DFVA Documentation

Versión 0.1

Luis Zárate y Marta Solano

10 de diciembre de 2018

Contenido:

CAPÍTULO 1

Descripción General

DFVA pretente ser un cliente para el firmado digital avanzado, la autenticación mediante certificados digitales y la validación de certificados y documentos.

Proporciona soporte para instituciones y personas utilizando:

- Para instituciones se utiliza una CA interna de certificados o Dogtag.
- Para personas PKCS11 para comunicarse directamente con la tarjeta de firma.

Tantos las personas como las instituciones pueden:

- Solicitar una autenticación.
- Firmar un documento XML (transacciones), ODF, MS Office, PDF.
- Validar un certificado emitido.
- Validar un documento XML, ODF, MS Office, PDF.

1.1 Clientes disponibles

Advertencia: Todos los clientes están en etapa de desarrollo

1.1.1 Personas

- **dfva_client** (https://github.com/luisza/dfva_client/) : Cliente en python para interactuar con DFVA con soporte a PKCS11 para personas.

1.1.2 Instituciones

- **dfva_java** (https://github.com/luisza/dfva_java/) : Cliente en java para interactuar con DFVA para instituciones.

- **dfva_php** (https://github.com/luisza/dfva_php/) : Cliente en php para interactuar con DFVA para instituciones.
- **dfva_python** (https://github.com/luisza/dfva_python/) : Cliente en python para interactuar con DFVA para instituciones.
- **dfva_c** (https://github.com/luisza/dfva_c/) : Cliente en c++ para interactuar con DFVA para instituciones.
- **dfva_c#** (https://github.com/luisza/dfva_csharp/) : Cliente en c# para interactuar con DFVA para instituciones.
- **dfva_html** (https://github.com/luisza/dfva_html/) : Cliente en javascript/html/css para captar información del usuario y mostrar los códigos provistos por dfva para instituciones.

1.1.3 Conexión con el BCCR (interno)

- **pyfva** (https://github.com/solvo/pyfva) : Interactura con el FVA del BCCR, útil en instituciones.

CAPÍTULO 2

Instalación

Nota: Sin hacer, esto solo para quienes estén familiarizados con django y para entorno de desarrollo

2.1 Instale las dependencias

Clone el repositorio desde

```
git clone https://github.com/luisza/dfva.git
```

Instale las dependencias (es recomendable utilizar un entorno virtual)

```
pip install -r requirements.txt
```

Modifique el archivo dfva/settings.py según la documentación.

Advertencia: Solo requiere **un manejador de CA**, puede usar una CA con Python CertBuilder o integrarse con DOGTAG (son excluyentes entre si)

2.2 Instalación de una CA dummy

Cree la carpeta internal_ca y ejecute el siguiente comando para generar la CA.

```
python manage.py  crea_ca
```

Se generará los siguientes archivos:

```
internal_ca/
├── ca_cert.pem
└── ca_key.pem
└── crl.pem
```

Cambie los permisos de los archivos

```
cd internal_ca/
chmod 600 ca_cert.pem  ca_key.pem crl.pem
```

En settings.py agregue:

```
CA_PATH = os.path.join(BASE_DIR, 'internal_ca')
CA_CERT = os.path.join(CA_PATH, 'ca_cert.pem')
CA_KEY = os.path.join(CA_PATH, 'ca_key.pem')
```

2.3 Ejecute la creación de la base de datos.

Asegurese de configurar la base de datos, y el motor de base de datos

```
python manage.py migrate
```

2.4 Corra el programa

```
python manage.py runserver
```

2.5 Corra las tareas syncrónicas

Para desarrollo puede no ser necessario corre las tareas asyncrónicas encargadas de notificar a aplicaciones externas.

Si se desea la funcionalidad de notificaciones con tiempo entonces deben ejecutarse en otra pestaña

```
celery -A dfva worker -l info -B
```

2.6 Settings de interés

- FVA_HOST = «http://localhost:8001/»: Url donde está corriendo el simulador del FVA BCCR
- STUB_SCHEME = “http”: Esquema para el Servicio SOAP del BCCR
- STUB_HOST = «localhost:8001»: Máquina del servicio SOAP del BCCR
- RECEPTOR_HOST = «http://localhost:8000/»: url base donde se recibirán las notificaciones del FVA BCCR
- DEFAULT_BUSSINESS = 1 Identificador del negocio del FVA BCCR
- DEFAULT_ENTITY = 1 Identificador de la entidad del FVA BCCR
- RECEPTOR_CLIENT = “receptor.client” cliente que recibe las respuestas del FVA BCCR

- DFVA_REMOVE_AUTHENTICATION = 5 tiempo en minutos de vida de la autenticación
- DFVA_REMOVE_SIGN = 20 tiempo en minutos de vida de la firma
- DFVA_PERSON_SESSION = 25 duración en minutos de la sesión de una persona

2.7 Instalación de Dogtag manager

Se requiere la biblioteca *dogtag-pki*, y esta a su vez depende de python-nss que tiene como dependencia *libnss3-dev* por eso ejecutar:

```
apt-get install libnss3-dev
pip install dogtag-pki
```

Agregue en settings.py

```
CAMANAGER_CLASS="corebase.ca_management.dogtag"
DOGTAG_HOST='localhost'
DOGTAG_PORT='8443'
DOGTAG_SCHEME='https'
DOGTAG_AGENT_PEM_CERTIFICATE_PATH=os.path.join(BASE_DIR, 'admin_cert.pem')
DOGTAG_CERTIFICATE_SCHEME={
'O': 'EXAMPLE.COM'
}
DOGTAG_CERT_REQUESTER='dfva'
DOGTAG_CERT_REQUESTER_EMAIL='dfva@example.com'
```

Nota: Puede instalar una sistema PKI para pruebas utilizando una imágen en docker de la siguiente forma.

Advertencia: Es recomendable correrla en una máquina con más de 2Gb de RAM

Permite activar IPV6

```
cat /etc/docker/daemon.json
{
  "ipv6": true,
  "fixed-cidr-v6": "2001:db8:1::/64"
}
```

```
docker run --name freeipa-server-container --privileged -ti   \
-h  ipa.mifirmacr.org  \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-p 53:53/udp -p 53:53 \
-p 80:80 -p 443:443 -p 389:389 -p 636:636 -p 88:88 -p 464:464 \
-p 88:88/udp -p 464:464/udp -p 123:123/udp -p 7389:7389 \
-p 8443:8443 -p 8080:8080 -p 9445:9445 \
--security-opt seccomp=unconfined \
--tmpfs /run --tmpfs /tmp \
-v /var/lib/ipa-data:/data:Z freeipa/freeipa-server \
--realm=mifirmacr.org \
--ds-password=LDAPPASSWORD \
--admin-password=ADMINPASSWORD
```

Se requiere que el usuario sea un agente de Dogtag, de lo contrario no se autenticará, para extraer el certificado pkcs12 del usuario admin que además es un agente dogtag debe buscar la llave en

```
docker exec -ti <nombre maquina> bash
cat /data/root/ca-agent.p12 | base64
```

para descomprimir y convertir a pem se recomienda algo como :

```
echo "codigo base64" | base64 -d > ca-agent.p12
openssl pkcs12 -in ca-agent.p12 -out admin_cert.pem -nodes
```

La contraseña es la misma que ds-password osea en este caso LDAPPASSWORD

CAPÍTULO 3

Aspectos de seguridad

3.1 Comunicación

DFVA es una aplicación RESTFull por lo que usa JSON como modelo de representación de objetos, los objetos son enviados vía POST sobre un canal de comunicación HTTPS. Las peticiones tienen el siguiente formato:

- **data_hash:** Suma hash de datos de tamaño máximo 130 caracteres, usando el algoritmo especificado
- **algorithm:** Algoritmo con que se construye data_hash, debe ser alguno de los siguientes: sha256, sha384, sha512
- **public_certificate:** Certificado de autenticación
- **data:** Datos de solicitud de autenticación encriptados usando AES con la llave de sesión encriptada con PKCS1_OAEP.
- **encrypt_method:** (opcional, por defecto: «aes_eax») Método de encripción del contenido («aes_eax», «aes-256-cfb»)

Además se envía un atributo identificador que depende del tipo de conversación que se quiera entablar, actualmente se pueden comunicar con DFVA: una institución (**institution**) con el uuid proporcionado en dfva y una persona (**person**) con su número de identificación (ver formato).

DFVA verifica que *data_hash* sea igual al generado a partir de data utilizando el algoritmo indicado.

Las respuestas tienen el siguiente formato:

- **data_hash:** Suma hash de datos de tamaño máximo 130 caracteres, usando el algoritmo especificado
- **algorithm:** Algoritmo con que se construye data_hash, debe ser alguno de los siguientes: sha256, sha384, sha512 *por defecto sha515*
- **data:** Datos de respuesta encriptados usando AES Modo EAX con la llave de sesión encriptada con PKCS1_OAEP.

DFVA intenta desencriptar lo suministrado en data utilizando los juegos de llaves explicados acontinuación:

3.1.1 Institución

Para manejar instituciones se debe crear una institución en la plataforma, al crearse se generará una llave privada, una llave pública y un certificado.

La llave privada corresponde a la llave privada de la institución y será utilizada para desencriptar las respuestas de dfva,

La llave pública corresponde a la llave pública de DFVA que se utilizará para encriptar la información que se envía en data.

En DFVA se construye dos pares de llaves por cada institución.

- **Llaves de la aplicación:** La llave privada se entrega y la llave pública se guarda y nunca es revelada.
- **Llaves del servicio:** La llave privada se guarda y nunca es revelada y la llave pública se entrega a la institución.

Los juegos de llaves encriptados y el certificado se guardan en la base de datos, excepto la llave privada de la institución, la cual nunca es almacenada en la báse de datos y solo se muestra una vez al usuario. Debido a que tanto las llaves privadas y las públicas son exclusivas de cada institución y generadas dentro de DFVA la llave pública de la institución nunca es expuesta asegurando aún más la comunicación RSA.

Nota: Se utiliza el SECRET_KEY de Django para generar una llave de encripción para los atributos almacenados en la base de datos, por lo que asegurece cambiar el valor por defecto.

3.1.2 Persona

Para la comunicación con personas se utiliza la encripción provista por los dispositivos PKCS11, de ahí se extrae:

- Un certificado de Autenticación.
- Un certificado de Firma.

En mis pruebas el certificado de firma tiene una llave privada que no permite desencriptar, por lo que solo se utiliza autenticación para comunicación.

Cuando una persona desea comunicarse con DFVA lo primero que reliza el programa es negociar el token de autenticación, el token es encriptado utilizando la llave pública del certificado de Autenticación en DFVA y enviado a la aplicación, dicho token solo puede ser desencriptado con el dispositivo PKCS11 y la llave privada de auteticación.

El token será utilizado como token de sesión en el algoritmo AES Modo EAX al encriptar data y deberá ser enviado firmado utilizando alguna de las llaves privadas del dispositivo (según contexto) y se comprobará en DFVA que el token sea correcto antes de proceder a desencriptar la información.

3.2 Almacenamiento

Nota: FALTA, Actualemente se guarda la información de verificación en forma plana en una base de datos y nunca se borra, autenticación y firma ya mueve a logs la información.

Advertencia: FALTA, Manejo información sobre los modelos de personas, la información descrita corresponde únicamente a institutiones.

DFVA se basa en Django por lo que soporta todas las bases de datos que este framework soporta, pero recomendamos Postgresql 9.4 o superior.

Las solicitudes de autenticación tiene un tiempo de vida de al menos 5 minutos (configurables), así una solicitud de autenticación se guardará en la base de datos por el tiempo definido, luego se extrae y se guarda en un archivo de log especiamente diseñado para guardar las solicitudes.

Las solicitudes de firma tienen un tiempo de vida de al menos 20 minutos (configurables), así una solicitud de firma se guardará por el tiempo definido y luego se guardará en un archivo de logs diseñado para guardar estas solicitudes. No se guarda el documento firmado, solo el registro de que se firmó y su suma hash.

Se pretende que las verificaciones de documentos y certificados y las revisiones de suscriptor conectado no se guarden en la base de datos (FALTA).

DFVA trabaja con tareas asincrónicas llamadas cada 5 y 20 minutos las cuales se encargan de revisar cuales peticiones han vencido el plazo y deben enviarse a logs, debido a este comportamiento algunas peticiones tendrán una duración entre 5-9 minutos para autenticación y 20 a 39 minutos para firma antes de ser enviadas a logs. Debido a que se provee la opción de solicitar la información de una petición para clientes no web esta información debe permanecer por un periodo de tiempo, además los mecanismos de control propios impedirá que una petición cuyo tiempo de vida haya caducado pueda ser obtenida por un cliente.

3.3 Bitácoras

DFVA usa al máximo el sistema de bitácoras de Django por lo que es ampliamente configurable. Actualmente genera los siguientes logs:

- **pyfva (debug, info, errors):** Bitácoras de comunicación con BCCR FVA
- **dfva (debug, info, errors):** Bitácoras de funcionamiento interno de DFVA.
- **dfva_authentication (info):** Bitácora de solicitudes de autenticación
- **dfva_sign (info):** Bitácora de solicitudes de firma.

Las bitácoras **dfva_authentication**, **dfva_sign** guardan los objecto en formato json. Actualmente la mayoría de los datos guardados son volcados a estas bitácoras.

3.4 Transporte

Se recomienda implementar HTTP Strict Transport Security (HSTS) en el sistemas en producción.

3.5 Disponibilidad

DFVA está basado en Django y utiliza todos los mecanismos provistos por este, así también posee todas las bondades en cuanto a escalabilidad. Por ello DFVA es escalable tanto Horizontal como Verticalmente.

Aunque AES EAX no es thread safe, solo se utiliza un hilo por encripción y abonando el hecho que Django es thread safe, se concidera que DFVA posee la capacidad de ejecutarse en entornos multi-hilo, con un pequeño impacto en los tiempos de encripción.

Nota: Más pruebas del comportamiento multihilo son recomendables.

3.6 Manejo de los certificados

DFVA es versatil y permite configurar el manejador de certificados, con ello permite comunicarse con la infraestructura de PKI que se desee.

Actualmente, se soporta la integración con Dogtag y también se soporta CA's creadas con Python Certbuilder para desarrollo utilizando el manejador **CA simple**

3.6.1 CA Simple

Advertencia: No utilizar en producción, este sistema está desarrollado para trabajar en desarrollo sin la necesidad de una infraestructura compleja.

DFVA trae un manejador y constructor de Autoridades de Certificados (CA), y debe construirse después de la instalación usando el comando:

```
python manage.py  crea_ca
```

Los de certificados, se generan en el directorio

```
internal_ca/
├── ca_cert.pem
└── ca_key.pem
└── crl.pem
```

Asegurar dichos archivos escapa al alcance de este documento, pero siempre es útil asegurar que solo el usuario tiene permiso para acceder a ellos.

```
chmod 600 ca_cert.pem  ca_key.pem crl.pem
```

3.6.2 DogTag

Nota: Para la instalación ver la sección de instalación de DFVA.

Dogtag es una aplicación que se integra con FreeIPA para proporcionar una robusta infraestructura PKI, actualmente el cliente desarrollado se integra con el API rest de Dogtag para generar, validar y revocar certificados.

Una aspecto importante de la implementación es que requiere que el usuario utilizado sea un agente de Dogtag capaz de solicitar y aprovar certificados, por lo que la aplicación será capaz de generar certificados en Dogtag de forma automática, debe contemparse esta situación dentro de la política de la PKI.

Dentro del proceso de solicitud de certificados se genera un objeto X509Request (certificate signing request csr) utilizando el esquema proporcionado en *DOGTAG_CERTIFICATE_SCHEME* exceptuando los campos OU y CN que corresponden a:

- OU = institution unit
- CN = domain

Ambos recolectados desde la interfaz.

En la validación del certificado se utiliza el *serial_number* del certificado para solicitar el estado del mismo en Dogtag. Además se valida el issuer sea identico al issuer de la instalación de Dogtag, así se garantiza que dicho certificado fue emitido por el issuer y que el serial_number es el adecuado.

3.7 Encripción

Se recomienda utilizar transporte https para la puesta en producción de esta plataforma, aún así DFVA posee una segunda capa de encripción, utilizando los algoritmos.

- **AES EAX:** Algoritmo simetrico, utilizado para encriptar el contenido, posee un token de sessión y un atributo IV (nonce), así como un Tag parameter. Este par debe ser único en cada encripción, osea no se puede repetir el IV con el mismo token de sessión. Actualmente tanto el token de sessión, el Tag parameter y el IV son de 16 bytes.
- **AES 256 CBF:** Algoritmo simetrico, utilizado para encriptar el contenido, posee un token de sessión y un atributo IV (nonce). Este par debe ser único en cada encripción, osea no se puede repetir el IV con el mismo token de sessión. Actualmente tanto el token de sessión es de 32 bytes y el IV son de 16 bytes. (usar solamente si EAX no está disponible en el lenguaje).
- **PKCS1 OAEP:** Algoritmo de encripción asimétrico, es utilizado para encriptar el token de sessión. También conocido como RSA/NONE/OAEPWithSHA1AndMGF1Padding en ambiente java.

Estructura de la encripción es: Token encriptado + IV (nonce) + datos encriptados.

Advertencia: en algunas implementaciones como en java se incluye dentro de los datos encriptados el IV al final, por lo que debe removerse y ponerse después del token encriptado.

CAPÍTULO 4

API de Comunicación

DFVA es una aplicación RESTFull por lo que usa JSON como modelo de representación de objetos, los objetos son enviados vía POST sobre un canal de comunicación HTTPS. Las peticiones tienen el siguiente formato:

- **data_hash:** Suma hash de datos de tamaño máximo 130 caracteres, usando el algoritmo especificado
- **algorithm:** Algoritmo con que se construye data_hash, debe ser alguno de los siguientes: sha256, sha384, sha512
- **public_certificate:** Certificado de autenticación
- **data:** Datos de solicitud de autenticación encriptados usando AES con la llave de sesión encriptada con PKCS1_OAEP.
- **encrypt_method:** (opcional, por defecto: «aes_eax») Método de encripción del contenido («aes_eax», «aes-256-cfb»)

Además se envía un atributo identificador que depende del tipo de conversación que se quiera entablar, actualmente se pueden comunicar con DFVA: una institución (**institution**) con el uuid proporcionado en dfva y una persona (**person**) con su número de identificación (ver formato).

DFVA verifica que *data_hash* sea igual al generado a partir de data utilizando el algoritmo indicado.

Las respuestas tienen el siguiente formato:

- **data_hash:** Suma hash de datos de tamaño máximo 130 caracteres, usando el algoritmo especificado
- **algorithm:** Algoritmo con que se construye data_hash, debe ser alguno de los siguientes: sha256, sha384, sha512 *por defecto sha515*
- **data:** Datos de respuesta encriptados usando AES Modo EAX con la llave de sesión encriptada con PKCS1_OAEP.

4.1 Instituciones

4.1.1 Autenticación

AuthenticateRequestViewSet.**institution**(*request*, **args*, ***kwargs*)

```
POST /authenticate/institution/
```

Solicita una petición de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **identification:** Identificación de la persona a autenticar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"
- **encrypt_method:** (opcional, default: «aes_eax») Método de encripción de segunda fase. («aes_eax», «aes-256-cfb»)

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **status:** Código de error de la transacción
- **status_text** Descripción para humanos del código de error
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification:** True si la autenticación ha sido procesada, False si está esperando al usuario
- **resume:** Resumen de la transacción.
- **hash_docsigned:** Normalmente Null, Base64 hash del documento firmado si el documento existe
- **hash_id_docsigned:** 0- no se ha firmado 1- Sha256, 2- Sha384, 3- Sha512

AuthenticateRequestViewSet.**institution_show**(*request*, **args*, ***kwargs*)

```
POST /authenticate/{id_transaction}/institution_show/
```

Solicita el estado de una petición de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

id_transaction Corresponde al id de la trasnacción del BCCR

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **status:** Código de error de la transacción
- **status_text** Descripción para humanos del código de error
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario
- **resume:** Resumen de la transacción.
- **hash_docsigned:** Base64 hash del documento firmado si el documento existe
- **hash_id_docsigned:** 0- no se ha firmado 1- Sha256, 2- Sha384, 3- Sha512

AuthenticateRequestViewSet.**institution_delete**(*request*, **args*, ***kwargs*)

```
POST /authenticate/{id_transaction}/institution_delete/
```

Elimina una petición de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

id_transaction Corresponde al id de la trasnacción del BCCR

Los valores devueltos son:

- **result** True/False si se eliminó la petición o no

4.1.2 Firma

SignRequestViewSet.**institution**(*request*, **args*, ***kwargs*)

```
POST /sign/institution/
```

Solicita una firma de un documento xml, odf o msoffice para un usuario

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **document:** Archivo en base64,

- **format:** tipo de archivo (xml_cofirma, xml_contrafirma, odf, msoffice, pdf),
- **algorithm_hash:** algoritmo usado para calcular hash,
- **document_hash:** hash del documento,
- **resumen:** Información de ayuda acerca del documento,
- **identification:** Identificación de la persona a autenticar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"
- **reason:** Razón de firma PDF (obligatorio solo en PDF)
- **place:** Lugar de firma en PDF (obligatorio solo en PDF)

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **sign_document:** Normalmente None, es un campo para almacenar el documento, pero no se garantiza que venga el documento firmado
- **status:** Código de error de la transacción
- **status_text** Descripción para humanos del código de error
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario
- **resume:** Resumen de la transacción.
- **hash_docsigned:** Normalmente Null, Base64 hash del documento firmado si el documento existe
- **hash_id_docsigned:** 0- no se ha firmado 1- Sha256, 2- Sha384, 3- Sha512

SignRequestViewSet.**institution_show**(*request*, **args*, ***kwargs*)

```
POST /sign/{id_transaction}/institution_show/
```

Verifica la firma dado un código y si respectiva identificación

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **identification:** Identificación de la persona a autenticar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

id_transaction Corresponde al id de la trasnacción del BCCR

Los valores devueltos son:

- **expiration_datetime:** hora final de validez

- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **sign_document:** Normalmente None, es un campo para almacenar el documento, pero no se garantiza que venga el documento firmado
- **status:** Código de error de la transacción
- **status_text** Descripción para humanos del código de error
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario
- **resume:** Resumen de la transacción.
- **hash_docsigned:** Normalmente Null, Base64 hash del documento firmado si el documento existe
- **hash_id_docsigned:** 0- no se ha firmado 1- Sha256, 2- Sha384, 3- Sha512

SignRequestViewSet.**institution_delete**(*request*, **args*, ***kwargs*)

```
POST /sign/{id_transaction}/institution_delete/
```

Elimina una petición de firma dado un código de transacción del BCCR

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

id_transaction Corresponde al id de la trasacción del BCCR

Los valores devueltos son:

- **result** True/False si se eliminó la petición o no

4.1.3 Validación

ValidateInstitutionViewSet.**institution_certificate**(*request*, **args*, ***kwargs*)

```
POST /validate/institution_certificate/
```

Solicita una de un certificado de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **document:** Archivo en base64 del certificado,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **identification:** Identificación del suscriptor
- **request_datetime:** Hora de recepción de la solicitud
- **code:** Código de identificación de la transacción (no es el mismo que el que se muestra en al usuario en firma)
- **status:** Estado de la solicitud
- **status_text:** Descripción en texto del estado
- **full_name:** Nombre completo del suscriptor
- **start_validity:** Inicio de la vigencia del certificado
- **end_validity:** Fin de la vigencia del certificado
- **was_successfully:** Si la verificación del certificado fue exitosa

Nota: Si la validación del certificado no fue exitosa, entonces los campos de identificación, full_name, start_validity, end_validity deben ignorase o son nulos.

`ValidateInstitutionViewSet.`**`institution_document`**(*request*, **args*, ***kwargs*)

```
POST /validate/institution_document/
```

Solicita una verificación de firma de un documento xml

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **document:** Archivo en base64 del certificado,
- **format:** Formato de documento a validar disponibles (cofirma, contrafirma, msoffice, odf)
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **identification:** Identificación del suscriptor
- **request_datetime:** Hora de recepción de la solicitud
- **code:** Código de identificación de la transacción (no es el mismo que el que se muestra en al usuario en firma)
- **status:** Estado de la solicitud
- **status_text:** Descripción en texto del estado
- **was_successfully:** Si la verificación del certificado fue exitosa
- **warnings:** Lista de advertencias
- **errors:** Lista de errores encontrados en el documento del tipo [{"code": "codigo","description": "descripción"}, ...]
- **signers:** Lista con la información de los firmantes

[{“identification_number”: “08-8888-8888”, “full_name”: “nombre del suscriptor”, “signature_date”: timezone.now()}, . . .]

Nota: Si la validación del documento no fue exitosa, entonces los campos de firmantes deben ignorase o son nulos.

ValidateSubscriptorInstitutionViewSet.**institution_suscriptor_connected**(*request*, **args*, ***kwargs*)

```
POST /validate/institution_suscriptor_connected/
```

Verifica si una persona está conectada (es contactable por el BCCR).

Los valores a suministrar en el parámetro data son:

- **institution:** uid de la institucion ver code en detalles de institución,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **identification:** Identificación de la persona a buscar,
- **request_datetime:** Hora de petición en formato “%Y-%m-%d %H:%M:%S”, osea “2006-10-25 14:30:59”

Data es un diccionario, osea un objeto de tipo clave -> valor

Retorna: **is_connected:** True si la persona está conectada, false si no lo está

4.2 Personas

4.2.1 Autenticación

AuthenticatePersonRequestViewSet.**person**(*request*, **args*, ***kwargs*)

```
POST /authenticate/person/
```

Solicita una petición de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **person:** identificación de la persona solicitante de autenticación,
- **identification:** Identificación de la persona a autenticar,
- **request_datetime:** Hora de petición en formato “%Y-%m-%d %H:%M:%S”, osea “2006-10-25 14:30:59”

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **status:** Código de error de la transacción
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario

- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario

AuthenticatePersonRequestViewSet.**person_show**(*request*, **args*, ***kwargs*)

```
POST /authenticate/{code}/person_show/
```

Solicita un estado de la solicitud de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **person:** identificación de la persona solicitante de autenticación,
- **identification:** Identificación de la persona a autenticar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **status:** Código de error de la transacción
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario

4.2.2 Firma

SignPersonRequestViewSet.**person**(*request*, **args*, ***kwargs*)

```
POST /sign/person/
```

Solicita una firma de un documento xml, odf o msoffice para un usuario

Los valores a suministrar en el parámetro data son:

- **person:** identificación de la persona solicitante de firma,
- **notification_url:** URL para la notificación (debe estar inscrita) o N/D si marca falso en not_webapp,
- **document:** Archivo en base64,
- **format:** tipo de archivo (xml_cofirma,xml_contrafirma, odf, msoffice),
- **algorithm_hash:** algoritmo usado para calcular hash,
- **document_hash:** hash del documento,
- **resumen:** Información de ayuda acerca del documento,
- **identification:** Identificación de la persona a firmar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **sign_document:** Normalmente None, es un campo para almacenar el documento, pero no se garantiza que venga el documento firmado
- **status:** Código de error de la transacción
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario

SignPersonRequestViewSet.**person_show**(*request*, **args*, ***kwargs*)

```
POST /sign/{code}/person_show/
```

Verifica la firma dado un código y su respectiva identificación

Los valores a suministrar en el parámetro data son:

- **person:** identificación de la persona solicitante de firma,
- **identification:** Identificación de la persona a autenticar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **expiration_datetime:** hora final de validez
- **request_datetime:** Hora de recepción de la solicitud
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **sign_document:** Normalmente None, es un campo para almacenar el documento, pero no se garantiza que venga el documento firmado
- **status:** Código de error de la transacción
- **identification:** Identificador del suscriptor
- **code:** Código para mostrar al usuario
- **received_notification** True si la autenticación ha sido procesada, False si está esperando al usuario

4.2.3 Validación

ValidatePersonViewSet.**person_certificate**(*request*, **args*, ***kwargs*)

```
POST /validate/person_certificate/
```

Solicita una de un certificado de autenticación para un usuario

Los valores a suministrar en el parámetro data son:

- **person:** Identificación de la persona validante,
- **document:** Archivo en base64 del certificado,

- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **identification:** Identificación del suscriptor
- **request_datetime:** Hora de recepción de la solicitud
- **code:** Código de identificación de la transacción (no es el mismo que el que se muestra en al usuario en firma)
- **status:** Estado de la solicitud
- **codigo_de_error:** Códigos de error del certificado, si existen
- **full_name:** Nombre completo del suscriptor
- **start_validity:** Inicio de la vigencia del certificado
- **end_validity:** Fin de la vigencia del certificado
- **was_successfully:** Si la verificación del certificado fue exitosa

Nota: Si la validación del certificado no fue exitosa, entonces los campos de identificación, nombre_completo, inicio_vigencia, fin_vigencia deben ignorase o son nulos.

`ValidatePersonViewSet.`**`person_document`**`(`*request*, **args*, ***kwargs*`)`

```
POST /validate/person_document/
```

Solicita una verificación de firma de un documento xml

Los valores a suministrar en el parámetro data son:

- **person:** Identificación de la persona validante,
- **document:** Archivo en base64 del certificado,
- **format:** Formato del documento a validar disponibles (cofirma, contrafirma, msoffice, odf)
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Los valores devueltos son:

- **identification:** Identificación del suscriptor
- **request_datetime:** Hora de recepción de la solicitud
- **code:** Código de identificación de la transacción (no es el mismo que el que se muestra en al usuario en firma)
- **status:** Estado de la solicitud
- **status_text:** Descripción en texto del estado
- **warnings:** Lista de advertencias
- **errors:** Lista de errores encontrados en el documento del tipo [{"codigo": "codigo","descripcion": "descripción"}, . . .]
- **signers:** Lista con la información de los firmantes [{"identification": "08-8888-8888", "full_name": "nombre del suscriptor", "signature_date": timezone.now()}, . . .]

- **was_successfully:** Si la verificación del certificado fue exitosa

Nota: Si la validación del documento no fue exitosa, entonces los campos de firmantes deben ignorase o son nulos.

`ValidateSubscriptorPersonViewSet.`**`person_suscriptor_connected`**(*request*, **args*, ***kwargs*)

```
POST /validate/person_suscriptor_connected/
```

Verifica si una persona está conectada (es contactable por el BCCR).

Los valores a suministrar en el parámetro data son:

- **person:** Identificación de la persona validante,
- **identification:** Identificación de la persona a buscar,
- **request_datetime:** Hora de petición en formato "%Y-%m-%d %H:%M:%S", osea "2006-10-25 14:30:59"

Data es un diccionario, osea un objeto de tipo clave -> valor

Retorna: **is_connected:** True si la persona está conectada, false si no lo está

4.2.4 Login

`PersonLoginView.`**`create`**(*request*, **args*, ***kwargs*)

Nota: Esta vista no está encriptada.

```
POST /login/
```

Permite a una persona autenticarse en DFVA, un token de sección es retornado y deberá ser usuado para encriptar la comunicación.

Los valores a suministrar son:

- **data_hash:** Suma hash de datos de tamaño máximo 130 caracteres, usando el algoritmo especificado
- **algorithm:** Algoritmo con que se construye data_hash, debe ser alguno de los siguientes: sha256, sha384, sha512
- **public_certificate:** Certificado de autenticación del dispositivo pkcs11
- **person:** Identificación de la persona,
- **code**: Identificación de la persona firmada con la llave privada del certificado de autenticación.

Los valores devueltos son:

- **identification**: Identificación del suscriptor
- **token**: Token de sección para encriptar atributo data posteriormente
- **expiration_datetime_token**: Hora máxima para usar el token
- **last_error_code**: Código de estado de la transacción
- **error_text**: Descripción de los errores encontrados

4.3 Notififcación para Instituciones

receptor.notify.**send_notification**(*data*, *serializer=None*, *request=None*, *encrypt_method='aes_eax'*)

Envia notificación a la institución, cuando se recibe una respuesta por parte del BCCR, este método reenvía la respuesta a la URL especificada en la petición.

La estructura de envío es:

Params id_transaction Id de transacción del BCCR

Params data Es un diccionario con los siguientes atributos

- **code:** Código de identificación de la transacción (no es el mismo que el que se muestra en al usuario en firma)
- **identification:** Identificador del suscriptor
- **id_transaction:** Id de trasnacción en el FVA del BCCR
- **request_datetime:** Hora de recepción de la solicitud
- **sign_document:** Almacena el documento, pero no se garantiza que venga el documento firmado, puede ser None
- **expiration_datetime:** Hora de recepción de la solicitud
- **received_notification:** True si la autenticación ha sido procesada, False si está esperando al usuario
- **duration:** tiempo que duró entre que fue enviada y fue recibida
- **status:** Código de error de la transacción
- **status_text:** Descripción en texto del estado

Params hashsum Suma hash realizada a data

Params algorithm Algoritmo con el que se realizó la suma hash

Por defecto se utiliza el método de encripción seleccionado al realizar la petición por parte de la institución, pero en caso de no lograrse identificar el método se utiliza por defecto "aes_eax"

CAPÍTULO 5

Flujos de interacción

5.1 Instituciones

5.1.1 Flujo de Autenticación

1. El usuario ingresa a la plataforma de la institución y se le despliega un modal donde se le pide la identificación
2. La identificación se envía a la plataforma de la institución
3. La aplicación mediante alguna de las bibliotecas de comunicación con dfva envía la petición utilizando un canal https vía JSON y encritpta en el atributo **data** la información, usando AES EAX + PKCS1_OEAP.
4. Dentro de DFVA se verifica que:
   - La institución está registrada y activa
   - La URL de notificación está registrada o la aplicación fue inscrita sin URL de notificación (para aplicaciones no web).
   - El certificado está vigente y es válido.
   - Los datos pueden ser desencriptados usando la llave privada de DFVA asignada a la aplicación.
   - La suma de verificación de los datos son iguales.

   En caso de que alguna de estas verificaciones falle DFVA devolverá un error.
5. La petición es almacenada en la base de datos y enviada mediante PyFVA al servicio de FVA del BCCR utilizando un canal SINPE.
6. Cuando se recibe la respuesta del FVA el código se guarda en la base de datos actualizando la información de la petición.
7. La respuesta se envía a la institución mediante el mismo canal usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. Se utiliza la llave pública de la aplicación para encriptar la comunicación (nunca publicada).

8. El código de identificación se le muestra al usuario y el usuario mediante el cliente de firma del BCCR firma la petición de autenticación.
9. El FVA del BCCR notifica a DFVA que el usuario ha firmado la autenticación, una vez que llega a DFVA se almacena en la base de datos.
10. Si la aplicación provee una URL de notificación se envía la notificación usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. Se utiliza la llave pública de la aplicación para encriptar la comunicación (nunca publicada).
11. Una vez la aplicación de la institución es notificada se autentica al usuario y se redirecciona al usuario a una vista autenticada.
12. Mientras tanto periódicamente se eliminan los datos de las peticiones vencidas guardando dentro de un archivo de log las peticiones eliminadas.

5.1.2 Flujo de estado de petición de Autenticación

El proceso de verificación inicia a partir del paso 11 del flujo de autenticación, osea a partir de la llegada del token a la aplicación de la institución.

11. La aplicación de la institución solicita el estado de la solicitud utilizando el id de transacción, para esto utilizará un canal https usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. La encripción se crea usando la llave publica del DFVA para la aplicación.
12. Se realizan las mismas verificaciones del paso 4 del proceso de autenticación y además se verifica que el id de transacción haya sido suministrado por DFVA. En caso de que alguna de estas verificaciones falle DFVA devolverá un error.
13. Se consulta en la Base de datos usando el id de transacción y se devuelve el resultado almacenado, mediante el mismo canal usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. Se utiliza la llave pública de la aplicación para encriptar la comunicación (nunca publicada).
14. En caso de que la información suministrada indique que el usuario está correctamente autenticado la aplicación redireccionará a una vista autenticada.

5.1.3 Flujo de Firma

1. El usuario navega dentro de la plataforma de la institución hasta que requiere alguna firma, en ese punto se le despliega un botón qien al hacer click envía una petición post y muestra un modal notificando al usuario que espere.
2. La plataforma de la institución es la encargada de saber que es lo que el usuario desea firma y debe enviarlo mediante alguna de las bibliotecas de comunicación con dfva utilizando un canal https vía JSON y encritpta en el atributo **data** la información, usando AES EAX + PKCS1_OEAP.

Dentro de DFVA se verifica que:

- La institución está registrada y activa
- La URL de notificación está registrada o la aplicación fue inscrita sin URL de notificación (para aplicaciones no web).
- El certificado está vigente y es válido.
- Los datos pueden ser desencriptados usando la llave privada de DFVA asignada a la aplicación.

- La suma de verificación de los datos son iguales.

En caso de que alguna de estas verificaciones falle DFVA devolverá un error.

3. La petición es almacenada en la base de datos y enviada mediante PyFVA al servicio de FVA del BCCR utilizando un canal SINPE.
4. Cuando se recibe la respuesta del FVA el código se guarda en la base de datos actualizando la información de la petición.
5. La respuesta se envía a la institución mediante el mismo canal usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. Se utiliza la llave pública de la aplicación para encriptar la comunicación (nunca publicada).
6. El código de identificación se le muestra al usuario y el usuario mediante el cliente de firma del BCCR firma la petición de firma.
7. El FVA del BCCR notifica a DFVA que el usuario ha firmado la autenticación, una vez que llega a DFVA se almacena en la base de datos.
8. Si la aplicación provee una URL de notificación se envía la notificación usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. Se utiliza la llave pública de la aplicación para encriptar la comunicación (nunca publicada).
9. Una vez la aplicación de la institución es notificada la aplicación realizará lo que necesite con el documento firmado.
10. Mientras tanto periódicamente se eliminan los datos de las peticiones vencidas guardando dentro de un archivo de log las peticiones eliminadas.

5.1.4 Flujo de estado de petición de firma

Este flujo es idéntico al flujo de chequeo del autenticación, la única diferencia corresponde a los datos entregados como respuesta, en los cuales se agrega el documento firmado si existe.

5.1.5 Flujo de verificación

1. El usuario interactua con la aplicación y por alguna razón la aplicación requiere verificar un certificado o un documento.
2. La aplicación mediante alguna de las bibliotecas de comunicación con dfva envía la petición utilizando un canal https vía JSON y encritpta en el atributo **data** la información, usando AES EAX + PKCS1_OEAP.
3. DFVA envía mediante PyFVA al servicio de FVA del BCCR utilizando un canal SINPE. **No se almacena en DB**
4. Se recibe la respuesta por parte de FVA del BCCR.
5. La respuesta se envía a la institución mediante el mismo canal usando JSON y encriptando el atributo **data** con AES EAX + PKCS1_OEAP. Se utiliza la llave pública de la aplicación para encriptar la comunicación (nunca publicada).

CAPÍTULO 6

Diagrama de clases

6.1 Paquetes

Existe una relación entre los paquetes que se describe a continuación

Institución

- `dot png` Serializadores
- `dot png` Vistas

Personas

- `dot png` Serializadores
- `dot png` Vistas

Manejador de certificados y Simple CA

- `dot png` Simple CA

6.2 Clases

DFVA se divide en 2 secciones; personas e instituciones, cada una de éstas se subdivide en 3 los cuales corresponde a

Institución

- `dot png` Modelos
- `dot png` Serializadores
- `dot png` Vistas

Personas

- `dot png` Modelos
- `dot png` Serializadores

- `dot` `png` Vistas

Manejador de certificados y Simple CA

- `dot` `png` Simple CA

CAPÍTULO 7

Riesgos identificados

Los riesgos aquí descritos responden a escenarios y comportamientos de DFVA que podrían llevar a un riesgo para la implementación, este lugar no es una lista de vulnerabilidades conocidas y todos los riesgos descritos han sido tratados y mitigados, pero en honor al trabajo realizado en este proyecto y en aras de demostrar que los desarrolladores de DFVA nos tomamos en serio la seguridad hacemos público aquellos riegos identificados que podrían poner en peligro la implementación de este software.

7.1 Almacenamiento

7.1.1 Almacenamiento de llaves y certificados

Las llaves privadas de los servicios se guardan en la base de datos, por ello todas llaves privadas usadas por los servicios (no las llaves privadas de las aplicaciones) son almacenadas en la base de datos.

Mitigación

Para mitigar este riesgos se realiza:

- Se Guardan encriptadas las llaves y certificados de la institución en la base de datos utilizando AES Modo EAX.
- Se utiliza *SECRET_KEY* de Django para generar una llave de encripción para AES.
- No se almacela la llave privada de la aplicación.
- Aseguramiento de la base de datos en producción

7.1.2 Almacenamiento de peticiones y respuestas de firma

Las peticiones al BCCR FVA deben esperar a la respuesta del BCCR que tiene como política “No nos llame, nosotros lo llamamos”. Por ello deben almacenarse en una base de datos. Por ello existe el riesgo que la información de las peticiones almacenadas en la base de datos puedan ser accedidas por extraño si la configuración del manejador de base de datos presenta alguna vulnerabilidad.

Además afecta el rendimiento de consulta de peticiones con respecto al número de peticiones por minuto en el sistema.

Mitigación

- Almacenar las peticiones por un tiempo fijo pero configurable, para autenticación por defecto se espera 5 minutos y para firma 20 minutos.
- Aseguramiento de la base de datos en producción
- Indexación de modelos de peticiones

7.1.3 Almacenamiento de bitácoras y auditoría

DFVA y PyFVA poseen un completo sistema de bitácoras, haciendo uso completo de Django logging emitiendo mensajes en los niveles Información, Debug, Warning, Error. Por defecto se guardan las bitácoras en disco provocando 2 afectaciones; 1) llenar el disco con logs, 2) fuga de información producto de los logs

Mitigación

- Configurar adecuadamente el sistema de logs, revisar Django logging y configurar un sistema de compresión y rotación de logs.

7.1.4 Almacenamientos de peticiones HTTP

Al firmar un documento se requiere que se envíen en el mensaje REST vía POST. Normalmente los servidores de contenido como Apache o Nginx almacenan las peticiones en memoria pero limitan el tamaño, y si se deseea aumentar el tamaño se utiliza el sistema de archivos. Así existe el riesgo de que alguien intente aprovechar esta particularidad para incrustar código malicioso e intentarlo ejecutar desde el sistema de archivos del sistema operativo.

Mitigación

- Debe mitigarse a nivel de implementación del servicio y configurando adecuadamente Nginx o Apache.

7.2 Serialización y deserialización de comunicaciones

7.2.1 Deserialización de Json

Toda la comunicación entre DFVA y las instituciones o personas se realiza utilizando JSON, por lo que alguien podría intentar insertar código malicioso para hacer romper los mecanismos de deserialización y con ello afectar el servicio.

Mitigación

- DFVA utiliza la última versión de Django Rest Framework y confía en la seguridad de este proyecto en el manejo de deserialización JSON.

7.2.2 Serialización de XML (Pyfva)

Un atacante podría intentar construir una petición maliciosa que rompa el serializador de XML para incrustar información que altere el comportamiento de la petición a nivel de BCCR FVA.

Mitigación

- Se confía en la biblioteca Soapfish para la elaboración de XML.
- La deserialización de XML se confía en que el único que envía XML es el BCCR.

7.3 Disponibilidad

7.3.1 Sustitución del servicio

DFVA es código abierto y libre, por lo que cualquiera puede configurar una instalación e intentar hacerse pasar por DFVA válido.

Mitigación

- Doble uso de llaves RSA y el cifrado de las comunicaciones ayuda a mitigarlo. La llave privada de la apliación solo la debería conocer la aplicación y la llave privada del servicio solo la conoce DFVA por lo que un sustituto no puede desencriptar las peticiones o enviar respuestas falsas.
- Cada institución debe crear mecanismos necesarios para resguardar las llaves RSA y los certificados.

7.3.2 Continuidad del servicio

DFVA no provee ninguna herramienta para prevenir ataques de denegación de servicio.

Mitigación

- DFVA está implementado para poder lanzarse en sistema distribuidos y balanceados, por lo que una buena infraestructura puede mitigar este tipo de ataques.
- DFVA No utiliza sesiones de estado, y cada petición se inicia y termina en una operación atómica.

Si los servicios del BCCR no están disponibles DFVA no funciona.

Mitigación

- DFVA siempre enviará un mensaje de error al cliente indicando que la transacción no se pudo realizar, por lo que el servicio de DFVA segurá disponible.

7.4 Integridad

7.4.1 Integridad de las peticiones

Algún atacante podría intentar un *Man in the Middle* para alterar el contenido de las peticiones y respuestas. Además una aplicación podría proponer hacer una petición por un canal no seguro.

Mitigación

- DFVA, podría ser implementado sobre HTTP sin perder la integridad y confidencialidad de los datos, gracias al doble mecanismo de encripción. Aun así se recomienda encarecidamente utilizar HTTPS en producción.

7.4.2 Integridad de las respuestas del BCCR FVA

Un atacante podría hacerse pasar por el BCCR FVA y emitir respuestas a peticiones para por ejemplo suplantar identidad o falsificación de documentos.

Mitigación

- DFVA posee una variable de configuración obligatoria llamada *ALLOWED_BCCR_IP* donde se especifican las direcciones IP de los servidores del BCCR que envían respuestas, cualquier IP que no esté listada en esta variable se le denegará el acceso.

7.5 Confidencialidad

7.5.1 Suplantación de identidad

- **Personas:**

Alguien podría intentar hacerse pasar por otra persona.

Mitigación

- Toda petición a DFVA es encriptada usando el mecanismo provisto por el dispositivo PKCS11 y verificado usando el certificado digital. Por lo tanto la autenticación de la persona es inequivoca.

- **Instituciones:**

Alguna institución enviar peticiones diciendo ser otra institución.

- Debido al doble sistema de encripción, para poder ser una institución valida se debe encriptar y desencriptar con las llaves RSA provistas, cualquier otra llave no importa si es válida para otra institución no lo será si el código de la institución es modificado.

CAPÍTULO 8

¿Cómo Contribuir?

8.1 Entorno de desarrollo

Para poder desarrollar DFVA core necesita tener:

- Lector PKCS11
- Firma digital de Costa Rica
- Simulador de FVA.
- Entorno instalado

Antes de proceder con cualquier cambio ejecute las pruebas en su entorno y asegurese que todas corran.

8.2 Testing

Debe crear las siguientes variables de entorno antes de correr una prueba.

```
export PKCS11_PIN=<PIN de desbloqueo de la tarjeta digital>
export PYTHONPATH=:$PYTHONPATH:<ruta del cliente DFVA de personas>/dfva_client
```

Ejecute las pruebas

```
python manage.py test
```

Nota: Es importante que el PIN sea correcto de lo contrario bloqueará la tarjeta y no podrá utilizarla más.

Advertencia: Asegurese de correr el simulador de FVA y que el la respuesta automática para el certificado sea exitosa con el número de cédula de la tarjeta.

Recuerde que una contribución con pruebas será mejor aceptada y contribuye a la estabilidad del sistema.

También puede hacer pruebas individuales ejecutando por ejemplo.

```
python manage.py test authenticator.tests.test_check_authenticator_institution.
↪CheckAuthenticatorInstitutionCase.test_authenticate_check
```

8.3 Reportar Issues

Utilize los issues de github para reportar cualquier problema encontrado en DFVA.

Por favor usar la sección de issues de cada cliente si el problema es del cliente y no del core de DFVA.

8.4 Pull Request

Todos son bienvenidos utilizando la sección de pull request de github.

CAPÍTULO 9

Índices y tablas

- genindex
- modindex
- search

Índice