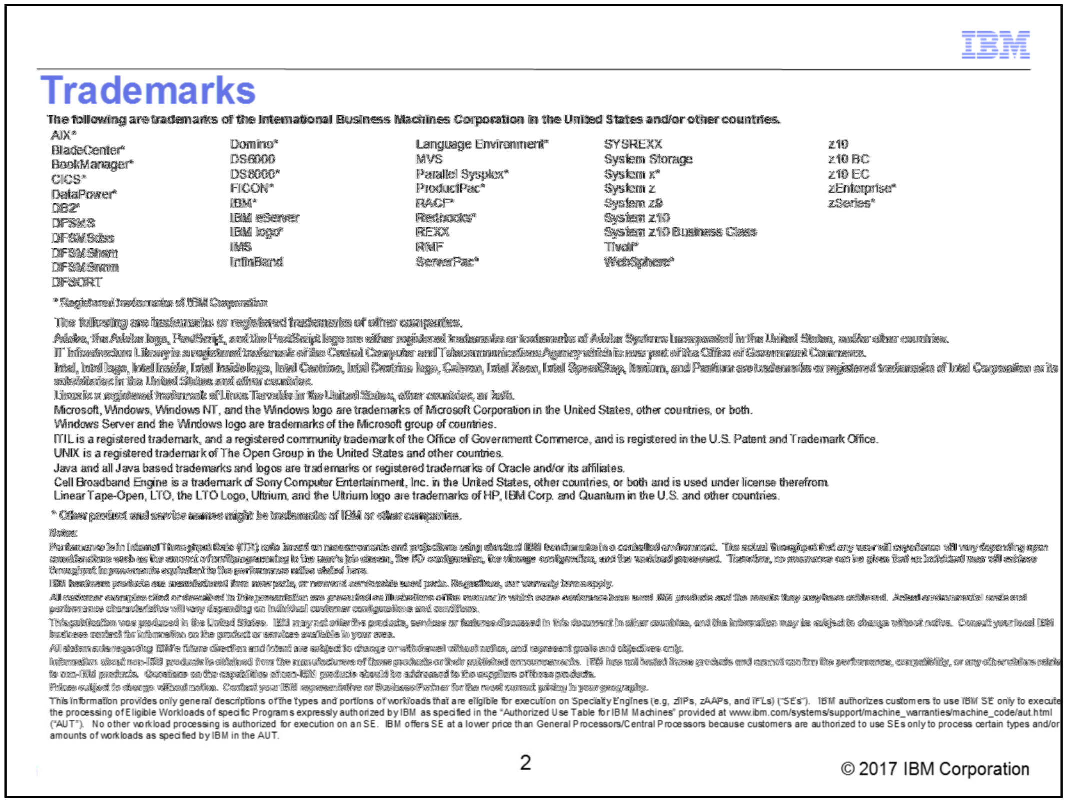
Where is my @#$%^&* Private Key ?!?!

NY-RUG 2017

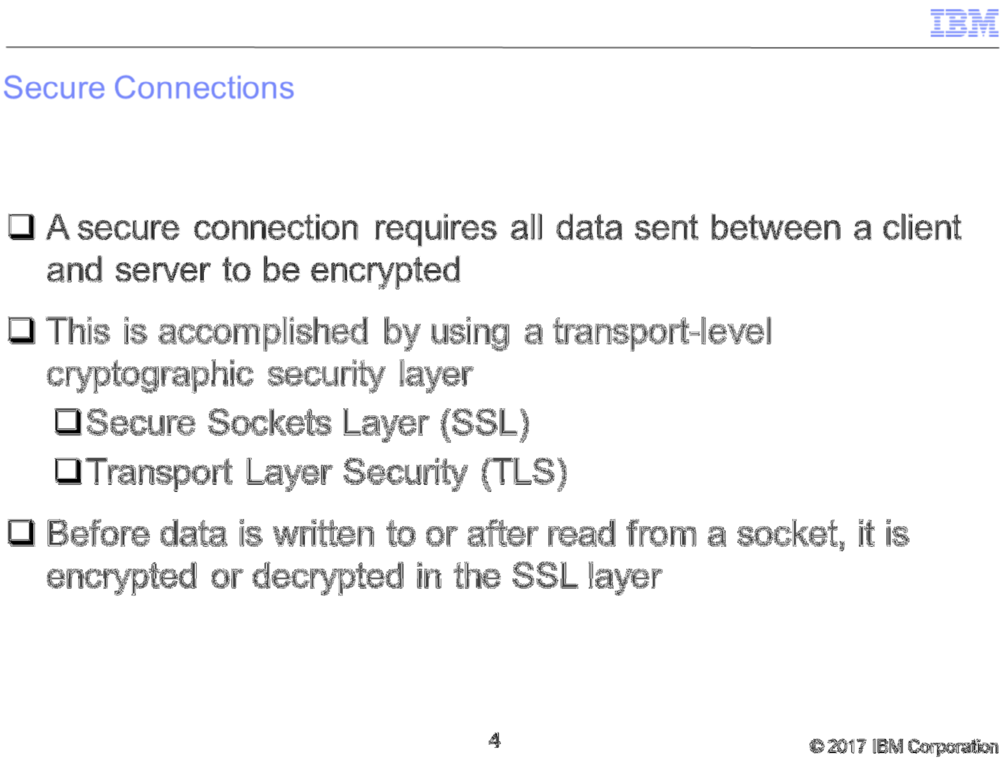
John Reale, III
Laura Sperling
z/OS Defect Support – RACF & SSL
IBM Poughkeepsie
jreale@us.ibm.com
lmdaniel@us.ibm.com



Trademarks

2

TOC



Secure Connections

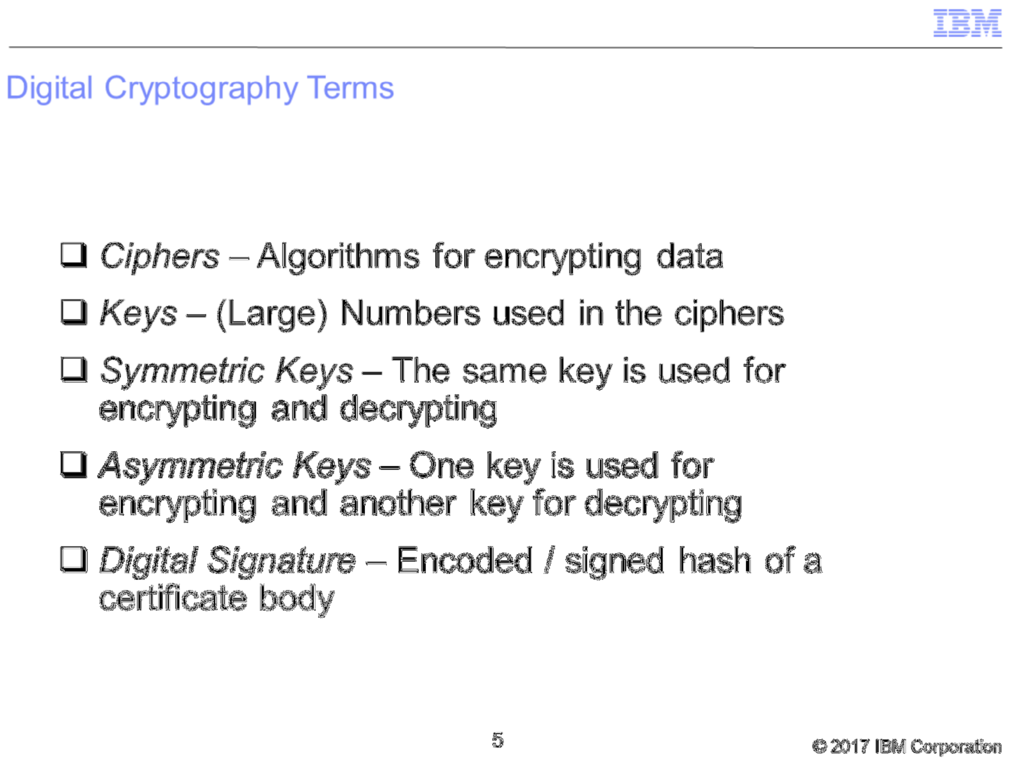
- A secure connection requires all data sent between a client and server to be encrypted
- This is accomplished by using a transport-level cryptographic security layer
  - Secure Sockets Layer (SSL)
  - Transport Layer Security (TLS)
- Before data is written to or after read from a socket, it is encrypted or decrypted in the SSL layer

Secure Connections

We must discuss the security layer in conjunction with X.509 v3 certificates.

The certificate is a way to deliver the Public Key that is used in the SSL negotiation (called the SSL handshake) prior to the sending of encrypted data.

For more on SSL and TLS, see links in appendix.


Digital Cryptography Terms

- ❑ *Ciphers* – Algorithms for encrypting data
- ❑ *Keys* – (Large) Numbers used in the ciphers
- ❑ *Symmetric Keys* – The same key is used for encrypting and decrypting
- ❑ *Asymmetric Keys* – One key is used for encrypting and another key for decrypting
- ❑ *Digital Signature* – Encoded / signed hash of a certificate body

5

© 2017 IBM Corporation

Digital Cryptography

These are some of the terms we will use in the rest of this presentation.

Ciphers include some like DES, 3DES, etc.

Keys are the numbers used by the algorithms in the different ciphers.

The keys have varied lengths (the larger the more secure).

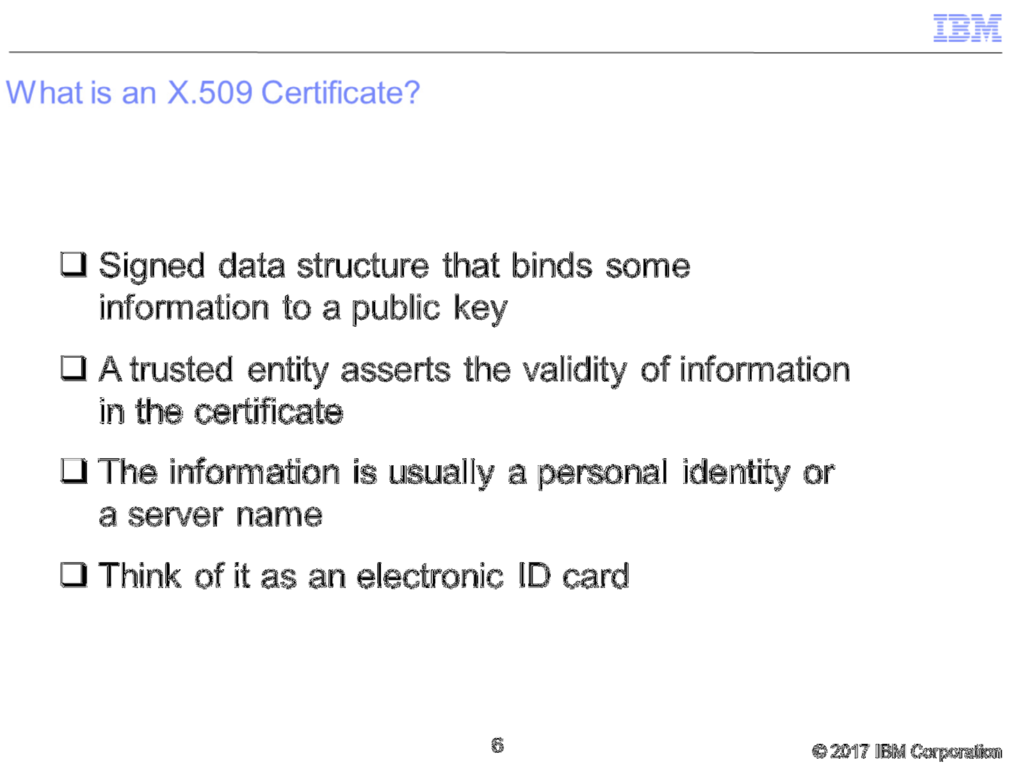
The **Symmetric Keys** are the ones used to do the real encrypting and decrypting of data in a secure connection.

The **Asymmetric Keys** are part of the Public-Key Infrastructure.

In PKI used by SSL/TLS, the asymmetric keys are pairs:

A Public Key (part of the certificate) and a Private Key – kept secret by the certificate owner.

Digital Signature is created by 'hashing' the certificate body to a fixed message digest length and then encoding this hash or checksum with the private key of the signing certificate.



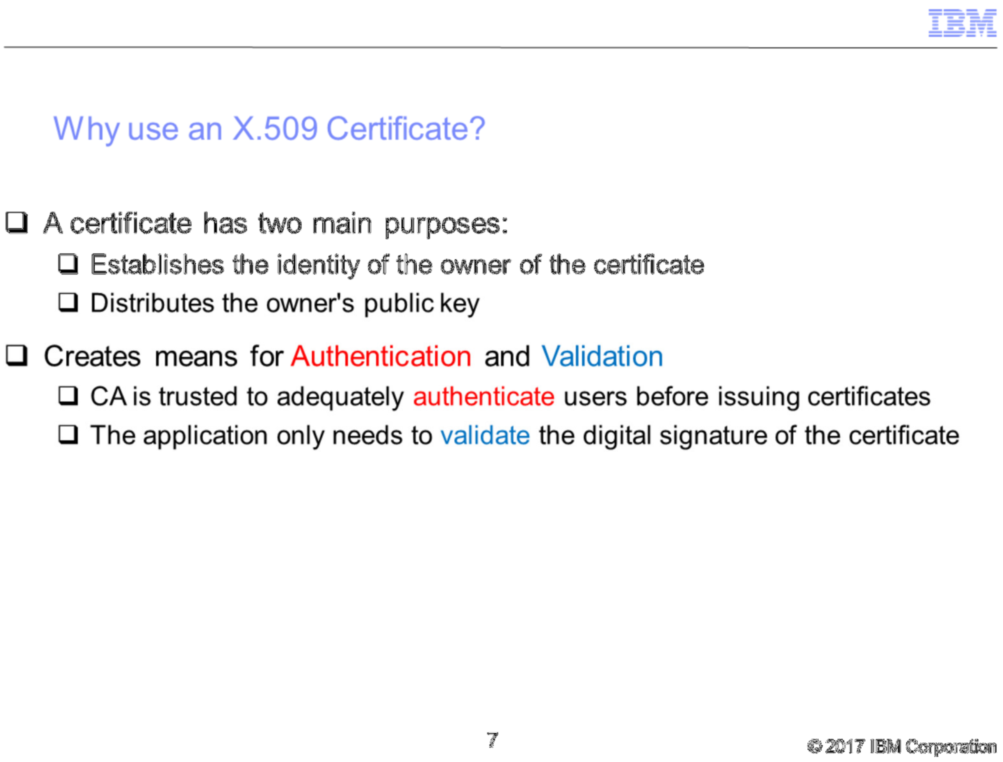
What is an X.509 Certificate?

- ❑ Signed data structure that binds some information to a public key
- ❑ A trusted entity asserts the validity of information in the certificate
- ❑ The information is usually a personal identity or a server name
- ❑ Think of it as an electronic ID card

6

Digital certificates

Certificates provide a way of authenticating users. Instead of requiring each participant in an application to authenticate every user, third-party authentication relies on the use of digital certificates.


IBM

Why use an X.509 Certificate?

- ❑ A certificate has two main purposes:
  - ❑ Establishes the identity of the owner of the certificate
  - ❑ Distributes the owner's public key
- ❑ Creates means for Authentication and Validation
  - ❑ CA is trusted to adequately authenticate users before issuing certificates
  - ❑ The application only needs to validate the digital signature of the certificate

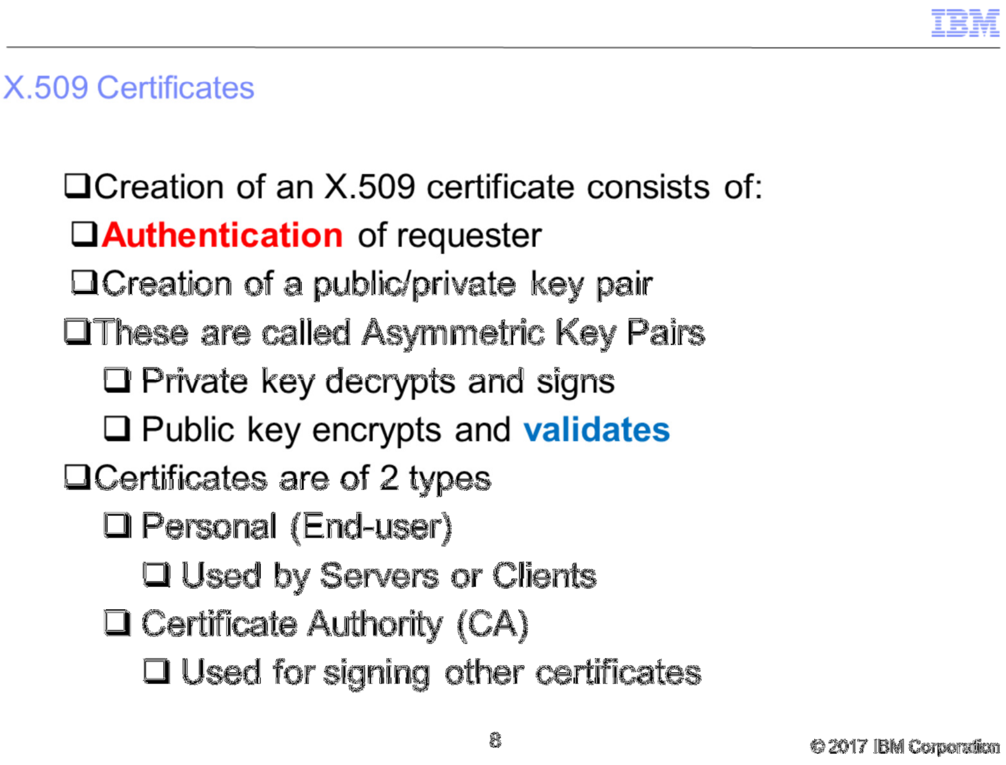
7

© 2017 IBM Corporation

A digital certificate is equivalent to an electronic ID card. The certificate serves two purposes:

- Establishes the identity of the owner of the certificate
- Distributes the owner's public key

Certificates are issued by trusted parties, called *certificate authorities* (CAs). These authorities can be commercial ventures ($$) or they can be local entities, depending on the requirements of your application. Regardless, the CA is trusted to adequately authenticate users before issuing certificates. A CA issues certificates with digital signatures. When a user presents a certificate, the recipient of the certificate validates it by using the digital signature. If the digital signature validates the certificate, the certificate is recognized as intact and authentic. Participants in an application need to validate certificates only; they do not need to authenticate users. The fact that a user can present a valid certificate proves that the CA has authenticated the user. The descriptor, *trusted third-party*, indicates that the system relies on the trustworthiness of the CAs.



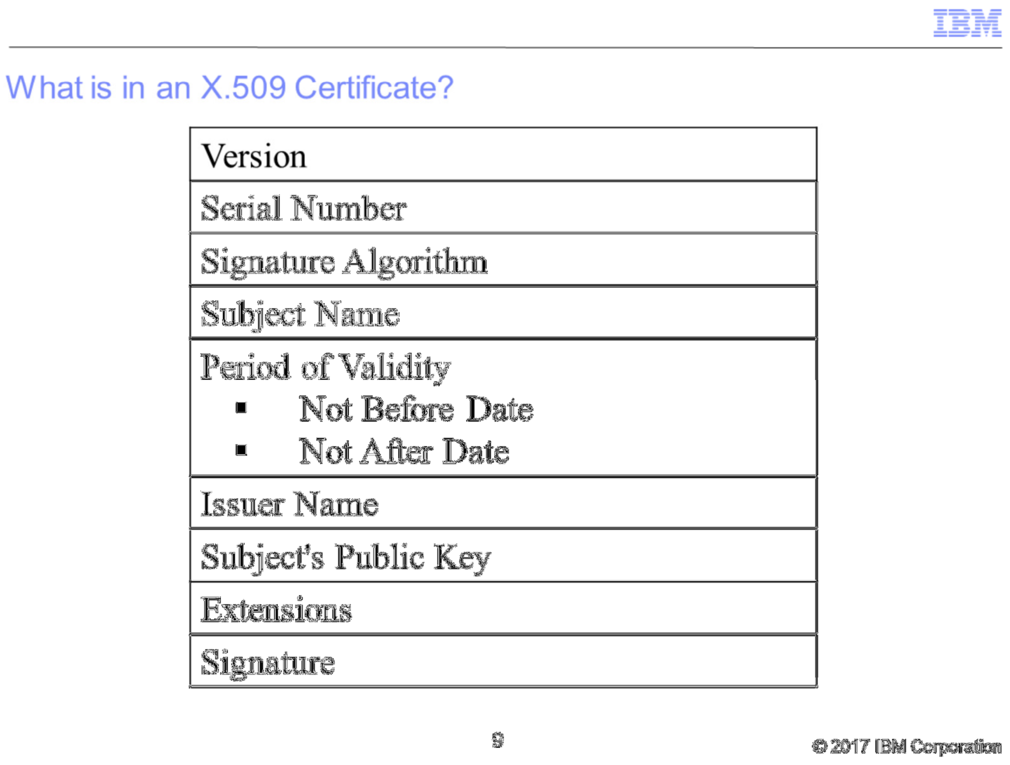
X.509 Certificates

- ❑Creation of an X.509 certificate consists of:
- ❑**Authentication** of requester
- ❑Creation of a public/private key pair
- ❑These are called Asymmetric Key Pairs
  - ❑ Private key decrypts and signs
  - ❑ Public key encrypts and **validates**
- ❑Certificates are of 2 types
  - ❑ Personal (End-user)
    - ❑ Used by Servers or Clients
  - ❑ Certificate Authority (CA)
    - ❑ Used for signing other certificates

8

© 2017 IBM Corporation

A Root CA is self-signed, while an Intermediate CA is signed by another CA.



IBM

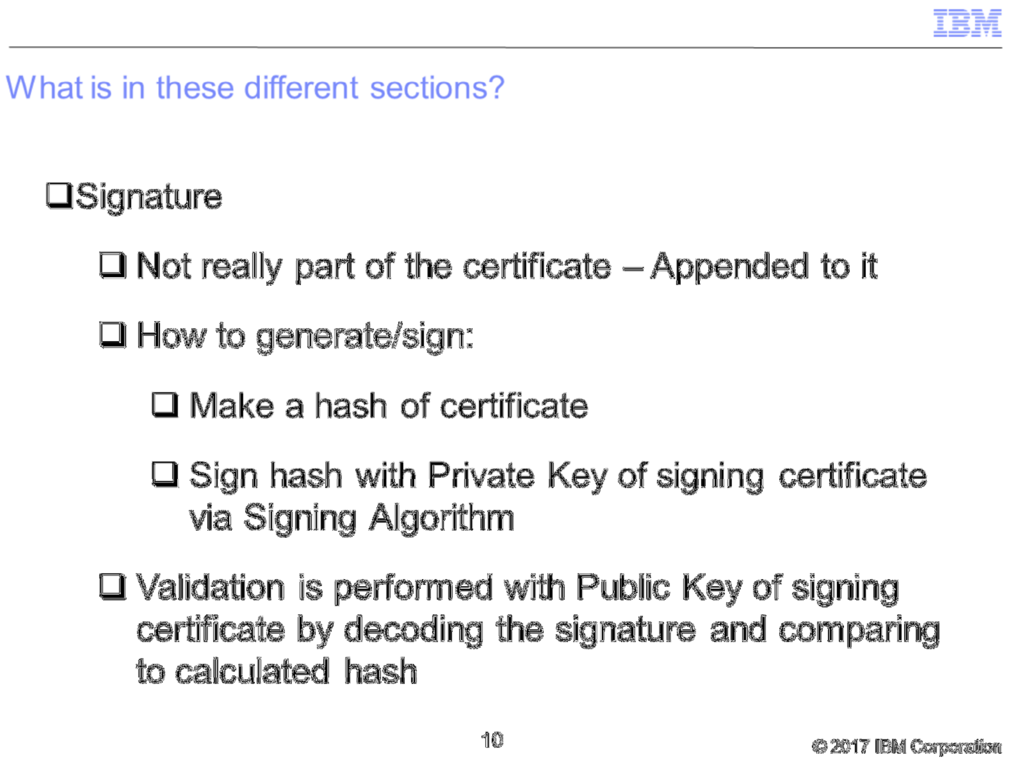
What is in an X.509 Certificate?

Version
Serial Number
Signature Algorithm
Subject Name
Period of Validity ▪ Not Before Date ▪ Not After Date
Issuer Name
Subject's Public Key
Extensions
Signature

9

© 2017 IBM Corporation

The Certificate body encompasses all parts of the certificate except for the Signature.



IBM

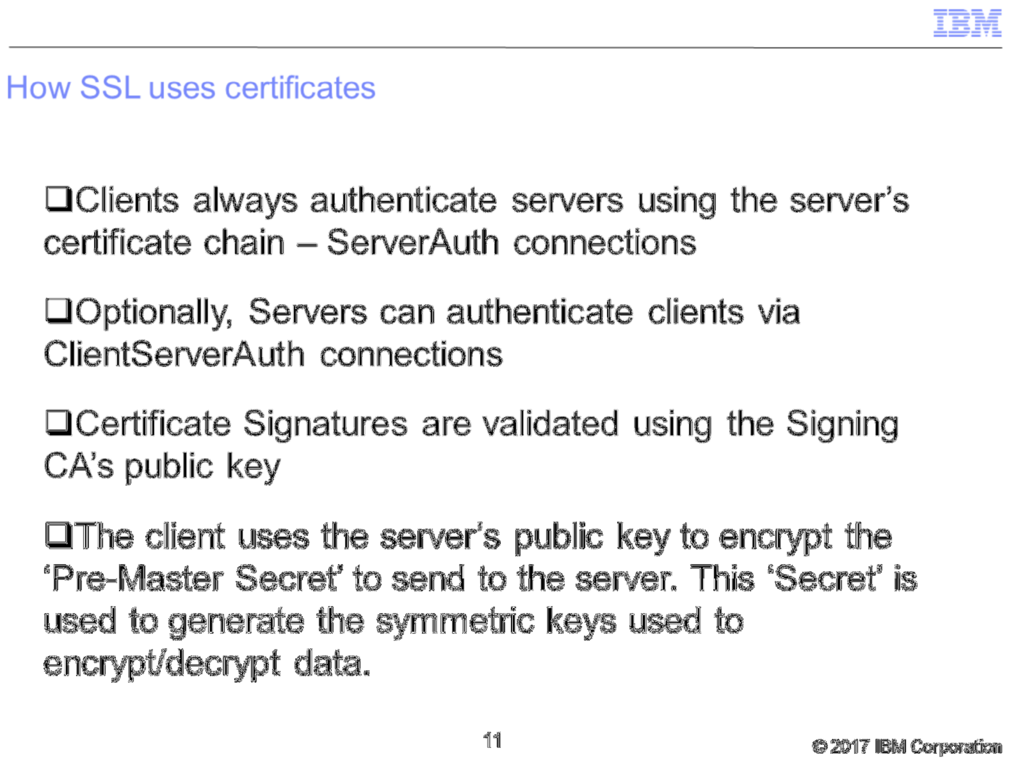
What is in these different sections?

- ❑Signature
  - ❑ Not really part of the certificate – Appended to it
  - ❑ How to generate/sign:
    - ❑ Make a hash of certificate
    - ❑ Sign hash with Private Key of signing certificate via Signing Algorithm
  - ❑ Validation is performed with Public Key of signing certificate by decoding the signature and comparing to calculated hash

10

© 2017 IBM Corporation

Signature Validation is done by the RECEIVING side of a certificate – not by the SENDING side.



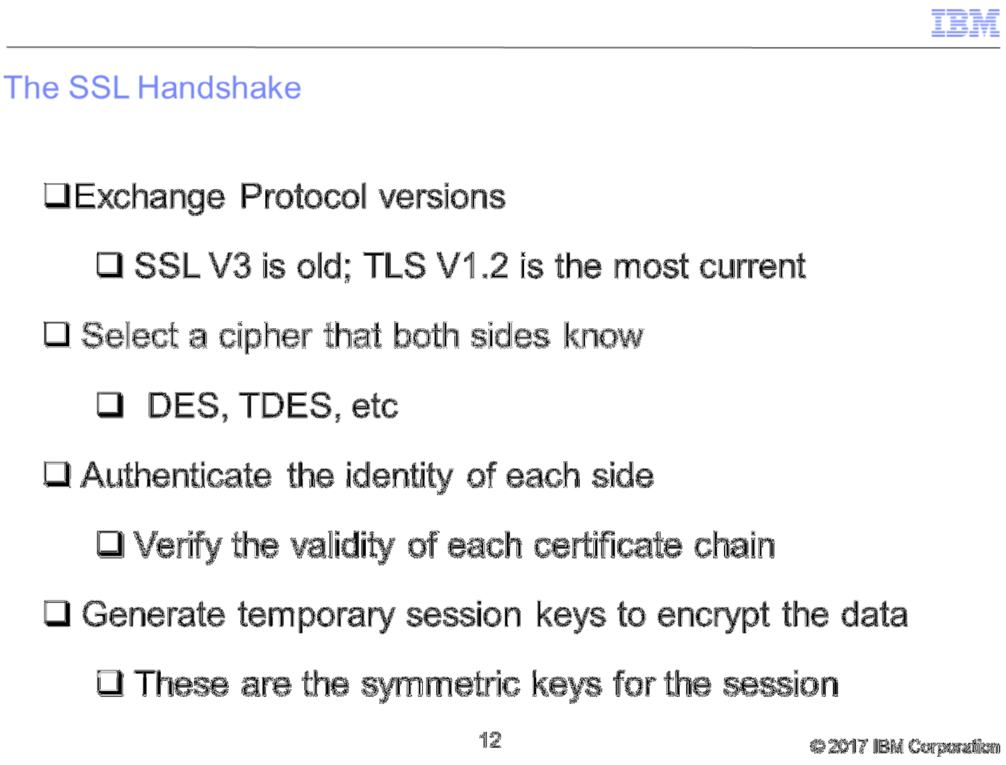
How SSL uses certificates

- ❑Clients always authenticate servers using the server's certificate chain – ServerAuth connections
- ❑Optionally, Servers can authenticate clients via ClientServerAuth connections
- ❑Certificate Signatures are validated using the Signing CA's public key
- ❑The client uses the server's public key to encrypt the 'Pre-Master Secret' to send to the server. This 'Secret' is used to generate the symmetric keys used to encrypt/decrypt data.

11

There are 2 symmetric keys created in the process – one for server data encrypt/decrypt and one for client data encrypt/decrypt.

They are first tested during the SSL handshake in the FINISHED message sent by the client and the FINISHED message sent by the server.



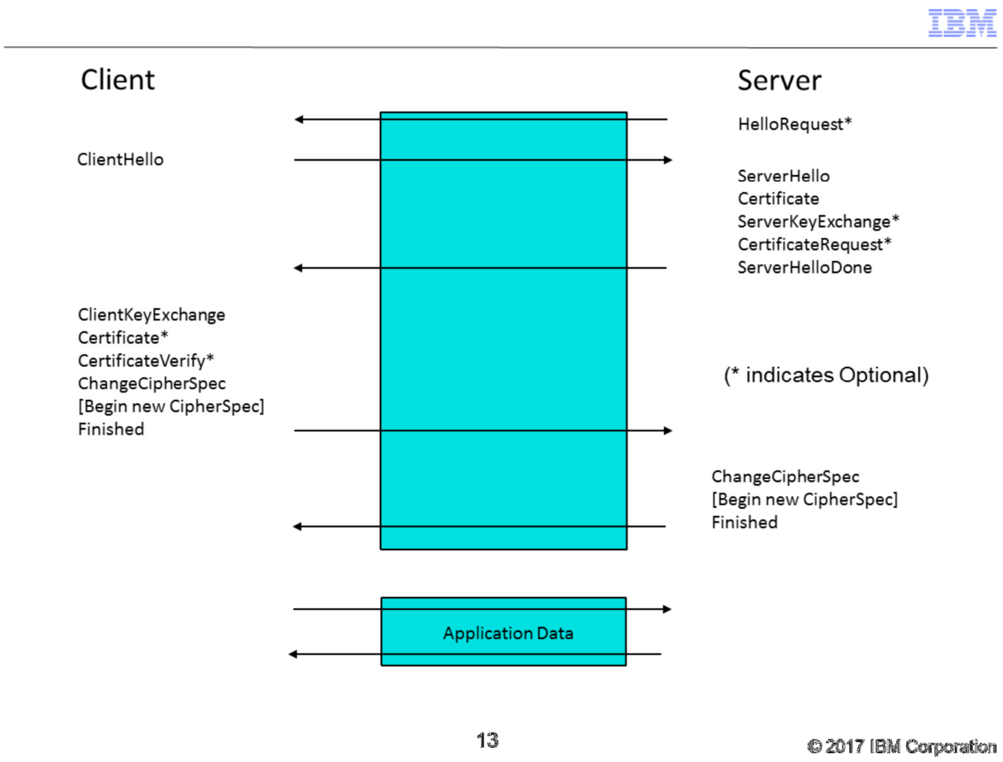
The SSL Handshake

- ❑Exchange Protocol versions
  - ❑ SSL V3 is old; TLS V1.2 is the most current
- ❑ Select a cipher that both sides know
  - ❑ DES, TDES, etc
- ❑ Authenticate the identity of each side
  - ❑ Verify the validity of each certificate chain
- ❑ Generate temporary session keys to encrypt the data
  - ❑ These are the symmetric keys for the session

12

The certificate chain / hierarchy is a structure of certificates that allows individuals to verify the validity of a certificate's issuer.

The chain begins with an End User / End Entity Certificate , and is followed by his signing certificate, and so on, until the Self-Signed Root is reached.



This charts shows the different communication elements that are transferred between a client and a server when setting up an SSL connection.

During the handshake, the client initiates the handshake by sending a hello message (**ClientHello**) containing the authentication/encryption ciphers it supports.

The server then sends the authentication/encryption cipher to be used (**ServerHello**) followed by its identity/certificate (**Certificate**) to the client.

After the Server has finished sending any optional messages, Server then sends **ServerHelloDone**.

The client will validate the server's certificate chain and look for its root CA cert on the client's "certificate repository" (keyring / kdb / etc).

(Alternately, it could find a "Trust Anchor" instead of the root. This is rarely used and we will not talk about it.)

This is where errors occur most often.

Once the cert is validated, the client generates "key material" (called the 'pre-master secret'), encrypts it with the server's public key, and sends it to the server (**ClientKeyExchange**).

Client uses “key material” to generate both session keys and notifies Server it is ready to switch to symmetric encryption of data (**ChangeCipherSpec**).

The first message the client sends using its session key and the agreed upon cipher is the **Finished** message.

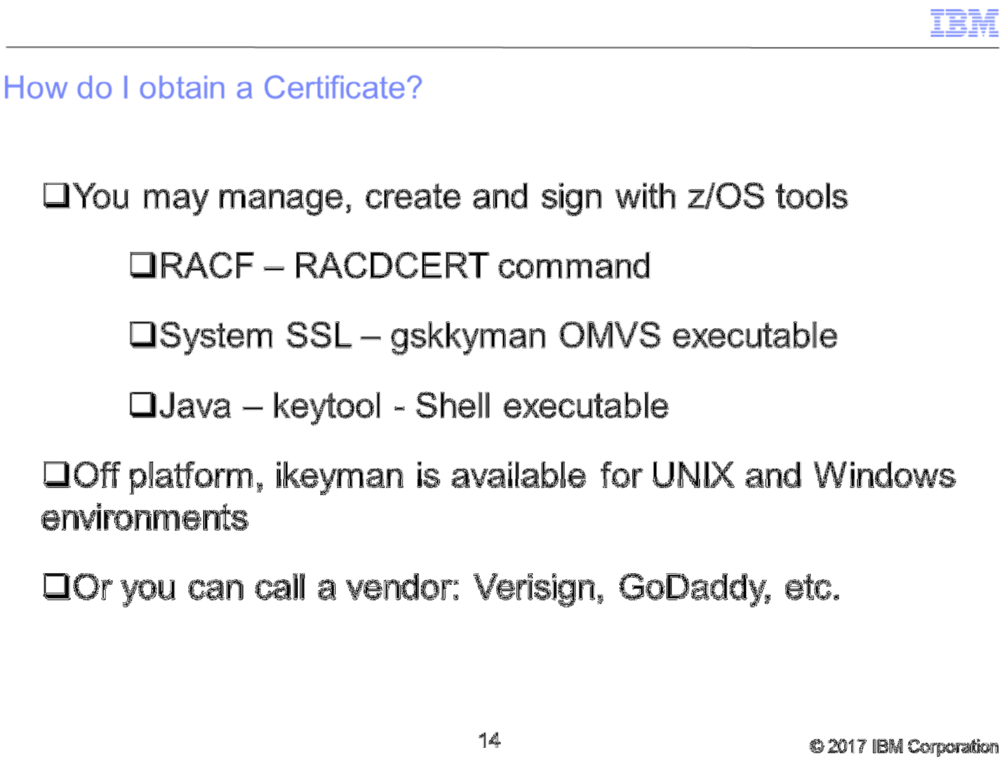
Server decrypts the "key material" using its private key, and also uses it to generate the same session keys.

Upon receiving the **Finished** message from client, and correctly decrypting it using the generated client session key, the server sends its own **ChangeCipherSpec** and **Finished** messages.

If the client correctly decrypts the **Finished** message from the server using the generated server session key, the handshake is complete.

Application data is now encrypted for this connection using symmetric encryption (via client and server session keys).

Handshake messages can be "stacked". “Stacked" means multiple messages can be sent in a single message to the partner application.

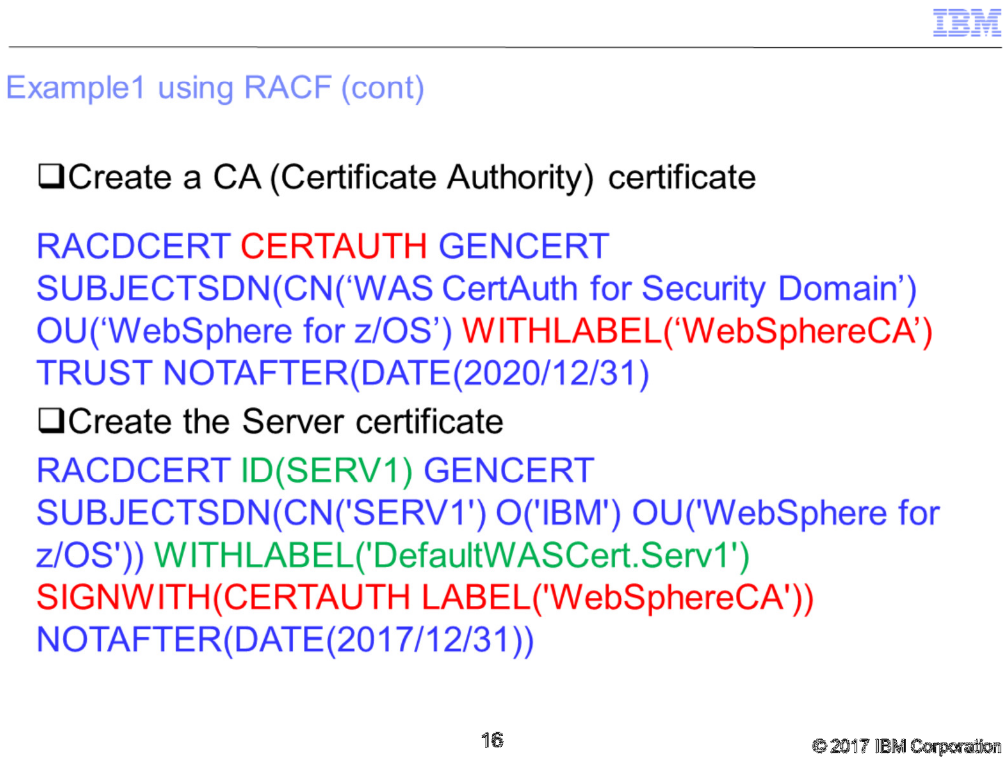

How do I obtain a Certificate?

- ❑You may manage, create and sign with z/OS tools
  - ❑RACF – RACDCERT command
  - ❑System SSL – gskkyman OMVS executable
  - ❑Java – keytool - Shell executable
- ❑Off platform, ikeyman is available for UNIX and Windows environments
- ❑Or you can call a vendor: Verisign, GoDaddy, etc.

14

For the purposes of this presentation, we will only be showing RACF-related examples.

Example 1 – Using RACF: Create your own Server Certificate and its signing CA



Example1 using RACF (cont)

❑Create a CA (Certificate Authority) certificate

```
RACDCERT CERTAUTH GENCERT
SUBJECTSDN(CN(‘WAS CertAuth for Security Domain’)
OU(‘WebSphere for z/OS’) WITHLABEL(‘WebSphereCA’)
TRUST NOTAFTER(DATE(2020/12/31)
```

❑Create the Server certificate

```
RACDCERT ID(SERV1) GENCERT
SUBJECTSDN(CN('SERV1') O('IBM') OU('WebSphere for
z/OS')) WITHLABEL('DefaultWASCert.Serv1')
SIGNWITH(CERTAUTH LABEL('WebSphereCA'))
NOTAFTER(DATE(2017/12/31))
```

16

Red text represents the CA.

Green text represents the End Entity.

Example1 using RACF (cont)

❑Create a keyring

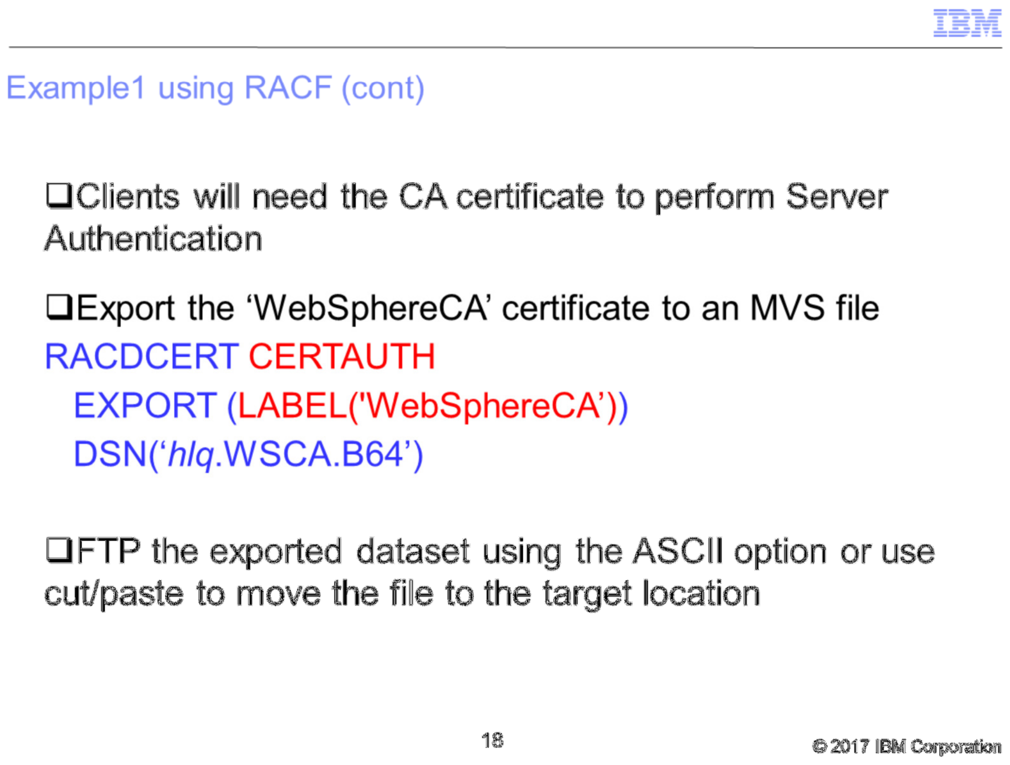
```
RACDCERT ID(SERV1) ADDRING(WASKeyring)
```

❑Connect the Server Certificate to the keyring

```
RACDCERT ID(SERV1) CONNECT
(LABEL('DefaultWASCert.Serv1') RING(WASKeyring)
DEFAULT)
```

❑Connect WAS CA Certificate to Server keyring

```
RACDCERT ID(SERV1) CONNECT (RING(WASKeyring)
LABEL('WebSphereCA') CERTAUTH)
```



IBM

Example1 using RACF (cont)

❑Clients will need the CA certificate to perform Server Authentication

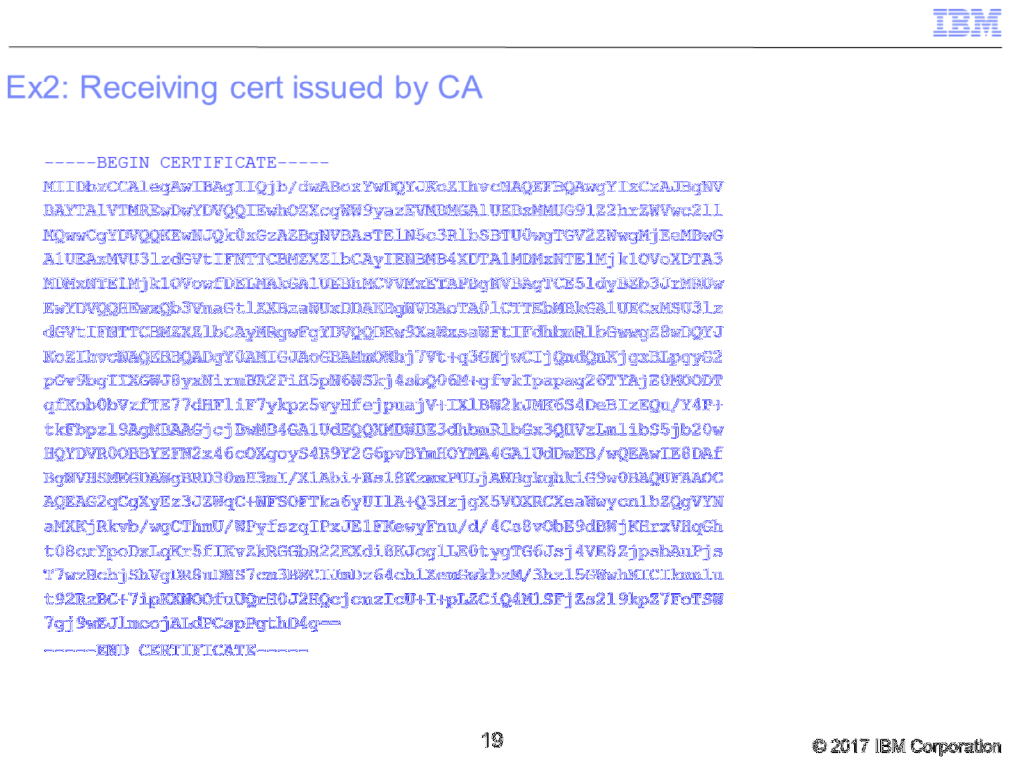
❑Export the ‘WebSphereCA’ certificate to an MVS file

```
RACDCERT CERTAUTH
  EXPORT (LABEL('WebSphereCA’))
  DSN(‘hlq.WSCA.B64’)
```

❑FTP the exported dataset using the ASCII option or use cut/paste to move the file to the target location

18 © 2017 IBM Corporation

The default FORMAT value is Base 64 encoding. If this default is overridden, you may have to FTP the file in BINARY mode.

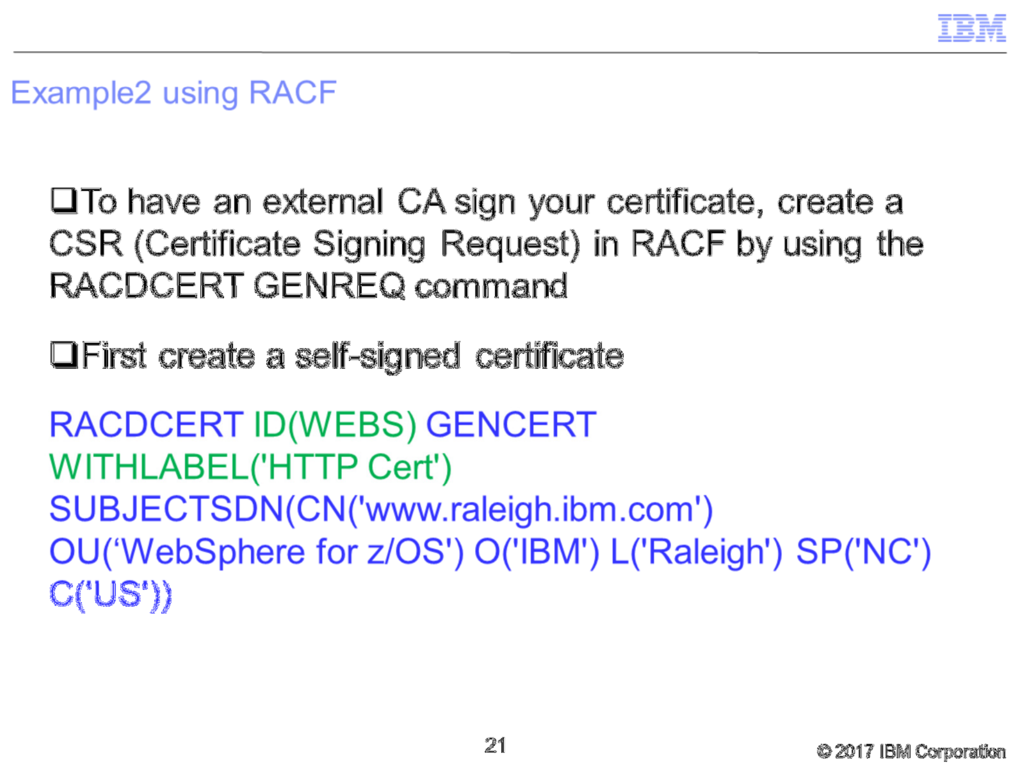


Ex2: Receiving cert issued by CA

```
-----BEGIN CERTIFICATE-----
MIIDbzCCAlegAwIBAgIIQjb/dwABoxYwDQYJKoZIhvcNAQEFBQAwgYIxCzAJBgNV
BAYTAlVTMREwDwYDVQQIEwhOZXcgWW9yazEVMBMGA1UEBxMMUG91Z2hrZWVwc2ll
MQwwCgYDVQQKEwNJQk0xGzAZBgNVBAsTElN5c3RlbSBTU0wgTGV2ZWwgMjEeMBwG
A1UEAxMVU31zdGVtIFNTTCBMZXZlbCAyIENBMB4XDTA1MDMxNTE1Mjk1OVoXDTA3
MDMxNTE1Mjk1OVowfDELMAkGA1UEBhMCVVMxETAPBgNVBAgTCE5ldyBZb3JrMRUw
EwYDVQQHEwxQb3VnaGtlZXBzaWUxDDAKBgNVBAoTA01CTTEbMBkGA1UECxMSU31z
dGVtIFNTTCBMZXZlbCAyMRgwFgYDVQQDEw9XaWxsaWFtIFdhbmRlbGwwgZ8wDQYJ
KoZIhvcNAQEBBQADgY0AMIGJAoGBAMmOXhj7Vt+q3GNjwCIjQndQnKjgxBLpgyG2
pGv9bgIIXGWJ8yxNirmBR2PiH5pN6WSkj4sbQ06M+gfvkIpapag26TYAjE0MOODT
qfKob0bVzfTE77dHFliF7ykpz5vyHfejpuajV+IX1BW2kJMK6S4DeBIzEQu/Y4P+
tkFbpzl9AgMBAAGjcjBwMB4GA1UdEQQXMDWBE3dhbmRlbGx3QHVzLmlibS5jb20w
HQYDVR0OBBYEFN2z46cOXgoyS4R9Y2G6pvBYmHOYMA4GA1UdDwEB/wQEAwIE8DAf
BgNVHSMEGDAWgBRD30mH3mI/X1Abi+Ns18KzmxPULjANBgkqhkiG9w0BAQUFAAOC
AQEAG2qCgXyEz3JZWqC+NFSOFTka6yUIlA+Q3HzjgX5VOXRCZeaWwycnlbZQgVYN
aMXKjRkvb/wgCThmU/WPyfszqIPxJE1FKewyFnu/d/4Cs8vObE9dBWjKHrxVHqGh
t08crYpoDxLqKr5fIKvZkRGGbR22EXdi8KJcglLE0tygTG6Jsj4VE8ZjpshAuPjs
T7wzHchjShVgDR8uNWS7cm3HWCIJmDz64chlXemGwkbzM/3hz15GWwhMICIkmnlu
t92RzBC+7ipKXMOOfuUQrH0J2HQcjcnzIcU+I+pLZCiQ4M1SFjZs219kpZ7FoTSW
7gj9wEJlmcojALdPCapPgthD4g==
-----END CERTIFICATE-----
```

This is a certificate file in Base64 format.

Example 2 – Using RACF:
Create your own Server/End Entity Certificate using an external CA



IBM

Example2 using RACF

- ❑To have an external CA sign your certificate, create a CSR (Certificate Signing Request) in RACF by using the RACDCERT GENREQ command
- ❑First create a self-signed certificate

```
RACDCERT ID(WEBS) GENCERT
WITHLABEL('HTTP Cert')
SUBJECTSDN(CN('www.raleigh.ibm.com')
OU('WebSphere for z/OS') O('IBM') L('Raleigh') SP('NC')
C('US'))
```

21

© 2017 IBM Corporation

Green text represents the End Entity.

Example2 using RACF

❑Now create the CSR from the certificate just created

RACDCERT ID(WEBS) GENREQ(LABEL('HTTP Cert')) DSN('CERTREQ.B64')

❑The CSR is in a PKCS#10 format and is now forwarded to the CA (eg, Verisign) by their instructions

- ❑ e-mail it, paste it into a web page, etc.

❑ Note: The CSR does *not* contain the private key!

Example2 using RACF

❑The PKCS#10 certificate request as Base64 encoded data. Must be handled as text/ASCII

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBfzCB6QIBADAQMQ4wDAYDVQQDEwV0ZXN0YTCBnzANBgkqhkiG9w0BAQE
FAAOBjQAwgYkCgYEA3qy4qFTb97+kefbgMysxIXpaRQVwqT0I2XeDmI0WmX
F6GkvK+i4k7wr/pto+cGtqCzHsQrm6aRKiJF6pdizYkf4xew0DqdeVArOyd
r/4HESVNlJRqxJ/jqY4IJ0uphnsbKKyfl7ny77u4M50YZBXRGq9VDAFpCaQ
bNW8xVkIUPECAwEAAaAwMC4GCSqGSIb3DQEJDjEhMAAwHQYDVR0OBBYEFFX
5QcyxyCL6q+NFFGjQpoCnP9mBMA0GCSqGSIb3DQEBBQUAA4GBAMyKoRZvGJ
AyVPummMMiRJgMQ4KMcYrraI79rz7LSWAq5/lPpbkue9ennlxaS3eJZOQNX
GaVk4Rem3rGM740/PpQIF/qMN1pJZfOEzyLrYxIaO6riFXM3Q2Y80m6C+X+
Vk69eRClLTvc8I5l5uz+EMCTd5x5PaGuzhXgjxkEQ5vt
-----END NEW CERTIFICATE REQUEST-----
```

Ex2: Receiving cert issued by CA

- ❑Follow CA's instructions for retrieving issued cert
  - ❑Download from web page, save to file, FTP to host
  - ❑Be aware of data type (ASCII vs Binary) when transferring to host
- ❑You may need to download the vendor's signing certificate chain and add them to RACF, using following command:

```
RACDCERT CERTAUTH ADD('cacert.b64') WITHLABEL('new ca')
```

- ❑This adds them to the CERTAUTH common repository

Ex2: Receiving cert issued by CA

❑Next, add newly signed cert to the same user-id (RACF will find the original label and overlay the changed parts)

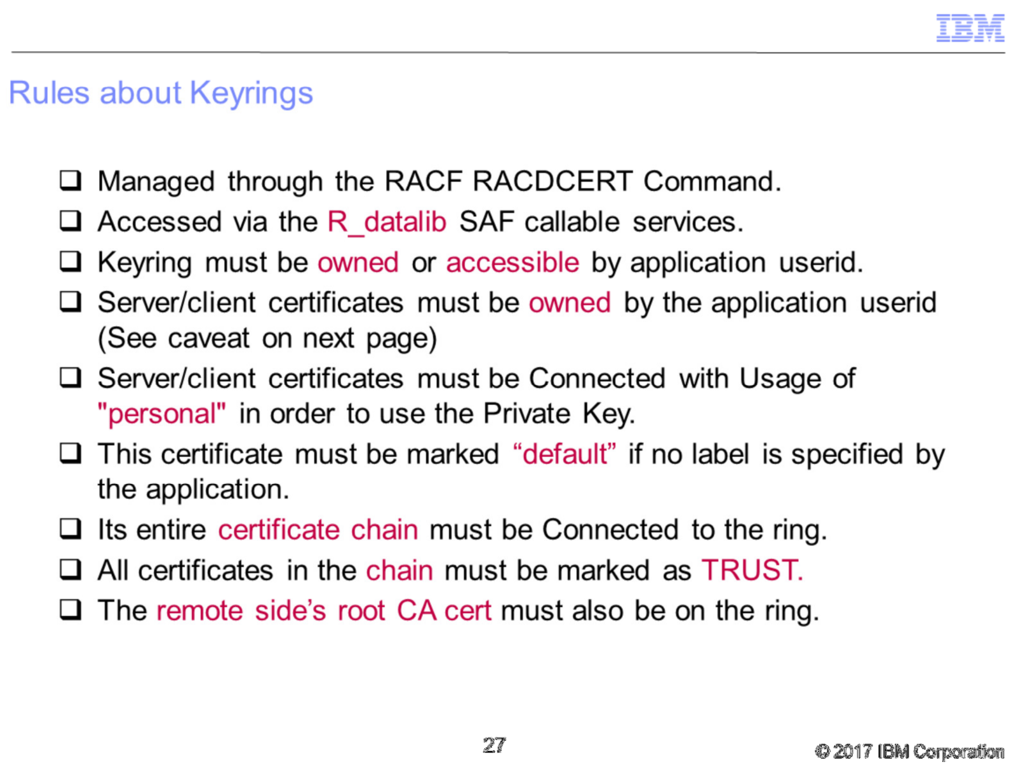
```
RACDCERT ID(WEBS) ADD(CERT.B64)
```

❑NOTE: Do *not* delete the original! (in z/OS 2.1, RACF now refuses to delete a cert that has an outstanding CSR. Msg IRRD198I is issued.)

❑All of these certs can then be added into a keyring as in earlier Example1

RACF Keyrings and Private Keys:

Some of the rules about Keyrings, plus profiles to grant the necessary access



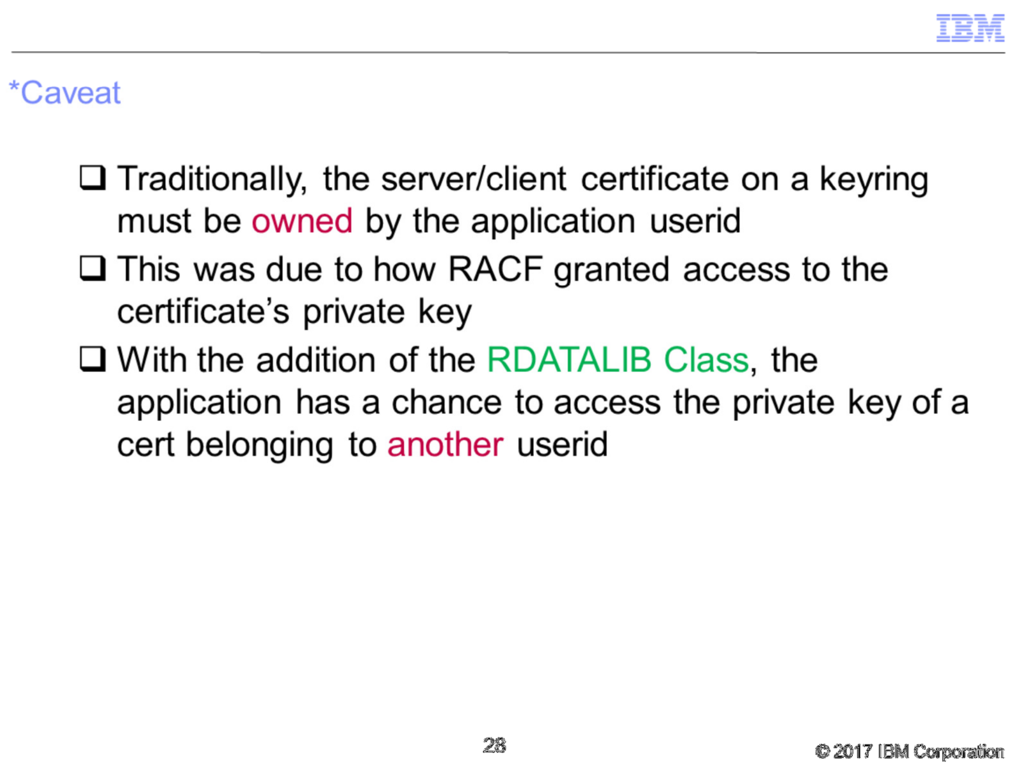
IBM

Rules about Keyrings

- ❑ Managed through the RACF RACDCERT Command.
- ❑ Accessed via the R_datalib SAF callable services.
- ❑ Keyring must be owned or accessible by application userid.
- ❑ Server/client certificates must be owned by the application userid (See caveat on next page)
- ❑ Server/client certificates must be Connected with Usage of "personal" in order to use the Private Key.
- ❑ This certificate must be marked "default" if no label is specified by the application.
- ❑ Its entire certificate chain must be Connected to the ring.
- ❑ All certificates in the chain must be marked as TRUST.
- ❑ The remote side's root CA cert must also be on the ring.

27

System SSL usage of RACF keyrings is through the SAF interfaces (R_datalib). This allows System SSL to work with any external security manager that supports the SAF calls.



*Caveat

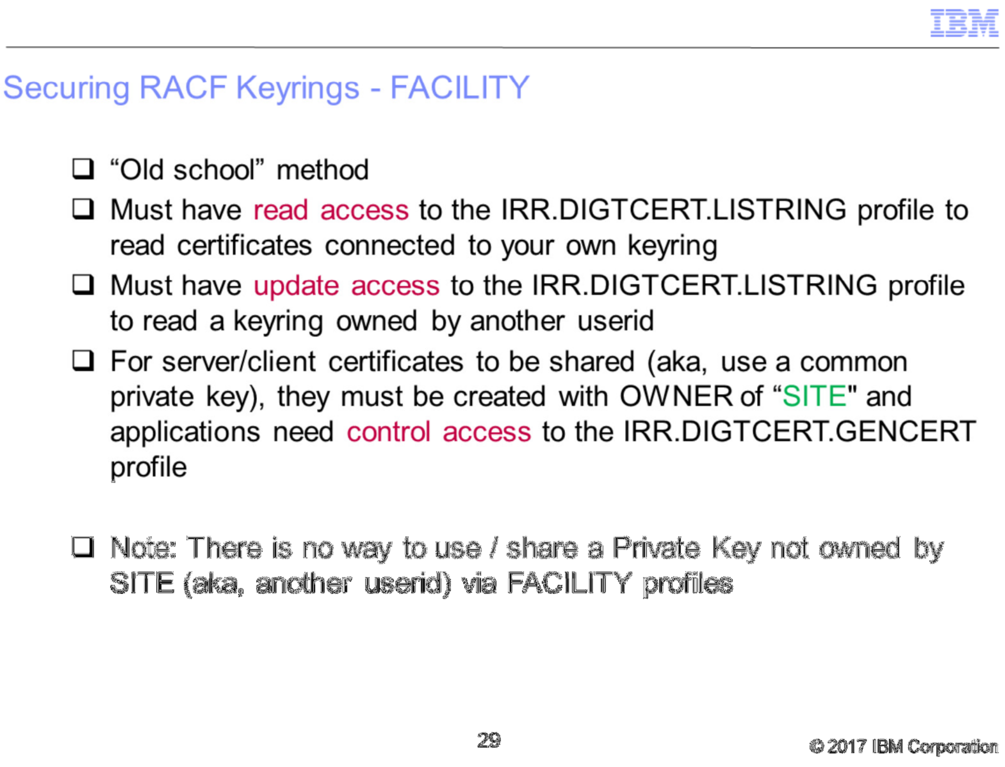
- ❑ Traditionally, the server/client certificate on a keyring must be owned by the application userid
- ❑ This was due to how RACF granted access to the certificate's private key
- ❑ With the addition of the RDATALIB Class, the application has a chance to access the private key of a cert belonging to another userid

28 © 2017 IBM Corporation

* Note: Certificates that are not owned by a userid do not usually have their private key associated with them. This means they cannot be used as client/server certificates; they can only be used in the certificate validation process.

Access to a private key was limited to just your own, or to one loaded into SITE or CERTAUTH (with additional permissions).

Now, the new RDATALIB Class protection allows access to another user's private key. This protection is very specific via the keyring; it is not a global access like the FACILITY Class profiles.



Securing RACF Keyrings - FACILITY

- ❑ "Old school" method
- ❑ Must have read access to the IRR.DIGTCERT.LISTRING profile to read certificates connected to your own keyring
- ❑ Must have update access to the IRR.DIGTCERT.LISTRING profile to read a keyring owned by another userid
- ❑ For server/client certificates to be shared (aka, use a common private key), they must be created with OWNER of "SITE" and applications need control access to the IRR.DIGTCERT.GENCERT profile

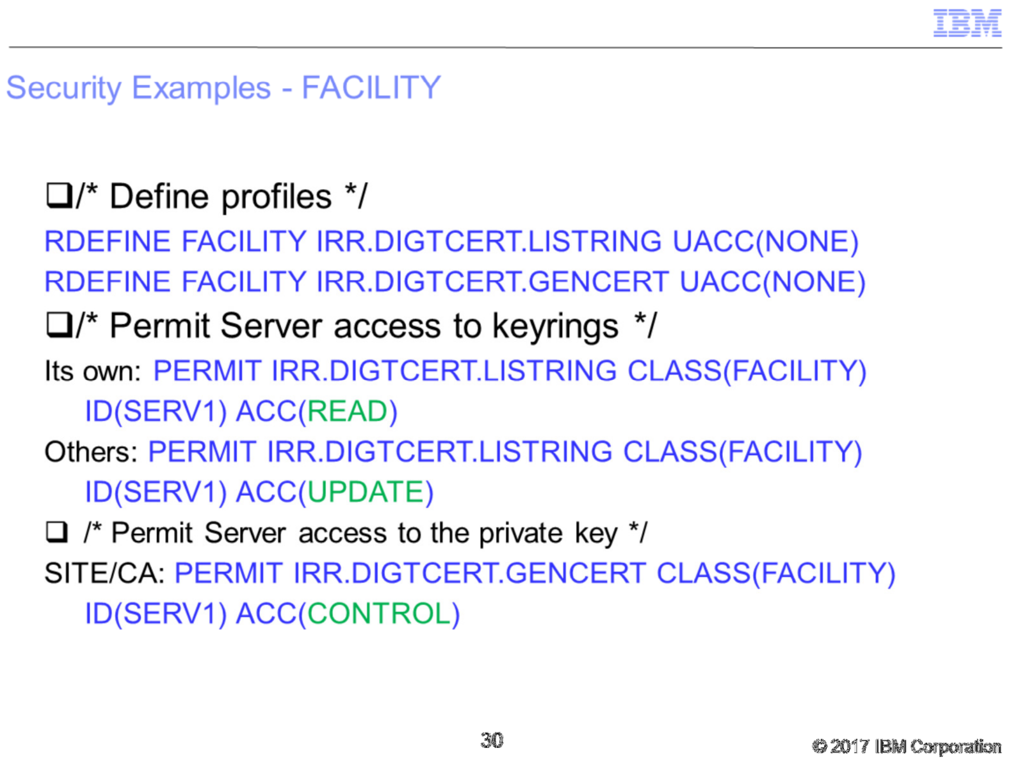
- ❑ Note: There is no way to use / share a Private Key not owned by SITE (aka, another userid) via FACILITY profiles

29 © 2017 IBM Corporation

OLD WAY

To obtain the private key of a SITE or CERTAUTH certificate, the user must CONNECT the SITE certificate with USAGE PERSONAL and the user must have CONTROL on IRR.DIGTCERT.GENCERT .

Only certificates marked as TRUST will be returned on the IRRSDL00 (R_DataLib) interface.



IBM

Security Examples - FACILITY

- /* Define profiles */

```
RDEFINE FACILITY IRR.DIGTCERT.LISTRING UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENCERT UACC(NONE)
```

- /* Permit Server access to keyrings */

Its own: `PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SERV1) ACC(READ)`

Others: `PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(SERV1) ACC(UPDATE)`

- /* Permit Server access to the private key */

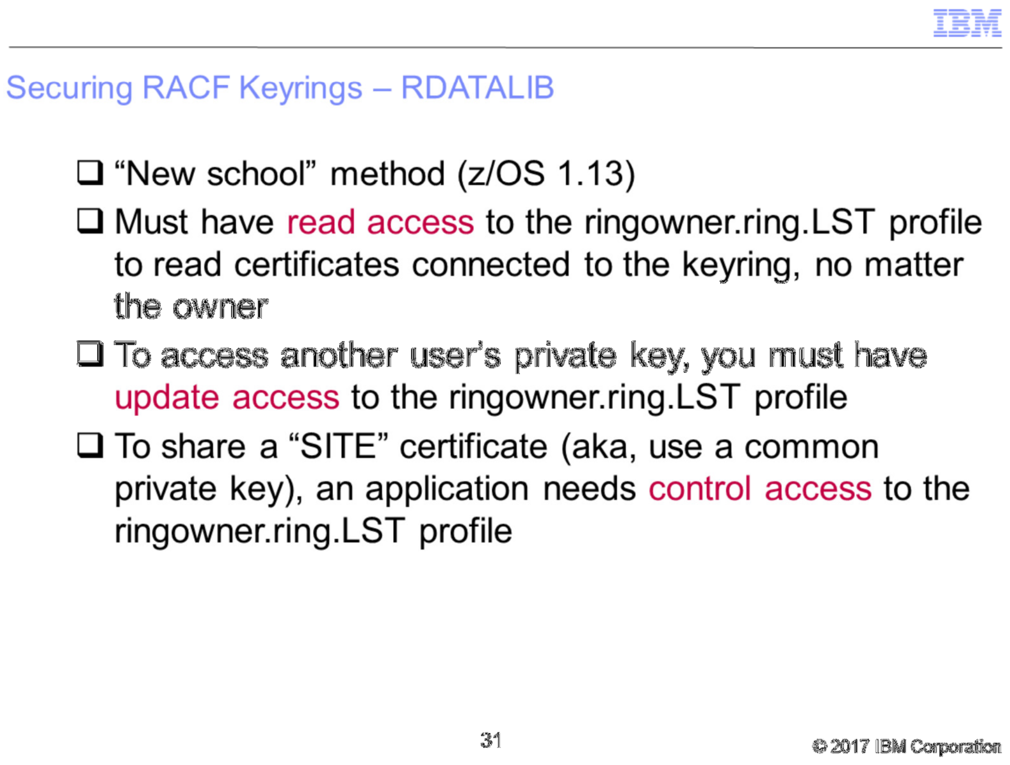
SITE/CA: `PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(SERV1) ACC(CONTROL)`

30 © 2017 IBM Corporation

OLD WAY

These are all of the defined FACILITY CLASS resources used to grant access to all the RACDCERT subcommands:

```
RDEFINE FACILITY IRR.DIGTCERT.ADD    UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ADDRING  UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.DELRING  UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTRING  UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.CONNECT  UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.REMOVE   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LIST    UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ALTER   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.DELETE   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENCERT  UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.GENREQ   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.EXPORT   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.EXPORTKEY UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.MAP    UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ALTMAP   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.DELMAP   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.LISTMAP  UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.REKEY   UACC(NONE)
RDEFINE FACILITY IRR.DIGTCERT.ROLLOVER  UACC(NONE)
```



IBM

Securing RACF Keyrings – RDATALIB

- ❑ “New school” method (z/OS 1.13)
- ❑ Must have read access to the ringowner.ring.LST profile to read certificates connected to the keyring, no matter the owner
- ❑ To access another user’s private key, you must have update access to the ringowner.ring.LST profile
- ❑ To share a “SITE” certificate (aka, use a common private key), an application needs control access to the ringowner.ring.LST profile

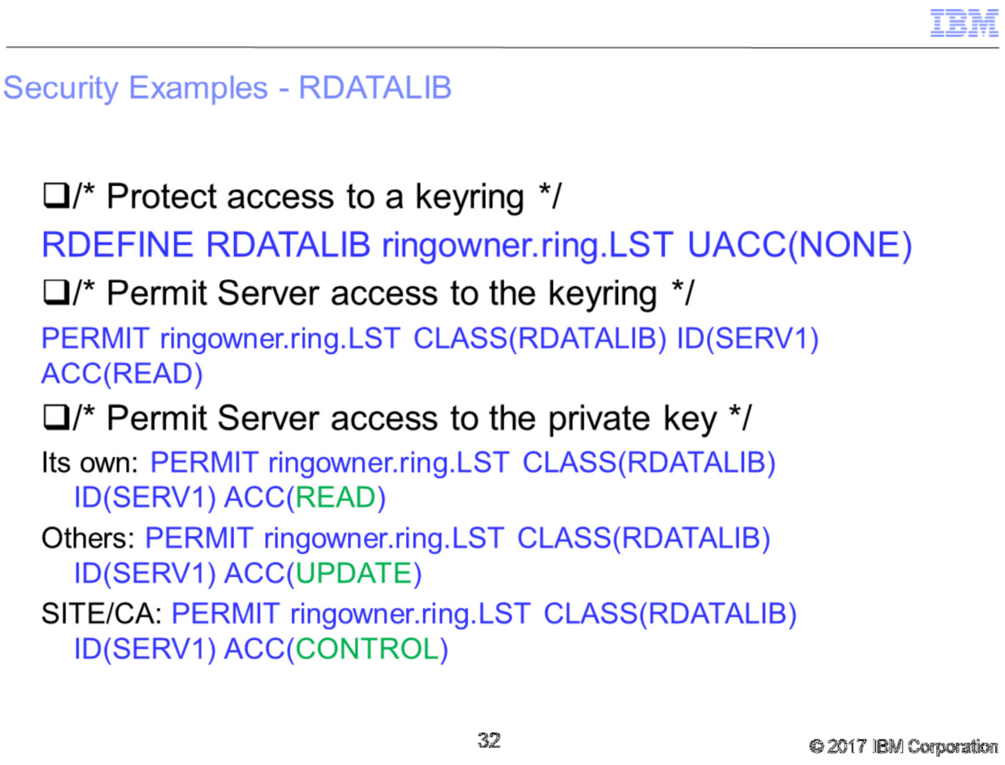
31

NEW WAY

To obtain the private key of another user’s certificate, the user must have UPDATE on ringowner.ring.LST .

To obtain the private key of a SITE or CERTAUTH certificate, the user must CONNECT the SITE or CERTAUTH certificate with USAGE PERSONAL and the user must have CONTROL on ringowner.ring.LST .

Only certificates marked as TRUST will be returned on the IRRSDL00 (R_DataLib) interface.



Security Examples - RDATALIB

- /* Protect access to a keyring */

```
RDEFINE RDATALIB ringowner.ring.LST UACC(NONE)
```

- /* Permit Server access to the keyring */

```
PERMIT ringowner.ring.LST CLASS(RDATALIB) ID(SERV1)
ACC(READ)
```

- /* Permit Server access to the private key */

Its own: `PERMIT ringowner.ring.LST CLASS(RDATALIB) ID(SERV1) ACC(READ)`

Others: `PERMIT ringowner.ring.LST CLASS(RDATALIB) ID(SERV1) ACC(UPDATE)`

SITE/CA: `PERMIT ringowner.ring.LST CLASS(RDATALIB) ID(SERV1) ACC(CONTROL)`

32 © 2017 IBM Corporation

NEW WAY – the RDATALIB Class

RDEFINE RDATALIB user.ring.LST UACC(NONE)

There are other functions for R_datalib that use:

```
RDEFINE RDATALIB user.ring.UPD UACC(NONE)
```

When in doubt…

- ❑ To list the certificates within a dataset:

RACDCERT CHECKCERT(‘dataset’)

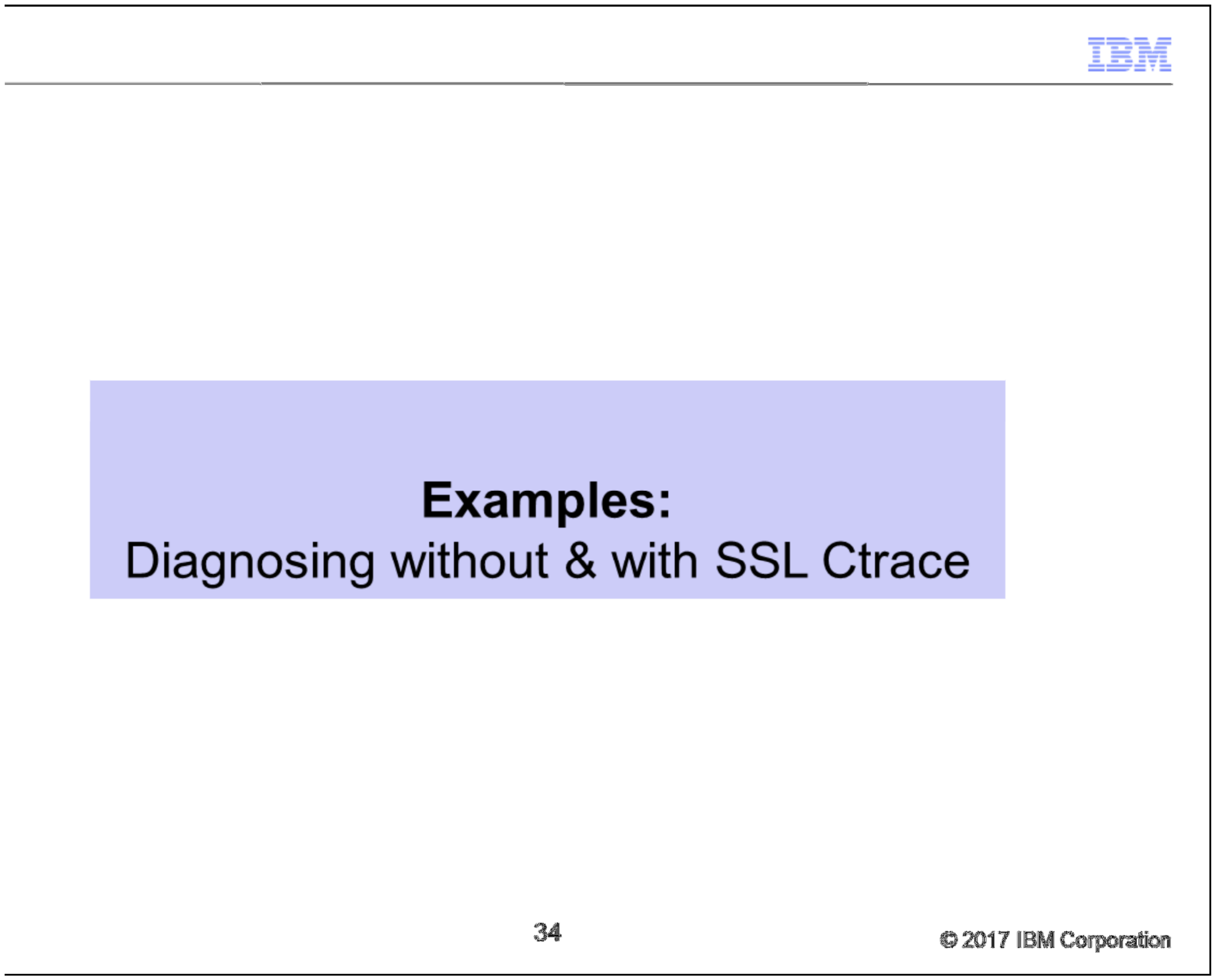
If a pkcs#12 package, add: PASSWORD(‘pswd’)

- ❑ To confirm you have a complete chain on a ring:

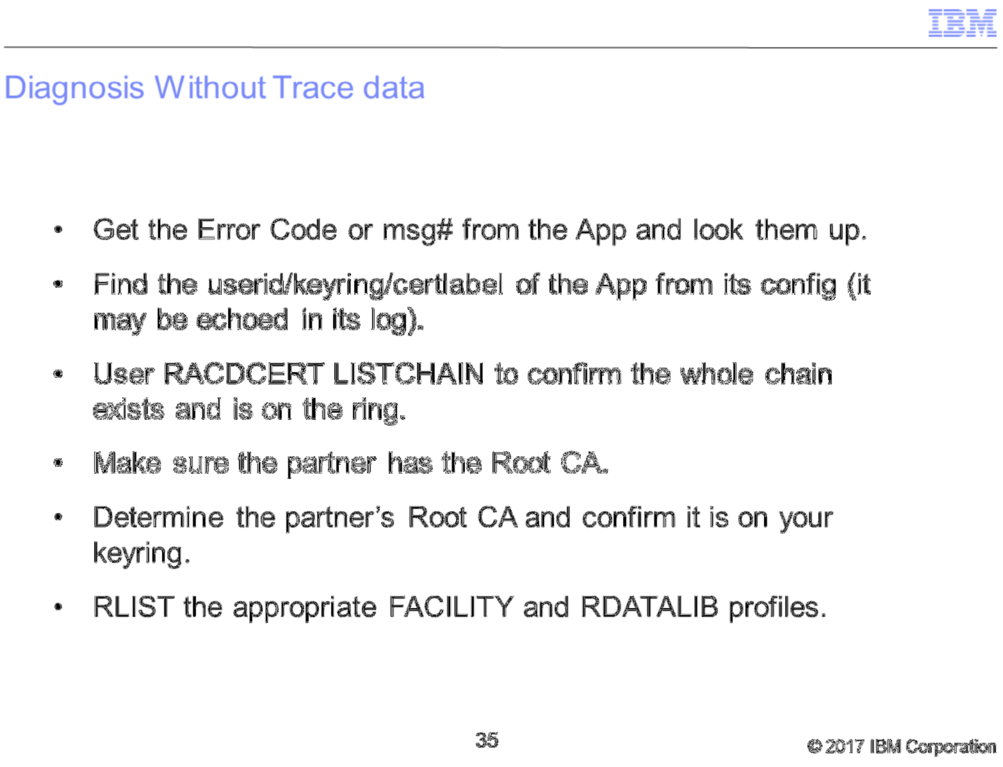
RACDCERT ID(user) LISTRING(myring)

RACDCERT ID(user) LISTCHAIN(LABEL(‘my cert’))

where ‘my cert’ denotes the Server/Client certificate.



Examples:
Diagnosing without & with SSL Ctrace


IBM

Diagnosis Without Trace data

- Get the Error Code or msg# from the App and look them up.
- Find the userid/keyring/certlabel of the App from its config (it may be echoed in its log).
- User RACDCERT LISTCHAIN to confirm the whole chain exists and is on the ring.
- Make sure the partner has the Root CA.
- Determine the partner's Root CA and confirm it is on your keyring.
- RLIST the appropriate FACILITY and RDATALIB profiles.

35 © 2017 IBM Corporation

If RDATALIB and FACILITY profiles both exist, the RDATALIB profiles take precedence.

Application Error

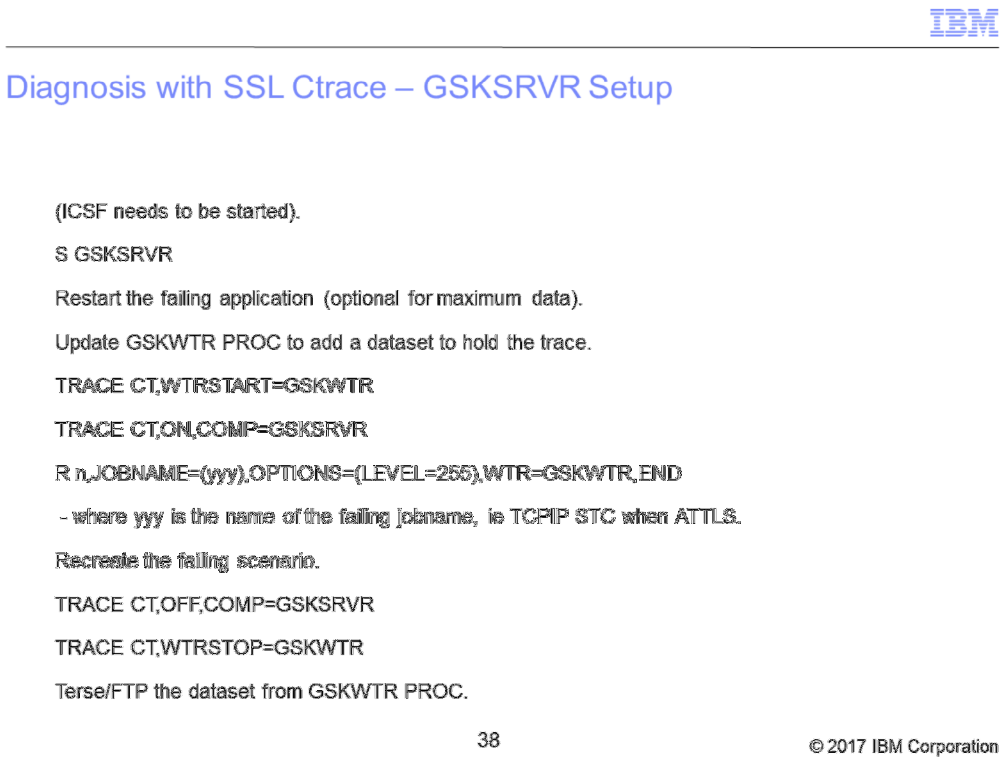
```
EZD1286I TTLS Error GRPID: 0000000B ENVID: 0000001D CONNID: 00005BDC
LOCAL: ::FFFF:123.45.67.890..0123 REMOTE: ::FFFF:234.567.89.12..6789
JOBNAME: abcdabcd USERID: mystc RULE: some-rule  RC:    6 Initial
Handshake 0000000000000000 0000005011421A30 0000000000000000 00000000

EZD1287I TTLS Error RC:  428 Initial Handshake 986
  LOCAL: 10.123.45.67..12345
  REMOTE: 123.456.78.901..1234
  JOBNAME: abcdabcd RULE: some-rule
  USERID: myuser GRPID: 00000037 ENVID: 00000070 CONNID: 0053CA78

IMW6802E SSL Handshake failed: return code 428 (Key entry does not contain a private key).
```

Diagnosis with SSL Ctrace

- ❑There are two methods.
  - ❑Method 1 – using ENVARs via the application;
  - ❑Method 2 – using the GSKSRVR instructions.



IBM

Diagnosis with SSL Ctrace – GSKSRVR Setup

(ICSF needs to be started).

S GSKSRVR

Restart the failing application (optional for maximum data).

Update GSKWTR PROC to add a dataset to hold the trace.

```
TRACE CT,WTRSTART=GSKWTR
TRACE CT,ON,COMP=GSKSRVR
R n,JOBNAME=(yyy),OPTIONS=(LEVEL=255),WTR=GSKWTR,END
```

- where yyy is the name of the failing jobname, ie TCPIP STC when ATTLS.

Recreate the failing scenario.

```
TRACE CT,OFF,COMP=GSKSRVR
TRACE CT,WTRSTOP=GSKWTR
```

Terse/FTP the dataset from GSKWTR PROC.

38

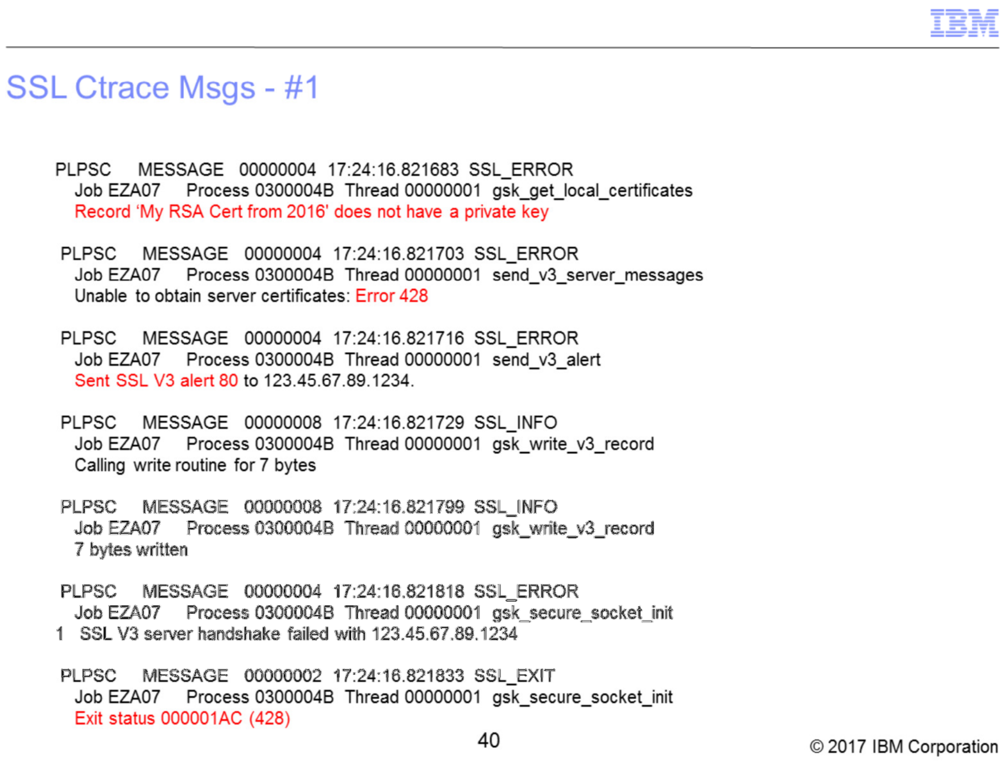
© 2017 IBM Corporation

DO NOT SPECIFY GSKSRVR IN THE JOBNAME= PARM!
Chapters 11 and 12 in the System SSL Programming book describe how to get the trace.
Here is the URL to the System SSL book:
http://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.gska100/sssl2srt1000643.htm (z/OS 2.2)

Chapter 11 describes the configuring the SSL started task, GSK Server. When you get to chapter 12, start with the section called "Capturing Component Trace Data".

Diagnosis with SSL Ctrace – Error codes

- There are several ways an SSL failure can be surfaced:
  - Status Codes
  - Function Return Codes
- An Alert is a standard value SSL reports to the partner to describe a failure. Some common ones we see are: 40, 42, 48, 80.
  - Alerts are documented in the RFCs (see appendix).
- On the peer, a received Alert is converted to an SSL Function Return Code to be passed back to caller.
  - Codes are doc'd in the z/OS System SSL Programming Guide.



SSL Ctrace Msgs - #1

```
PLPSC    MESSAGE   00000004  17:24:16.821683  SSL_ERROR
   Job EZA07     Process 0300004B  Thread 00000001  gsk_get_local_certificates
   Record 'My RSA Cert from 2016' does not have  a private key

 PLPSC    MESSAGE   00000004  17:24:16.821703  SSL_ERROR
   Job EZA07     Process 0300004B  Thread 00000001  send_v3_server_messages
   Unable  to obtain server certificates: Error 428

 PLPSC    MESSAGE   00000004  17:24:16.821716  SSL_ERROR
   Job EZA07     Process 0300004B  Thread 00000001  send_v3_alert
   Sent SSL V3 alert 80 to 123.45.67.89.1234.

 PLPSC    MESSAGE   00000008  17:24:16.821729  SSL_INFO
   Job EZA07     Process 0300004B  Thread 00000001  gsk_write_v3_record
   Calling  write routine  for 7 bytes

 PLPSC    MESSAGE   00000008  17:24:16.821799  SSL_INFO
   Job EZA07     Process 0300004B  Thread 00000001  gsk_write_v3_record
   7 bytes written

 PLPSC    MESSAGE   00000004  17:24:16.821818  SSL_ERROR
   Job EZA07     Process 0300004B  Thread 00000001  gsk_secure_socket_init
1   SSL V3 server handshake  failed with 123.45.67.89.1234

 PLPSC    MESSAGE   00000002  17:24:16.821833  SSL_EXIT
   Job EZA07     Process 0300004B  Thread 00000001  gsk_secure_socket_init
   Exit status 000001AC  (428)
```

40

© 2017 IBM Corporation

The situation:

The Server is starting, has sent its SERVER_HELLO and is now trying to read its own certificate. Whether it was explicitly listed in the app's config file, or obtained as the default on the ring/kdb,

it appears that the private key was not given to SSL upon retrieval.

The error is 428 (GSK_ERR_NO_PRIVATE_KEY). It becomes Alert 80 (INTERNAL_ERROR) which is sent to the peer (Client).

How to resolve this issue

RACDCERT ID(userid) LIST(LABEL('My RSA Cert from 2016'))

- Does the task's assigned userid own the cert?
- Does cert contain private key?

RACDCERT ID(userid) LISTCHAIN(LABEL('My RSA Cert from 2016'))

RACDCERT USER(ringowner) LISTRING(ring)

- Which keyring are they all on? Is the cert marked PERSONAL, etc?

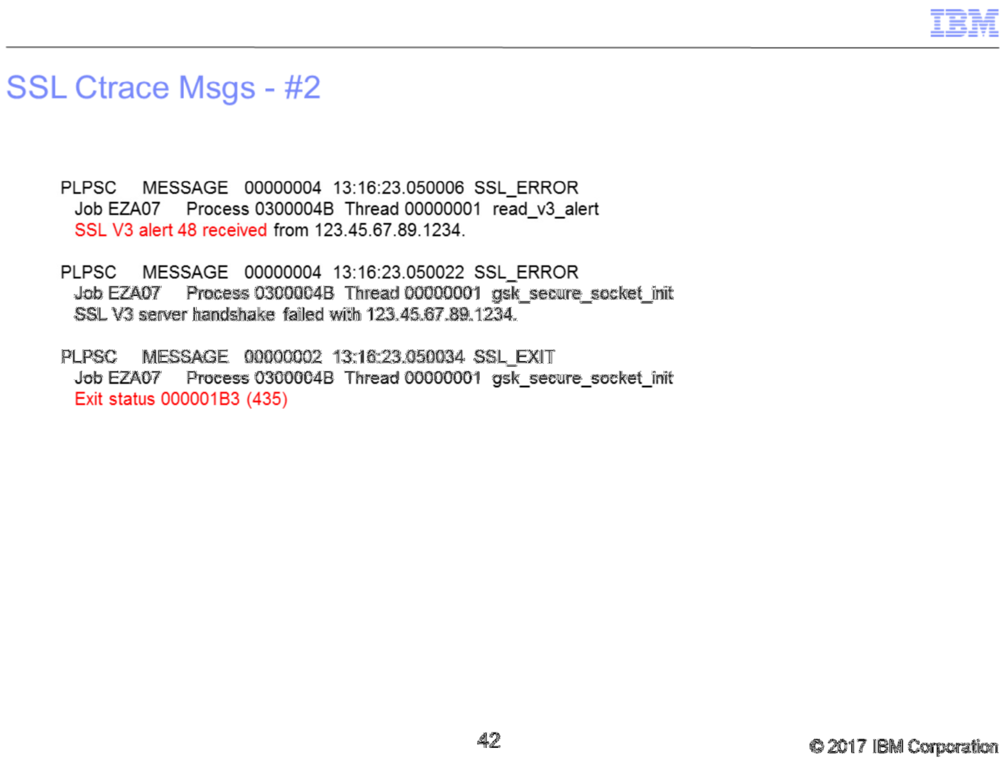
RLIST RDATALIB ringowner.ring.LST AUTH

RLIST FACILITY IRR.DIGTCERT.GENCERT AUTH

RLIST FACILITY IRR.DIGTCERT.LIST AUTH

RLIST FACILITY IRR.DIGTCERT.LISTRING AUTH

- Does the task's userid have appropriate access?



SSL Ctrace Msgs - #2

```
PLPSC    MESSAGE   00000004  13:16:23.050006  SSL_ERROR
  Job EZA07     Process 0300004B  Thread 00000001  read_v3_alert
  SSL V3 alert 48 received from 123.45.67.89.1234.

PLPSC    MESSAGE   00000004  13:16:23.050022  SSL_ERROR
  Job EZA07     Process 0300004B  Thread 00000001  gsk_secure_socket_init
  SSL V3 server handshake failed with 123.45.67.89.1234.

PLPSC    MESSAGE   00000002  13:16:23.050034  SSL_EXIT
  Job EZA07     Process 0300004B  Thread 00000001  gsk_secure_socket_init
  Exit status 000001B3 (435)
```

42

© 2017 IBM Corporation

The situation:

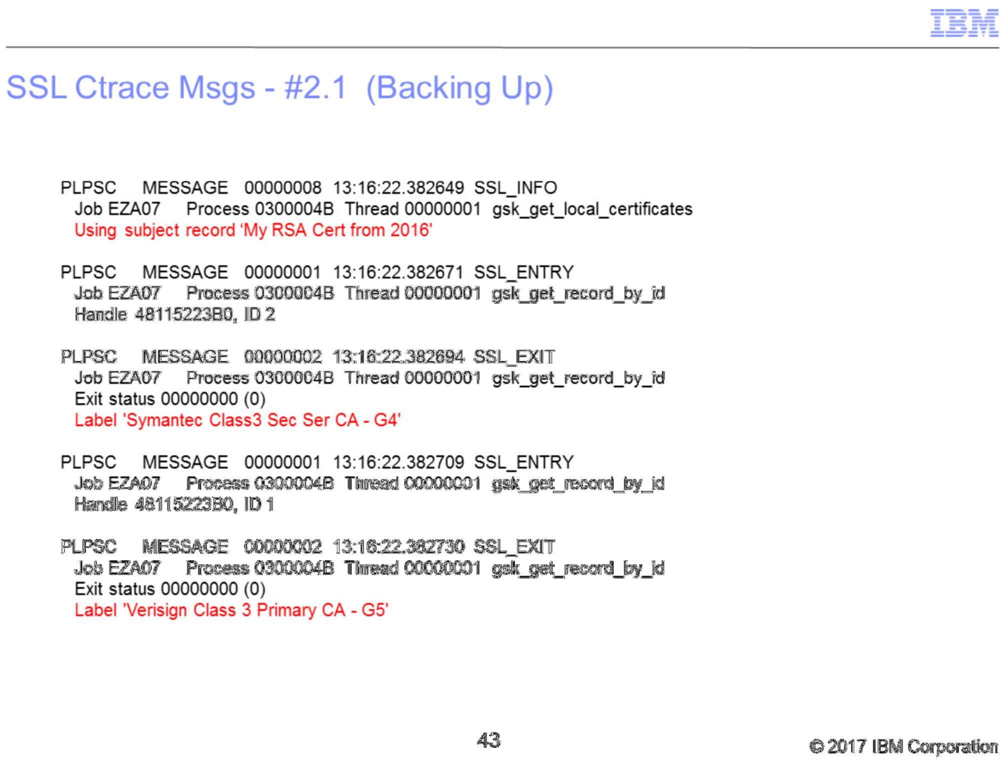
Here, the Server sent its SERVER_HELLO with the Server's Certificate Chain.

It immediately receives Alert 48 (UNKNOWN_CA).

This is turned into Return Code 435 (GSK_ERR_UNKNOWN_CA).

They both mean: Unknown CA.

The Client could not validate the Server's certificate down to a root (or trusted base).


IBM

SSL Ctrace Msgs - #2.1 (Backing Up)

```
PLPSC    MESSAGE   00000008  13:16:22.382649  SSL_INFO
  Job EZA07    Process 0300004B  Thread 00000001  gsk_get_local_certificates
  Using subject record 'My RSA Cert from 2016'

PLPSC    MESSAGE   00000001  13:16:22.382671  SSL_ENTRY
  Job EZA07    Process 0300004B  Thread 00000001  gsk_get_record_by_id
  Handle 48115223B0, ID 2

PLPSC    MESSAGE   00000002  13:16:22.382694  SSL_EXIT
  Job EZA07    Process 0300004B  Thread 00000001  gsk_get_record_by_id
  Exit status 00000000 (0)
  Label 'Symantec Class3 Sec Ser CA - G4'

PLPSC    MESSAGE   00000001  13:16:22.382709  SSL_ENTRY
  Job EZA07    Process 0300004B  Thread 00000001  gsk_get_record_by_id
  Handle 48115223B0, ID 1

PLPSC    MESSAGE   00000002  13:16:22.382730  SSL_EXIT
  Job EZA07    Process 0300004B  Thread 00000001  gsk_get_record_by_id
  Exit status 00000000 (0)
  Label 'Verisign Class 3 Primary CA - G5'
```

43 © 2017 IBM Corporation

The situation continued:

Earlier in the Server Ctrace, we can see the certs that were pulled together to be sent in the SERVER_HELLO.

Compile this list of names and review associated cert data in RACF. In fact, later in the trace (not shown), one can also find the actual certs in DER format, which can be used to derive DN information such as Subject and Issuer.

How to resolve this issue

Get concurrent trace from client.

Use timestamp of Alert on server to orient yourself.

Back up to see where the certificate validation failed.

Review certificates on client keyring to verify necessary root and intermediates exist.

Summary

- We've reviewed what a certificate is an how SSL uses them in a handshake
- We looked at how to create your own certs and add them to keyrings in RACF
- We examined a couple failure scenarios and showed how to diagnose them with and without an SSL Ctrace.
- We showed all the RACF commands you would need along the way.

Appendix

- Additional reference materials
  - **System SSL Programming**
    - https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.gska100/toc.htm
  - The SSL Protocol - Version 3.0 (Netscape)
    - http://wp.netscape.com/eng/ssl3/draft302.txt (obsolete)
    - http://www.ietf.org/rfc/rfc6101.txt
  - RFC 2246 The TLS Protocol Version 1.0
    - http://www.ietf.org/rfc/rfc2246.txt
  - RFC 4346 The TLS Protocol Version 1.1
    - http://www.ietf.org/rfc/rfc4346.txt
  - RFC 5246 The TLS Protocol Version 1.2
    - http://www.ietf.org/rfc/rfc5246.txt
  - RFC 3280 Internet X.509 Public Key Infrastructure Certificate and CRL Profile
    - http://www.ietf.org/rfc/rfc3280.txt
  - FIPS 186 Digital Signature Standard
    - http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

Appendix – Cont.

- Introduction to Public-Key Cryptography
  - http://www.rsasecurity.com/rsalabs/node.asp?id=2165
- PKCS #7 Standard Message Syntax
  - http://www.rsasecurity.com/rsalabs/node.asp?id=2129
- PKCS#12
  - http://www.rsasecurity.com/rsalabs/node.asp?id=2138
- ICSF Application Programmer's Guide
  - Hardware Crypto callable services
- RACF Command Language Reference – RACDCERT
  - https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.icha400/radcertg.htm#radcertg
- RACF Callable Services – R_datalib
  - https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.ichd100/datalib.htm
- RACF Security Administration Guide – using a SITE cert
  - https://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.icha700/scen7.htm

Credits

- ❑ Bill Wandell – original author, pre-2006
- ❑ Dave Hilliard & Laura Sperling – updated 2014
- ❑ John Reale, III & Laura Sperling – updated 2017