

Firma el hash y el digest pasados como parámetros con el certificado del DNIe que se le indique en certToUse.

Definición de la función:

```
public func signHashDNIe( accessKey : String, pin : String, hash: [UInt8],
digest: [UInt8], signPadding: DNIeSingPadding = .PKCS, certToUse:
DNIeCertificates, passport: NFCPassportModel?, paceKeyReference: UInt8,
tags: [DataGroupId] = [], skipSecureElements :Bool = true,
customDisplayMessage: ((NFCViewDisplayMessage) -> String?)? = nil,
operacion: DNIeOperations, completed: @escaping (NFCPassportModel?,
NFCPassportReaderError?)->()) {}
```

Parámetros de entrada:

Parámetro	Tipo	Observaciones
accessKey	String	Espera recibir el código can del DNIe.
pin	String	Espera recibir el pin del DNIe.
hash	[UInt8]	Contiene el hash del elemento a firmar.
signPadding	DNIeSingPadding	Especifica el tipo de padding que se utilizará al firmar el hash del texto especificado. Se permiten los siguientes valores: DNIeSingPadding.PKCS o DNIeSingPadding.PSS. Por defecto se utiliza el padding PKCS que es el formato nativo del DNIe.
certToUse	DNIeCertificates	Permite especificar el certificado a usar en la firma del hash generado. Se permiten los siguiente valores: DNIeCertificates.AUTENTICACION o DNIeCertificates.FIRMA
passport	NFCPassportModel	Espera recibir el objeto NFCPassportModel que se ha obtenido en la lectura del DNIe utilizando la función readPassport.
paceKeyReference	UInt8	Indica el tipo de canal PACE que se debe montar. Este campo acepta el valor: PACEHandler.CAN_PACE_KEY_REFERENCE
tags	[DataGroupId]	Recibe un array con los datagroups a leer. Si no se indica este parámetro o se envía un array vacío ([]) se realizará la lectura de todos los datagroups. Por ejemplo puede enviarse el array [.COM, .SOD, .DG1, .DG2, .DG7, .DG11, .DG12, .DG13, .DG14, .DG15]
skipSecureElements	Bool	Indica si queremos evitar la lectura de los elementos de seguridad (DG3 que contiene la huella digital y DG4 que contiene el iris) aunque se incluyan entre los Datagroups a leer. Estos datos están protegidos y no se pueden por apps de terceros. El valor por defecto es true.

customDisplayMessage	((NFCViewDisplayMessage) -> String?)?	Con este parámetro es posible indicar la función que utilizaremos para traducir los mensajes enviados desde la tarjeta a la app. iDNIe contiene una función que permite traducir estos mensajes al castellano.
operacion	DNIeOperations	Este parámetro permite especificar la opción de firma que se va a realizar. Se utiliza este parámetro para diferenciar el mensaje que se muestra al usuario tras la firma del hash. Acepta los siguientes valores: DNIeOperations.FIRMA_DATOS o FIRMA_DATOS.FIRMA_DOCUMENTO

Resultado:

Parámetro	Tipo	Observaciones
passport	NFCPassportModel	Objeto que contiene los datos firmados dentro del atributo signedMessage.
error	NFCPassportReaderError	Objeto que contiene los diferentes tipos de error y sus textos correspondientes traducidos al castellano en la librería iDNIe.

Ejemplo de llamada a la función:

```
passportReader.signHashDNIe(accessKey: Tramite.usuarioDNIe!.can, pin:
Tramite.pinDNIe!, hash: hash, digest: digest, certToUse:
DNIeCertificates.FIRMA, passport: passport, paceKeyReference:
PACEHandler.CAN_PACE_KEY_REFERENCE, tags: [], skipSecureElements: true,
customDisplayMessage: { (displayMessage) in
                return NFCUtils.customDisplayMessage(displayMessage:
displayMessage)
            }, operacion: .FIRMA_DOCUMENTO, completed: { (passport, error)
in
            if let passport = passport {
                //passport.signedMessage contiene los datos firmados
            } else {
                //procesamos error
            }
        })
    }
```

Autenticación

Permite preparar la app para poder establecer la autenticación con diferentes sistemas externos que basan la autenticación en el uso de certificados digitales, como los contenidos en el DNIe.
Este módulo incluye las siguientes funciones:

authenticationDNIeOpenSession

Establece los canales de conexión necesarios para realizar operaciones de autenticación con el DNIe dejándolos abiertos a la espera del desafío enviado por el sistema en el que se desea realizar la autenticación.

Definición de la función:

```
public func authenticationDNIeOpenSession( accessKey : String, pin : String, passport: NFCPassportModel?, paceKeyReference: UInt8, tags: [DataGroupId] = [], skipSecureElements :Bool = true, customDisplayMessage: ((NFCViewDisplayMessage) -> String?)? = nil, completed: @escaping (NFCPassportModel?, NFCPassportReaderError?)->()) {}
```

Parámetros de entrada:

Parámetro	Tipo	Observaciones
accessKey	String	Espera recibir el código can del DNIe.
pin	String	Espera recibir el pin del DNIe.
passport	NFCPassportModel	Espera recibir el objeto NFCPassportModel que se ha obtenido en la lectura del DNIe utilizando la función readPassport.
paceKeyReference	UInt8	Indica el tipo de canal PACE que se debe montar. Este campo acepta el valor: PACEHandler.CAN_PACE_KEY_REFERENCE
tags	[DataGroupId]	Recibe un array con los datagroups a leer. Si no se indica este parámetro o se envía un array vacío ([]) se realizará la lectura de todos los datagroups. Por ejemplo puede enviarse el array [.COM, .SOD, .DG1, .DG2, .DG7, .DG11, .DG12, .DG13, .DG14, .DG15]
skipSecureElements	Bool	Indica si queremos evitar la lectura de los elementos de seguridad (DG3 que contiene la huella digital y DG4 que contiene el iris) aunque se incluyan entre los Datagroups a leer. Estos datos están protegidos y no se pueden por apps de terceros. El valor por defecto es true.
customDisplayMessage	((NFCViewDisplayMessage) -> String?)?	Con este parámetro es posible indicar la función que utilizaremos para traducir los

		mensajes enviados desde la tarjeta a la app. iDNIe contiene una función que permite traducir estos mensajes al castellano.

Resultado:

Parámetro	Tipo	Observaciones
passport	NFCPassportModel	Objeto que contiene los datos firmados dentro del atributo signedMessage.
error	NFCPassportReaderError	Objeto que contiene los diferentes tipos de error y sus textos correspondientes traducidos al castellano en la librería iDNIe.

Ejemplo de llamada a la función:

```
passportReader.authenticationDNIeOpenSession(accessKey:
Tramite.usuarioDNIe!.can, pin: Tramite.pinDNIe!, passport: passport,
paceKeyReference: PACEHandler.CAN_PACE_KEY_REFERENCE, tags: [],
skipSecureElements: true, customDisplayMessage: { (displayMessage) in
                return NFCUtils.customDisplayMessage(displayMessage:
displayMessage)
            }, completed: { (passport, error) in
            if let passport = passport {
                //passport contiene todos los datos del DNIe
            } else {
                //procesamos error
            }
        })
    }
```

signChallengeDNIe

Firma el hash y el digest pasados como parámetros con el certificado de autenticación del DNIe.

Definición de la función:

```
public func signChallengeDNIe(hash: [UInt8], digest: [UInt8], signPadding:
DNIeSingPadding = .PKCS, passport: NFCPassportModel?, paceKeyReference:
UInt8, customDisplayMessage: ((NFCViewDisplayMessage) -> String?)? = nil,
completed: @escaping (NFCPassportModel?, NFCPassportReaderError?)->()) { }
```

Parámetros de entrada:

Parámetro	Tipo	Observaciones
hash	[UInt8]	Contiene el hash del elemento a firmar.
digest	[UInt8]	Contiene la cabecera del algoritmo de creación del hash utilizado.
signPadding	DNIeSingPadding	Especifica el tipo de padding que se utilizará al firmar el hash del texto

		especificado. Se permiten los siguientes valores: DNIeSingPadding.PKCS o DNIeSingPadding.PSS. Por defecto se utiliza el padding PKCS que es el formato nativo del DNIe.
passport	NFCPassportModel	Espera recibir el objeto NFCPassportModel que se ha obtenido en la lectura del DNIe utilizando la función readPassport.
paceKeyReference	UInt8	Indica el tipo de canal PACE que se debe montar. Este campo acepta el valor: PACEHandler.CAN_PACE_KEY_REFERENCE
customDisplayMessage	((NFCViewDisplayMessage) -> String?)?	Con este parámetro es posible indicar la función que utilizaremos para traducir los mensajes enviados desde la tarjeta a la app. iDNIe contiene una función que permite traducir estos mensajes al castellano.

Resultado:

Parámetro	Tipo	Observaciones
passport	NFCPassportModel	Objeto que contiene los datos firmados dentro del atributo signedMessage.
error	NFCPassportReaderError	Objeto que contiene los diferentes tipos de error y sus textos correspondientes traducidos al castellano en la librería iDNIe.

Ejemplo de llamada a la función:

```
passportReader.signChallengeDNIe(hash: hash, digest: digest, signPadding:
signPadding, passport: passport, paceKeyReference:
PACEHandler.CAN_PACE_KEY_REFERENCE, customDisplayMessage: {
(displayMessage) in
                return NFCUtils.customDisplayMessage(displayMessage:
displayMessage)
            }, completed: { (passport, error) in
            if let passport = passport {
                //passport.signedMessage contiene los datos firmados
            } else {
                //procesamos error
            }
        })
    }
```