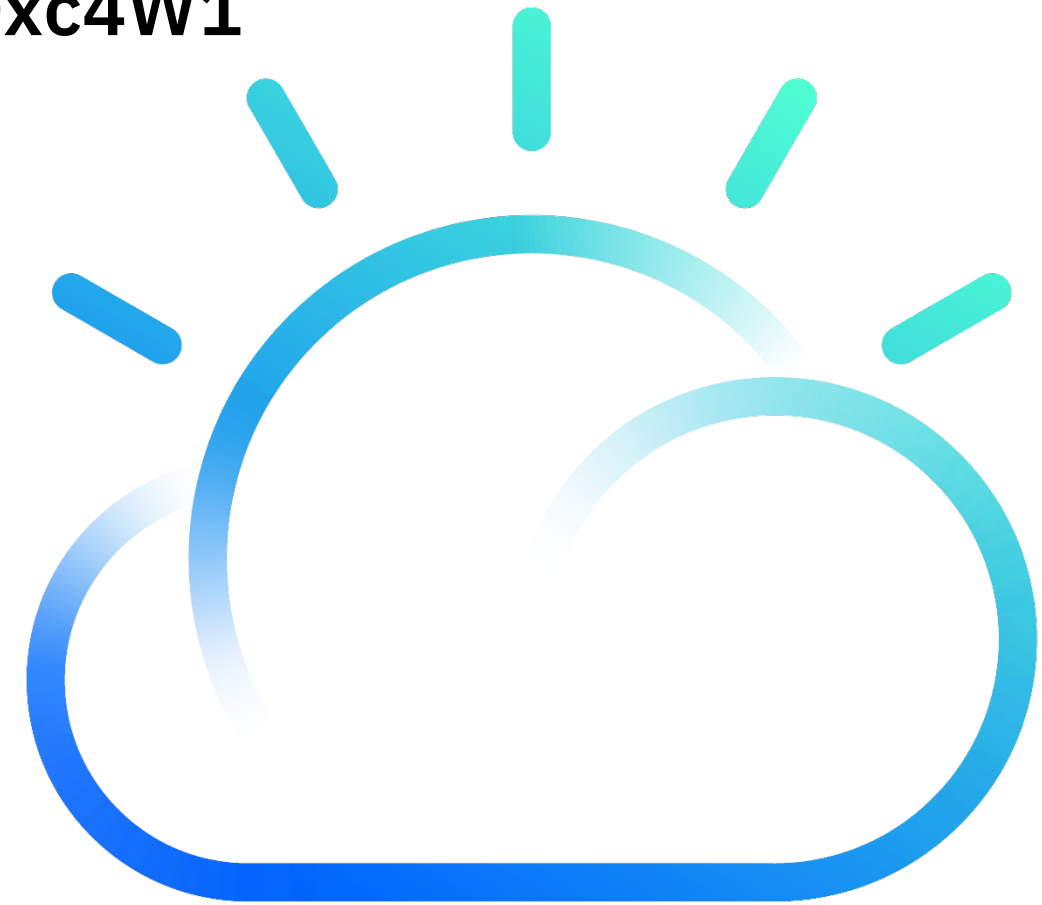
OpenSSL changes in Client SDK4.50xc4W1



Wednesday , 21st October 2020

Sunil Kajarekar
Informix Test Engineer
HCL Software

IBM Cloud

Disclaimers

IBM’s statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM’s sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user’s job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

Agenda

What is changed in SSL communication.
How to migrate existing client keystore.
How to create new client keystore.
How to create new (Connection Manager) keystore.

What is changed

SSL/Crypto libraries
IBM GSKit
Newer OpenSSL

Why openssl instead of Gskit

Reasons that are advantageous for customers:

GSKit is IBM proprietary and as such neither freely nor separately available

Instead, GSKit is packaged and - if required - installed with each product that needs it

Delivering (security) fixes requires a product release or special build

OpenSSL is Open Source: publicly available and with short release/improvement cycles

Customer has more control over:

What release version of OpenSSL to use,

How and where it is installed and how it is configured

Customer can get and install new OpenSSL release versions and benefit from them immediately, without waiting for an IBM product release or special build

ClientSDK4.50.xc4W1

Before ClientSDK 4.50.xC4W1 (i.e. up to 4.50.xC3):

All ClientSDK installations only use GSKit

Beginning with ClientSDK 4.50.xC4W1:

A ***stand-alone*** installation of a database client uses **OpenSSL** as encryption library

Whereas a database client installation ***co-located with*** a database server installation still uses **GSKit.**

"*co-located with*" means installed in the same directory as the database server.
and therefore the GSKit from the database server installation is available for the Client SDK as well.

Informix Server always uses GSKit only, even with 14.10.xC4W1.

In the upcoming 4.50.xC5 Client SDK fixpack, the default install will be GSKit with the option to switch to OpenSSL.

SSL/TLS Keystore and its handling

Database client using a **stand-alone Client SDK**:

	CSDK 4.50.xC3 and older	Action	CSDK 4.50.xC4W1
Keystore	"CMS" (GSKit proprietary ***.kdb*** file)	Convert	PKCS#12 (****.p12*** file) only
	PKCS#12 (open standard ****.p12*** file)	N/A	
Optional password stash file	****.sth*** file (GSKit secret)	Create new file with "onkstash"	****.stl*** file (Informix secret)

Database client using server **co-located Client SDK** installation: no change

SSL/TLS keystore and its handling (continued)

SSL/TLS keystore for database client:

- Contains CA certificate(s) for the authentication of the database server during the SSL handshake when connecting to the server
- A keystore for Connection Manager also contains its user private key and matching user certificate (During SSL handshake this user certificate is sent to connecting database client to authenticate CM itself).
- Keystore access is always password protected

Keystore handling:

- Database clients only read the keystore, but need the password to access it
- To avoid having to provide the password for each keystore access, the password can be "stashed"
- Administration (i.e. creation) of keystores is done with GSKit or OpenSSL tools

SSL/TLS keystore and its handling (continued)

Keystore password stashing:

- Password is saved in an encrypted format in a "stash file“
- The encryption algorithm is secret
- The SSL function layer can retrieve the stashed password and thus use it to access the keystore during the SSL handshake
- There is no way for the user to retrieve the password from the stash file

Differences between openssl and gskit for clientsdk

Keystore:

- GSKit supports the standard PKCS#12 format (*.p12 file) as well as the GSKit proprietary "CMS" format (*.kdb file)
- OpenSSL supports the standard PKCS#12 format (*.p12 file)
- Existing *.kdb keystores need to be converted for use with OpenSSL
- Existing *.p12 files do not need conversion
- Unfortunately, up to 14.10.xC3 our documentation propagated "CMS" keystores in *.kdb files (even though GSKit already supports the PKCS#12 format)
- (no difference in directory location of the keystore)

Differences between OpenSSL and GSKit for Client SDK (continued)

Password stashing:

- The secret password encryption algorithms used with GSKit and OpenSSL are different
- Consequently, the stash files are named differently:
  - GSKit stash files: *.sth
  - OpenSSL stash files: *.stl
- As the encryption algorithms are secret and different:
  - A password stashed with GSKit cannot be used with OpenSSL
  - And vice versa
- OpenSSL itself does not know about password stashing, but CSDK does!
- For use with OpenSSL, the password must be stashed anew using "onkstash"
- (no difference in directory location of the stash file)

Differences between OpenSSL and GSKit for Client SDK (continued)

Keystore administration:

- For GSKit, generally the "gsk8capicmd" utility is used:
  - (only) GSKit knows the secret password encryption algorithm, "gsk8capicmd" can do the password stashing as well as retrieve the stashed password to access the keystore
- For OpenSSL, generally the "openssl" utility is used:
  - But, as OpenSSL does not know about password stashing, the utility "onkstash" must be used to stash the password.
  - In order to use the "openssl" utility for an existing keystore, the password must be known and supplied to the "openssl" utility.
- To stash the password for OpenSSL during a keystore conversion from GSKit, the password for the PKCS#12 keystore must be known when running "onkstash".
- If the password of a GSKit *.p12 keystore is not known (but only in the GSKit stash file), then GSKit must be used to change the password. That way the new password is known (again). Then it can be stashed for OpenSSL using "onkstash".

Migrating an existing client keystore that was created with GSKit

If your database client installation is co-located with the database server installation, the database client continues to use GSKit as encryption library.

In this case, keystore migration is not necessary.

Migrating an existing client keystore that was created with GSKit (continued)

If your database client uses a stand-alone installation of Client SDK 4.50.xC4W1, then it will now use OpenSSL as encryption library.

In this case:

Make sure, an appropriate version of OpenSSL is installed before you install Client SDK 4.50.xC4W1.

OpenSSL version requirements (from CSDK release notes):

- For standalone installations, minimum OpenSSL version 1.0.x needs to be installed on the system.

Migrating an existing client keystore that was created with GSKit (continued)

If your client keystore has the GSKit-proprietary format "CMS" (usually with file extension "*.kdb"), then this keystore needs to be converted to a PKCS#12 keystore:

- As the CMS format is GSKit-specific, you need the GSKit command "gsk8capicmd"(or "gsk7capicmd") in order to convert the keystore.

Use a command like:

- ```
  gsk8capicmd -keydb -convert -db KEYSTOREFILE.kdb -pw PASSWORD \
  -old_format cms -new_db KEYSTOREFILE.p12 -new_pw PASSWORD \
  -new_format pkcs12
  ```

If the password for the existing *.kdb keystore is not known (but only in the stash file), then use "-stashed" instead of "-pw PASSWORD".

In any way, the password for the converted PKCS#12 keystore will be as given with "-new_pw ...", and therefore is known for the following step.
  ```

Migrating an existing client keystore that was created with GSKit (continued)

Create a stash file with the keystore password for use with OpenSSL:

- Use the new utility "onkstash" contained with Client SDK 4.50.xC4W1 (or newer) to stash the keystore password:
- *onkstash KEYSTOREFILE.p12 PASSWORD*
- This step is also needed in case your keystore already had the PKCS#12 format.

If you do not know the password for an existing PKCS#12 keystore created with GSKit (but the password is only in the stash file), then:

- First change the password with a command like:
- *gsk8capicmd -keydb -changepw -db KEYSTOREFILE.p12 -stashed \ -new_pw PASSWORD*

Then run the above onkstash command

Creating a new client keystore from scratch using OpenSSL

Log on to the database server machine and extract the certificate from the server's PKCS#12 keystore:

- Server's CA certificate is signed by CA authority:
  - `openssl pkcs12 -in $INFORMIXSERVER.p12 -passin pass:SERVERPASSWD \`
    `-out SSL_KEYSTORE_LABEL.cert.pem -cacerts –nokeys`
- Server's CA certificate is self signed:
  - `openssl pkcs12 -in $INFORMIXSERVER.p12 -passin pass:SERVERPASSWD \`
    `-out SSL_KEYSTORE_LABEL.cert.pem –nokeys`

If the server's certificate is self-signed, then the resulting output PEM file most probably only contains this single certificate.

If the server's keystore is a *.kdb file of "CMS" format, you need to use an equivalent "gsk8capicmd", e.g.:

- `gsk8capicmd -cert -extract -db $INFORMIXSERVER.kdb \`
  `-pw SERVERPASSWD -label SSL_KEYSTORE_LABEL \`
  `-target SSL_KEYSTORE_LABEL.cert.pem -format ascii`

... (continued on next slide)

Creating a new client keystore from scratch using OpenSSL (continued)

If your server's **“CMS” format** keystore contains additional CA certificates that were used to sign the server's certificate:

- If you do not already have such additional CA certificates in a separate PEM file, then you need to extract each of these CA certificates with a command like the above - using the appropriate label for each one of them.
  - hint: a command of the following form lists the labels contained in a keystore:
  - `gsk8capicmd -cert -list -db $INFORMIXSERVER.kdb -pw PASSWORD`
- Then you can combine all the CA certificates in the PEM files into a single PEM file,e.g. with an OS command like "cat".

Creating a new client keystore from scratch using OpenSSL (continued)

Transfer the output file SSL_KEYSTORE_LABEL.cert.pem (and, if applicable, the combined PEM file with the additional CA certificates) from the server machine to the client machine

Create the client keystore using the exported certificate(s) in the PEM file as input:

- *openssl pkcs12 -export -out client.p12 -passout pass:CLIENTPASSWD \ -in SSL_KEYSTORE_LABEL.cert.pem -caname LABEL1 [-caname LABEL2 …]*
- If you want to include additional (default) CA certificates from another PEM file into your keystore, add the option "-certfile <PEMFILE>" to the above command.
- If you have multiple certificates, you need to specify the option "-caname …“ multiple times to give a label to each CA certificate contained in the input PEM file(s).

Use the utility "onkstash" to stash the keystore password:

- *onkstash client.p12 CLIENTPASSWD*

Connection Manager (CM) and its keystore

CM is “in the middle” between database client and database server

Regarding SSL/TLS communication, it is both:

- Client (towards the DB server) and
- Server (towards the DB client)

CM is part of Client SDK:

- In 4.50.xC4W1, a stand-alone CM uses OpenSSL (but CM installed in same directory as Informix server continues to use GSKit)
- Keystore handling is basically the same as for a database client

(continued ...)

Connection Manager (CM) and its keystore (... continued)

Due to its double role, CM's keystore contains:

- CA certificate(s) needed to authenticate the DB server (i.e. for CM's client role)
  - Obtain such certificates and put them in the CM's keystore as you would for a DB client
- Its own user certificate and corresponding private key (i.e. for CM's server role to authenticate itself towards the DB client)
  - Obtain or create these as you would for the DB server and put them into CM's keystore
  - If CM's user certificate is self-signed:
    - Get CM's self-signed certificate and put it into the DB client's keystore
  - If necessary, obtain (additional) CA certificate(s) needed to authenticate CM and put them into the DB client's keystore (just as you would put the DB server CA certificates in the DB client's keystore)

Creating a new CM keystore from scratch using OpenSSL

Log on to the database server machine and extract the certificate from the server's PKCS#12 keystore:

- Server's CA certificate is signed by CA authority:
  - `openssl pkcs12 -in $INFORMIXSERVER.p12 -passin pass:SERVERPASSWD \`
    `-out SSL_KEYSTORE_LABEL.cert.pem -cacerts –nokeys`
- Server's CA certificate is self signed:
  - `openssl pkcs12 -in $INFORMIXSERVER.p12 -passin pass:SERVERPASSWD \`
    `-out SSL_KEYSTORE_LABEL.cert.pem –nokeys`

If the server's certificate is self-signed, then the resulting output PEM file most probably only contains this single certificate.

If the server's keystore is a *.kdb file of "CMS" format, you need to use an equivalent "gsk8capicmd", e.g.:

- `gsk8capicmd -cert -extract -db $INFORMIXSERVER.kdb \`
  `-pw SERVERPASSWD -label SSL_KEYSTORE_LABEL \`
  `-target SSL_KEYSTORE_LABEL.cert.pem -format ascii`

... (continued on next slide)

Creating a new CM keystore from scratch using OpenSSL (continued)

If your server's **“CMS” format** keystore contains additional CA certificates that were used to sign the server's certificate:

- If you do not already have such additional CA certificates in a separate PEM file, then you need to extract each of these CA certificates with a command like the above - using the appropriate label for each one of them.
  - Hint: a command of the following form lists the labels contained in a keystore:
  - *gsk8capicmd -cert -list -db $INFORMIXSERVER.kdb -pw PASSWORD*

Creating a new CM keystore from scratch using OpenSSL (continued)

Create a self-signed certificate for CM:

- *openssl req -x509 -newkey rsa:4096 -keyout cmKey.pem -out \ cmCert.pem -days 365 –nodes*
- It will ask you questions like Country, State, Locality, Organisation, Organisation Unit, **Common Name**, E-mail address, provide the details and proceed.

Then you can combine the CA certificates from server as well as CM's certificate generated in previous step (cmCert.pem) into a single PEM file, e.g. with an OS command like "cat".

Create a keystore for CM:

- *openssl pkcs12 -export -out $CMNAME.p12 -passout pass:CMPASSWD \ -inkey cmKey.pem -in certFile.pem -caname SERVERLABEL -caname CMLABEL*
- CMLABEL is value provided in **Common Name** while creating CM certificate.
- Use the utility "onkstash" to stash the keystore password:
  - *onkstash $CMNAME.p12 CMPASSWD*

Questions