



The Complete (?) View of z/OS Certificates

Wai Choi, CISSP

System SSL Design and Development

June 11, 2025

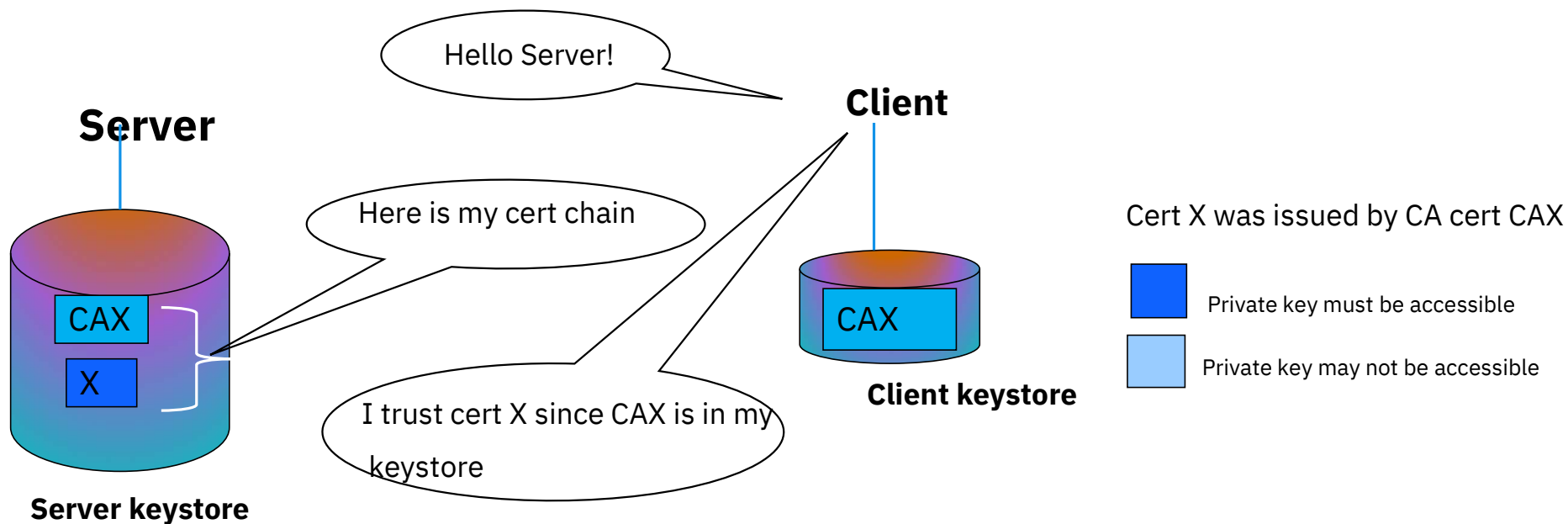
Agenda

- A typical workflow to set up secure communication through TLS from the keystore perspective
- How certificate and key in the keystore involve multiple components on z/OS
 - RACF, ICSF and CLiC, System SSL, Communication Server AT-TLS, PKI Services and JSSE
- Highlight the importance in each component

**This presentation is modified from the one I presented in the WW Security Conference and Guided Europe Share (GSE) last year.*

Feel free to post questions in RACF_L

Keystores are the basic set up needed for TLS handshake



Different terms are used as keystores:

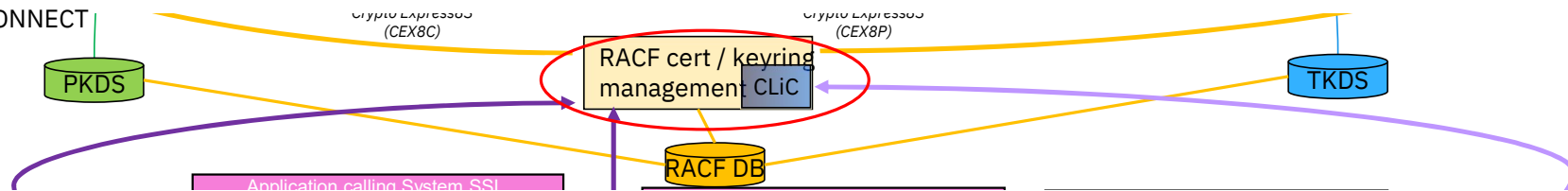
- Keyring (RACF), key database (System SSL), keystore+truststore (Java) PKCS#12 file (System SSL, Java), Token (ICSF)...
- The role is the same – a collection of certificate(s) and sometimes private key

RACF

The big picture – 1st stop – RACF keyring

RACF RACDCERT command – populating the key ring

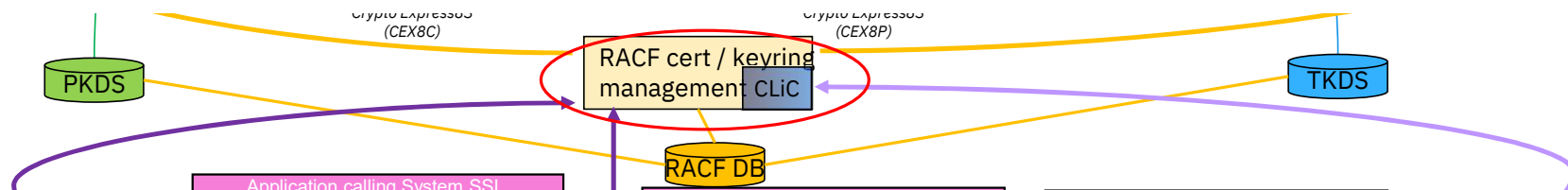
- There are 26 functions operating on certificate and the related objects including certificate request, keyring, token, mapping filter
- The most used functions are
 - GENCERT
 - GENREQ
 - ADD
 - ADDRING
 - CONNECT



- These functions get the certificate and the associated objects into the RACF DB
- The private key associated with the certificate can also be stored in the RACF DB, or in ICSF's PKDS or TKDS

Populate the keyring

- Series of RACDCERT functions is the first preparation for TLS communication using RACF keyring
 - RACDCERT ADDRING(...)
 - RACDCERT GENCERT /ADD (...) (multiple times for each certificate)
 - RACDCERT CONNECT(<cert>... <keyring>) (multiple times for each certificate)



- RACDCERT LISTRING(<keyring>)

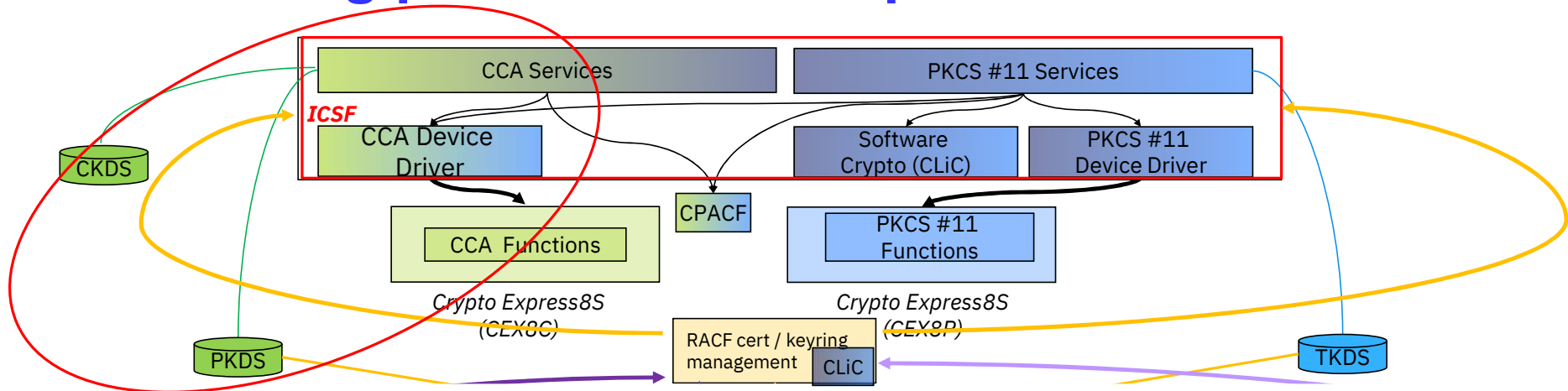
Ring:

```
>myRing<
```

Certificate Label Name	Cert Owner	USAGE	DEFAULT
myCA	CERTAUTH	CERTAUTH	NO
myIntCA	CERTAUTH	CERTAUTH	NO
myServerCert	ID(ftpd)	PERSONAL	YES

RACF, ICSF and PKI Services

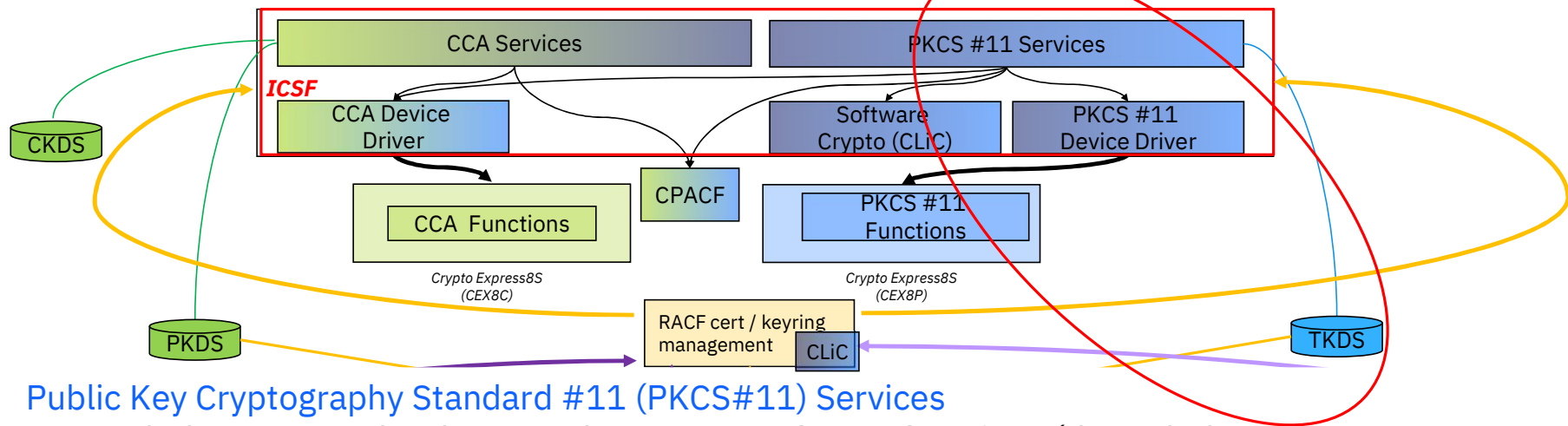
The big picture – next step ICSF



Integrated Cryptographic Service Facility (ICSF) is the driver of almost all the cryptographic operations. It provides two cryptographic services

- **Common Cryptographic Architecture (CCA) Services**
 - CCA manages two key datasets to store key materials
 - Cryptographic Key Data Set (CKDS) – for symmetric keys
 - Public Key Data Set (PKDS) – for asymmetric keys
 - All keys are secure keys protected under a master key

The big picture – next step ICSF



- **Public Key Cryptography Standard #11 (PKCS#11) Services**

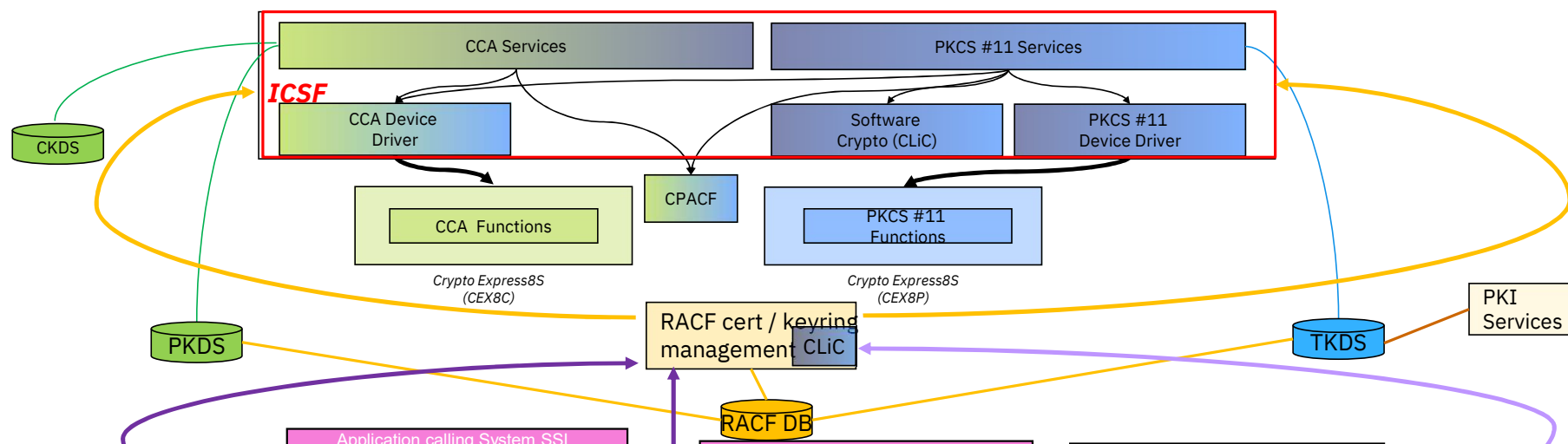
- Include PKCS#11 hardware and PKCS#11 software functions (through the Crypto Lite library in C (CLiC))
- Manage one key dataset to store key materials
 - Token Key Dataset (TKDS) – for asymmetric keys
 - Objects managed by CLiC are clear
 - That's why TKDS can contain both secure and clear objects



*You may use PKCS#11 services without the cryptographic coprocessors

*CPACF has a limited set of functions for the handshake process to improve the performance without hardware

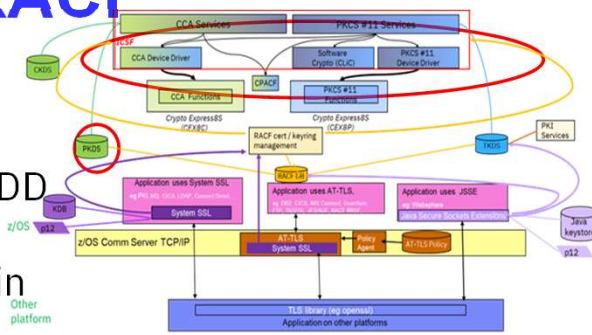
PKDS and TKDS managed by ICSF, RACF and PKI Services



- PKDS and TKDS are managed by ICSF
- But RACF can create PKDS/TKDS key that is associated with the certificate in RACF through the RACDCERT command
- PKI Services can create certificate with its associated key in TKDS too
- **Important advice – don't manipulate the private key object in the KDS created through RACDCERT or PKI Services from ICSF!!!**

ICSF PKDS secure key management through RACF

- Two key datasets are used by ICSF to store the asymmetric key materials
 - Public Key Dataset (PKDS)
 - When the keyword PKDS is specified in RACDCERT GENCERT or ADD command, the private key associated with certificate would be stored in the ICSF PKDS, referenced by a private key label stored in RACF DB
 - Eg, racdcert id(xxx) gencert ... rsa(pkds(*)) withlabel('pkdscert')
(Note: pkds(*) indicates the certificate label would be used as key label too)

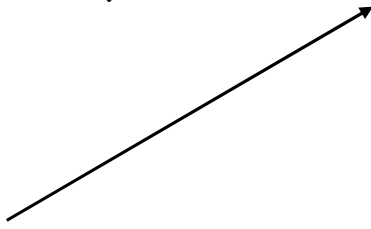


```
RACDCERT id(xxx)
list(label('pkdscert')):
```

```
Label: pkdscert
...
Key Type: RSA
Key Size: 2048
Private Key: YES
PKDS Label: PKDSCERT
Ring Associations:
*** No rings associated ***
```

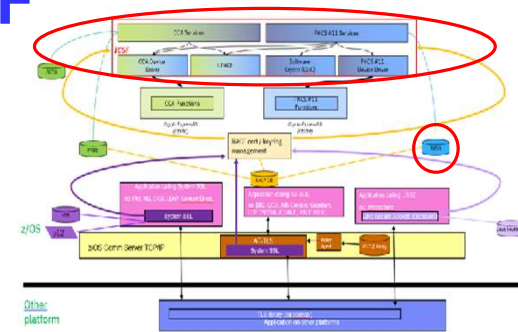
ICSF Utility panel - option 6 - PKDS KEYS ICSF - PKDS Key Attributes

```
Label: pkdscert
...
Key Usage:      KEYM SIGN NO-XLATE FR-NONE
Sections:      PRIVATE PUBLIC
Token Format:   CRT
Public Exponent:
00010001
Modulus:
D65B3DF6195205AC34F669AA56D8CF4...
```



ICSF TKDS secure key management through RACF

- o Token Key Dataset (TKDS) – use to store the secure private key
 - o When the keyword TOKEN is specified in RACDCERT GENCERT command, the private key associated with certificate would be stored in the ICSF TKDS, referenced by a private key label stored in RACF DB, eg
 - o `racdcert id(xxx) gencert ... rsa(token(mytoken)) withlabel('tkdscert')`
(Note: the token must exist , may be created through RACDCERT ADDTOKEN)



```
RACDCERT id(XXX)
list(label('tkdscert')):
```

```
Label: tkdscert
...
Key Type: RSA
Key Size: 2048
Private Key: YES
TKDS Token: MYTOKEN
TKDS ID: 00000004y
Ring Associations:
*** No rings associated ***
```

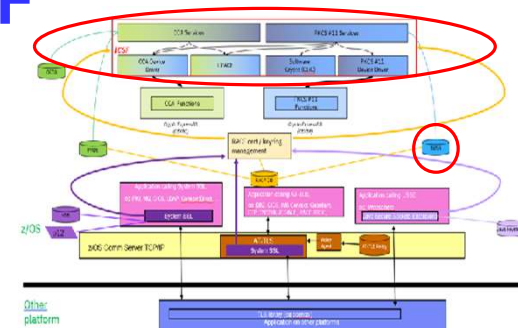
ICSF Utility panel - option 7 - PKCS11 TOKEN
ICSF Token Management - Token Details

```
Token name: MYTOKEN
...
_ Object 00000004Y PRIVATE KEY PRIVATE: TRUE MODIFIABLE: TRUE
EXTRACTABLE: NEVER SENSITIVE:ALWAYS
LABEL: IRRRSAPRIVKEY
...
```

Note: The object ends with 'Y' indicates secure object, 'T' indicates clear one. TKDS can contain both secure and clear objects

ICSF TKDS secure key management through RACF

- If you want to store the certificate in the same token where the private key is, you need to do a BIND
- BIND copies the certificate from RACF to the token
 - `racdcert id(xxx) bind(token(mytoken) label('tkdscert'))`
 - After the bind, RACDCERT LISTTOKEN will show the certificate
 - *Note: Before the BIND, LISTTOKEN is empty although the token contains the private key.*

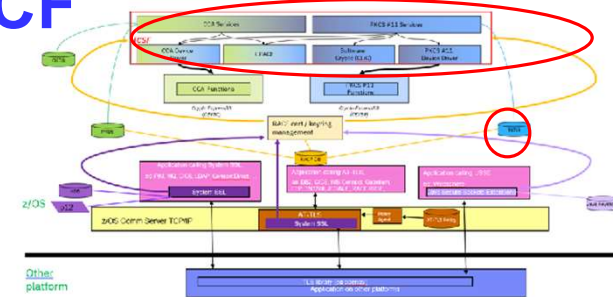


RACDCERT LISTTOKEN (mytoken)

Seq Num	Attributes	Labels
00000005	Default: NO Usage: PERSONAL Owner: ID(xxx)	Priv Key: SECURE TKDS: tkdscert Pub Key: YES RACF: tkdscert

RACF – certificate based
ICSF – key based

ICSF TKDS clear key management through RACF



- Token Key Dataset (TKDS) – use to store the clear private key
 - You can generate a certificate without specifying the token keyword first
 - Then BIND that certificate to a TOKEN, the certificate and the private key will be stored in TKDS as clear objects, eg
 - `racdcert id(xxx) gencert ... rsa withlabel('tkdscert2')`
 - `racdcert id(xxx) bind(token(mytoken) label('tkdscert2') default)`
(Note: the token must exist , may be created through RACDCERT ADDTOKEN)
 - RACDCERT LIST won't show the TKDS information, but ICSF would show the key and certificate objects
 - RACDCERT LISTTOKEN shows the certificate object in TKDS objects

ICSF TKDS clear key management through RACF

```
RACDCERT id(xxx)
list(label(`tkdscert2`):
```

```
Label: tkdscert2
...
Key Type: RSA
Key Size: 2048
Private Key: YES
Ring Associations:
*** No rings associated ***
```

Notice that ICSF creates public key object together with the private key

```
ICSF Utility panel - option 7 - PKCS11 TOKEN
ICSF Token Management - Token Details
```

```
Token name: MYTOKEN
...
_ Object 00000007T PRIVATE KEY    PRIVATE: TRUE      MODIFIABLE: TRUE
                                EXTRACTABLE: TRUE SENSITIVE: FALSE
                                LABEL:            tkdscert2
...
_ Object 00000008T PUBLIC KEY    PRIVATE: FALSE     MODIFIABLE: TRUE
                                LABEL:            tkdscert2
...
_ Object 00000009T CERTIFICATE   PRIVATE: FALSE     MODIFIABLE: TRUE
                                DEFAULT: TRUE      CATEGORY: User
                                LABEL:            tkdscert2
```

Note: The object ends with 'Y' indicates secure object, 'T' indicates clear one. TKDS can contain both secure and clear objects

```
RACDCERT LISTTOKEN(mytoken)
```

Seq Num	Attributes	Labels
00000009	Default: YES Usage: PERSONAL Owner: ID(XXX)	Priv Key: CLEAR TKDS: tkdscert2 Pub Key: YES RACF: tkdscert2

PKI Services Certificate Generation Application

Choose one of the following:

- Request a new certificate using a model

Select the certificate template to use as a model

Request Certificate

- Pick up a previously requested certificate

Enter the assigned transaction ID

Select the certificate return type

Pick up Certificate

- Renew or revoke a previously issued certificate

Renew or Revoke Certificate

- Recover a previously issued certificate whose key was generated by PKI Services

Recover Certificate

- 2-Year PKI Generated Key Certificate
- 1-Year PKI SSL Browser Certificate
- 1-Year PKI S/MIME Browser Certificate
- 2-Year PKI Windows Logon Certificate
- 2-Year PKI Browser Certificate For Authenticating
- 5-Year PKI SSL Server Certificate
- 5-Year PKI IPSEC Server (Firewall) Certificate
- 5-Year PKI Intermediate CA Certificate
- 2-Year PKI Authenticode - Code Signing Certificate
- 5-Year SCEP Certificate - Preregistration
- 2-Year EST Certificate - Preregistration
- 2-Year PKI Generated Key Certificate
- n-Year PKI Certificate for Extensions Demonstration
- 1-Year SAF Browser Certificate
- 1-Year SAF Server Certificate
- 2-Year EV SSL Server Certificate

Keypair is generated using ICSF PKCS11 for the PKI Generated Key Certificate type - private key can be clear or secure, controlled by PKI configuration or ICSF set up

View from ICSF Utility panel – option 7 – PKCS11 TOKEN
 ICSF Token Management – Token Details
 Token name: PKISRVD.PKITOKEN - when SecureKey=T

Object 000009DY PUBLIC KEY PRIVATE: FALSE 1
 MODIFIABLE: TRUE
 LABEL: 1+bEBQUy/WSZVknDWBrf3lg+
 SUBJECT: <Not-specified>
 ID:
 394C9492A298E91B158250A75D098A785099EF8C
 EC PARAMS: Named Curve - secp256r1
 EC POINT:
 044104FB7CB7F7F874443FE8E76A2A5D011C0009B7BB8280D84C8FD
 8...
 USAGE FLAGS: Enc(F), Verify(T), Wrap(F), Derive(T)

Object 000009FT CERTIFICATE PRIVATE: FALSE 3
 MODIFIABLE: TRUE
 DEFAULT: FALSE
 CATEGORY: Authority
 LABEL: feb17test@pki
 SUBJECT: CN=feb17test, OU=fvt, O=pki, L=pok,
 C=us
 ID:
 394C9492A298E91B158250A75D098A785099EF8C
 ISSUER: CN=Subcal, OU=test, C=us
 SERIAL NUMBER: 72

Object 000009EY PRIVATE KEY PRIVATE: TRUE 2
 MODIFIABLE: TRUE
 EXTRACTABLE: TRUE
 SENSITIVE: ALWAYS
 LABEL: 1+bEBQUy/WSZVknDWBrf3lg+
 SUBJECT: <Not-specified>
 ID:
 394C9492A298E91B158250A75D098A785099EF8C
 EC PARAMS: Named Curve - secp256r1
 USAGE FLAGS: Dec(F), Sign(T), Unwrap(F), Derive(T)

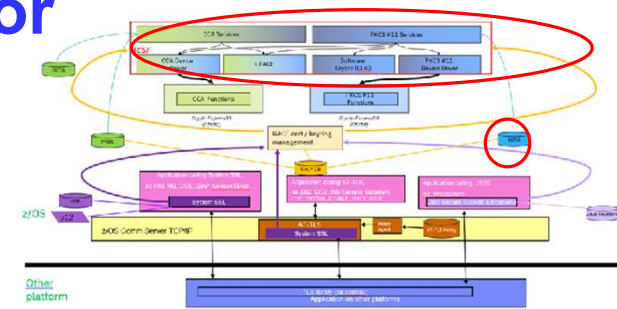
Object 00000A0T DATA PRIVATE: TRUE 4
 MODIFIABLE: TRUE
 LABEL: 0000000000000000000000000000000072
 APPLICATION: Z/OS PKI SERVICES
 OBJECT ID: 1 2 840 113549 1 12
 VALUE:
 30821413020103308213B306092A864886F70D010701A08213A40
 48213A
 03082139C308201A706092A864886F70D010701A0820198048201
 943082
 01903082018C060B2A864886F70D010C0A0102A082010B3082010
 730...

RACDCERT LISTTOKEN(PKISRVD.PKITOKEN)

000009F Default: NO Priv Key: **SECURE** TKDS: feb17test@pki
 Usage: CERTAUTH Pub Key: YES

ICSF TKDS token can be used as a keystore for handshake

- Token Key Dataset (TKDS) – use to store certificate chain as in a keystore
 - You can use the BIND command to bind certificates without the private key, copies of the certificates stored in TKDS can be used to build up the chain for handshake
 - Eg, `raccert certauth bind(token(mytoken) label('myCA'))`
`raccert certauth bind(token(mytoken) label('myIntCA'))`
 (Note: bind to the same token which contains the certificate and the private key bound previously)
 - After all the certificates and private key set up in the token, you may use it as input to System SSL to perform handshake, just like a keyring



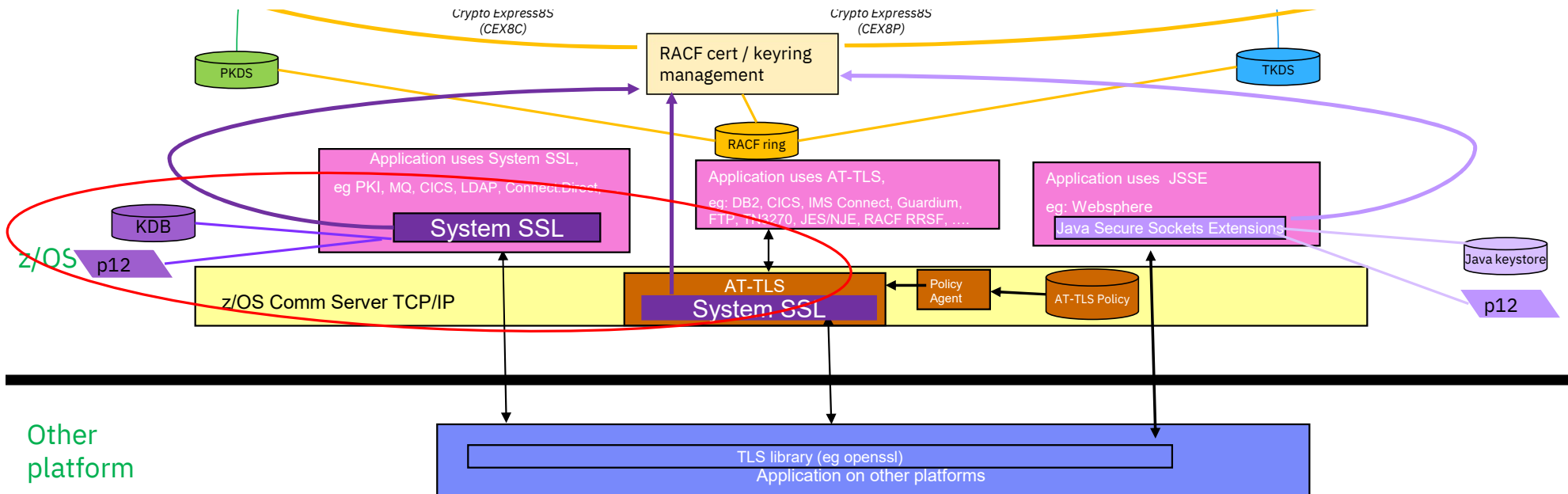
E.g. `RACDCERT LISTTOKEN(mytoken)`

Seq Num	Attributes	Labels
0000000B	Default: NO Usage: CERTAUTH Owner: CERTAUTH	Priv Key: NONE Pub Key: YES RACF: myCA TKDS: myCA
0000000D	Default: NO Usage: CERTAUTH Owner: CERTAUTH	Priv Key: NONE Pub Key: YES RACF: myIntCA TKDS: myIntCA
00000009	Default: YES Usage: PERSONAL Owner: ID(xxx)	Priv Key: CLEAR Pub Key: YES RACF: tkdscert2 TKDS: tkdscert2

System SSL

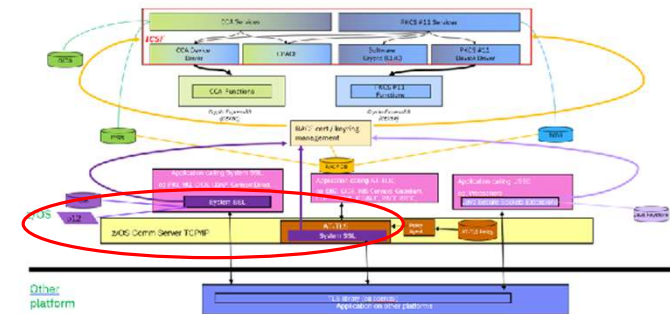
The big picture – next stop System SSL

- System SSL provides APIs for the Secure Socket Layer (SSL)/ Transport Layer Security(TLS) protocol
- The protocol requires each of the communicating parties to have a certificate/key store
- System SSL supports the following certificate/key stores
 - its native gskkyman KDB
 - SAF key ring.
 - PKCS#11 token
 - PKCS#12 file
 - GSKIT CMS V4 database (seldomly used)

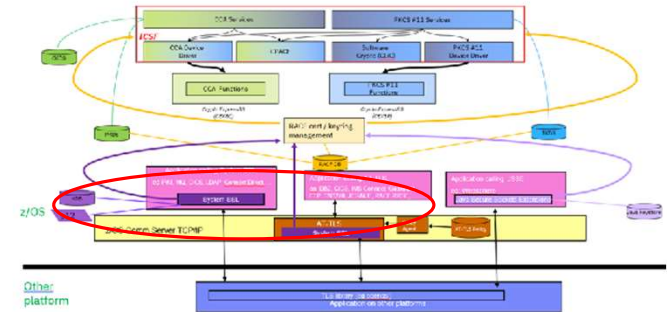


Different keystores input to System SSL

- The support of gskkyman KDB and PKCS#12 is through System SSL's own implementation. They are both files on zFS.
 - Use <KDB name> and <KDB password or a stash file> for gskkyman kdb (usually it uses a suffix .kdb, but not required)
 - Use <PKCS#12 file> and <PKCS#12 password>
 - System SSL can differentiate between these two formats not judging from the file extension – myfile.p12 can be a KDB, myp12.kdb can be a PKCS#12 file
- The support of Keyring and PKCS#11 are based on the RACF R_datalib dataGet functions
 - Use <ring owner id>/<ring name> if keystore is a keyring
 - Use *TOKEN*/<token name> if keystore is a token in TKDS



System SSL gskkyman



Can gskkyman display the token populated by RACF or PKI? Yes

gskkyman -dc -t <token> -l <label>

E.g. gskkyman -dc -t mytoken -l tkdscert

Label:

<tkdscert>

...

Trusted:

Non-critical Extension: 1

Yes

subjectAltName:

Version:

DNS: 1

3

<sysplex1.com>

Serial number:

DNS: 2

...

<sysplex2.com>

Issuer's Name:

Non-critical Extension: 2

...

subjectKeyIdentifier:

Subject's Name:

24 7B F3 FD D4 FD DF 7B 49 F2 C9 E8 2F 08 5C A3

AB CF 59 F7

...

Effective Date:

Non-critical Extension: 3

authorityKeyIdentifier:

Expiration Date:

Key ID:

...

24 7B F3 FD D4 FD DF 7B 49 F2 C9 E8 2F 08 5C A3

AB CF 59 F7

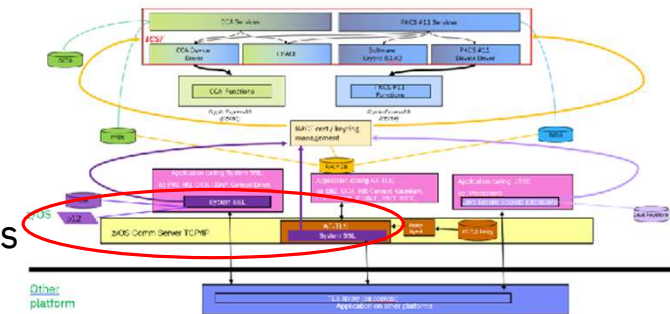
Note: RACDCERT LIST starts listing multiple subject alternate names, SKID and IKID in v3r2.

System SSL and PKI supported these a long time ago.

Note: while System SSL can display a PKI token, the attributes of the certificate object and the private key object are not compatible with the SSL handshake process out of the box.

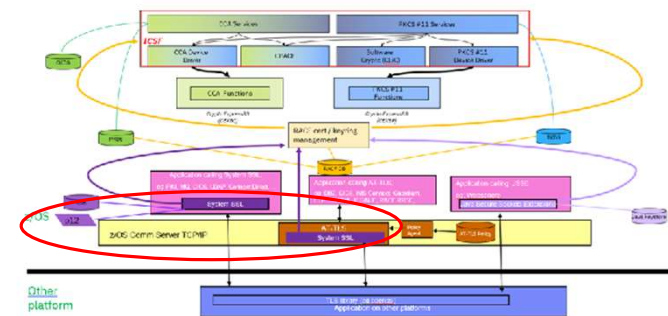
System SSL environment variables

- The basic set of information needed for handshake
 - TLS version
 - Cipher suites
 - Name of the keystore
- The first set of System SSL function calls, `gsk_environment_open()` picks up the environment variables for the handshake, like
 - `GSK_PROTOCOL_SSLV/TLSV<version #>`
 - `GSK_V3_CIPHERS`
 - `GSK_KEYRING_FILE`
 -
- These environment variable values can be overwritten by the corresponding input parameters during the API call, for example
 - Statement `CEEDPRMxx: CEEDOPT(..., ENVAR(GSK_PROTOCOL_TLS1_3=OFF)...`
 - Application: `gsk_attribute_set_enum(... GSK_PROTOCOL_TLSV1_3, GSK_PROTOCOL_TLSV1_3_ON)`→ Effective protocol is TLSv1.3



System SSL exploiters

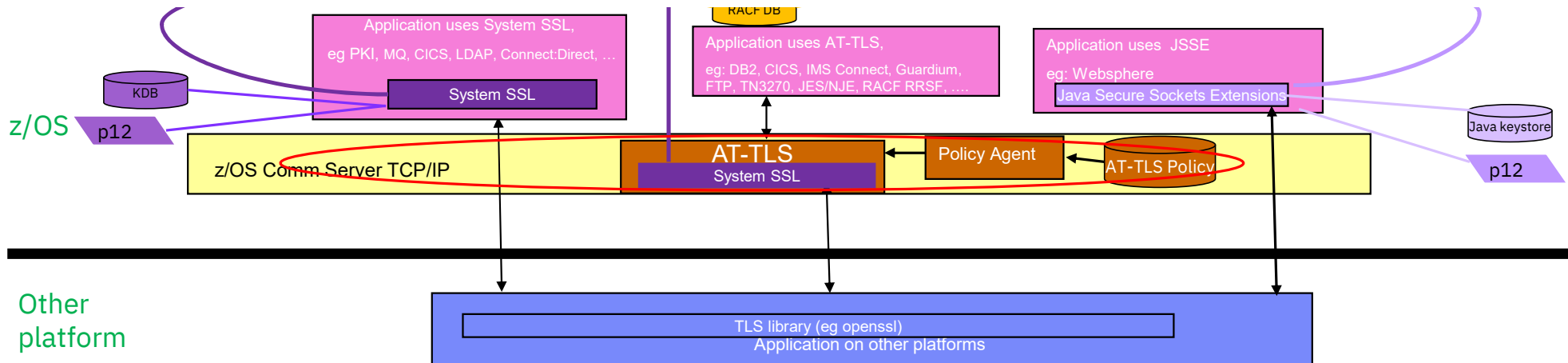
- Some IBM exploiters
 - HTTP Server
 - MQ
 - LDAP
 - PKI Services
 - Connect:Direct



Communication Server AT-TLS

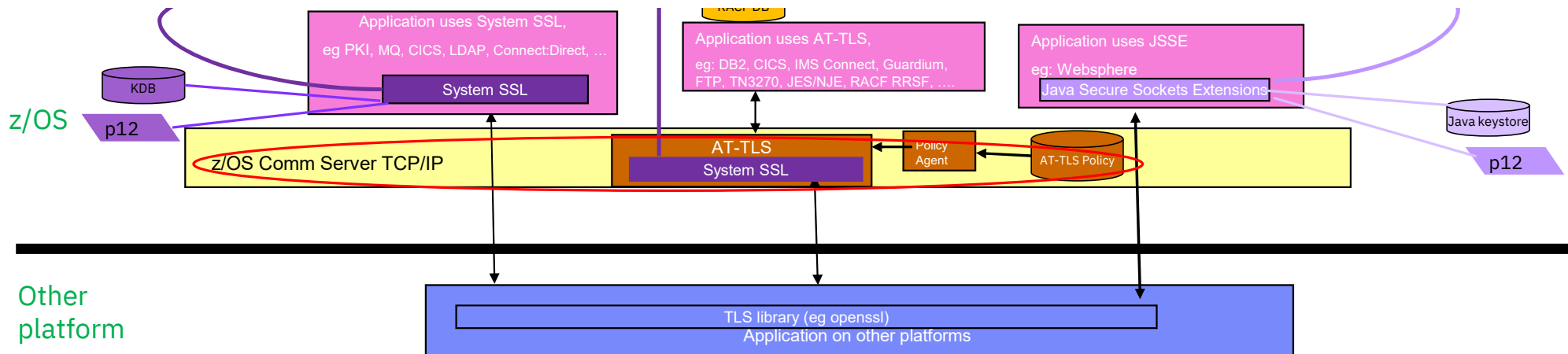
The big picture – next step AT-TLS

- Communication Server **Application Transparent TLS (AT-TLS)** is the internal SSL application that z/OS provides for the other z/OS components and customer applications
- It makes System SSL calls on behalf of the application. It handles handshake based on the policy rules in the **policy file** read by the **policy agent**
- AT-TLS creates and manages the LE environments under which System SSL executes. These environments are organized as TTLSSGroup objects in the policy file.
- Since System SSL functions are executed under a certain TTLSSGroup, most of its environment variables do not take effect under AT-TLS



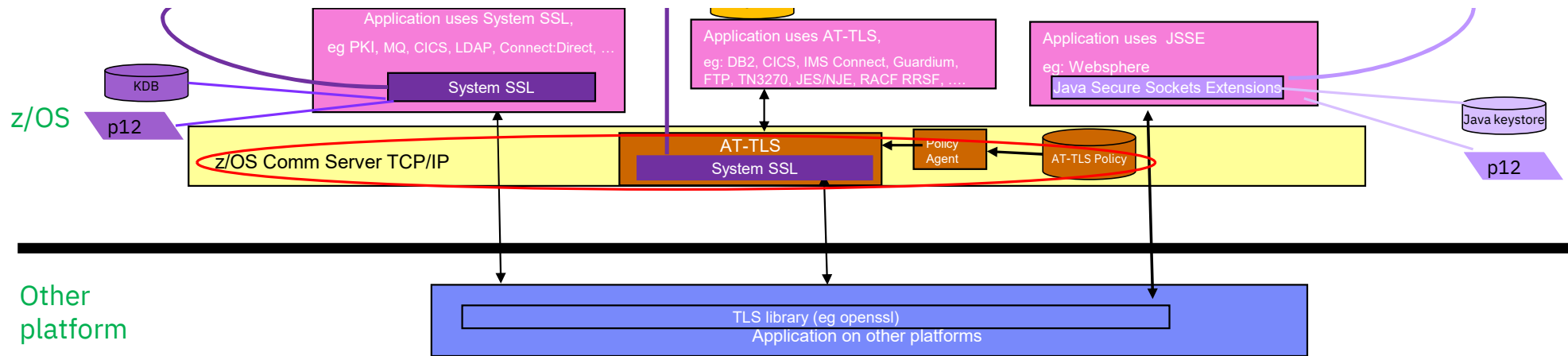
AT-TLS vs SSL setting

- Policy agent specifies majority of System SSL settings according to the policy statements, choose its own if it is not specified and pass it to SSL API
- Policy agent can also specify a parameter value when it invokes SSL API
- Either way the values from the policy agent will trump the value set from the SSL environment variables
- Some corresponding variables (System SSL in purple, AT-TLS in brown)
 - GSK_PROTOCOL_TLS<version> - TLS<version>
 - GSK-TLS_SIG_ALG_PAIRS – SignaturePairs
 - GSK_OCSP_RESPONSE_SIGALG_PAIRS – OCSPResponseSigAlgPairs



AT-TLS Configuration

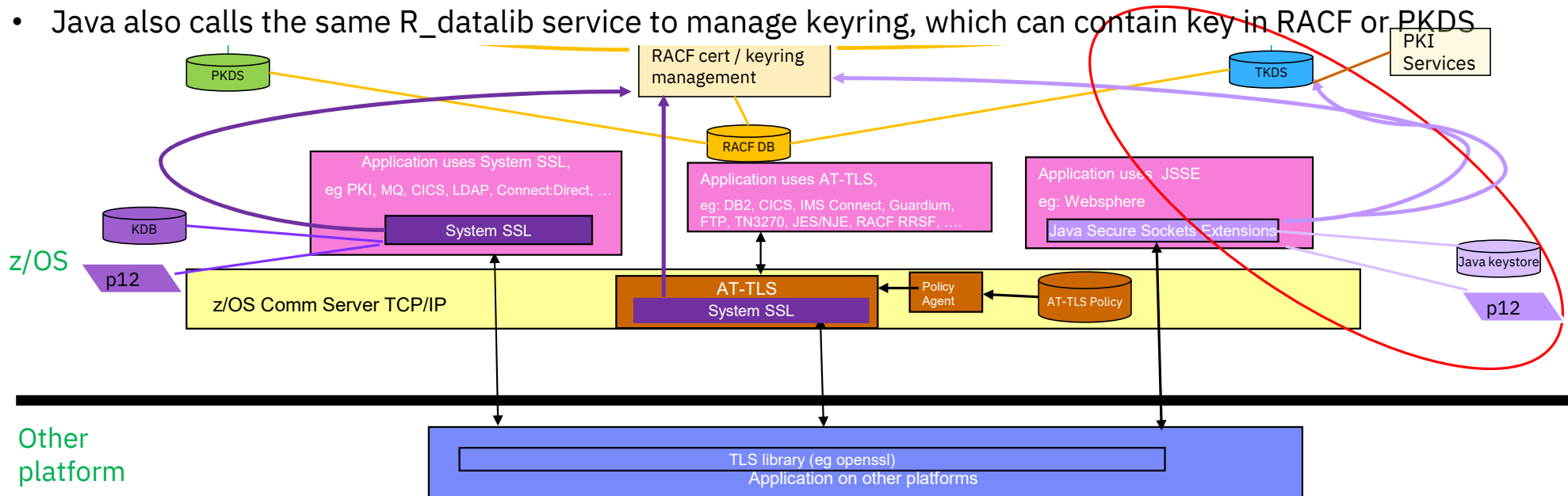
- AT-TLS policy statements set up
 - coded by hand
 - z/OSMF plugin IBM Network Configuration Assistant (NCA)



Java Secure Socket Extension (JSSE)

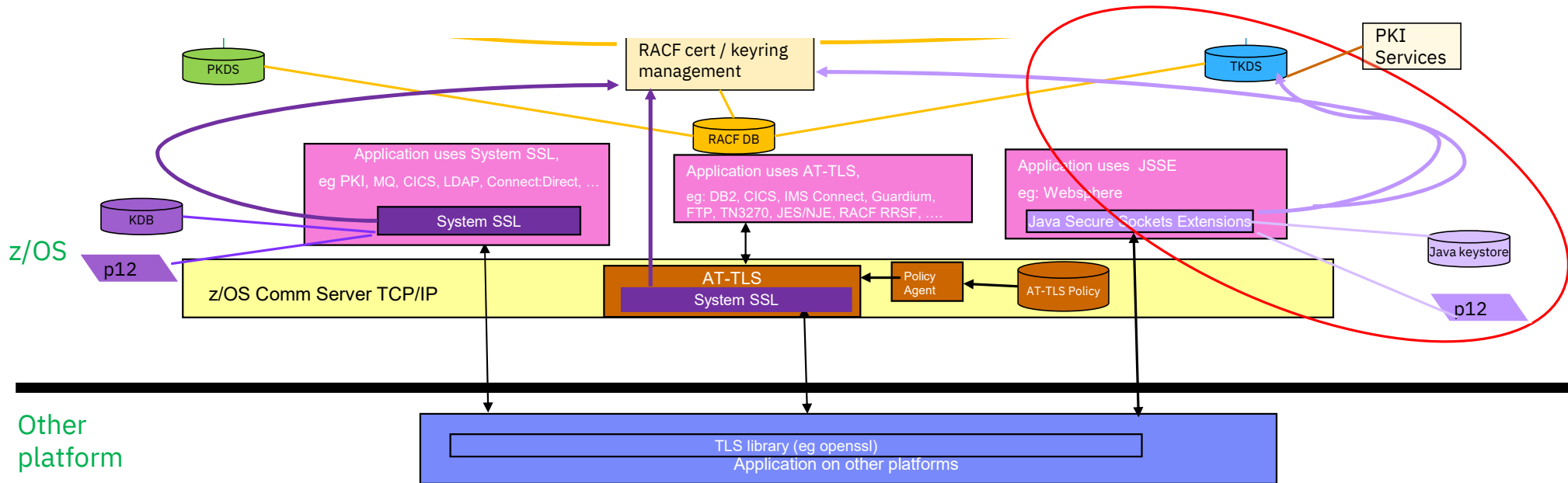
The big picture – next stop Java Secure Socket Extension

- Java provides TLS implementation via Java Secure Sockets Extension (JSSE), similar role as System SSL
- JSSE invokes Java Cryptographic Extension (JCE) providers (eg, IBMJCECCA, SunPKCS11), for cryptographic operations
- Other than its own keystore, truststore and PKCS#12 file, it also supports SAF keyrings, and z/OS token
- Java also calls the same R_datalib service to manage keyring, which can contain key in RACF or PKDS



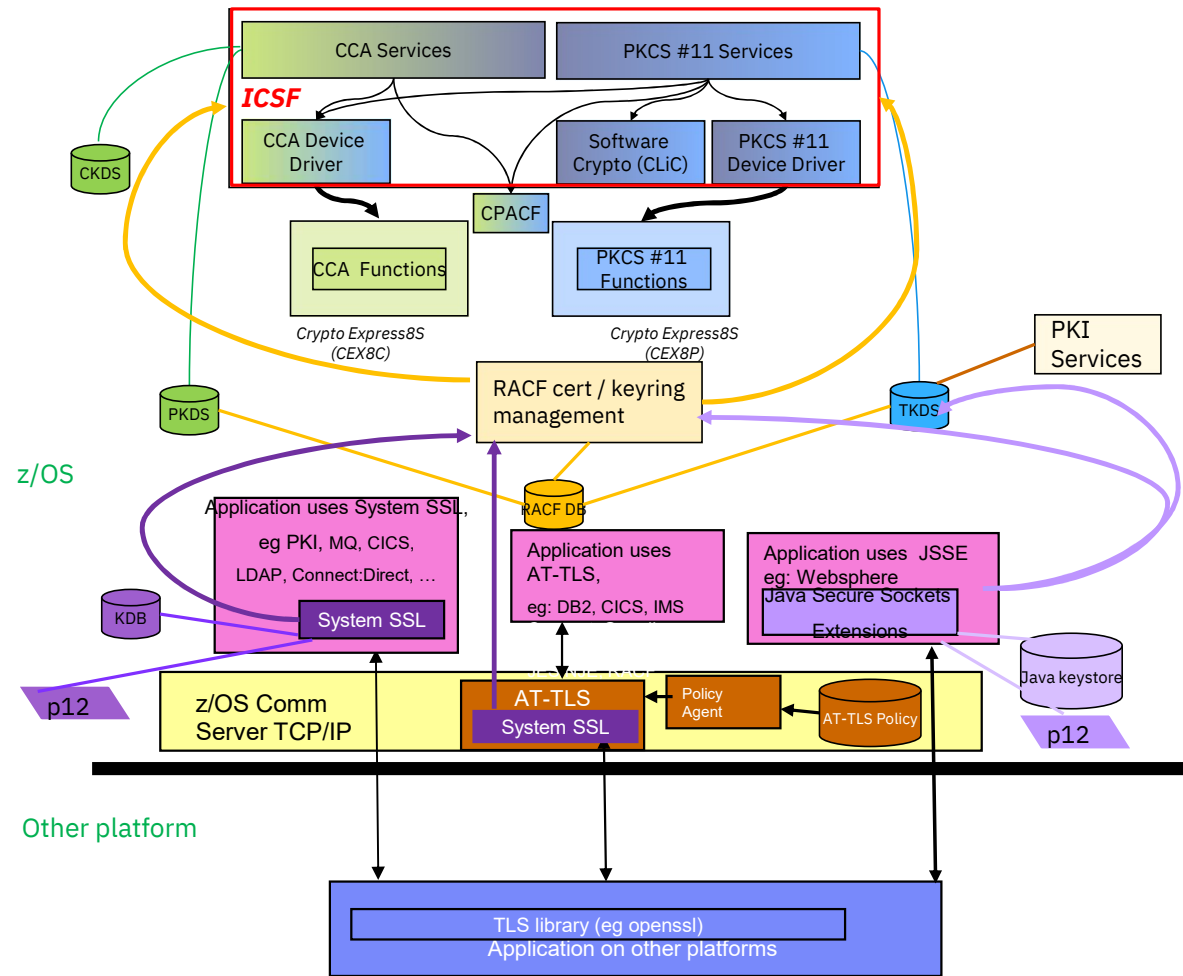
JCEE exploiters

- WebSphere and Liberty are the main exploiters of JCEE
- Notice that Java token support is not through RACF like System SSL does. It uses the SunPKCS11 provider



Summary

- ICSF provides crypto functions through hardware or software. Software is supported by CLiC
- RACF set up certificate/key stores with keys in RACF keyring, PKDS or TKDS
- System SSL and JSSE can set up certificates/key stores with keys in their own DBs
- During handshake, System SSL or JSSE retrieves the certificate and key from the certificate/key stores. If the store is a RACF keyring or a PKCS#11 token, the RACF callable service R_datalib is used
- The retrieval of key is based on the input attribute when R_datalib is called
- An application can call System SSL directly or use Communication Server AT-TLS to implement the TLS protocol
- Main role:
 - ICSF – keys
 - RACF, PKI – certificates
 - SSL, AT-TLS, JSSE – keystores



References

- **Articles Controlling TLS settings on z/OS**
<https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/flora-gui1/2023/08/14/zos-tlssl-config-info-hub?CommunityKey=406e5630-08ab-45a7-8592-d1c960f86311>
- **RACF Command Language Reference (The RACDCERT chapter)**
<https://www.ibm.com/docs/en/SSLTBW 3.1.0/pdf/icha400 v3r1.pdf>
- **RACF Callable Services (R_datalib, initACEE)**
<https://www.ibm.com/docs/en/SSLTBW 3.1.0/pdf/ichd100 v3r1.pdf>
- **RACF Security Administrator's Guide (Chapter on managing certificates scenarios)**
<https://www.ibm.com/docs/en/SSLTBW 3.1.0/pdf/icha700 v3r1.pdf>
- **ICSF Application Programmer's Guide**
https://www.ibm.com/docs/en/SSLTBW 3.1.0/pdf/csfb400 icsf_apg_hcr77e0.pdf
- **System SSL Programming**
<https://www.ibm.com/docs/en/SSLTBW 3.1.0/pdf/gska100 v3r1.pdf>
- **Communication Server IP System Administrator's Commands**
<https://www.ibm.com/docs/en/SSLTBW 3.1.0/pdf/halu101 v3r1.pdf>
- **IBM Semeru Runtime Certified Edition for z/OS 17**
<https://www.ibm.com/docs/en/semeru-runtime-ce-z/17>