Configuring IG for AM Tokens (and KeyStores)

Background

Issue Summary

The reason for producing this wiki is to document configuration of AM and IG token handling, with regards to the necessary key creation, keystore handling and AM and IG configuration required to secure them. This follows issues with finding the correct keystore configuration, procedures and keys in completing feature **OPENIG-2722** - Getting issue details... STATUS (pertaining to stateless access-tokens). The PR associated with this JIRA is also very informative on how to configure the environments.

KeyStores Basics

A **keystore** is a repository for securely storing:

- private keys - with a certificate chain containing the public key (asymmetric keys).
- and "secret" (symmetric) keys.
- identity certificates - containing public key certificates belonging to another party. Examples include PGP and X509.

This repository may be a specifically formatted text file or backed by some other mechanism. keystore access is available through the `java.security.KeyStore` class.

Keys are stored in a keystore with an associated "alias" that is used for identifying the key. The keystore may be password protected (with a storepass) and each private or secret key may be individually protected with its own password (keypass). There is no need to password-protect public keys.

A **truststore** is a keystore specifically for managing remote authentication (see JSSE Ref Guide). It's use is not important in the context of this discussion, so it won't be covered any further.

`keytool` is a java supplied command-line tool for performing operations on a keystore, such as creating keys, listing keys and importing and exporting.

More information on `KeyStores` can be found here:

- `KeyStore` javadoc
- `keytool` tech note
- Java cryptography tutorial

KeyStore Types

A `KeyStore`, as mentioned, may be formatted in a specific way, according to the its type:

- JKS (default pre-Java 9):
    - Java-based format.
    - Can store public/ private keys and certificates.
    - Cannot store secret keys.
- JCEKS:
    - Java-based format.
    - More secure than JKS.
    - Can store public/ private keys and certificates.
    - Can store secret keys.
- PKCS12 (default as-of Java 9):
    - Public cryptograhic standard format.
    - Can store public/ private keys and certificates.
    - Cannot store secret keys.

When using the keytool, it is necessary to specify the `-storetype` if not using JKS, otherwise you will get unhelpful messages about keystore formats and tampering.

More useful information on keystore types:

- pixelstech.net.
- The full set of available ksystore types is available in an java(8) security technote.

AM Configuration

Tokens

AM 6.0 supports the following client requests (Response Type):

- `code`. Specifies that the client application requests an authorization code grant.
- `token`. Specifies that the client application requests an implicit grant type and requests a token from the API.
- `id_token`. Specifies that the client application requests an ID token.
- `code token`. Specifies that the client application requests an access token, access token type, and an authorization code.
- `token id_token`. Specifies that the client application requests an access token, access token type, and an ID token.
- `code id_token`. Specifies that the client application requests an authorization code and an ID token.
- `code token id_token`. Specifies that the client application requests an authorization code, access token, access token type, and an ID token.

KeyStores

AM comes with two pre-configured `KeyStores`: `keystore.jks` and `keystore.jceks`. Which is used can be configured through the AM console:

- For full AM KeyStore configuration, see the AM Setup and Maintenance Guide.
    - `Deployment (top-level menu)  Servers  http://.../openam  Security (left-hand menu)  Key Store`
- To change the keystore, edit:
    - Keystore File: Uncheck lock and edit to ".jceks" file extension.
    - Keystore Type: Uncheck lock and change to "JCEKS".
    - Restart AM.
- The current KeyStore config is located in file `path/to/am/boot.json`.
- The AM KeyStore storepass and keypass are located in:
    - storepass: `path/to/am/.storepass`
    - keypass: `path/to/am/.keypass` - default keypass used ("changeit").
- The sample keystore set-up is located in AM source, here:
    - openam-server-only/src/main/webapp/WEB-INF/template/keystore
    - with an associated README (though it's not that helpful).

Keys (and Related Issues)

AM `KeyStores` come complete with provided test keys, as documented in the AM Setup and Maintenance Guide (specifically Table 5.1):

- Note that there is no further information on these keys, in order to recreate your own.
- Not all keys are available for export to a third-party (e.g. IG) keystore.

Supplied sample keys are

- AM test class `com.forgerock.openam.functionaltest.api.keys.TestKey` (openam-functional-test-apis) sets up the `keystore.jceks` keys.
    - Notably `rsajwtsigningkey` which can be used for JWT signing.
- AM test class `com.sun.identity.setup.TestKeys` (openam-core) serves to set up some test keys for demo purposes:
    - `SecretKey directenctest` - is anticipated to test token encryption:
        - `directenctest` is expected to be the HMAC encryption key for testing encryption but seemingly has an algorithm of RAW and cannot be imported into a PKCS12 keystore.
    - `SecretKey selfservingsigntest`

Other notes of interest:

- AM has an endpoint that can be used to list keys known to the AM keystore in use:
    - jwk_uri endpoint:

      AM jwk_uri endpoint

      ```
      curl "http://openam.example.com:8082/openam/oauth2/connect/jwk_uri" | jq
      ```

- It is necessary to restart AM after any key change - as AM caches keystore content on load.

> **Key changes**
>
> It is necessary to restart AM after any key change - as AM caches keystore content on load.

IG KeyStores

IG doesn't expressly need the configuration of any `KeyStores`, except if required by a particular `Filter` or component (e.g. the new `StatelessAccessTokenResolver`).

In the case that IG must use AM provided keys (e.g. the above example) then the correct keystore type must be created and AM-generated keys must be made available to it. In the case of then either a signing key (for verification) or an encryption key (for decryption) must be provided.

Sharing Keys between AM and IG (KeyStores)

Using the example of the `StatelessAccessTokenResolver`, AM would be configured to produce access tokens that are either Encrypted *or* Signed JWTs (Encrypted takes precedence in AM config). As such, depending on how the JWT is secured, we would need to generate an encryption key or a signature key in the AM `KeyStore`. Again, depending on how the JWT is secured, we would then need to import the respective key into the IG keystore:

- If the access-token is Signed with an RSA private key, then we would need to import the corresponding public key into the IG keystore.
- If the access-token is Encrypted - which AM always does with a `SecretKey` - then we would need to import the self-same `SecretKey` into the IG keystore.

Some points to note - or emphasise - here:

- AM signs Signed access-tokens with the private key and IG would need the corresponding public key to verify the signature
- AM encrypts access-tokens (currently and for the forseeable future) with a `SecretKey` (symmetric) so IG must use this same key to decrypt it.
- Selection of which type of keystore AM should use is determined by the means of securing the token:
    - for a Signed access-token, a JKS keystore is sufficient (private key with a certificate).
    - for Encrypted access-tokens, the choice of AM keystore to configure is decided by our need to store `SecretKeys`. JKS keystores cannot store `SecretKeys` so AM has to be configured to use a JCEKS keystore at a minimum.
- Selection of which type of keystore IG should use is determined by the means used by AM of securing the token *as well as*being able to import the corresponding key into its keystore:
    - For a Signed access-token, a JKS keystore is sufficient (as it can store the trusted certificate).
    - For an Encrypted access-token, it is necessary to be able to export the `SecretKey` from the AM keystore and import it into the IG keystore.
        - This means the IG keystore must be of type JCEKS or PKCS12.
        - In testing `StatelessAccessResolver`, PKCS12 was used as the IG keystore.

With this in mind, the following snippets show how to configure a signature or encryption key in an AM keystore (JCEKS) and import the corresponding key into an IG keystore (JCEKS or PKCS12).

Configuring Signature keys

The following code-block illustrates how to export a signature certificate from AM (keystore.jks) to an IG PKCS12 keystore (or a JCEKS keystore would be applicable):

Configuring a signature key

```
# create verify-key03 (RSA 2048) in AM JKS keystore
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -genkey -alias verify-key03 \
    -dname "CN=openig.example.com, OU=example, O=com, L=fr, ST=fr, C=fr" \
    -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openam/openam-embedded-DJ/openam
/keystore.jceks" \
    -storetype JCEKS \
    -storepass "qWPzxXdIF0IaD/6Q9Bp7vr32oUK0H8h8" \
    -keypass changeit \
    -keyalg RSA -keysize 2048

# export verify-key03 to .pem
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -exportcert -rfc -alias verify-key03 \
    -file "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/openig-container/apache-tomcat-
8.0.46/conf/verify-key03-cert.pem" \
    -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openam/openam-embedded-DJ/openam
/keystore.jceks" \
    -storetype JCEKS \
    -storepass "qWPzxXdIF0IaD/6Q9Bp7vr32oUK0H8h8" \
    -keypass changeit

# import verify-key03 .pem to IG PKCS12 keystore
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -import -trustcacerts -rfc -alias
verify-key03 \
    -file "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/openig-container/apache-tomcat-
8.0.46/conf/verify-key03-cert.pem" \
    -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/ig_instance_dir/config
/IG_keystore.p12" \
    -storetype PKCS12 \
    -storepass "keystore"

# list content of IG PKCS12 to confirm key present
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -list \
    -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/ig_instance_dir/config
/IG_keystore.p12" \
    -storetype PKCS12 \
    -storepass keystore
```

Configuring Encryption key

The following code-block illustrates how to export an encryption `SecretKey` from AM (keystore.jks) to an IG PKCS12 keystore (or a JCEKS keystore would be applicable):

Configuring an encryption key

```
# create secret key enckey07 (RSA 2048) in AM JKS keystore
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -genseckey -alias enckey07 \
    -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openam/openam-embedded-DJ/openam
/keystore.jceks" \
    -storetype JCEKS \
    -storepass "qWPzxXdIF0IaD/6Q9Bp7vr32oUK0H8h8" \
    -keypass changeit \
    -keyalg AES \
    -keysize 256

# export enckey07 to .pem - using keytool exportseckey --> !!!doesn't work!!!
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -exportseckey -alias enckey07 \
   -file "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/openig-container/apache-tomcat-
8.0.46/conf/enckey03-secretkey.pem" \
   -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openam/openam-embedded-DJ/openam
/keystore.jceks" \
   -storetype JCEKS \
   -storepass "qWPzxXdIF0IaD/6Q9Bp7vr32oUK0H8h8"  \
   -keypass changeit

# list content of AM keystore.jceks to confirm key present
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -list  -v \
   -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openam/openam-embedded-DJ/openam
/keystore.jceks" \
   -storetype JCEKS \
   -storepass "qWPzxXdIF0IaD/6Q9Bp7vr32oUK0H8h8"

# import enckey07 key (direct from keystore) to IG PKCS12 keystore
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -importkeystore \
   -srcalias enckey07 \
   -srckeystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openam/openam-embedded-DJ/openam
/keystore.jceks" \
   -srcstoretype JCEKS \
   -srcstorepass "qWPzxXdIF0IaD/6Q9Bp7vr32oUK0H8h8" \
   -destalias enckey07 \
   -destkeystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/ig_instance_dir/config
/IG_keystore.p12" \
   -deststoretype PKCS12 \
   -deststorepass "keystore" \
   -destkeypass "keystore"

# list content of IG PKCS12 to confirm key present
/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/keytool -list  -v \
    -keystore "/Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig/ig_instance_dir/config
/IG_keystore.p12" \
    -storetype PKCS12 \
    -storepass "keystore"
```

Further Steps

- After adding new keys to an AM keystore, it is necessary to restart AM (as keystore contents are cached on load by AM).

Configuring AM

This assumes the keystore is configured correctly:

- References the keystore where the keys have been created (refer back to KeyStores section).
- AM has been restarted following keystore and key changes.

Additionally:

- Token security is configured in two places, depending on what is being configured:
  - *OAuth2 Provider service* configuration: *Advanced* tab.
  - *OAuth2 Client application* configuration: *Advanced* and *Signing and Encryption* tabs.
  - N.B. There is also a n *OAuth2 Provider* config screen under *Configuration* (which is also slightly different to the service UI) but this seems to be overridden by the *OAuth2 Provider service* configuration.
- AM configures access-token security such that encryption takes precedence over signature. That is, if both are configured, encryption is used if switched on, otherwise signature is used.

Securing Access Tokens

OAuth2 Provider Configuration

OAuth2 Provider configuration: `Realm  Configure OAuth2 Provider`

Core tab (and Create screen)

`Use Stateless Access & Refresh Tokens` = true

`Access Token Lifetime (seconds)` = <sensible lifetime for testing - 10secs>

`Issue Refresh Tokens` = False (unimportant but does not need to be set)

`Issue Refresh Tokens on Refreshing Access Tokens` = False (unimportant but does not need to be set)

`Issue Refresh Tokens` = False (unimportant but does not need to be set)

It is also necessary to add appropriate scopes. For example:

```
"email|Your email address", "openid|", "address|Your postal address", "phone|Your telephone number(s)",
"profile|Your personal information"
```

Securing an Access Token with a Signature

Configuring signature

`OAuth2 Token Signing Algorithm` = RS256 as per verify-key03 config

`Token Signing RSA Public/Private Key Pair` = verify-key03 Asymmetric key configured earlier.

`Stateless Token Compression` = can be either True or False

`Enable Stateless Token Encryption` = False essential to ensure signing configuration is used

Securing an Access Token with Encryption

Configuring encryption

`Enable Stateless Token Encryption` = True

`Token Encryption Secret Key Alias` = enckey07 Symmetric key as you configured earlier

`Subject Identifier Hash Salt` = changeme

OAuth2 Client Configuration

OAuth2 Client configuration: `Realm  Applications  OAuth2`

Ensure support for access-tokens

`Advanced` (tab)

    `Grant Types` = should include "Resource Owner Password Credentials"

    `Response Types` = should include "code token" for access-tokens.

`Signing and Encryption` (tab)

    `Public key selector` = X509

Additional notes:

- The `Signing and Encryption` tab includes a lot of token and "algorithm fields". For access tokens, these do not need to be configured. For instance:
    - "ID Token" fields may be left default or empty
        - If empty, ID Token Signing Algorithm = none
    - "User info ... algorithm" fields may be left default or empty
    - "Request parameter ... algorithm" fields may be left default or empty

Configuring IG

IG configuration `KeyStore` heaplet would reference the above IG keystore and keystore type. For example:

Sample heap config

```
    "heap": [
        {
            ...
        },
        {
            "config": {
                "type": "PKCS12",
                "password": "keystore",
                "url": "file:///Users/wayne.morrison/dev/pyforge/results/20180723-114228/Filters/openig
/ig_instance_dir/config/IG_keystore.p12"
            },
            "name": "IgP12KeyStore",
            "type": "KeyStore"
        },
        ...
```

Specific key configuration would be added in the `Filter` or other component configuration. For example, for `StatelessAccessTokenResolver`:

StatelessAccessTokenResolver config example

```
{
    "type" : "StatelessAccessTokenResolver",
    "config" : {
        "issuer" : "http://openam.example.com:8082/openam/oauth2/realms/root/realms/filters_realm",
        "tokenName" : "access_token",
        "signature" : {
             "alias" : "verify-key03",
             "password" : "keystore",
             "keystore" : "IgP12KeyStore"
        },
        "_encryption" : {
            "alias" : "enckey07",
            "password" : "keystore",
            "keystore" : "IgP12KeyStore"
        }
    }
}
```

- This references alias "verify-key03" to use as the public key to verify the signature of the access token.
- There is also a commented section ("_encryption") indicating the `SecretKey` to use to decrypt the access token (if encrypted).

Demonstration and Testing

You can obtain a token from AM (oauth2/access_token endpoint) using the following:

```
http -a test_OAuth2ResourceServerFilter:password POST openam.example.com:8082/openam/oauth2/realms/root/realms
/filters_realm/access_token username=demo password=changeit grant_type=password scope=openid --form  | jq  '.
["access_token"]'


# capture it in $TOKEN for later use
TOKEN=`http -a test_OAuth2ResourceServerFilter:password POST openam.example.com:8082/openam/oauth2/realms/root
/realms/filters_realm/access_token username=demo password=changeit grant_type=password scope=openid --form  | jq
'.["access_token"]' | tr -d '"'`
```

- This example is configured inside a realm named "filters_realm".
- The $TOKEN can then be used in calls to IG.

Please refer to Sample `StatelessAccessTokenResolver` setup and configuration (PR) for more thorough instructions on testing (upon which this document was based).

Documentation

- AM
  - Setup and Maintenance Guide: Configuring KeyStores
  - OAuth2 Guide: Stateless OAuth 2.0 Access and Refresh Tokens
  - OAuth2 Guide: Configuring Stateless OAuth 2.0 Token Encryption
  - OAuth2 Guide: Configuring Digital Signatures
  - OAuth2 Guide: Reference: Provider Settings
  - OAuth2 Guide: Reference: Client Settings
- IG
  - Gateway Guide: Acting as an OAuth 2.0 Resource Server
  - Sample `StatelessAccessTokenResolver` setup and configuration (PR)

Troubleshooting

1. **Key configured in AM not found:**
   a. Confirm the correct configured keystore is the same as the one where the keys were added.
   b. Restart AM after keys are added.
2. **Issue creating a specific key type in a keystore:**
   a. The keystore type does not support the type of key being created:
      i. For instance JKS does not support `SecretKey`
3. **Issue importing a specific key type into a new keystore:**
   a. The new keystore type does not support the type of key being imported:
      i. For instance JKS does not support `SecretKey`
4. **`keytool` operation fails with error indicating incorrect format or that tampering has occurred:**
   a. `java.io.IOException: Invalid keystore format`

   b. Use the `-storetype` setting to ensure you're performing the operation on an expected keystore type.
5. **Issue importing keys from one keystore to another:**
   a. Ensure the `-srcstoretype` and `-deststoretype` settings are correct for the source and target keystores.
   b. Ensure that the destination keystore can accommodate the algorithm used in the source key.
      i. If it cannot then... ??????
      ii. This is the issue that occurred while trying to import the AM test key "directenctest". It wasn't clear what algorithm was used.
6. **AM fails to encrypt a key, complaining of a "Invalid offset/length combination":**
   a. The key specified has a keysize that cannot be used in combination with the key type. Try a larger key size
   b. I found that an AES key of 256b was fine.
7. **AM issue when creating an access token - complains of no signature even though encryption is being used:**
   a. This actually relates to the id-token configuration
      i. Ensure that if `ID Token Signing Algorithm` is not being used then it is set to "none" (not left empty).