Web Services Option Pack

nShield Web Services v3.3.1 User Guide

21 October 2024

© Entrust Corporation. All rights reserved.

Table of Contents

1. Introduction . . . 1
2. Prerequisites . . . 2
    2.1. Security World software . . . 2
    2.2. MongoDB . . . 3
3. Install the Web Services Option Pack . . . 4
    3.1. WSOP Server Installation . . . 4
4. Start the Web Services Option Pack . . . 7
5. Set up the WSOP TLS connection . . . 8
    5.1. Create Web Services server certificate and private key . . . 9
        5.1.1. Use an HSM-protected key for the Web Services server . . . 9
        5.1.2. Use a certificate signed by a CA for production deployments . . . 10
        5.1.3. Use a non-HSM-protected key for the Web Services server . . . 11
        5.1.4. Revocation of Web Services server's certificates and key . . . 13
    5.2. Create MongoDB database certificate . . . 13
    5.3. Create MongoDB database client certificate . . . 13
        5.3.1. Use the TLS certificate to authenticate clients when X.509 authentication is enabled . . . 14
6. Web Services client authentication . . . 15
    6.1. Client identification . . . 15
        6.1.1. Virtual Partitioning . . . 15
    6.2. Configure client authentication . . . 16
        6.2.1. Mutual authentication . . . 16
        6.2.2. Enable and disable client authentication . . . 16
    6.3. Client certificate revocation . . . 17
        6.3.1. Configure the CRL directory . . . 17
        6.3.2. CRLs for WSOP client certificates . . . 17
        6.3.3. General notes on CRLs . . . 17
7. Configure the Web Services Option Pack . . . 19
    7.1. Default web services configuration . . . 19
    7.2. Server listening host . . . 19
    7.3. Server listening port . . . 19
    7.4. WSOP TLS server configuration . . . 20
        7.4.1. WSOP TLS certificate . . . 20
        7.4.2. WSOP TLS private key . . . 20
        7.4.3. API Gateway . . . 21
        7.4.4. TLS server ciphersuites configuration . . . 22
    7.5. TLS client authentication . . . 22

7.6. Logging . . . 23
7.6.1. Corecrypto service logging . . . 23
7.6.2. Database driver logging level . . . 23
7.7. WSOP health endpoint . . . 24
7.8. Security World options . . . 25
7.8.1. HKNSO . . . 26
7.9. Request limiter options . . . 26
7.9.1. RSA key concurrent requests limit . . . 26
7.10. Caching options . . . 26
7.10.1. WSOP key material expiration . . . 26
7.10.2. Key cache capacity . . . 27
7.10.3. Key TTL period . . . 27
7.10.4. Group cache capacity . . . 27
7.10.5. Group TTL period . . . 27
7.10.6. Maximum number of active protection domains . . . 28
7.10.7. Domain TTL period . . . 28
7.11. Database configuration . . . 28
7.11.1. Hosts . . . 28
7.11.2. Logging level . . . 28
7.11.3. Timeout . . . 29
7.11.4. Maximum Listed Keys . . . 29
7.11.5. Database name . . . 29
7.11.6. Stand-alone Web Services Option Pack installation . . . 29
7.11.7. Virtual Partitioning database . . . 30
7.11.8. Virtual Partitioning collection . . . 30
7.11.9. Authentication method . . . 30
7.11.10. Disable TLS . . . 31
7.11.11. Certificate Authority (CA) files for TLS . . . 31
7.11.12. Database type . . . 31
7.11.13. MongoDB configuration . . . 32
7.12. Group options . . . 33
7.12.1. Allowing all group deletions . . . 33
8. Key Management with WSOP . . . 34
8.1. Protection domains . . . 34
8.1.1. "Well-Known" protection domain . . . 34
8.1.2. Consistent requests across multiple WSOP instances . . . 35
8.2. Key groups . . . 35
8.2.1. Management of multiple key groups . . . 35
8.3. Client segregation . . . 35

9. Database . . . 37
9.1. General architecture . . . 37
9.2. Database type . . . 37
9.3. MongoDB . . . 37
9.3.1. User roles . . . 38
9.3.2. Authentication methods . . . 39
10. Virtual Partitioning . . . 40
10.1. Introduction . . . 40
10.2. Corecrypto configuration file . . . 40
10.2.1. Virtual Partitioning database . . . 40
10.2.2. Virtual Partitioning collection . . . 40
10.3. Virtual Partitioning database mapping . . . 41
10.3.1. Issuer . . . 41
10.3.2. Subject . . . 41
10.3.3. Label . . . 42
10.3.4. Index . . . 42
10.3.5. Example of a valid mapping record . . . 43
10.3.6. Example of setting a mapping record using mongo shell . . . 43
10.4. Migration . . . 43
10.5. Logs and errors corecrypto usage . . . 43
10.5.1. Virtual Partitioning disabled . . . 43
10.5.2. Virtual Partitioning enabled . . . 44
10.5.3. Virtual Partitioning enabled and the client is not mapped . . . 44
10.6. Example script for Virtual Partitioning database generation . . . 44
11. WSOP Database Management Tool . . . 48
11.1. Introduction . . . 48
11.2. Installation . . . 48
11.3. Configuration file . . . 49
11.3.1. Database hostname . . . 49
11.3.2. Database port number . . . 49
11.3.3. Database name . . . 49
11.3.4. Database type . . . 50
11.3.5. MongoDB settings . . . 50
11.3.6. Disable TLS . . . 50
11.3.7. Authentication method . . . 50
11.3.8. Username and password file . . . 51
11.3.9. Authentication database . . . 51
11.3.10. Certificate Authority (CA) files for TLS . . . 51
11.3.11. Virtual Partitioning database and collection . . . 51

11.4. Environment variables . . . 52
11.5. Database initialization . . . 52
11.5.1. Examples . . . 53
11.6. Migration . . . 53
11.6.1. Examples . . . 54
11.7. DB check . . . 55
11.7.1. Examples . . . 56
11.8. Migration For Virtual Partitioning . . . 56
11.8.1. Prerequisites . . . 56
11.8.2. Example of successful migration . . . 57
11.8.3. Examples of migrations with setup errors . . . 57
11.9. Upgrade . . . 57
11.9.1. Examples . . . 58
11.10. Migrate schema . . . 58
11.10.1. Example . . . 59
11.11. PKCS #11 Object Migration . . . 59
12. REST API . . . 61
12.1. Common fields . . . 61
12.1.1. Key identifiers . . . 62
12.1.2. Key group identifiers . . . 62
12.1.3. Protection domain identifiers . . . 62
12.1.4. Protection domain types . . . 62
12.1.5. Plaintext, ciphertext, payloads and signatures . . . 62
12.1.6. Digest option . . . 63
12.1.7. IV and AAD parameters . . . 63
12.1.8. Algorithm identifiers . . . 63
12.1.9. Key type identifiers . . . 64
12.1.10. Elliptic Curve identifiers . . . 64
12.1.11. Protection domain activation option . . . 64
12.2. Examples . . . 64
12.2.1. List keys . . . 65
12.2.2. List keys in a group . . . 68
12.2.3. Public key export . . . 69
12.2.4. Public key import . . . 69
12.2.5. Create keys . . . 70
12.2.6. Delete keys . . . 70
12.2.7. List key groups . . . 71
12.2.8. Create key groups . . . 71
12.2.9. Delete key groups . . . 72

12.2.10. Create protection domains . . . 72
12.2.11. Delete protection domains . . . 73
12.2.12. List protection domains . . . 73
12.2.13. Activate protection domains . . . 74
12.2.14. Deactivate protection domains . . . 74
12.2.15. Sign payload . . . 74
12.2.16. Verify signature . . . 75
12.2.17. Sign a payload with a passphrase . . . 75
12.2.18. Encrypt plaintext . . . 76
12.2.19. Decrypt ciphertext . . . 76
12.2.20. Random Number Generation . . . 76
12.2.21. Health check . . . 77
12.3. Error handling . . . 78
12.4. Language bindings . . . 78
13. Security guidance for the Web Services Option Pack . . . 79
14. Base64url Encoding . . . 81
15. Uninstalling the Web Services Option Pack . . . 82
16. Upgrading the Web Services Option Pack . . . 83
16.1. Upgrading the Web Services Option Pack from v3.0 onwards . . . 83
16.2. Upgrading the Web Services Option Pack from releases before v3.0 . . . 83
17. Supported algorithms . . . 85
17.1. Key types . . . 85
17.2. Signing algorithms . . . 85
17.2.1. Product-specific signing algorithms . . . 86
17.3. Encryption algorithms . . . 86
18. Supported TLS cipher suites . . . 87

1. Introduction

The Web Services Option Pack (WSOP) provides users of Security World Software with the ability to remotely use their Security World keys via a REST (Representational State Transfer) API for signing, verification, encryption, and decryption. Only authenticated clients are permitted access to the service, providing assurance that your Security World keys remain usable only by clients that are permitted access.

WSOP is installed on top of your Security World Software, allowing you to use your existing applications and keys via the REST API, and to carry out remote management of keys, for example key creation. All Security World keys are protected by an nShield Hardware Security Module (HSM), ensuring the confidentiality and integrity of these keys is maintained.

2. Prerequisites

Before you install WSOP:

- Refer to the latest Release Notes at Release Notes for hardware and software compatibility, and known and fixed issues.
- Review Security guidance for the Web Services Option Pack. For additional security guidance, see also the Security Manual available with the Security World software.
- Check you have a supported version of Security World software installed, see Release Notes for details, and a working Security World configured. See Security World software.
- Check you have MongoDB (v7.0.6) running with a replica set configured. See MongoDB.
- If you already have a previous version of WSOP installed, this must first be removed before installing a new version. See Upgrading the Web Services Option Pack.

This release of WSOP is compatible with Linux only. See the Release Notes for the list of compatible operating systems and versions.

A WSOP server can only use a single HSM. High availability can be achieved by deploying multiple WSOP servers behind a third-party load balancer, with each WSOP server communicating with a single HSM.

2.1. Security World software

WSOP requires nShield Security World software to be installed, and a working Security World to be configured. To confirm that there is a usable Security World, use the `nfkminfo` command:

```
nfkminfo
```

If the Security World is usable then the state line in the `nfkminfo` output shows `Usable`.

```
World
    generation 2
    state 0x37270008 Initialised Usable Recovery !PINRecovery !ExistingClient RTC NVRAM FTO AlwaysUseStrongPrimes
!DisablePKCS1Padding !PpStrengthCheck !AuditLogging SEEDebug
...
```

For further information on installing Security World software and creating a Security World, see the *User Guide* shipped with your nShield Security World software.

2.2. MongoDB

WSOP has been tested against version 7.0.6.

Replica Set setup is required. For details about MongoDB replica set configuration, see the MongoDB documentation: Replica Set Configuration

Minimum number of replica set instances supported is 1.

Minimum action privileges required for corecrypto databases and collections are: `find`, `insert`, `remove`, `createIndex` and `update`.

For more information please refer to the MongoDB documentation https://www.mongodb.com/docs/.

3. Install the Web Services Option Pack

Before proceeding with the installation, a MongoDB database must already be running. You need the hostname, replica set name, and any database authentication details for the WSOP configuration. If a replica set is not in use, this option must be commented out.

1. Open a terminal window and create a temporary directory to unpack the WSOP tar to:

```
mkdir wsop_install
```

2. Extract the WSOP tar to the temporary directory created above:

```
sudo tar -xzf wsop-p11-X.X.X.tar.gz -C wsop_install
```

The WSOP release tarball contains the following installation archives:

Filename	Package
corecrypto.tar.gz wsop-common.tar.gz	WSOP Server
dbmt.tar.gz	Database Management Tool
nShield-WebServicesClient-Linux-1.1.0.tar.gz	PKCS #11 library and utilities for Linux
nShield-WebServicesClient-Windows-1.1.0.tar.gz	PKCS #11 library and utilities for Windows

To install the WSOP Server, see WSOP Server Installation for more information.

To install the Database Management tool, see WSOP Database Management Tool.

To install the PKCS #11 library, see the *nShield® Web Services PKCS #11 Provider User Guide*.

3.1. WSOP Server Installation

To install the Web Services Option Pack Server:

1. Change to the root directory.
2. Extract the following files from the unpacked WSOP tar. This installs all files required by the WSOP service to `/opt/nfast`.

```
sudo tar -xzf /path/to/wsop_install/corecrypto.tar.gz
```

```
sudo tar -xzf /path/to/wsop_install/wsop-common.tar.gz
```

3. Update relevant sections of `config.yaml` for your server and MongoDB environment.

   An example configuration is provided at

   ```
   /opt/nfast/webservices/corecrypto/conf/config.yaml.example
   ```

   If `config.yaml` is not found in the `webservices/corecrypto/conf` directory during the installation, then the `config.yaml.example` file will be copied over as the initial `config.yaml`.

   For assistance in configuring a new deployment, the Health Check configuration option `allow_unauthenticated_clients` can be disabled to allow unrestricted access to the health check endpoint.

   Entrust recommends using a secure connection to the database at all times. In the default configuration, TLS for database connection is on and the authentication method with database is set to `X509`.

   The following database fields must be configured correctly before starting the WSOP service:

   - `hosts` field specifies the addresses and ports of the database hosts:

     ```
     # List of database hosts
     hosts:
       - database1.ncipher.com:30001
       - database2.ncipher.com:30002
       - database3.ncipher.com:30003
     ```

   - `db_ca_file`, `db_cert_file` and `db_key_file` specify the Certificate Authority (CA) files for database TLS.

     ```
     # Path to the mongoDB TLS certificate
     db_ca_file: /opt/nfast/webservices/corecrypto/tls/db/db_ca.crt

     # Path to the corecrypto client certificate (used when Mutual Authentication is enabled)
     db_cert_file: /opt/nfast/webservices/corecrypto/tls/db/db_client.pem

     # Path to the corecrypto client private key (used when Mutual Authentication is enabled)
     db_key_file: /opt/nfast/webservices/corecrypto/tls/db/db_client.key
     ```

   - `replica_set` field specifies the name of the MongoDB replica set used for WSOP server. The default value must be changed to the correct replica set for your configuration.

```
        # Name of the Replication Set
        replica_set: rs1
```

4. Install the Database Management Tool (DBMT) and initialize the database. For details, see the WSOP Database Management Tool chapter.
5. The Web Services Option Pack service can now be installed using the command:

```
sudo /opt/nfast/webservices/sbin/install
```

The installer will create the following, as required:

- Either a SysV-style init script or systemd script for automatically starting and stopping the service.
- The `wsopd` user. This user is dedicated to running `corecrypto` service.

Restarting the service

To restart the WSOP service:

```
/opt/nfast/scripts/init.d/corecrypto restart
```

Each time the configuration file is changed, you must restart the WSOP service to take the new configuration into use.

4. Start the Web Services Option Pack

The installation process creates one service, `corecrypto`, which needs further configuration before starting. See Configure the Web Services Option Pack.

The service can be controlled using the scripts from the directory:

```
/opt/nfast/scripts/init.d/
```

WSOP service can be controlled using the following commands:

To start the service:

```
sudo /opt/nfast/scripts/init.d/corecrypto start
```

To stop the service:

```
sudo /opt/nfast/scripts/init.d/corecrypto stop
```

To restart the service:

```
sudo /opt/nfast/scripts/init.d/corecrypto restart
```

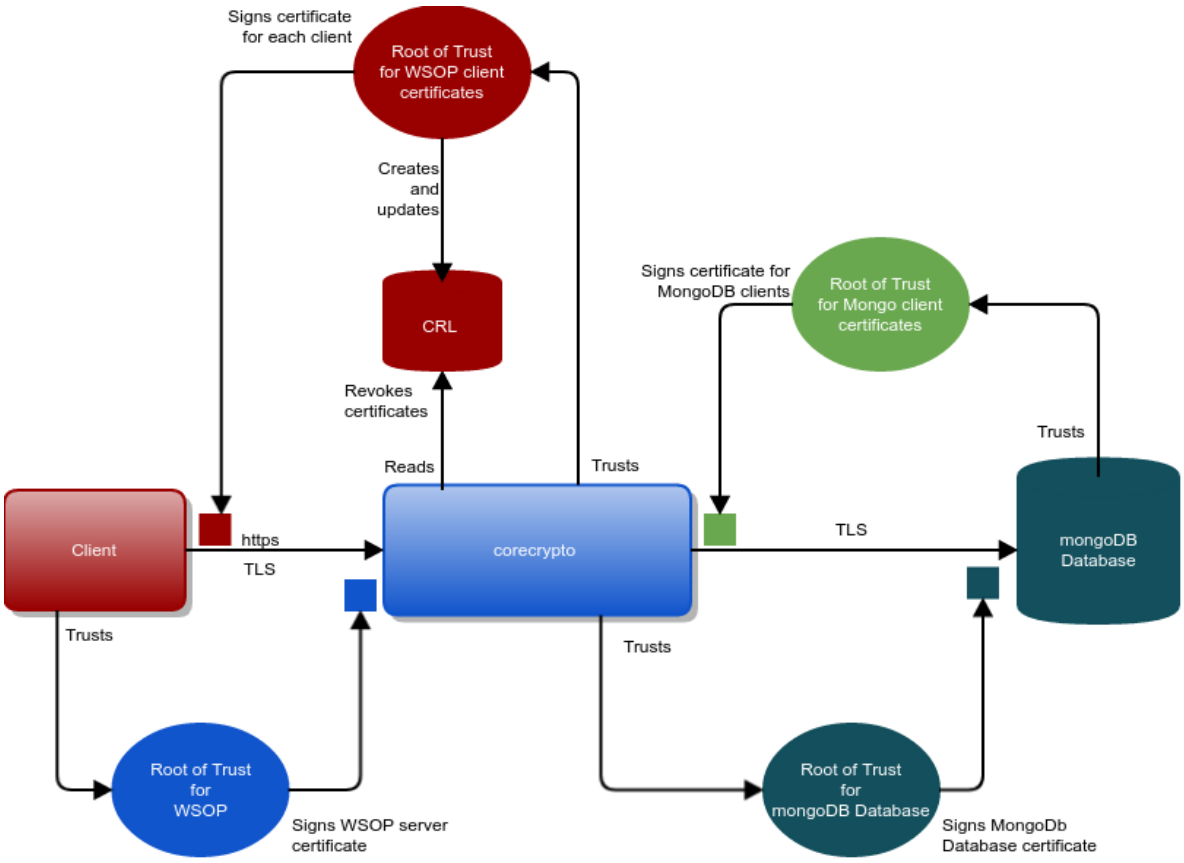
The installation scripts add the prefix `nc_` in front of corecrypto service name when used by the system and service manager. For example, you can restart WSOP using the commands:

```
sudo service nc_corecrypto restart
```

5. Set up the WSOP TLS connection

This page is for direct TLS authentication between a client and the WSOP server. If an API Gateway is being used it corresponds to the client in this information. For further API Gateway information see Client Authentication.

Certificate handling with WSOP:



The Certificate Authority (CA) that is trusted by clients to sign the WSOP corecrypto server certificate can be either an external commercial CA or an internal CA owned by the customer's organization and widely trusted within the organization.

Before deploying WSOP in a production environment the external interface between `corecrypto` (WSOP) and external clients must be properly configured.

The private TLS key for the external interface can be an HSM-protected Security World key, but the internal TLS connection will use private TLS keys that are stored in the file system.

To configure WSOP for production use, there are four steps to follow:

1. Create a server certificate (signed by a CA) for the external interface.
2. Create WSOP client certificates.

3. Create MongoDB Database Certificate.
4. Create MongoDB Database Client Certificate.

5.1. Create Web Services server certificate and private key

The two main options available for creating the Web Services server certificate for the external interface between `corecrypto` (WSOP) and WSOP clients and for protecting its associated private key:

- Using the HSM to create and store the Web Services server's private key. This is achieved by generating a key pair on the HSM, exporting the public key, and then generating a CSR to request the Web Services Server's certificate. The CSR can be sent either to an external CA or processed in your organization.
- Using WSOP's file system store the Web Services server's private key. This is achieved by generating a key pair, and then generating a Web Services server's certificate, signed by the private key of your own Web Services server root certificate. (OpenSSL can be used for both these operations). This means you will need to give the client read-access to a trusted copy of your Web Services server root certificate, so that the client can verify the Web Services server's certificate correctly.

Once you have created a private key and server certificate you will need to update the `corecrypto` configuration file located at `/opt/nfast/webservices/corecrypto/conf/config.yaml` to include information about the location of the certificate and the private key. If the default locations are not used, you must ensure that the access to these files is restricted to `wsopd` user.

It is strongly recommended that the Web Services server's private TLS key, which is used to secure communications with WSOP clients, is protected by the nShield HSM.

5.1.1. Use an HSM-protected key for the Web Services server

To generate a certificate with an HSM-protected key you should use the nShield OpenSSL that is shipped with nShield Security World Software. This version of OpenSSL is installed to `/opt/nfast/openssl/bin/openssl` by default.

Before proceeding you need to ensure that the nShield distributed openssl shared libraries in `/opt/nfast/openssl/lib` are loaded and available, and the nShield openssl engine is also available. To do this, run the following command:

```
/opt/nfast/openssl/bin/openssl engine -c -t nfkm
(nfkm) nShield NFKM engine
[RSA, rsaEncryption, id-ecPublicKey]
```

```
    [ available ]
```

For further information on the nShield OpenSSL NFKM engine consult the User Guide shipped with your nShield Security World software.

5.1.2. Use a certificate signed by a CA for production deployments

For production deployments, the Web Services server's TLS certificate should be signed by a Certificate Authority, either external or part of your own organization. To generate an HSM-protected certificate signed by a CA:

1. Generate the WSOP server's TLS private key:

   ```
   generatekey -q simple protect=module type=RSA size=2048 \
   ident=wsop-server-key plainname=wsop-server-key nvram=no pubexp=""
   ```

   WSOP only supports module-protected TLS private keys.

2. Generate the certificate signing request (CSR):

   ```
   /opt/nfast/openssl/bin/openssl req -engine nfkm -keyform engine \
   -key simple_wsop-server-key -new -subj \
   "/C=US/ST=CA/O=Acme, Inc./CN=example.com" -reqexts SAN \
   -config <(cat /opt/nfast/openssl/openssl.cnf <(printf \
   "\n[SAN]\nsubjectAltName=DNS:example.com,DNS:www.example.com"))
   ```

3. Submit the CSR to your CA to obtain a CA-signed certificate.
4. Import the signed certificate, with the full certificate chain into the WSOP external TLS folder.
5. Initialize the database using the Database Management Tool:

   ```
   dbmt db-init --config config.yaml
   ```

   For more information about the initialization of the database, see the Database Management Tool section of this User Guide.

6. Migrate the WSOP TLS key into the database using the Database Management Tool:

   ```
   dbmt migrate --config config.yaml --library /opt/nfast/bin/libcknfast.so -K simple wsop-server-key
   ```

   For more information about the migration of a key, see the Database Management Tool section of this User Guide.

7. Change the corecrypto configuration `/opt/nfast/webservices/corecrypto/conf/config.yaml` to use the new certificate and the generated HSM key. In the `external_tls` section of the config file:
   a. Set `key_appname_ident: simple:wsop-server-key`
   b. Set `cert_file: /opt/nfast/webservices/corecrypto/tls/external/server-corecrypto-cert.crt`
   c. Unset (comment out) `key_file`
8. Restart the `corecrypto` service for the change to take effect.

5.1.3. Use a non-HSM-protected key for the Web Services server

If it has been decided not to use the HSM to protect Web Services server's private key, this key must be stored in WSOP's file system. You will need to manually generate the server's private key and TLS server certificate file using tools such as OpenSSL (without using the OpenSSL nShield NFKM engine).

A script is provided with the WSOP installation to create a Web Services server's TLS root certificate and generate the server private key and certificates using OpenSSL. Alternatively, use the TLS root certificate of your own Web Services server, and generate the server TLS private key and certificate using other tools.

To use the provided script, first create two directories, one to store the generated TLS root certificate of your own Web Services server, and one to store the generated server private key and certificate. Then run the `mkcerts_externalcorecrypto.sh` script located in `/opt/nfast/wsop/scripts`.

```
mkdir -p ca_dir server_dir
/opt/nfast/webservices/scripts/mkcerts_externalcorecrypto.sh --cadir ca_dir \
--hostname <hostname> --serverdir server_dir <validity in days>
```

You can optionally replace `--cadir ca_dir` with `--deleteca` to delete the private key of the server's TLS root certificate, used to sign the server certificate, but if you do this you will not be able to generate a new server certificate with the same Web Services server's TLS root certificate. The consequence of this is, when your Web Services server's TLS certificate expires, a new Web Services server certificate hierarchy will need to be generated. It is therefore recommended to either use your own CA or save the generated server TLS root certificate's private key.

After execution the `server_dir` folder will contain two sets of server certificates available for use, one based on an RSA key, the other one using an ECDSA key.

The certificates and the keys parameters, like curve or size, can be altered by changing the

scripts or by generating the certificates directly.

5.1.3.1. Use a self-signed certificate for testing the setup

To test your setup, you can generate a temporary self-signed certificate and private key. This is not recommended for use in a production deployment.

1. Generate the WSOP TLS key:

```
generatekey -q simple protect=module type=RSA size=2048 \
ident=wsop-server-key plainname=wsop-server-key nvram=no pubexp=""
```

2. Generate the certificate:

```
/opt/nfast/openssl/bin/openssl req -engine nfkm -keyform engine \
-key simple_wsop-server-key -new -x509 \
-subj "/C=US/ST=CA/O=Acme, Inc./CN=example.com" -reqexts SAN \
-config <(cat /opt/nfast/openssl/openssl.cnf <(printf \
"\n[SAN]\nsubjectAltName=DNS:example.com,DNS:www.example.com")) \
-out server-corecrypto-cert.crt
```

3. Copy the file `server-corecrypto-cert.crt` to `/opt/nfast/webservices/core-crypto/tls/external`.
4. Initialize the database using the Database Management Tool:

```
dbmt db-init --config config.yaml
```

   For more information about the initialization of the database, see WSOP Database Management Tool.

5. Migrate the WSOP TLS key into the database using the Database Management Tool:

```
dbmt migrate --config config.yaml --library /opt/nfast/bin/libcknfast.so -K simple wsop-server-key
```

   For more information about the migration of a key, see the Database Management Tool section of this User Guide.

6. Change the `corecrypto` configuration `/opt/nfast/webservices/corecrypto/conf/config.yaml` to use the new certificate and the generated HSM key. In the `external_tls` section of the config file:
   a. Set `key_appname_ident: simple:wsop-server-key`.
   b. Set `cert_file: /opt/nfast/webservices/corecrypto/tls/external/server-core-crypto-cert.crt`.

c. Unset (comment out) `key_file`.

7. Restart the `corecrypto` service for the change to take effect.

5.1.4. Revocation of Web Services server's certificates and key

If any of the private keys for the Web Services server's certificate hierarchy are compromised, immediately replace the existing Web Services server's certificate hierarchy and keys with new values, using the procedures indicated in the previous sections.

5.2. Create MongoDB database certificate

MongoDB database TLS transport requires that the mongo cluster be configured to use TLS and the certificate provisioned. Refer to MongoDB documentation Configure mongod and mongos for TLS/SSL for a detailed presentation of how to configure TLS transport security.

The `corecrypto` service will need to be configured to use the TLS transport by setting the `disable_tls` field from `database` section to `false`.

The CA certificate needs to be made accessible to the `corecrypto` and its path needs to be configured, by setting the `db_ca_file` field from `database` section.

5.3. Create MongoDB database client certificate

`MongoDB` supports mutual TLS authentication as described in MongoDB documentation. In this case, the `corecrypto` service will authenticate to the `MongoDB` database using by using TLS client certificates.

For details of how the public certificates are generated and the required MongoDB configuration, refer to the MongoDB documentation.

After generating the client certificates, it will be necessary to configure the following fields in the `corecrypto` configuration file:

1. Set the client certificate path `db_cert_file` or copy the certificate file to its default location.
2. Set the client private key path `db_key_file` or copy the private key file to its default location.

File permissions must be set so that they can only be accessed by authorized users. This means that only the file owner has read permission to private TLS certificate keys.

If the MongoDB client certificate private key is compromised, refer to the steps in the MongoDB documentation.

Entrust recommends that the default ownerships and permissions that the installer sets should not be changed.

5.3.1. Use the TLS certificate to authenticate clients when X.509 authentication is enabled

MongoDB supports X.509 certificate authentication for use with a secure TLS connection. The X.509 client authentication allows clients to authenticate to servers with certificates rather than with a username and password.

Follow this tutorial as guidance of how to use X.509 for client authentication with a stand-alone mongod instance.

In this case the client certificates must have the following properties:

1. A single Certificate Authority (CA) must issue the certificates for both the client and the server.
2. Client certificates must contain the following fields:
   a. `keyUsage = digitalSignature`
   b. `extendedKeyUsage = clientAuth`
3. Each unique MongoDB user must have a unique certificate.
4. A client X.509 certificate's `subject`, which contains the Distinguished Name (`DN`), must differ from that of a member X.509 certificate, as presented in MongoDB documentation.
5. The X.509 certificate must not be expired.

Once the client certificates are generated and the `corecrypto` configuration file is set, ensure that:

1. WSOP server configuration file has the TLS transport security enabled, by setting the `disable_tls` field from `database` section to `false`.
2. The `auth_type` field from `database` section is set to `X509`.

6. Web Services client authentication

The Web Services REST API can be accessed only via HTTPS. By default, all endpoints of the REST API require client authentication. The health endpoint can be configured to not require client authentication – see WSOP Health Endpoint for more information.

If the WSOP server is not configured to operate behind an API Gateway, clients are identified and authorized via the client TLS certificate supplied with an HTTPS request.

If the WSOP server is configured to operate behind an API Gateway, clients can be identified and authorized via the client TLS certificate or the JSON Web Token (JWT) supplied with an HTTPS request. In both authorization methods, the client's TLS connection is terminated in the API Gateway. The `issuer` and `subject` fields are extracted by the API Gateway and passed on to the WSOP server as part of the request.

Third-party solutions exist to aid creation and management of client identity information. Alternatively (and only when using client TLS certificates), OpenSSL is able to create and manage certificates. It is shipped with most distributions, and also distributed as part of nShield Security World Software.

If using client certificates, Entrust strongly recommends that a root certificate and client certificate hierarchy are created specifically for WSOP.

Entrust strongly advises against using WSOP client authentication certificates or tokens in regular web browsers, since this may enable a Cross Site Request Forgery attack.

TLS certificates must contain Subject Alternative Name extension fields.

6.1. Client identification

Clients in WSOP are uniquely identified by the `issuer` and `subject` fields. These are provided either from the Web Services client TLS certificate or an API Gateway.

6.1.1. Virtual Partitioning

To enforce virtual partioning of clients, each distinct client must have a unique combination of `issuer` and `subject` fields. If this is not the case, a client that is not authorized to access keys in a protection domain will be able to access the keys if they share the same identity

as the authorizing client, as defined by the provided `issuer` and `subject` fields.

See Virtual Partitioning for more information.

6.2. Configure client authentication

Client authentication is configured by editing the `external_tls` section of the `corecrypto` configuration file located at `/opt/nfast/webservices/corecrypto/conf/config.yaml`.

6.2.1. Mutual authentication

If using client certificates, it is necessary for your REST clients to have individual certificates with an established chain of trust to the client root certificate to be able to authenticate with Web Services Option Pack.

To configure this chain of trust, set `ca_certificate_file` to the path of the certificate file for the Certificate Authority that issues the Web Services client certificates. For example, after installation this is set to:

```
ca_certificate_file: /opt/nfast/webservices/corecrypto/tls/external/corecrypto-external-ca-cert.pem
```

6.2.2. Enable and disable client authentication

Client authentication is enabled by default.

It is possible to disable client authentication (for example, for initial commissioning or testing purposes), and permit any client to perform operations using your Security World keys. This is achieved by changing the configuration file and restarting the Web Services Option Pack server.

Disabling client authentication permits the use of your Security World keys by any REST client that is able to access the machine that is hosting your Web Services Option Pack installation. While the key material is protected by the HSM, the use of the keys is not.

Client authentication should not be disabled in a production environment.

If not using an API Gateway, to disable client authentication, set `client_auth_enabled: false` To enable it, set `client_auth_enabled: true`

If using an API Gateway, client authentication is determined by the gateway's settings. In this context, `client_auth_enabled` instead controls whether authentication of the gateway

to WSOP is enabled and its setting is deployment dependent.

6.3. Client certificate revocation

6.3.1. Configure the CRL directory

To enable the revocation of client certificates you must configure a directory to store the revocations.

This is done by editing the value of `crl_directory` in the `external_tls` section of the configuration file `/opt/nfast/webservices/corecrypto/conf/config.yaml`.

For example:

```
crl_directory: /opt/nfast/webservices/corecrypto/tls/external/crls
```

If the `crl_directory` is not configured in the config file, then no revocation checking will happen.

Note that corecrypto consults its `config.yaml` only on start-up. Therefore, for any updates to the configuration file to take effect, the WSOP server must be restarted.

6.3.2. CRLs for WSOP client certificates

A CRL file in the CRL directory will be loaded only if its extension is `.crl`, `.pem`, or `.der` and it is signed by the private key of the root certificate for the revoked client TLS certificates. The `CRL sign` extension must be enabled for the signing CA. `corecrypto` checks the validity of the CRL that is revoking the client certificates by consulting the root certificate that was used for validating the client TLS certificates.

If the directory contains multiple CRL files, a client certificate is treated as revoked if it is listed in one or more valid CRL files. You do not need to restart the service after changing or adding to the contents of the CRL directory. The software will automatically use the new set of files.

6.3.3. General notes on CRLs

A Certificate Revocation List (CRL) file is signed by a trusted CRL issuer, typically the CA for the revoked certificates. The file either lists identifiers of the revoked certificates or denotes that none have been revoked. The file also holds other data, such as the time the

CRL was updated.

Tools such as OpenSSL may be used to maintain this list and create CRLs.

The specification for CRLs is in RFC-5280: https://tools.ietf.org/html/rfc5280#section-5.

7. Configure the Web Services Option Pack

Web Services Option Pack is configured by configuring the `corecrypto` service. The configuration file is `/opt/nfast/webservices/corecrypto/conf/config.yaml`. Use this file to set the WSOP external interface, corecrypto logging and database settings.

Entrust does not recommend changing the default permissions of the configuration file.

Only `wsopd` user shall be granted write access to the `/opt/nfast/webservices/corecrypto/conf/` folder and its content.

7.1. Default web services configuration

The service is installed using a default configuration. Entrust recommends reviewing and updating the initial configuration before the WSOP service is brought into service to ensure all configuration settings are appropriate for the deployment environment.

Special attention needs to be given to TLS connection and logging so that the system is used securely.

7.2. Server listening host

```
host: 0.0.0.0
```

This should be set to the interface that WSOP will listen for client requests on. It can be an IP address of the interface or a host name which must be available in the local system.

Location: The `external_tls` section of the corecrypto `config.yaml` file.

7.3. Server listening port

```
Port: 18001
```

Web Services Option Pack server runs as a low-privileged user and, in typical environments, is unable to bind to "privileged" ports (port numbers in the range of 1 to 1023). To use a low numbered port number, yet continue using a low-privileged user, a firewall can be configured to enable port forwarding. Consult the user guides of your internal/external firewall for further information.

Location: The `external_tls` section of the corecrypto `config.yaml` file.

7.4. WSOP TLS server configuration

This section contains details of Web Services Option Pack's TLS server certificate and private key. If using Security World to protect your private key, the named key will be corresponding to a Security World key.

All the settings refer to the `external_tls` section of the corecrypto `config.yaml` file.

7.4.1. WSOP TLS certificate

```
cert_file: /opt/nfast/webservices/corecrypto/tls/external/server-corecrypto-cert.pem
```

This contains the name of the file the server TLS certificate. If the user has a chain of certificates they must be packed together.

7.4.2. WSOP TLS private key

```
key_appname_ident: appname:ident
```

This directive contains the name of the Security World TLS server key. This is an HSM-protected key and Entrust recommends that you enable this option when possible, because it offers a higher level of security.

If it is decided that the WSOP server's TLS private key is not to be protected by an HSM, this option can be disabled by either deleting the line, or by commenting it out.

Entrust recommends the use of HSM-protected keys.

```
exclude_tls_key: true
```

If an HSM-protected key is used, when set to `true`, this option hides the TLS key from queries to prevent access or use of the key through WSOP. The default is `true`.

An alternative to the HSM-protected TLS key is to provide a file-based TLS private key. The confidentiality and integrity of this TLS private key file is protected by the Operating Systems access controls/permissions.

```
key_file: /opt/nfast/webservices/corecrypto/tls/external/server-corecrypto-privkey.pem
```

`key_file` and `key_appname_ident` directives are mutually exclusive. `key_file` specifies the location of the WSOP server TLS private key (when it is not protected by the HSM).

Entrust does not recommend changing the default permissions of the TLS private key files. The TLS private key file should be restricted such that only the `wsopd` user has read access.

Entrust recommends excluding the TLS private key files from any back-ups of the system and for the TLS private key to not be stored in any shared location.

7.4.3. API Gateway

```
api_gateway: false
```

Set to true to enable the Web Services Option Pack server to operate behind an API Gateway.

When `api_gateway` is set to `true`:

- The Web Services Option Pack server is enabled to operate behind an API Gateway.
- The API Gateway should authenticate the WS client and terminate the client's TLS con nection.
- On successful client authentication, the received client's identity information should be passed from the API Gateway to WSOP via a secure channel.
- If WS client authentication is disabled in the API Gateway, no client's identity informa-tion will be passed to WSOP.
- For more information please refer to Client Authentication.

```
headers:
    issuer_header:
    subject_header:
```

This option is only used when Web Services Option Pack is enabled to work behind an API Gateway. By default, the Web Services Option Pack will look into authorization headers passed by the gateway to identify its clients by `issuer` and `subject`. If Authorization head-ers are not provided, then webservices will identify a clients' `issuer` and `subject` values from the request headers defined in the `issuer_header` and `subject_header` configuration fields.

7.4.4. TLS server ciphersuites configuration

It is possible for the WSOP administrator to control the acceptable ciphersuites and the order the TLS handshakes negotiate them.

```
cipher_suites:

    - ECDHE-ECDSA-AES128-GCM-SHA256
    - ECDHE-RSA-AES128-GCM-SHA256
    - ECDHE-ECDSA-AES256-GCM-SHA384
    - ECDHE-RSA-AES256-GCM-SHA384

...
```

The `cipher_suites` directives control the order the ciphersuite is chosen during the TLS con nection initialization.

Entrust does not recommend adding weaker TLS ciphersuites as this will reduce the security provided by WSOP's TLS secure channel.

7.5. TLS client authentication

All the settings refer to the `external_tls` section of the corecrypto `config.yaml` file.

```
ca_certificate_file: /opt/nfast/webservices/corecrypto/tls/external/external-ca-cert.pem
crl_directory: /opt/nfast/webservices/corecrypto/tls/external/crls
```

If the WSOP server is authenticating the client using a client certificate, then the client certificate and authentication parameters are configured via the `external_tls` section in the `corecrypto` service config file. The `ca_certificate_file` contains the Web Services client root certificate, which forms the trust anchor for authenticating the client certificates received during TLS handshake. REST clients, as part of the same handshake, will need to present a client certificate that can be verified using this client root certificate.

The `crl_directory` contains the certificate revocation list for certificates that are no longer valid.

```
client_auth_enabled: true
```

If using an API Gateway, authorization of client requests is handled by the API Gateway. Set to `true`, to enable, or `false`, to disable authentication of the gateway to WSOP. If not using an API Gateway, set to `false` to disable client authentication. REST clients that are able to connect to your Web Services Option Pack server can use your Security World keys without any authentication or authorization.

Entrust recommends that client authentication is always enabled within a production environment.

7.6. Logging

`corecrypto` service allows setting of the log level separately for `server` and for `database driver` integration.

7.6.1. Corecrypto service logging

The corecrypto logging is controlled from `logging` section within `corecrypto` configuration file. `corecrypto` is able to output logs simultaneously to console, log file and syslog daemon.

`corecrypto` server logging is controlled by configuring `loglevel` field from the `logging` section:

```
loglevel: Warning
```

Available log levels are:

- `Trace`
- `Debug`
- `Info`
- `Warning`
- `Error`

This will set the logging level for the logger independent of the output method chosen.

Info, Debug, and Trace logging levels are very verbose — it is recommended that such logs are sent to an external syslog server with sufficient capacity to store the logs.

This parameter sets the logging level for all logging outputs.

7.6.2. Database driver logging level

`corecrypto` server logging is controlled by configuring `loglevel` field from `database driver` integration:

```
# Database Options
database:
  ...
    # The loglevel of the database driver component
```

```
    # Valid values: Trace, Debug, Info, Warning, Error
    loglevel: Info
```

The `loglevel` field specifies the logging level for the database driver. The database driver is a module of the WSOP server which handles the database connectivity and requests to the database.

You have the following logging options:

- Logging to console.

  By default, `corecrypto` service has console logging enabled. In order to disable this, update the following parameter in the console subsection:

  ```
  console:
      output: discard
  ```

- Logging to file.

  By default, `corecrypto` service outputs the logs to a file on the local file system. In order to configure set the `filepath` and the activation state in the `file` subsection:

  ```
  file:
      enabled: true
      filepath: /opt/nfast/log/corecrypto.log
  ```

- Logging to Syslog.

  ```
  syslog:
      enabled: false
      network: udp
      host: localhost:514
  ```

  `corecrypto` service logs can be directed to a syslog server. The configuration directives provide a means to activate the output to syslog, set the host, port, and type of transport. If the transport, set via parameter `network`, is not set then the logs will be sent to the local syslog daemon. The allowed `network` parameters values are `tcp` and `udp`.

7.7. WSOP health endpoint

The health endpoint checks the availability of the WSOP service.

This configuration resides in the `health` section of the corecrypto `config.yaml` file.

The estate health check configuration options relate to the modules belonging to the Secu-

rity World.

```
# Interval in seconds between estate health check
# Estate means the modules which belong of the security world
estate_check_interval: 5s

# Period in seconds after which the estate health check will timeout
estate_check_timeout: 30s
```

The Security World health check configuration value should be a multiple of the `estate_check_interval` as the Security World check will also execute an estate check.

```
# Interval in seconds between each security world check
# Its value is a multiple of the estate_check_interval value
# Note: each time a security world check takes place, an estate
# health check also takes place
sworld_check_interval: 300s
```

The database check can be set to different intervals and timeout values than the estate and Security World checks with the following options.

```
# Interval in seconds between each database health check
database_check_interval: 5s

# Period in seconds after which a database health check will timeout
database_check_timeout: 30s
```

Access to the health check endpoint can be controlled with the following configuration option.

```
# Allow unauthenticated clients to probe the health check endpoint.
# Only applicable when tls.client_auth_enabled is true
allow_unauthenticated_clients: false
```

The health check endpoint can be extended to also probe the availability of FIPS-authentica tion.

```
# Extend health check status to include whether the service can acquire FIPS-authentication
include_fips_ready_check: false
```

7.8. Security World options

This configuration resides in the `world` section of the corecrypto `config.yaml` file and relates to the Security World used by the WSOP deployment.

7.8.1. HKNSO

The hash of the nShield Security Officer key for the Security World in use. A valid and correct value must be present for successful WSOP installation and operation.

```
# Hash of the security world Officer
# It needs to match the world in the database and needs to be set before starting corecrypto.
hknso: 0000000000000000000000000000000000000000
```

7.9. Request limiter options

This configuration resides in the `request_limiter` section of the corecrypto `config.yaml` file.

7.9.1. RSA key concurrent requests limit

The following option sets the maximum number of concurrent RSA key creation requests allowed by the system. When this limit is exceeded, a `TooManyRequests` error is sent to the client.

```
# Maximum number of concurrent requests for creating RSA keys.
# Too many concurrent requests can lead to a denial of service.
rsa_key_concurrent_requests_limit: 10
```

7.10. Caching options

The following configuration options relating to cache behaviour reside in the `cache` section of the corecrypto `config.yaml` file.

7.10.1. WSOP key material expiration

In a deployment that uses Client Segregation, key material enabled by a client is loaded on the HSM in a client space that is separate from the key material of other clients. All key material that had been loaded into the client's space on the HSM is unloaded if the client hasn't performed any key management activities, for example encrypting or signing, for the period configured in key_manager_inactivity.

When you are changing this value, consider the number of unique clients that WSOP will serve and the amount of keys that each client will use.

```
# Period of inactivity in minutes after which a key manager will be closed
```

```
# Only applicable when tls.client_auth_enabled is true
# Setting of 0 is used to disable closing an inactive key manager.
key_manager_inactivity: 1440m
```

7.10.2. Key cache capacity

The `key_cache_capacity` field determines the total capacity of the cache. It is the maximum number of keys that can be stored in the cache before old keys are evicted.

```
# Capacity of the key cache. When the number of keys in the cache reaches this
# capacity, then the keys are evicted based on a least recently used (LRU) policy.
# Minimum value 100
key_cache_capacity: 30000
```

7.10.3. Key TTL period

The `key_TTL_period` field defines how long a key stays in the cache before it is evicted (regardless of key_cache_capacity).

```
# Period of time for which a key stays in the cache before it is evicted
# Minimum value 1m
key_TTL_period: 60m
```

7.10.4. Group cache capacity

The `group_cache_capacity` field defines the capacity of the cache of groups. It is the maximum number of groups that can be stored in the cache before old groups are evicted.

```
# Capacity of the group cache: when the number of groups in the cache reaches the
# capacity, then the groups are evicted based on least recently used (LRU) policy
# Minimum value 100
group_cache_capacity: 20000
```

7.10.5. Group TTL period

The `group_TTL_period` field is the time difference between a group being added to the cache and it being evicted regardless of cache capacity.

```
# Time to live value for a group in the cache
# Minimum value 1m
group_TTL_period: 120m
```

7.10.6. Maximum number of active protection domains

The `max_number_of_active_protection_domains` field defines the maximum number of protection domains stored in the cache. Once this limit is reached, any attempts to activate additional protection domains will return the `TooManyActiveDomains` error.

```
# Maximum number of protection domains which are stored in the corecrypto cache
# Minimum value 100
max_number_of_active_protection_domains: 10000
```

7.10.7. Domain TTL period

The `domain_TTL_period` field defines the time difference between a protection domain being added to the cache and it being evicted regardless of cache capacity. A protection domain is only evicted after it has been deleted or deactivated. Active protection domains are never evicted.

```
# Time to live value for protection domains in the cache
# Minimum value 1m
domain_TTL_period: 720m
```

7.11. Database configuration

This configuration resides in the `database` section of the corecrypto `config.yaml` file.

7.11.1. Hosts

`hosts` field specifies the addresses and ports of the database hosts:

```
# List of database hosts
hosts:
  - database1.ncipher.com:30001
  - database2.ncipher.com:30002
  - database3.ncipher.com:30003
```

7.11.2. Logging level

The `loglevel` field specifies the logging level for the database driver. The database driver is a module of the WSOP server which handles the database connectivity and requests to the database.

```
# The loglevel of the database driver component
# Valid values: Trace, Debug, Info, Warning, Error
```

```
    loglevel: Info
```

Possible log levels are:

- `Trace`
- `Debug`
- `Info`
- `Warning`
- `Error`

7.11.3. Timeout

Each database request from the WSOP server is executed with a timeout. If the request is not served in the time specified by the timeout parameter, then an error is returned. The `timeout` field from the `database` section specifies the respective timeout.

```
    # Time before a database request should fail
    timeout: 5s
```

7.11.4. Maximum Listed Keys

The `listkeys_max_limit` field specifies the maximum permitted listed keys at a time for the list keys endpoint, /km/v1/keys. This field is optional, however, if present and the queried limit or keys to be returned exceeds the configured limit, an error will be returned.

```
    # Maximum returned keys when listing, large queries can hurt the service.
    # If you wish to retrieve more than the maximum limit then you may make multiple API
    # requests and combine the results within your application using the offset and limit.
    listkeys_max_limit : 300000
```

7.11.5. Database name

The `db_name` field specifies the name of the database. This field is mandatory, meaning that the WSOP server will not start until a database name is provided in the corecrypto `config.yaml` file.

```
    # Database name
    db_name : nshield-corecrypto
```

7.11.6. Stand-alone Web Services Option Pack installation

The `is_WSOP_standalone` flag specifies whether the Web Services Option Pack shares the database with any other services.

```
# Disable this option when WSOP is running with KeySafe 5
is_WSOP_standalone: true
```

7.11.7. Virtual Partitioning database

The `segregation_db_name` field specifies the name of the Virtual Partitioning database.

```
segregations_db_name: segregation_db
```

7.11.8. Virtual Partitioning collection

The `segregations_collection_name` field specifies the collection inside the Virtual Partitioning database that contains the mappings of the virtual partitions.

```
segregations_collection_name: segregations
```

7.11.9. Authentication method

WSOP supports the following authentication mechanisms when connecting to the database:

- By username and password (SCRAM): The value of `auth_type` is set to `pwd`.
- X.509 certificate authentication: The value of `auth_type` is set to `tls`. For this type of authentication, the Certificate Authority (CA) files for TLS must be also provided in the configuration file.
- No authentication: The value of `auth_type` is set to `none`.

Entrust doesn't recommend setting the `auth_type` to `none` for production deployments.

For details regarding database authentication see Authentication Methods.

```
# Authentication method with database. Valid values: [none, pwd, tls]
# none - no authentication
# pwd  - username and password authentication using mongodb SCRAM
# tls  - x509 authentication
auth_type: tls
```

If the authentication is performed by username and password, then the `auth_username_file`

and auth_password_file fields specify the location of a secure file containing the username and passphrase to use for authentication.

```
auth_username_file: /opt/nfast/webservices/corecrypto/pwd-auth/config-username-auth
auth_password_file: /opt/nfast/webservices/corecrypto/pwd-auth/config-password-auth
```

The auth_source field specifies the name of the authentication database. The username and password provided by the file name specified by the auth_username_file and auth_password_file fields are verified against the username and password stored in the authentication database.

```
auth_source: userdb
```

7.11.10. Disable TLS

The disable_tls field from database section specifies if TLS for database connection is on or off as follows: value true means that TLS is not used while value false means that TLS is used.

```
# Transport Layer Security. Default is false.
disable_tls: false
```

7.11.11. Certificate Authority (CA) files for TLS

When TLS option is used as method of authentication, then db_ca_file, db_cert_file and db_key_file specify the Certificate Authority (CA) files for TLS.

```
# Path to the mongoDB TLS certificate
db_ca_file: /opt/nfast/webservices/corecrypto/tls/db/db_ca.crt

# Path to the corecrypto client certificate (used when Mutual Authentication is enabled)
db_cert_file: /opt/nfast/webservices/corecrypto/tls/db/db_client.pem

# Path to the corecrypto client private key (used when Mutual Authentication is enabled)
db_key_file: /opt/nfast/webservices/corecrypto/tls/db/db_client.key
```

7.11.12. Database type

The db field from database section specifies the type of the database management system. Currently the only supported value is mongodb.

```
# Type of database. Supported values: mongodb
db: mongodb
```

7.11.13. MongoDB configuration

MongoDB configuration is specified in the `mongodb` section of the corecrypto `config.yaml` file:

- Replication set name:

  The `replica_set` field specifies the name of the MongoDB replica set used for WSOP server. The default value must be changed to the correct replica set for your configuration.

```
# Name of the Replication Set
replica_set: rs1
```

- Connection timeout:

  The `connect_timeout` specifies the number of seconds WSOP server will wait before the database connection attempt is aborted.

```
# Timeout for connection to the database server
connect_timeout: 5s
```

- Selection timeout:

  The `selection_timeout` specifies the number of seconds the WSOP server will wait to select a server for a database operation before giving up and raising an error.

```
# Timeout for selecting a connection from the pool
selection_timeout: 5s
```

- Read/write timeout:

  `socket_timeout` specifies the number of seconds that a read or a write operation from or to the database can take before the operation is timed out. This setting is a socket specific setting.

```
# Timeout waiting for read/write in the socket
socket_timeout: 5s
```

- Minimum and maximum pool size:

  The `min_pool_size` and `max_pool_size` specify the minimum and maximum number of connections, respectively, of the connection pool.

```
# Minimum and maximum connections to use in mongodb's connection pool
min_pool_size: 1
```

```
        max_pool_size: 100
```

7.12. Group options

This configuration resides in the `group` section of the corecrypto `config.yaml` file.

7.12.1. Allowing all group deletions

When `allow_all_group_deletions` is set to `true`, any key group within the database can be deleted. When set to `false`, users cannot delete any key groups within the `Module` or `Well-Known` protection domains that have the same name as their parent protection domain.

```
    # Allow all group deletions.
    # Disable this option to prevent the deletion of the reserved groups belonging
    # to either the Module Protection or Well-Known Key Protection domains.
    # A reserved group is defined as any group which has the same name as
    # the protection domain it belongs to.
    allow_all_group_deletions: true
```

8. Key Management with WSOP

Security World keys in the Web Services Option Pack environment are managed by using the Web Services Option Pack REST API. Non-confidential key data which exists and can be accessed outside the HSM is stored in a database management system. For details about the key storage in the database management system, see the Database chapter.

Security World keys are identified using an `appname` and an `ident`. The appnames supported by Web Services Option Pack are `simple`, `wspkcs11` and `pkcs11`. `simple` keys can be generated and used through the WSOP API. `simple` keys can also be migrated into WSOP if the algorithm is supported. PKCS #11 keys can be generated and used through the WSOP PKCS #11 library. PKCS #11 keys can also be migrated into WSOP if the algorithm is supported by the WSOP PKCS #11 library. `wspkcs11` designates keys which have been generated through the WSOP PKCS #11 library and `pkcs11` are legacy PKCS #11 keys which have been migrated into WSOP. The `ident` can only contain digits and lowercase letters. It cannot contain spaces, underscores (`_`), or hyphens (`-`).

8.1. Protection domains

Web Services Option Pack provides support for Softcards through the concept of protection domains. A single protection domain is available for each Softcard, in addition to a "Module" protection domain which is the set of all HSM-protected keys.

A protection domain can be "activated" and "deactivated", which is equivalent to loading and unloading a Softcard.

The "Module" protection domain is always activated and cannot be deactivated.

Once a protection domain is activated all key groups that are protected by the active domains are able to load and create keys for use, until that protection domain is deactivated.

A protection domain does not need to be activated to load the public half of an asymmetric key or delete a key.

8.1.1. "Well-Known" protection domain

The "Well-Known" protection domain can only be used for public key import, key generation is not allowed. It exhibits the same properties as the "Module" protection domain in that it is created by default, is always active, and it cannot be deactivated.

8.1.2. Consistent requests across multiple WSOP instances

WSOP supports consistent requests across multiple instances. This means that attempts to activate the Softcard protection domain only occur if they are necessary as part of cryptographic or key creation requests. It provides the ability to load balance multiple instances of WSOP, which improves performance and horizontal scaling. It also eliminates stateful behaviour, facilitating the implementation of cloud-friendly architects.

8.2. Key groups

Key groups hold a collection of keys that are linked by a single common protection domain. All Security World keys belong to exactly one key group and each protection domain can have multiple key groups. A key group is created automatically for each available protection domain. Key group names are non-unique.

8.2.1. Management of multiple key groups

Key group management provides the ability to create and delete groups within protection domains.

You can assign protection domains and groups to virtual partitions to control which users can see and use them, see Virtual Partitioning. If you assign a protection domain to a virtual partition, all of its groups and their keys inherit that virtual partition. If a group has not inherited a virtual partition from its protection domain, you can assign one to the group and it will be inherited by the group's keys.

In a public protection domain (one that does not require a passphrase), any user can see and use the groups and their keys. To restrict access to a group in a public protection domain, you can assign that group to a virtual partition. This means that even though the protection domain is public, access to the group and its keys depends on the virtual partition settings.

8.3. Client segregation

The Web Services Option Pack (WSOP) introduces the concept of WSOP client segregation, making it possible for the keys protected by Softcards to be used only after each client presents the authentication passphrase (associated with the particular protection domain).

WSOP provides client segregation at the protection domain level only when TLS Client

Authentication is enabled. Individual clients are identified by the unique combination of the client `issuer` and `subject` fields. Each WSOP client has to activate individually the protection domains they want to use.

As the activated state of a protection domain is specific to a WSOP client, any commands that require the protection domain to be activated will only be successful if the client identity information can be successfully validated, and this identity information contains the correct `issuer` and `subject` fields.

When client segregation is in operation, any key or token handles loaded in the HSM will be loaded in a separate client space to other Web Services Option Pack clients and nShield applications.

Client segregation does not restrict the visibility of keys to clients. They can still be listed and deleted regardless of the protection domain activated state. To restrict the visibility of keys to particular clients see Virtual Partitioning.

The access and usage of the public keys are not restricted by the client segregation, therefore any client is able to use the public part of an asymmetric key.

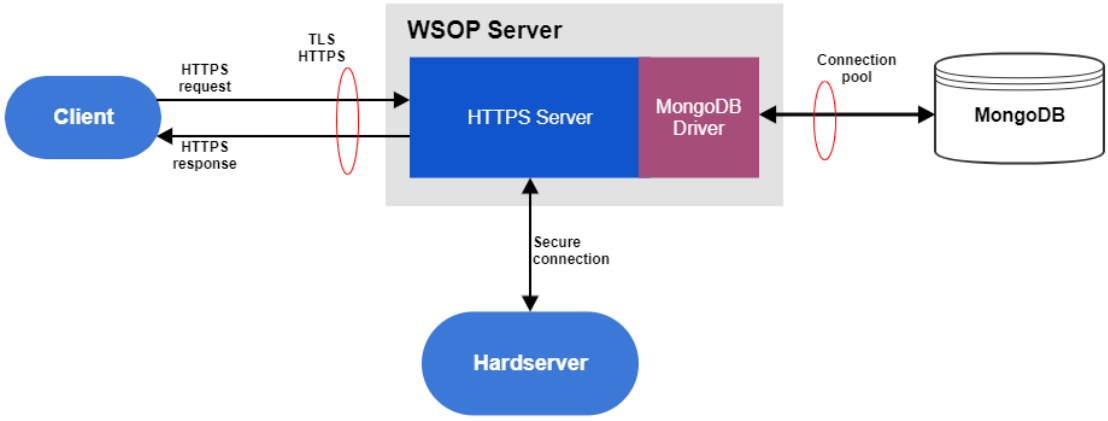
9. Database

9.1. General architecture

This chapter provides information about key storage. Security World keys and token data in the Web Services Option Pack environment are stored in a standalone database management system. The respective data is non-confidential data which exists and can be accessed outside the HSM.

The WSOP server accesses the database management system by using a database driver. The database driver depends on the database management system implementation.

The figure below is a high-level picture of the WSOP server and the database management system in which the database management system is implemented with MongoDB. The MongoDB driver allows the access of the WSOP server to the MongoDB database.



Details about key properties and key management are provided in the Key Management with WSOP chapter. WSOP database configuration options are explained in the Database Configuration subchapter.

9.2. Database type

Each Security World has one database management system. The database type is specified in the WSOP server configuration file by the `db` field. The name of the database and other database configuration options are also provided in the WSOP server configuration file.

9.3. MongoDB

When `db` field from WSOP configuration file type is set to `mongodb`, then database management system is implemented by using MongoDB. MongoDB configuration options are also provided in the WSOP server configuration file under the `mongodb` section.

MongoDB stores Security World keys and token data in the following collections:

- Keys
- Groups
- Domains
- Softcards

9.3.1. User roles

MongoDB has the notion of roles, where each role has a defined set of allowed actions. A user of a MongoDB database can be given a role which then determines what the user can and cannot do to the data.

MongoDB comes with three built-in roles, namely cluster-administrator, database-administrator, and a database-user.

Give the WSOP server the database-user role. The WSOP server should only have the privileges necessary for it to function as a user of the MongoDB database.

To operate MongoDB, the WSOP server needs the following actions:

- find
- insert
- remove
- createIndex
- update

The MongoDB administrator will configure the MongoDB database with the following actions and privileges for WSOP data-user role:

```
dbname = nshield-corecrypto
actions = ["find", "insert", "remove", "createIndex", "update"]
privileges=[
    {"resource" : {"db" :  dbname, "collection" : ""}, "actions" : actions},
    ]
```

For details about MongoDB roles, see the MongoDB documentation: Role-based access control

9.3.2. Authentication methods

The WSOP MongoDB deployment supports the following authentication mechanisms to verify the identity of the WSOP server:

- none
- pwd
- tls

The type of authentication is specified by `auth_type` option in the WSOP server configuration file in the database section using one of the following values: `none`, `pwd`, `tls`.

none

No authentication. This option is used during development. It should not be used during production.

pwd

Using username and password files for Salted Challenge Response Authentication Mechanism (SCRAM), MongoDB verifies the supplied WSOP server credentials against the MongoDB's username, password, and authentication database. The `auth_username_file` and `auth_password_file` options define the location of a secure file containing the username and passphrase to use for authentication. The value of `auth_source` option specifies the name of MongoDB's authentication database.

tls

WSOP server can use tls X.509 certificates instead of usernames and passwords to authenticate to the MongoDB database. In this configuration the value of the `auth_type` option in the WSOP server configuration file is set to `X509`. Also, in the same configuration file, the TLS must be enabled by setting `disable_tls` to `false`.

The Certificate Authority (CA) files to use for TLS are specified by the following options: `db_ca_file`, `db_cert_file` and `db_key_file`.

10. Virtual Partitioning

10.1. Introduction

Virtual Partitioning allows a WSOP administrator to manage the access to database records by controlling their visibility on a client-by-client basis.

A client is identified with the `issuer` and `subject` fields, which are mapped to a virtual partition label. The virtual partition label is used to filter the client's view of the database. Multiple clients can be assigned to share the same virtual partition label. Records can also be assigned to a common label that is visible to all users.

See Client Authentication for more information about client identification.

10.2. Corecrypto configuration file

Virtual Partitioning database and the mapping collection need to specified be in the corecrypto `config.yaml`.

10.2.1. Virtual Partitioning database

The `segregation_db_name` field specifies the name of Virtual Partitioning database.

```
# WSOP Corecrypto Segregation (WCS) Options
# The segregation database and collection must be defined in order
# to enable WCS. When enabled, corecrypto objects
# will be segregated based on the mappings defined in the collection.
# If the segregation database and collection are not defined,
# WCS is disabled.
#
segregations_db_name: segregation_db
```

10.2.2. Virtual Partitioning collection

Inside the Virtual Partitioning database, the `segregations_collection_name` field specifies the collection that contains the mappings of the virtual partitions.

```
# WSOP Corecrypto Segregation (WCS) Options
# The segregation database and collection must be defined in order
# to enable WCS. When enabled, corecrypto objects
# will be segregated based on the mappings defined in the collection.
# If the segregation database and collection are not defined,
# WCS is disabled.
#
segregations_db_name: segregation_db
```

```
    segregations_collection_name: segregations
```

Virtual Partitioning is enabled once both fields, `segregation_db_name` and `segregations_collection_name`, are defined. If one of the fields is commented out or undefined, Virtual Partitioning is disabled.

10.3. Virtual Partitioning database mapping

The mapping inside Virtual Partitioning collection should contain the following fields:

- `ISSUER`
- `SUBJECT`
- `label`

The mappings will be created by the database administrator as WSOP's rights on Virtual Par titioning database are `readonly`.

10.3.1. Issuer

The `"ISSUER"` field contains relative distinguished names of the X.509 certificate issuer for the client that is mapped to a specific virtual partition.

When `api_gateway` is set to true, the `"ISSUER"` field needs to match the format in which the X.509 certificate issuer is passed by the gateway in the headers.

10.3.2. Subject

The `"SUBJECT"` field contains relative distinguished names of the X.509 certificate subject for the client that is mapped to a specific virtual partition.

When `api_gateway` is set to true, the `"SUBJECT"` field needs to match the format in which the X.509 certificate subject is passed by the gateway in the headers.

10.3.2.1. Relative distinguished names

Only the following relative distinguished names should be used in the `"ISSUER"` and `"SUBJECT"` fields:

- `CN` for common name
- `OU` for organizational unit
- `O` for organization name
- `PC` for postal code / zip code
- `STREET` for street / first line of address
- `L` for locality name
- `ST (or SP or S)` for state or province name
- `C` for country

The order of these names matters. They should be listed in the order defined above. If a name is missing, it should not be listed in the `"ISSUER"` and `"SUBJECT"` fields.

10.3.3. Label

The `"label"` field contains the virtual partition name of the client. If the `"label"` field does not exist or is empty `"label":""`, the virtual partition will be set to common.

The common virtual partition can be accessed and modified by all clients that are mapped in Virtual Partitioning collection.

If the client in use does not have a mapping in Virtual Partitioning database, all operations performed by that client will give a 403:Forbidden error.

```
{
  "error": "Forbidden",
  "message": "unknown issuer and subject in segregation database",
  "path": "/km/v1/groups",
  "status": "403",
  "timestamp": "2021-12-15T16:48:21Z"
}
```

A client should not have the X.509's issuer and subject mapped to more the one label in the database.

10.3.4. Index

Issuer and subject index must be set as case insensitive and unique.

An example using mongo shell for virtual partitioning database set to "segregation_db" and collection set to "segregations":

```
use segregation_db

db.createCollection("segregations", { collation: { locale: 'en', strength: 2 } } )

db.segregations.createIndex({ "issuer": 1, "subject": 1 }, { unique: true } )
```

locale must be set to the correct language: https://docs.mongodb.com/manual/reference/collation-locales-defaults/

10.3.5. Example of a valid mapping record

An example of a valid mapping record:

```
{
  "_id": {
    "$oid": "documentID"
  },
  "label": "VirtualPartitionLabel",
  "subject": "CN=CommonName,OU=OrganizationalUnit,O=Organization,C=Country",
  "issuer": "CN=CommonName,OU=OrganizationalUnit,O=Organization,L=Locality,C=Country"
}
```

10.3.6. Example of setting a mapping record using mongo shell

```
use segregation_db

db.segregations.insert({
  "label": "VirtualPartitionLabel",
  "subject": "CN=CommonName,OU=OrganizationalUnit,O=Organization,C=Country",
  "issuer": "CN=CommonName,OU=OrganizationalUnit,O=Organization,L=Locality,C=Country"
})
```

10.4. Migration

Virtual Partitioning is supported by the Database Management Tool. For more information about the migration of a partitioned RFS, see the Database Management Tool section of this User Guide.

10.5. Logs and errors corecrypto usage

10.5.1. Virtual Partitioning disabled

This is an example output where at least one of the fields, `segregation_db_name` and `segregations_collection_name`, is commented out or undefined, Virtual Partitioning is disabled.

```
[INFO] [DBADAPTER] [1129] TLS CA key loaded
[INFO] [DBADAPTER] [1129] TLS client keys loaded
[INFO] [DBADAPTER] [1129] NewMongoAdapter: Created new MongoDB Adapter client - Hosts: [mongo1:30001 mongo2:30002
mongo3:30003] ReplicaSet: rs1 Database name: nshield-corecrypto Segregation: disabled
[INFO] [WSOP] [1129] [server] configureAPI: database initialised
[DEBUG] [WSOP] [1129] [DB] Adapter[ncore]: databaseHealthCheck: started
```

This is not an error, the service will run with Virtual Partitioning disabled

10.5.2. Virtual Partitioning enabled

This is an example output where both fields, `segregation_db_name` and `segregations_collection_name`, are defined, Virtual Partitioning is enabled.

```
[INFO] [WSOP] [1180] [config] loglevel set to: debug
[INFO] [DBADAPTER] [1180] TLS CA key loaded
[INFO] [DBADAPTER] [1180] TLS client keys loaded
[INFO] [DBADAPTER] [1180] NewMongoAdapter: Created new MongoDB Adapter client - Hosts: [mongo1:30001 mongo2:30002
mongo3:30003] ReplicaSet: rs1 Database name: nshield-corecrypto Segregation: enabled. Using mappings from
segregation_db.segregations
[INFO] [WSOP] [1180] [server] configureAPI: database initialised
[DEBUG] [WSOP] [1180] [DB] Adapter[ncore]: databaseHealthCheck: started
```

10.5.3. Virtual Partitioning enabled and the client is not mapped

This is an example output where a client is not mapped.

```
[ERROR] [DBADAPTER] [1180] getOneRecord: record not found where [{issuer
CN=CommonName,OU=OrganizationalUnit,O=Organization,L=Locality,C=Country} {subject
CN=CommonName,OU=OrganizationalUnit,O=Organization,C=Country}]
[ERROR] [DBADAPTER] [1180] GetSegregationLabel: database record not found
[ERROR] [WSOP] [1180] [KM] NCoreKeyManager[c66a5fcc-be23-5d3b-aa7a-997ec27e3b71]: NewNCoreKeyManager: cannot get
the segregation label: database record not found
[ERROR] [WSOP] [1180] [KM] NCoreKeyManager[c66a5fcc-be23-5d3b-aa7a-997ec27e3b71]: NewNCoreKeyManager: cannot get
the segregation label: database record not found
[ERROR] [WSOP] [1180] [request] [9e36eedd-acc9-430b-abec-7e0b54a8fd4b] &models.RestErrorObject{Error:"Forbidden",
Message:"unknown issuer and subject in segregation database", Path:"/km/v1/groups", Status:"403",
Timestamp:"2021-12-15T16:48:21Z"}
[INFO] [WSOP] [1180] [request] [9e36eedd-acc9-430b-abec-7e0b54a8fd4b] IPaddress - "GET /km/v1/groups HTTP/2.0"
403
```

10.6. Example script for Virtual Partitioning database generation

The following is an example python script that demonstrates how the segregation db and collection is created and populated with clients and their corresponding virtual partitions.

The `segregation_map` dictionary defines the label for each certificate. The `issuer` and `sub-`

ject fields are extracted from the certificate. This information is then inserted into the data base with the corresponding label.

```
#!/usr/bin/python3
"""
This example script adds segregation mappings for TLS certs as defined in
segregation_map
"""
from pymongo import MongoClient
from pymongo.collation import Collation
from cryptography import x509
from os import path

config = {
    "MONGO_HOST": "mongo1",
    "MONGO_PORT": 30001,
    "SEGREGATION_DB": "segregation_db",
    "SEGREGATION_COLLECTION": "segregations",
}

# When no label is provided, the client belongs to the common virtual partition
segregation_map = {
    # client certificate : partition label
    "client1.crt": "Label_A",
    "client2.crt": "Label_A",
    "client3.crt": "Label_B",
    "client4.crt": "Label_C",
    "client5.crt": None,
    "client6.crt": None
}

try:
    # Create the Segregation database and collection
    with MongoClient(host=config["MONGO_HOST"], port=config["MONGO_PORT"]) as mongoClient:
        mongoClient[config["SEGREGATION_DB"]
                    ][config["SEGREGATION_COLLECTION"]].drop()
        # The index of Subject and Issuer is case insensitive
        collection = mongoClient[config["SEGREGATION_DB"]].create_collection(
            config["SEGREGATION_COLLECTION"], collation=Collation(locale='en', strength=2))
        collection.create_index([("issuer", 1), ("subject", 1)], unique=True)

        documents_to_add = []

        # Process certificates
        for certificate in segregation_map:
            filename = path.join(path.dirname(
                path.abspath(__file__)), certificate)
            with open(filename, 'rb') as pfile:
                cert = x509.load_pem_x509_certificate(pfile.read())

                subject = []

                subject_serial_number = cert.subject.get_attributes_for_oid(
                    x509.NameOID.SERIAL_NUMBER)
                if subject_serial_number:
                    subject.append(subject_serial_number[0].rfc4514_string())

                subject_common_name = cert.subject.get_attributes_for_oid(
                    x509.NameOID.COMMON_NAME)
                if subject_common_name:
                    subject.append(subject_common_name[0].rfc4514_string())

                subject_organization_unit_name = cert.subject.get_attributes_for_oid(
                    x509.NameOID.ORGANIZATIONAL_UNIT_NAME)
                if subject_organization_unit_name:
```

```
            subject.append(
                subject_organization_unit_name[0].rfc4514_string())

        subject_organization_name = cert.subject.get_attributes_for_oid(
            x509.NameOID.ORGANIZATION_NAME)
        if subject_organization_name:
            subject.append(
                subject_organization_name[0].rfc4514_string())

        subject_postal_code = cert.subject.get_attributes_for_oid(
            x509.NameOID.POSTAL_CODE)
        if subject_postal_code:
            subject.append(subject_postal_code[0].rfc4514_string())

        subject_street_address = cert.subject.get_attributes_for_oid(
            x509.NameOID.STREET_ADDRESS)
        if subject_street_address:
            subject.append(subject_street_address[0].rfc4514_string())

        subject_locality_name = cert.subject.get_attributes_for_oid(
            x509.NameOID.LOCALITY_NAME)
        if subject_locality_name:
            subject.append(subject_locality_name[0].rfc4514_string())

        subject_province_name = cert.subject.get_attributes_for_oid(
            x509.NameOID.STATE_OR_PROVINCE_NAME)
        if subject_province_name:
            subject.append(subject_province_name[0].rfc4514_string())

        subject_country_name = cert.subject.get_attributes_for_oid(
            x509.NameOID.COUNTRY_NAME)
        if subject_country_name:
            subject.append(subject_country_name[0].rfc4514_string())

        subject = ','.join(subject)

        issuer = []

        issuer_serial_number = cert.issuer.get_attributes_for_oid(
            x509.NameOID.SERIAL_NUMBER)
        if issuer_serial_number:
            issuer.append(issuer_serial_number[0].rfc4514_string())

        issuer_common_name = cert.issuer.get_attributes_for_oid(
            x509.NameOID.COMMON_NAME)
        if issuer_common_name:
            issuer.append(issuer_common_name[0].rfc4514_string())

        issuer_organization_unit_name = cert.issuer.get_attributes_for_oid(
            x509.NameOID.ORGANIZATIONAL_UNIT_NAME)
        if issuer_organization_unit_name:
            issuer.append(
                issuer_organization_unit_name[0].rfc4514_string())

        issuer_organization_name = cert.issuer.get_attributes_for_oid(
            x509.NameOID.ORGANIZATION_NAME)
        if issuer_organization_name:
            issuer.append(issuer_organization_name[0].rfc4514_string())

        issuer_postal_code = cert.issuer.get_attributes_for_oid(
            x509.NameOID.POSTAL_CODE)
        if issuer_postal_code:
            issuer.append(issuer_postal_code[0].rfc4514_string())

        issuer_street_address = cert.issuer.get_attributes_for_oid(
            x509.NameOID.STREET_ADDRESS)
        if issuer_street_address:
```

```
                    issuer.append(issuer_street_address[0].rfc4514_string())

                issuer_locality_name = cert.issuer.get_attributes_for_oid(
                    x509.NameOID.LOCALITY_NAME)
                if issuer_locality_name:
                    issuer.append(issuer_locality_name[0].rfc4514_string())

                issuer_province_name = cert.issuer.get_attributes_for_oid(
                    x509.NameOID.STATE_OR_PROVINCE_NAME)
                if issuer_province_name:
                    issuer.append(issuer_province_name[0].rfc4514_string())

                issuer_country_name = cert.issuer.get_attributes_for_oid(
                    x509.NameOID.COUNTRY_NAME)
                if issuer_country_name:
                    issuer.append(issuer_country_name[0].rfc4514_string())

                issuer = ','.join(issuer)

            record = {}
            record["subject"] = subject
            record["issuer"] = issuer
            if segregation_map[certificate]:
                record["label"] = segregation_map[certificate]
            else:
                # No label value indicates the common partition
                pass

            documents_to_add.append(record)

        collection.insert_many(documents_to_add)

        # Reporting
        for i in collection.find({}):
            # display 'null' when no label
            if 'label' not in i:
                i['label'] = 'null'
            print("Added label: '{0}' subject: '{1}', issuer: '{2}'".format(
                i['label'], i['subject'], i['issuer']))

except Exception as ex:
    print(ex)
    import sys
    sys.exit(1)
```

11. WSOP Database Management Tool

11.1. Introduction

As described previously, in the current WSOP release the keys and tokens are stored in a database.

Older WSOP versions (WSOP2.1 and older) store the keys in file in `/opt/nfast/kmdata/local` folder.

The Database Management Tool (DBMT) is part of WSOP installation package and must be used for database collection initialization. It can also be used for migration and integrity checking. For more advanced database operations, contact your database administrator.

The Database Management Tool is implemented as a Python3 package. Specify the path of the package in the `PYTHONPATH` environment variable.

The Database Management Tool imports PKCS #11 objects using an external PKCS #11 library specified in `PKCS11_MODULE` environment variable or `--library` argument. If no library is specified, the Security World PKCS #11 library will be used. If no library is installed, a warning message will be displayed and the migration will continue without including the PKCS #11 objects.

The log file is generated in the local directory as `nshieldlogs/database_management_tool.log`

To see the help text of the Database Management Tool you can run the tool with `--help` option.

11.2. Installation

Prerequisites

- Security World 12.80+ installed and running
- Mongo DB
- Root access

Install in local system by unpacking and running the install script:

```
tar -xf dbmt-X.X.X.tar.gz
cd opt/nfast/webservices/dbmt-X.X.X
./install.sh
```

The dbmt is installed to the PYTHONPATH directory (for example, /opt/nfast/python3/bin/dbmt).

Install script must be called with correct user permissions.

An empty configuration file is supplied in the DBMT package: opt/nfast/webservices/dbmt-X.X.X/config_example.yaml

11.3. Configuration file

The database migration tool requires a configuration file which provides details about the destination database. The name and location of the configuration file is passed as the value of --config command line argument. The configuration file is in yml format.

This subchapter describes the sections and the fields in the configuration file.

11.3.1. Database hostname

The db_host field specifies the name or the IP address of the host where the database is cur rently running.

```
# Database hostname (ip address)
db_host: localhost
```

11.3.2. Database port number

The db_port field specifies the port number of the database.

```
# Database port number
db_port : 30001
```

11.3.3. Database name

The db_name field specifies the name of the database. This field is mandatory, meaning that running the migration tool will fail if a name is not provided.

```
# Name of the database
```

```
db_name : nshield-corecrypto
```

11.3.4. Database type

The `db` field specifies the type of the database management system. Currently the only sup ported value is `mongodb`.

```
# Database Management System. Valid values: [mongodb]
db : mongodb
```

11.3.5. MongoDB settings

The following settings are under `mongodb` section of the configuration file.

11.3.6. Disable TLS

The `disable_tls` field specifies if TLS for database connection is on or off as follows:

- `true`: TLS is not used.
- `false`: TLS is used.

```
# Transport Layer Security. Default is enabled
disable_tls: false
```

11.3.7. Authentication method

The `auth_type` field specifies one of the possible authentication methods:

- none
- username, password, and authentication database.
- TLS

The possible values are `none`, `pwd`, and `tls`.

For details about each of these options see section Authentication Methods from this chap ter.

```
#
# Authentication method with database. Valid values: [none, pwd, tls]
auth_type: none
```

11.3.8. Username and password file

In case the authentication is performed by username and password, then `auth_pwd_file` field specifies the file where the username and password are stored.

```
auth_pwd_file: /path_to/config-pwd-auth.yaml
```

11.3.9. Authentication database

The `auth_source` field specifies the name of the authentication database. The username and password provided by the file name specified by the `auth_pwd_file` field are verified against the username and password stored in the authentication database.

```
auth_source: userdb
```

11.3.10. Certificate Authority (CA) files for TLS

When TLS option is used as an authentication method, then `db_ca_file`, `db_cert_file` and `db_key_file` fields specify the Certificate Authority files for TLS.

```
# Certificate Authority files to use for TLS
db_ca_file: /path_to/db_ca.crt
db_cert_file: /path_to/db_client.pem
db_key_file: /path_to/db_client.key
```

11.3.11. Virtual Partitioning database and collection

To enable Virtual Partitioning the user must define the Virtual Partitioning database and collection that contains the client mappings. For more information about Virtual Partitioning see the Virtual Partitioning section of this User Guide.

```
# WSOP Corecrypto Segregation (WCS) Options
# The segregation database and collection must be defined in order
# to enable WCS. When enabled, corecrypto objects
# will be segregated based on the mappings defined in the collection.
# If the segregation database and collection are not defined,
# WCS is disabled.
#
segregations_db_name: segregation_db
segregations_collection_name: segregations
```

As previously stated, to enable Virtual Partitioning, both fields, `segregation_db_name` and `segregations_collection_name`, must be defined. If one of the fields is defined and the other is undefined, then the migra-

> tion with virtual partitions will raise an error.

11.4. Environment variables

Hardserver environment variables that might be needed for migration.

- To import Softcards:

```
:# export CKNFAST_LOADSHARING=1
```

- To use alternative kmdata local directory:

```
:# export NFAST_KMLOCAL=/opt/nfast/kmdata/local_custom/
```

You can also define these environment variables in the hardserver configuration file: `/etc/nfast.conf`.

11.5. Database initialization

Initializes the database with an initial structure. This also imports required Security World files and generates default domains and groups. If run on an existing setup, the database will be synced with the current Security World files.

Usage

```
dbmt db-init [-h] --config CONFIG -n HOSTNAME -p PORT [-d DB] [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
```

Optional arguments

Arg	Description
`-h|--help`	Shows this help message and exits.
`--config <CONFIG>`	Specifies the YAML configuration file used to configure the Database Management Tool.
`-n|--hostname <HOSTNAME>`	The hostname of the database server. Overrides the value in `config.yaml`.
`-p|--port <PORT>`	The port of the database server. Overrides the value in `config.yaml`.
`-d|--db <DB>`	The database type to use. Available types: `mongodb` (default)

Arg	Description
`--log-level`	The severity level to log. One of: • `DEBUG` • `INFO` (default) • `WARNING` • `ERROR` • `CRITICAL`

11.5.1. Examples

- Database initialization:

```
:# dbmt db-init --config  config.yaml
```

11.6. Migration

Imports kmdata using nfast to the MongoDB server.

Usage

```
dbmt migrate [-h] --config CONFIG -n HOSTNAME -p PORT [-d DB] [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
[-K APPNAME KEY_IDENT | --softcard SOFTCARD | --module | --wellknown] [--dry-run] [--library LIBRARY] [--
segregate CERT]
```

Optional arguments

Arg	Description
`-h|--help`	Shows this help message and exits.
`--config <CONFIG>`	Specifies the YAML configuration file used to configure the Database Management Tool.
`-n|--hostname <HOSTNAME>`	The hostname of the database server. Overrides the value in `config.yaml`.
`-p|--port <PORT>`	The port of the database server. Overrides the value in `config.yaml`.
`-d|--db <DB>`	The database type to use. Available types: `mongodb` (default)

Arg	Description
`--log-level`	The severity level to log. One of: • `DEBUG` • `INFO` (default) • `WARNING` • `ERROR` • `CRITICAL`
`-K|--key <APPNAME KEY_IDENT>`	Filter the keys to import by appname and key-ident.
`-K|--key <APPNAME KEY_IDENT> --group <GROUP_UUID>`	Filter the keys to import by appname and key-ident. The respective keys are imported to the provided group.
`--softcard <SOFTCARD>`	Only add specified Softcard.
`--module`	Only add Module protected keys.
`--wellknown`	Only add Well-Known public keys.
`--dry-run`	Dry run without committing to database.
`--library <LIBRARY>`	PKCS #11 library path
`--segregate <CERT>`	Migrate objects with segregation using the supplied CERT file for issuer and subject information. Requires WSOP segregation to be enabled in the config file.

11.6.1. Examples

- Simple migration:

```
:# dbmt migrate --config  config.yaml --library /opt/nfast/bin/libcknfast.so
```

- Migration of Module protected keys:

```
:# dbmt migrate --config  config.yaml --module --library /opt/nfast/bin/libcknfast.so
```

- Migration of Softcard protected keys:

```
:# dbmt migrate --config  config.yaml --softcard ACCOUNTANCY-SOFTCARD --library
/opt/nfast/bin/libcknfast.so
```

- Migration of a key:

```
:# dbmt migrate --config config.yaml --library /opt/nfast/bin/libcknfast.so -K simple ACCOUNTANCY-RSA-2048-
KEY
```

```
* Migration of a key to the provided group:
+
[source]
:# dbmt migrate --config config.yaml --library /opt/nfast/bin/libcknfast.so -K simple ACCOUNTANCY-RSA-2048-KEY
--group c26e5efc-6ef3-4d9b-b9ef-c888acea1452
```

- Migration with segregation:

```
:# dbmt migrate --config  config.yaml --library /opt/nfast/bin/libcknfast.so --segregate cert.pem
```

- Dry run:

```
:# dbmt migrate --config  config.yaml --log-level DEBUG --library /opt/nfast/bin/libcknfast.so --dry-run
```

The password must be entered for every password protected Softcard. If incorrect password is entered migration is aborted. Migration data is only uploaded to MongoDB after all the passwords are entered correctly.

11.7. DB check

Checks the data integrity of the WSOP database in the Mongo DB server.

Usage:

```
dbmt db-check [-h] --config CONFIG -n HOSTNAME -p PORT [-d DB] [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
```

Optional arguments:

Arg	Description
`-h|--help`	Shows this help message and exits.
`--config <CONFIG>`	Specifies the YAML configuration file used to configure the Database Management Tool.
`-n|--hostname <HOSTNAME>`	The hostname of the database server. Overrides the value in `config.yaml`.
`-p|--port <PORT>`	The port of the database server. Overrides the value in `config.yaml`.

Arg	Description
`-d|--db <DB>`	The database type to use. Available types: `mongodb` (default).
`--log-level`	The severity level to log. One of: • `DEBUG` • `INFO` (default) • `WARNING` • `ERROR` • `CRITICAL`

11.7.1. Examples

To perform a database integrity check:

```
:# dbmt db-check --config config.yaml
```

11.8. Migration For Virtual Partitioning

The Database Management Tool migration also supports Virtual Partitioning. This can be used to maintain RFS directory partitions of Softcards and key grouping in the standalone database.

Entrust recommends that you use a separate machine for the migration. This migration machine must have Security World 12.80+ installed, as well as the `DBMT` utility. Set `NFAST_KMLOCAL` to the directory containing the keys to be migrated.

11.8.1. Prerequisites

- The Database Management Tool is installed.
- MongoDB servers are running.
- Security World of version at least 12.80 is installed.
- The MongoDB address is defined in the Database Management Tool `config.yaml`.
- Virtual Partitioning is enabled in the Database Management Tool `config.yaml`.
- The client's mapping is defined in Virtual Partitioning Collection.
- `NFAST_KMLOCAL` is defined to be the directory containing the keys to be migrated for this client.
- The database has been initialized

11.8.2. Example of successful migration

This is the example output for a successful migration with Virtual Partitioning.

```
:# export NFAST_KMLOCAL=/opt/nfast/kmdata/local_custom/
:# dbmt migrate --config config.yaml --segregate cert.crt
Starting migration of the database...
Establishing connection to hardserver
TLS enabled
No Authentication specified
WSOP Client segregation ENABLED (segregation_db.segregations)
Establishing mongo connection to: mongo1:30001
Set the database name: nshield-corecrypto
Retrieving segregation information from TLS certificate cert.crt
Looking up subject: CN=CommonName,OU=OrganizationalUnit,O=Organization,C=Country
Migrating using segregation label 'VirtualPartitionLabel'
Collecting kmdata artifacts...
```

11.8.3. Examples of migrations with setup errors

This is the example output when Virtual Partitioning has not been enabled in the Database Management Tool `config.yaml` but migration with Virtual Partitioning is attempted.

```
:# dbmt migrate --config config.yaml --segregate cert.crt
Starting migration of the database...
Establishing connection to hardserver
TLS enabled
No Authentication specified
Establishing mongo connection to: mongo1:30001
Set the database name: nshield-corecrypto
Segregation is not enabled. Please set segregation_db and segregation_collection_name in config
```

This is the example output when Virtual Partitioning is enabled and configured in the Database Management Tool `config.yaml` but Virtual Partitioning database and collection have not been created.

```
:# dbmt migrate --config config.yaml --segregate cert.crt
Starting migration of the database...
Establishing connection to hardserver
TLS enabled
No Authentication specified
WSOP Client segregation ENABLED (segregation_db.segregations)
Establishing mongo connection to: mongo1:30001
Set the database name: nshield-corecrypto
Retrieving segregation information from TLS certificate cert.crt
Looking up subject: CN=CommonName,OU=OrganizationalUnit,O=Organization,C=Country
Segregation database is empty or does not exist
```

11.9. Upgrade

Upgrades the key records between versions of the Database Management Tool `con-`

`fig.yaml`.

Usage:

```
dbmt upgrade [-h] --config CONFIG -n HOSTNAME -p PORT [-d DB] [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
[--dry-run [DRY_RUN]]
```

Optional arguments:

Arg	Description
`-h|--help`	Shows this help message and exits.
`--config <CONFIG>`	Specifies the YAML configuration file used to configure the Database Management Tool.
`-n|--hostname <HOSTNAME>`	The hostname of the database server. Overrides the value in `config.yaml`.
`-p|--port <PORT>`	The port of the database server. Overrides the value in `config.yaml`.
`-d|--db <DB>`	The database type to use. Available types: `mongodb` (default).
`--log-level`	The severity level to log. One of: • `DEBUG` • `INFO` (default) • `WARNING` • `ERROR` • `CRITICAL`
`--dry-run [<DRY_RUN>]`	Dry run without committing to database.

11.9.1. Examples

To perform a database upgrade:

```
:# dbmt upgrade --config config.yaml
```

11.10. Migrate schema

Updates previous versions of databases to be supported by the latest WSOP release. This tool is available from the WSOP v3.3.0 release onwards.

Usage:

```
dbmt migrate-schema [-h] --config CONFIG -n HOSTNAME -p PORT [-d DB] [--log-level
{DEBUG,INFO,WARNING,ERROR,CRITICAL}] [--dry-run]
```

Optional arguments:

Arg	Description
`-h|--help`	Shows this help message and exits.
`--config <CONFIG>`	Specifies the YAML configuration file used to configure the Database Management Tool.
`-n|--hostname <HOSTNAME>`	The hostname of the database server. Overrides the value in `config.yaml`.
`-p|--port <PORT>`	The port of the database server. Overrides the value in `config.yaml`.
`-d|--db <DB>`	The database type to use. Available types: `mongodb` (default).
`--log-level`	The severity level to log. One of: • `DEBUG` • `INFO` (default) • `WARNING` • `ERROR` • `CRITICAL`
`--dry-run [<DRY_RUN>]`	Dry run without committing to database.

11.10.1. Example

To update previous versions of the database using `migrate-schema`:

```
:# dbmt migrate-schema --config config.yaml
```

11.11. PKCS #11 Object Migration

The following table lists the PKCS #11 objects currently supported by the Database Management Tool for migration.

Object	Supported Types
`CKO_DATA`	
`CKO_CERTIFICATE`	`CKC_X_509`

Object	Supported Types
CKO_SECRET_KEY	CKK_AES CKK_SHA256_HMAC CKK_SHA384_HMAC CKK_SHA512_HMAC
CKO_PRIVATE_KEY	CKK_RSA
CKO_PUBLIC_KEY	CKK_RSA

12. REST API

The REST API offered by Web Services Option Pack is made up of three parts: crypto for cryptographic operations, km for key management, and a health check endpoint. Full details of the REST API are available in the *Web Services Option Pack OpenAPI spec* file `/opt/nfast/webservices/docs/openapi.yaml`.

The currently defined set of operations is as follows:

- Crypto
  - Sign
  - Verify
  - Encrypt
  - Decrypt
- Key Management
  - List Keys
  - List Keys in a key group
  - Get Key information
  - Create Key
  - Delete Key
  - List key groups
  - Get key group information
  - Get key group name
  - Create key group
  - Delete key group
  - Create protection domain
  - Delete protection domain
  - List protection domains
  - Get protection domain information
  - Activate a protection domain
  - Deactivate a protection domain
- Random Number Generation
- Health check endpoint

12.1. Common fields

Several of these fields are common to many cryptographic and key management operations.

12.1.1. Key identifiers

Security World keys are uniquely identified in REST calls through a resource identifier. A key's resource identifier includes which key group it is a member of. An example key identifier is:

```
km/v1/groups/ee3b7c4b-c6a7-3f2b-9d1c-0fc6c76168bb/keys/f9388a1e-b1c6-3e0e-bb5e-8eb887aab4be
```

12.1.2. Key group identifiers

All Security World keys belong to exactly one key group. Each key group belongs to, and is protected by, a single protection domain. An example key group identifier is:

```
ee3b7c4b-c6a7-3f2b-9d1c-0fc6c76168bb
```

12.1.3. Protection domain identifiers

A protection domain is either a Softcard or the set of all module keys. An example protection domain identifier is:

```
ee3b7c4b-c6a7-3f2b-9d1c-0fc6c76168bb
```

12.1.4. Protection domain types

A protection domain is one of the following types:

- Module
- Softcard
- Well-Known

12.1.5. Plaintext, ciphertext, payloads and signatures

Data sent in requests or responses is encoded in base64url format. Base64url encoding is a variation of standard base64 encoding and uses a slightly different character set that are safe for using in URLs without additional escaping of reserved characters, see Base64url

Encoding

Data that forms part of cryptographic operations must be sent in a single request. Responses will similarly be sent from Web Services Option Pack as a single response body.

12.1.6. Digest option

Applicable for signing and verification requests, the optional `digest` Boolean flag indicates whether the provided `payload` has been hashed prior to sending the request.

When the `digest` flag is set to `true`, the server-side hashing of the payload is skipped, and length checking is performed to ensure that the supplied `payload` is the correct length for the hash that corresponds to the signature or verification algorithm.

The `digest` flag is not applicable for HMAC signing and verification requests.

12.1.7. IV and AAD parameters

Supplemental `iv` and `aad` parameters can be used with supported mechanisms as described below:

Mechanism	Parameters
AES-CBC	• If `iv` is omitted, a 16 byte random IV will be internally generated. • If `iv` is specified, it must be 16 bytes in length.
AES-GCM	• If `iv` is omitted, a 12 byte random IV will be internally generated. This is a FIPS 140-2 approved mode of operation for this mechanism. • If `iv` is specified, it must be 12 bytes in length. • `aad` is optional and if specified, must be less than 65535 bytes in length. • A fixed value of 16 bytes is used for `taglen`.
RSA-OAEP	• The label `L` parameter can be passed in as the `iv` parameter. • If `iv` (`L` value) is omitted, a `null string` is used. • If `iv` (`L` value) is specified, it must be less than 1024 bytes in length.

12.1.8. Algorithm identifiers

The majority of the algorithms are identified using JSON Web Algorithm identifiers. For example, `RS256` specifies RSA signatures with SHA-256 for hashing. For further information, see Supported algorithms in this *User Guide* and the *Web Services Option Pack OpenAPI Specification*.

12.1.9. Key type identifiers

The types of keys that can be created are identified using nCore identifiers. Refer to the *Web Services Option Pack OpenAPI Specification* for further information.

12.1.10. Elliptic Curve identifiers

Elliptic Curves are identified using defined names from the Digital Signature Standard (DSS). For example, `P-256`. Refer to *Web Services Option Pack OpenAPI Specification* for further information.

The `name` input parameter should not contain any HTML character, for example `<`, `>` or `&`, as this can result in those values becoming corrupted when they are retrieved later.

12.1.11. Protection domain activation option

Applicable for crypto and key creation requests, the optional `protectiondomain` parameter activates the protection domain if required.

When `passphrase` is provided as a part of the `protectiondomain` parameter and the protection domain is not activated, the passphrase will be used to attempt to activate the protection domain before completing the request.

12.2. Examples

Given client identity information and a corresponding key, it is possible to use command line tools such as `curl` to demonstrate use of the REST API. These examples assume the fol lowing:

- The Web Services Option Pack server's CA certificate is located in `server_ca.pem`.
- A valid client certificate is located in `client_crt.pem`.
- The private key corresponding to the client certificate is located in `client_key.pem`.

12.2.1. List keys

List all keys:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/keys?sort=created"
```

Result:

```
{
    "keys": [
        {
            "created": "2017-07-03T13:17:56Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/90050af4-9b8c-5731-bbf2-
08e3af797e2c",
            "length": 256,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes"
        },
        {
            "created": "2017-07-06T15:44:12Z",
            "keyType": "RSA",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/e51a13d1-b775-3129-b1f9-
b8642a212475",
            "public": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/e51a13d1-b775-3129-b1f9-
b8642a212475/public",
            "length": 1024,
            "sworldAppname": "simple",
            "sworldIdent": "test-rsa"
        }'
        {
            "created": "2018-07-06T15:44:12Z",
            "curve": "P-521",
            "keyType": "ECDSA",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/6fdc73bb-db0e-517b-962d-
a378f270fd41",
            "public": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/6fdc73bb-db0e-517b-962d-
a378f270fd41/public",
            "sworldAppname": "simple",
            "sworldIdent": "test-ecdsa"
        }
        //...
    ]
}
```

List one key:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/keys?limit=1"
```

Result:

```
{
    "keys": [
        {
            "created": "2017-07-03T13:17:56Z",
            "keyType": "AES"'
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/90050af4-9b8c-5731-bbf2-
08e3af797e2c",
            "length": 256,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes"
        }
    ]
}
```

List keys, with an offset:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/keys?offset=2"
```

Result:

```
{
    "keys": [
        {
            "created": "2017-07-06T08:12:51Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/69fc1c75-81eb-58ad-83ff-
c3fd517c27cf",
            "length": 192,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes192"
        },
        {
            "created": "2017-07-07T17:24:31Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/7f34e38a-be3a-5378-a163-
3313ce82ea02",
            "length": 256,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes256"
        }
        //...
    ]
}
```

List all keys created since the beginning of 2017:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/keys?filter=created%20ge%20datetime'2017-01-01T00:00:00Z'"
```

Result:

```
{
    "keys": [
        {
            "created": "2017-07-03T13:17:56Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/90050af4-9b8c-5731-bbf2-
08e3af797e2c",
            "length": 256,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes"
        },
        {
            "created": "2017-07-06T15:44:12Z",
            "keyType": "RSA",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/e51a13d1-b775-3129-b1f9-
b8642a212475",
            "length": 1024,
            "sworldAppname": "simple",
            "sworldIdent": "test-rsa"
        }
    ]
}
```

The use of `%20` to escape spaces in URIs. Date and time stamps are in ISO-8601 format.

List all keys which have a certain name:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/keys?name=engineering"
```

Result:


```
{
    "keys": [
        {
            "created": "2021-12-17T16:44:52Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/19e40e8c-2188-5f9f-a1c8-8007424a4157/keys/589e4486-9da6-53ec-92ff-
b2444c61ee02",
            "length": 192,
            "name": "engineering",
            "sworldAppname": "simple",
            "sworldIdent": "test-aes192"
        },
        {
            "created": "2021-12-17T16:44:52Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/19e40e8c-2188-5f9f-a1c8-8007424a4157/keys/470d26c7-ad9d-5e49-9910-
6e0c4fa9dad5",
            "length": 256,
            "name": "engineering",
            "sworldAppname": "simple",
            "sworldIdent": "test-aes256"
        },
        {
            "created": "2021-12-17T16:45:08Z",
```

```
            "keyType": "RSA",
            "kid": "/km/v1/groups/a8668e0d-ef17-5a36-9dff-12a885dcb738/keys/1afdf157-056a-561e-8280-
9b1b8e3aa817",
            "length": 1024,
            "name": "engineering",
            "public": "/km/v1/groups/a8668e0d-ef17-5a36-9dff-12a885dcb738/keys/1afdf157-056a-561e-8280-
9b1b8e3aa817/public",
            "sworldAppname": "simple",
            "sworldIdent": "test-rsa"
        }
    ]
}
```

List all keys with a matching substring of that named.

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/keys?filter=kid%20contains'269f0189'"
```

Result:

```
{
    "keys": [
        {
            "created": "2017-07-03T13:17:56Z",
            "keyType": "AES",
            "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/90050af4-9b8c-5731-bbf2-
08e3af797e2c",
            "length": 256,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes"
        }
    ]
}
```

12.2.2. List keys in a group

List all keys belonging to a certain group, for instance 19e40e8c-2188-5f9f-a1c8-8007424a4157 group:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/groups/19e40e8c-2188-5f9f-a1c8-8007424a4157/keys"
```

Result:

```
{
    "keys": [
        {
            "created": "2021-12-17T16:44:52Z",
```

```
            "keyType": "AES",
            "kid": "/km/v1/groups/19e40e8c-2188-5f9f-a1c8-8007424a4157/keys/4a59aec6-1cb4-5f27-9546-
249b6e241f6f",
            "length": 256,
            "sworldAppname": "simple",
            "sworldIdent": "test-aes192"
        },
        {
            "created": "2021-12-17T16:44:51Z",
            "keyType": "HMACSHA256",
            "kid": "/km/v1/groups/19e40e8c-2188-5f9f-a1c8-8007424a4157/keys/dc2b10ee-dc7b-521f-b732-
db252f7c13c2",
            "length": 256,
            "name": "accounting",
            "sworldAppname": "simple",
            "sworldIdent": "test-hmac256"
        },
        //...
    ]
}
```

All HTTP queries described in List keys also work when listing the keys of a certain group.

12.2.3. Public key export

The following example exports the public key from an ECDSA key:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/6ff3a5ef-c12f-5e82-b1d6-
d955caef2730/public"
```

Result:

```
{
    "crv": "P-384",
    "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/6ff3a5ef-c12f-5e82-b1d6-d955caef2730”,
    "kty": "EC",
    "x": "JDvhhAqlSpn2xrjq7C1iOYKvPZl1PVARsXlsQc9EyP9dxhcZSJ0bkJcYXuu-3PWx",
    "y": "_B3o_PZx2jsYA4nsKG82sCoEyCzdbeFTtIPrO5Y8PhkRc4owIiO_7BOUu97BkRsZ"
}
```

12.2.4. Public key import

Public Key import can only be performed in the `Well-Known` Protection key group.

```
curl --cacert server_ca.pem \
    --cert client_crt.pem —-key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
```

```
    --request POST \
-d '{"appname":"simple", "ident":"test-import", "public": {"alg": "ES384","crv": "P-384","kty": "EC","x":
"JDvhhAqlSpn2xrjq7C1iOYKvPZl1PVARsXlsQc9EyP9dxhcZSJ0bkJcYXuu-3PWx", "y":
"_B3o_PZx2jsYA4nsKG82sCoEyCzdbeFTtIPrO5Y8PhkRc4owIiO_7BOUu97BkRsZ"}}'
    'https://wsop.server:18001/km/v1/groups/2bd40730-85b1-5deb-8417-fb78a7735743/keys'
```

Result:

```
{
    "kid": "/km/v1/groups/2bd40730-85b1-5deb-8417-fb78a7735743/keys/170ff5ca-9d46-522c-84d1-84074955b94f"
}
```

12.2.5. Create keys

Create a key in the `Module Protection` key group:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
-d '\{"keytype":"ECDSA","curve":"P-256", "appname":"simple","ident":"test-p256"}'
    'https://wsop.server:18001/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys'
```

Result:

```
{
    "kid": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/54872032-a16b-5c5e-a1c5-97f2656b7c24"
}
```

Create a key in the `Softcard Protection` key group with a passphrase:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
-d '{"keytype":"ECDSA","curve":"P-384", "appname":"simple","ident":"test-
p384","protectiondomain":{"passphrase":"passphrase1"}}'
    'https://wsop.server:18001/km/v1/groups/22fe7f69-ce90-5b75-b00b-69d55e840b3a/keys'
```

Result:

```
{
    "kid": "/km/v1/groups/22fe7f69-ce90-5b75-b00b-69d55e840b3a/keys/54872032-a16b-5c5e-a1c5-97f2656b7c24"
}
```

12.2.6. Delete keys

The following example deletes a key:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --request DELETE \
    'https://wsop.server:18001/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/54872032-a16b-5c5e-a1c5-
97f2656b7c24'
```

On successful deletion, `204 No Content` is returned.

12.2.7. List key groups

List all key groups:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/groups"
```

Result:

```
{
    "groups": [
        {
            "groupid": "ccba7ee7-9032-5217-9db2-58d07fd4c045",
            "keys": "/km/v1/groups/ccba7ee7-9032-5217-9db2-58d07fd4c045/keys",
            "name": "soft1",
            "protection": "/km/v1/protectiondomains/ccba7ee7-9032-5217-9db2-58d07fd4c045",
            "type": "Softcard"
        },
        {
            "groupid": "366f38ec-64e1-503c-9950-18cc324d67c6",
            "keys": "/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys",
            "name": "Module Protection",
            "protection": "/km/v1/protectiondomains/366f38ec-64e1-503c-9950-18cc324d67c6",
            "type": "Module"
        },
    //...
    ]
}
```

12.2.8. Create key groups

Create a key group within a protection domain. The following example creates a key group with name `newGroup`.

```
curl --cacert server_ca.pem \
    -cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
```

```
    -d '\{"name": "newGroup", "protection": "/km/v1/protectiondomains/bf3cab05-b27c-5627-9b44-1d32b6f7148f"}' \
    'https://test-server:18001/km/v1/groups'
```

Result:

```
{
  "id": "/km/v1/groups/31894214-4ec9-4418-889e-609b6d65751b"
}
```

12.2.9. Delete key groups

Delete a key group within a protection domain. The following example deletes a key group and all keys within the group.

```
curl --cacert server_ca.pem \
    -cert client_crt.pem –-key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request DELETE \
    'https://test-server:18001/km/v1/groups/4759eac6-9a22-5a8d-8dda-ef3e94d08161?cascade=keys'
```

On successful deletion, `204 No Content` is returned.

By default a group will not be deleted if it has any keys within it. The cas cade parameter allows a user to set this behaviour. The options are "none" (default) and "keys". If "keys" is set then all keys in the group will be deleted alongside the group.

12.2.10. Create protection domains

Create protection domain can only be performed with `Softcard` protection domain types. The following example creates a protection domain of type `Softcard`.

```
curl --cacert server_ca.pem \
    --cert client_crt.pem –-key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
     -d '{"name":"softcard1","passphrase":"passphrase1","type":"Softcard"}' \
     'https://wsop.server:18001/km/v1/protectiondomains' | jq
```

Result:

```
{
  "id": "e3307c07-300e-585a-95da-ed4f3d0c226e"
}
```

12.2.11. Delete protection domains

Delete protection domain can only be performed with `Softcard` protection domain types. The following example deletes a protection domain and all associated keys.

```
curl --cacert server_ca.pem \
    --cert client_crt.pem —-key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request DELETE \
    'https://wsop.server:18001/km/v1/protectiondomains/025ad489-3ec8-5bbc-b251-f0e497f7cd48?cascade=keys' | jq
```

On successful deletion, `204 No Content` is returned.

By default a protection domain will not be deleted if it has any groups within it. The cascade parameter allows a user to set this behaviour. The options are "none" (default), "groups" and "keys". If "groups" is set then all groups will be deleted alongside the protection domain as long as the groups do not contain any keys. If "keys" is set then all keys in all of the groups will be deleted alongside the protection domain.

12.2.12. List protection domains

List all protection domains:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Accept: application/json' \
    --request GET \
    "https://wsop.server:18001/km/v1/protectiondomains"
```

Result:

```
{
    "protectiondomains": [
        {
            "active": true,
            "id": "366f38ec-64e1-503c-9950-18cc324d67c6",
            "name": "Module Protection",
            "type": "Module"
        },
        {
            "active": false,
            "id": "ccba7ee7-9032-5217-9db2-58d07fd4c045",
            "name": "soft1",
            "type": "Softcard"
        },
        //...
    ]
}
```

12.2.13. Activate protection domains

Activate a protection domain:

```
    curl --cacert server_ca.pem \
        --cert client_crt.pem --key client_key.pem \
        --header 'Content-Type: application/json' \
        --header 'Accept: application/json' \
        --request POST \
        -d '\{"passphrase":"passphrase1"}' \
        "https://127.0.0.1:18001/km/v1/protectiondomains/ccba7ee7-9032-5217-9db2-58d07fd4c045/activate"
```

Result:

```
{
    "protectiondomain": {
        "active": true,
        "id": "ccba7ee7-9032-5217-9db2-58d07fd4c045",
        "name": "soft1",
        "type": "Softcard"
    }
}
```

12.2.14. Deactivate protection domains

Deactivate a protection domain:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
    "https://127.0.0.1:18001/km/v1/protectiondomains/ccba7ee7-9032-5217-9db2-58d07fd4c045/deactivate"
```

Result:

```
{
    "protectiondomain": {
        "active": false,
        "id": "ccba7ee7-9032-5217-9db2-58d07fd4c045",
        "name": "soft1",
        "type": "Softcard"
    }
}
```

12.2.15. Sign payload

The following example signs a payload:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
```

```
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
    -d '\{"kid":"/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/e51a13d1-b775-3129-b1f9-b8642a212475",
        "alg":"RS256",
        "payload":"dGVzdCBkYXRhCg"}'
    'https://wsop.server:18001/crypto/v1/sign'
```

Result:

```
{
"signature": "<base64url-encoded signature omitted>"
}
```

12.2.16. Verify signature

The resulting signature can be verified as follows:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
    -d '\{"kid":"/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/e51a13d1-b775-3129-b1f9-b8642a212475",
        "alg":"RS256",
        "payload":"dGVzdCBkYXRhCg",
        "signature": "<base64url-encoded signature omitted>"}'
    'https://wsop.server:18001/crypto/v1/verify'
```

Result:

```
{
"valid":true
}
```

12.2.17. Sign a payload with a passphrase

The following example signs a payload with a passphrase:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
    -d '\{"kid":"/km/v1/groups/22fe7f69-ce90-5b75-b00b-69d55e840b3a/keys/54872032-a16b-5c5e-a1c5-97f2656b7c24",
        "alg":"ECDSA-SHA384",
        "payload":"dGVzdCBkYXRhCg",
        "protectiondomain": {"passphrase":"passphrase1"}}'
    'https://wsop.server:18001/crypto/v1/sign'
```

Result:

```
{
"signature": "<base64url-encoded signature omitted>"
}
```

12.2.18. Encrypt plaintext

The following example encrypts some plaintext:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
    -d '\{"kid":"/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/90050af4-9b8c-5731-bbf2-08e3af797e2c",
        "alg":"RSA-OAEP-256",
        "plaintext":"dGVzdCBkYXRhCg"}'
    'https://wsop.server:18001/crypto/v1/encrypt'
```

Result:

```
{
"ciphertext": "<base64url-encoded ciphertext omitted>",
}
```

12.2.19. Decrypt ciphertext

The resulting ciphertext can be decrypted as follows:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request POST \
    -d '\{"kid":"":"/km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/90050af4-9b8c-5731-bbf2-08e3af797e2c
",
        "alg":"RSA-OAEP-256",
        "ciphertext":"<base64url-encoded ciphertext omitted>"}'
    'https://wsop.server:18001/crypto/v1/decrypt'
```

Result:

```
{
"plaintext": "dGVzdCBkYXRhCg"
}
```

12.2.20. Random Number Generation

The random number endpoint can be used to generate random bytes from the HSM, using a

single length input parameter for the number of bytes to be generated.

The random numbers are returned encoded in a base64RawURL format. They can be generated as follows:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request GET \
    'https://wsop.server:18001/random/v1/random-bytes?length=20'
```

Result:

```
{
"bytes": "C26xQCt0zI0MrH5LdsIJINMlMEg"
}
```

The length parameter takes a minimum value of 1 and a maximum of 65535.

Setting no length parameter corresponds to a default length of 1 byte.

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request GET \
    'https://wsop.server:18001/random/v1/random-bytes'
```

Result:

```
{
"bytes": "PA"
}
```

12.2.21. Health check

The health of the system can be checked as follows:

```
curl --cacert server_ca.pem \
    --cert client_crt.pem --key client_key.pem \
    --header 'Content-Type: application/json' \
    --header 'Accept: application/json' \
    --request GET \
    'https://wsop.server:18001/health'
```

Result:

```
{
    "releaseId": "1.5.0",
    "status": "pass",
    "version": "1"
}
```

The health check endpoint can be configured to not require TLS mutual authentication while retaining mutual authentication on all other end-points. For more information, see Configure the Web Services Option Pack.

12.3. Error handling

In the event of a failed request, errors are reported to the client by using the HTTP response codes that indicate errors and an appropriate error message and error code in the response body in JSON format.

For example:


```
{
    "error": "ResourceNotFound",
    "message": "loading key /km/v1/groups/366f38ec-64e1-503c-9950-18cc324d67c6/keys/98487743-2ac4-5fe9-81ba-
95eb4a93cae4 failed: unknown key",
    "path": "/crypto/v1/sign",
    "status": "404",
    "timestamp": "2019-12-05T09:58:40Z"
}
```


12.4. Language bindings

An OpenAPIv3-compliant API description is provided (`/opt/nfast/webservices/docs/openapi.yaml`). This represents the Web Services Option Pack REST API, which can be used for generating bindings for a number of programming languages using tools provided as part of Smartbear's Swagger.

The correctness of the generated bindings from Smartbear's Swagger tooling cannot be guaranteed due to numerous issues with the generators. For more information, see the documentation provided by Smartbear.

13. Security guidance for the Web Services Option Pack

The key material for your Security World keys is protected by your nShield HSM. However, as with all secure systems, administrators must remain diligent when it comes to who uses the system. All network traffic between Web Services Option Packs and clients using the REST API is passed via a secure channel, setup using TLS 1.2 with mutual authentication. Entrust make the following security related recommendations:

- Protect your server's TLS certificate private key by storing this in the nShield HSM. This will provide extra protection for the confidentiality of this key, even in the event of server comprise.
- Do not change the default ownerships or permissions for the TLS directories and files set by the WSOP installer:
  - owner (root): Full permission.
  - group: Read permission, not write.
  - others: No permissions.
  - All TLS certificate private keys are protected using file permissions. When providing your own keys, `corecrypto` keys should be owned by the `root` user and the `wsopd` group. Other users should not be able to read or write these files. Care should be taken that private key files that live on the disk are not included in any backup, as this will compromise the confidentiality of the keys.
- The path under `/opt/nfast/webservices/corecrypto` must be owned by `wsopd` user and group. Only `wsopd` user should have permission to write to files in this directory.
- Ensure log levels are set appropriately for your environment. More verbose log levels may expose information that is useful for auditing who uses Web Services Option Pack, but the log information will also contain which REST crypto API operations have been performed. While this may be useful for diagnostics, be aware that this log information could be considered sensitive and should be suitably protected when stored.
- The logs files may grow quickly if left unattended for long time. Whoever maintains the web service is responsible for log rotation.
- Private key material for Security World keys is not exposed by Web Services Option Pack, it is however, usable by remote clients. Ensure client authentication is enabled at all times.
- If using client certificates, create and use a separate root certificate and client certificate hierarchy that is dedicated to servicing clients of the Web Services Option Pack.
- Maintain the certificate revocation list, restricting clients that should no longer have access to your Security World keys.

- Use issued client certificates only in client applications, avoiding having them imported into regular web browsers, since this may enable a Cross Site Request Forgery attack.
- The Web Services tar file integrity can be verified by using the published hash. It is recommended that you verify the integrity of this file before executing it.
- The network environment of Web Services Option Pack should be suitably protected to maintain its availability (for example, through use of firewalls, intrusion detection/prevention systems).
- Ensure that the Web Services Option Pack's system clock is set accurately and can only be modified by an authorized system administrator, so that certificate lifetimes are correctly interpreted.
- Only authorized system administrators should have access to the Web Services Option Pack and only trusted software should be run on the platform hosting the Web Services Option Pack.
- Standard virus prevention and detection measures should be taken on the platform hosting the Web Services Option Pack.
- CRL revocation list shall be maintained continuously by pulling the CRL at regular intervals, or whenever it is updated. Web service operation should not be permitted if this cannot be performed.
- The Web Services Client Certificates must be created only for approved applications.
- Ensure that the integrity of the non-sensitive Security World resource data that exist in WSOP's database is protected and that access to this database is restricted to authorised and trusted entities.
- Ensure that regular backups are taken of WSOP's database and that these backups are stored in a secure location that only authorised entities can access.
- Ensure that any authorised entity accessing WSOP's database is provided with the least privileges required to perform their legitimate function.
- Ensure that all the components of the WSOP system, including the database, are protected from physical or logical access by unauthorised entities.

14. Base64url Encoding

Base64url encoding is a method of encoding binary data for safe, unambiguous transmission as plain text, for example in URLs. Full details of how to encode and decode are detailed in https://tools.ietf.org/html/rfc4648#section-5. Since standard base64 encoding uses characters that are reserved for other uses in URLs, a variant has been created that uses alternative characters that do not require escaping when forming part of a URL. Padding characters are also removed and can be derived through the length of the base64url-encoded data, as specified in https://tools.ietf.org/html/rfc7515#appendix-C.

The python script below demonstrates how to encode a file to base64url-encoded format:

```
#!/usr/bin/python

import base64
import sys

with open(sys.argv[1], "rb") as input_file:
    in_data = input_file.read()
    out_b64 = base64.urlsafe_b64encode(in_data).strip("=")
    print out_b64
```

The python script below demonstrates how to pad and decode:

```
#!/usr/bin/python

import base64
import sys

pad = {0: "", 2: "==", 3: "="}

with open(sys.argv[1], "rb") as input_file:
    in_b64u_stripped = input_file.read()
    mod4 = (len(in_b64u_stripped)) % 4
    if (mod4 == 1):
        raise ValueError("Invalid input: its length mod 4 == 1")
    b64u = in_b64u_stripped + pad[mod4]
    out_data = base64.urlsafe_b64decode(b64u)
    print(out_data)
```

15. Uninstalling the Web Services Option Pack

Before uninstalling Web Services Option Pack, Entrust advises that you create a full back up of `/opt/nfast/webservices`. Removing Web Services Option Pack will not affect your existing Security World, its keys, or your Security World Software installation. To uninstall Web Services Option Pack, run the following command:

```
/opt/nfast/webservices/sbin/install -u
```

This ensures that the Web Services Option Pack server is stopped and will not automatically restart if the machine is rebooted. To remove Web Services Option Pack fully, remove the following files and directories:

- `/opt/nfast/webservices/`
- `/opt/nfast/scripts/install.d/67corecrypto`
- `/opt/nfast/lib/versions/webservices-atv.txt`
- `/opt/nfast/lib/versions/corecrypto-atv.txt`

If required, you can also remove the `wsopd` user and group that was created as part of the installation.

16. Upgrading the Web Services Option Pack

16.1. Upgrading the Web Services Option Pack from v3.0 onwards

If you are upgrading from WSOP v3.0, you need to run `dbmt upgrade` first. See Upgrade.

1. Back up the WSOP configuration and the database.
2. Stop all instances of the Web Services Option Pack and ensure that nothing is using the database.
3. Uninstall the currently installed version of the DBMT and install the new version (see WSOP Database Management Tool).
4. Run `migrate-schema`. See Migrate schema for more information.

The migrate-schema tool must be run before installing the latest WSOP release or migrating anything new into the database.

5. Uninstall any currently installed versions of the Web Services Option Pack (see Uninstalling the Web Services Option Pack).
6. Install the required version of the Web Services Option Pack (see Install the Web Services Option Pack).
7. Start the new corecrypto service (see Start the Web Services Option Pack).

16.2. Upgrading the Web Services Option Pack from releases before v3.0

If you are upgrading from WSOP v1.0, Java is no longer required and can be removed from the system if needed.

1. Back up the WSOP configuration.
2. Stop all instances of the Web Services Option Pack.
3. Uninstall the currently installed version of the DBMT and install the new version (see WSOP Database Management Tool).
4. Run `migrate`. See Migration for more information.
    - The migrate tool must be run before installing the latest WSOP release.
5. Uninstall any currently installed versions of the Web Services Option Pack (see Unin-

stalling the Web Services Option Pack).

6. Install the required version of the Web Services Option Pack (see Install the Web Services Option Pack).
7. Start the new corecrypto service (see Start the Web Services Option Pack).

As described previously, in the current WSOP release the keys and tokens are stored in a database.

Older WSOP versions (WSOP2.1 and earlier) store the keys in file in `/opt/nfast/kmdata/local` folder.

Before using the current WSOP release, you must migrate your `kmdata` folder using the Data base Management Tool. See WSOP Database Management Tool for more information.

Older WSOP versions (WSOP2.1 and earlier) supported OCS-protected keys, however the current release of WSOP does not. Your OCS-protected keys **will not** be migrated.

The current WSOP release will install a new configuration file, which includes new configuration options. You may need to transfer some configuration options from your existing configuration. Special attention should be paid to your security configuration, including certificates and, if using an API Gateway, any other client identity information to be passed to the gateway. Configure the Web Services Option Pack describes the configuration in more detail.

For previous versions of WSOP, the configuration file was installed to `/opt/nfast/wsop/corecrpto/conf/`. Please note any path changes when transferring previous values to the new WSOP 3.3.1 configuration.

Once migration is completed, WSOP is ready to use.

17. Supported algorithms

17.1. Key types

This version of WSOP supports the following key algorithms:

Algorithm	Key Type	Notes
AES	AES or Rijndael	
ECDSA	ECDSA	Supported curves are: • P-256 • P-384 • P-521
RSA	RSA	
SHA-256 HMAC	HMACSHA256	
SHA-384 HMAC	HMACSHA384	
SHA-512 HMAC	HMACSHA512	

17.2. Signing algorithms

The following supported signing and verification algorithm identifiers are aligned to the JOSE IANA registry:

- RS256
- RS384
- RS512
- PS256
- PS384
- PS512
- HS256
- HS384
- HS512
- ES256
- ES384
- ES512

17.2.1. Product-specific signing algorithms

The following algorithms are available from the ECDSA-SHAXXX suite:

- ECDSA-SHA256
- ECDSA-SHA384
- ECDSA-SHA512

Each of these elliptic curve (EC) algorithms for signing and verification uses the correspond ing hash algorithm from the SHA2 family, and each allows the user to specify a key created using any one of the curves P-256, P-384, or P-521. This free choice contrasts with the Ellip tic Curve JSON Web Algorithms (ES256, ES384, ES521). Each of those uses a matched pair of EC and hash function (P-256 with SHA-256, P-384 with SHA384, P-521 with SHA512) so that there is a close match in cryptographic security strength between the EC public/pri- vate key cryptography and the hash algorithm.

The security strength of any signature created using the ECDSA-SHAXXX suite will be the strength of its weakest part.

17.3. Encryption algorithms

The following supported encryption and decryption algorithm identifiers are aligned to the JOSE IANA registry:

- RSA1_5
- RSA-OAEP
- RSA-OAEP-256
- RSA-OAEP-384
- RSA-OAEP-512
- A128CBC
- A192CBC
- A256CBC
- A128CBC-NOPAD
- A192CBC-NOPAD
- A256CBC-NOPAD
- A128GCM
- A192GCM
- A256GCM

18. Supported TLS cipher suites

This appendix and the WSOP configuration file both use the OpenSSL project's identifiers for TLS Cipher Suites.

Recommended Cipher Suites: The Default List

The following TLS Cipher Suites are supported by WSOP and are configured for use by default. It is strongly recommended that this default set of cipher suites, or a subset of it, is used.

- ECDHE-ECDSA-AES128-GCM-SHA256
- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES256-GCM-SHA384
- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-ECDSA-CHACHA20-POLY1305
- ECDHE-RSA-CHACHA20-POLY1305

Less Secure Cipher Suites: Not Recommended

The following TLS Cipher Suites are supported by WSOP, but only if explicitly configured for use by the user. These are less secure cipher suites and should only be configured for use after a thorough threat analysis of the operating environment.

- ECDHE-RSA-AES128-SHA256
- ECDHE-ECDSA-AES128-SHA256
- ECDHE-RSA-AES256-SHA
- ECDHE-RSA-AES128-SHA
- ECDHE-RSA-DES-CBC3-SHA
- ECDHE-ECDSA-AES256-SHA
- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-RC4-SHA
- ECDHE-ECDSA-RC4-SHA
- AES256-GCM-SHA384
- AES128-GCM-SHA256
- AES128-SHA256
- AES256-SHA
- AES128-SHA
- DES-CBC3-SHA

- RC4-SHA