
# Digital Certificates - What & How On ESMs

**Jamieson Walker - Software Engineer - Broadcom**

**jamieson.walker@broadcom.com**

**Seamus hayes - Software Engineer - Broadcom**

**seamus.hayes@broadcom.com**

# What is a certificate?

- File that contains a cryptographic key, details about the organization to which it belongs, and a signature verifying the validity of its contents.
- Used in communication Transport Layer Security (TLS) as well as HTTPS web browsing.
- Communicating with the use of certificates provides :
  - **Authentication** - The receiver has reason to believe the message was created and sent by a specific sender.
  - **Non-Repudiation** - The sender cannot deny having sent the message.
  - **Integrity** - Ensures the message was not altered in transit.
  - **Privacy** - Only the intended recipient can decipher the message

# Cryptography

# Symmetric Encryption

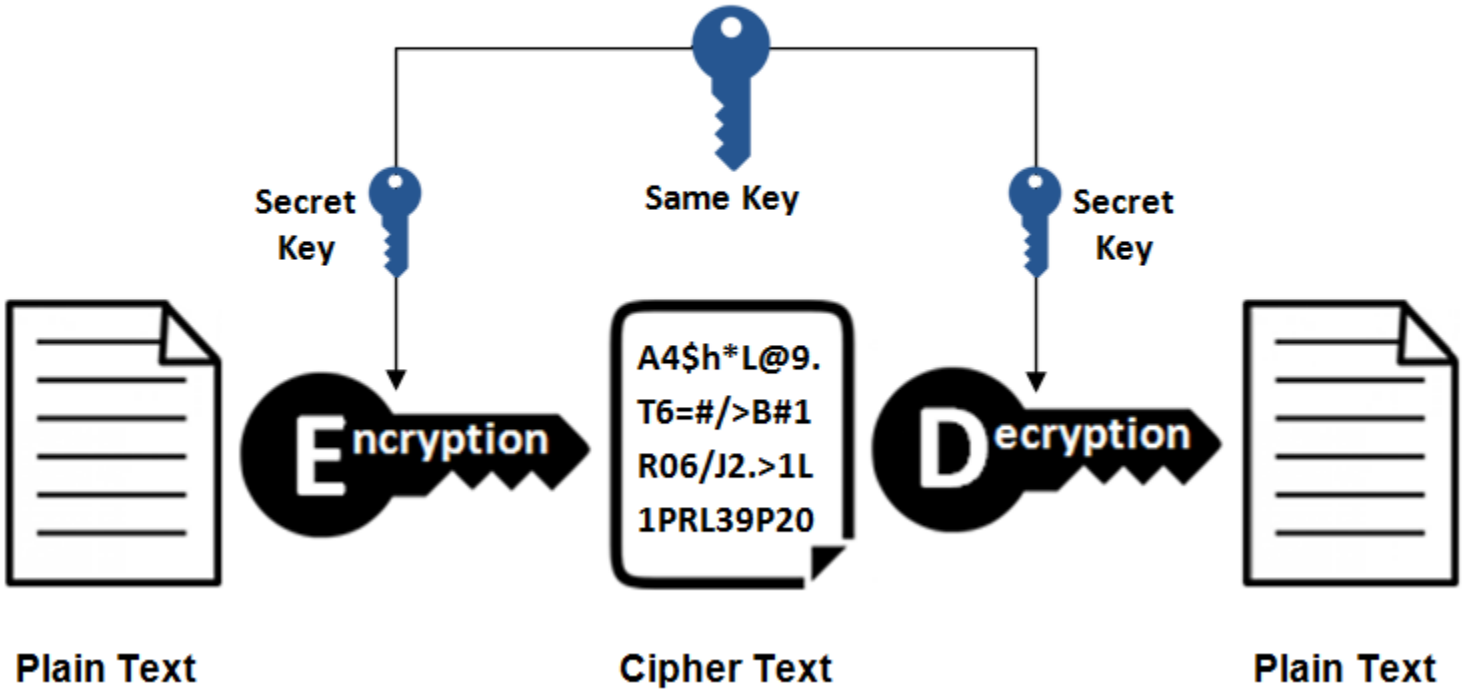



Diagram Credit: https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

Alice

Bob

Sensitive document

Source: Asymmetric encryption - Simply explained Savjee

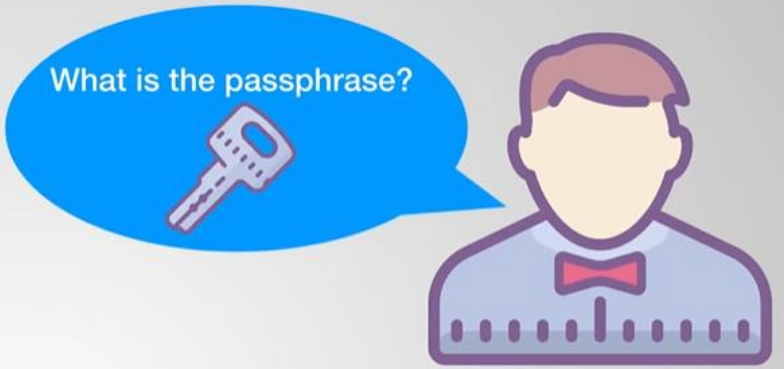
Alice

Bob

Encrypted, senstive document

Source: Asymmetric encryption - Simply explained Savjee

Alice




Bob

Source: Asymmetric encryption - Simply explained Savjee

Alice

Bob

Passphrase

Source: Asymmetric encryption - Simply explained Savjee

Alice

Bob

Source: Asymmetric encryption - Simply explained Savjee

Alice

Bob

Source: Asymmetric encryption - Simply explained Savjee

# How can we securely share a key?

# Asymmetric Encryption

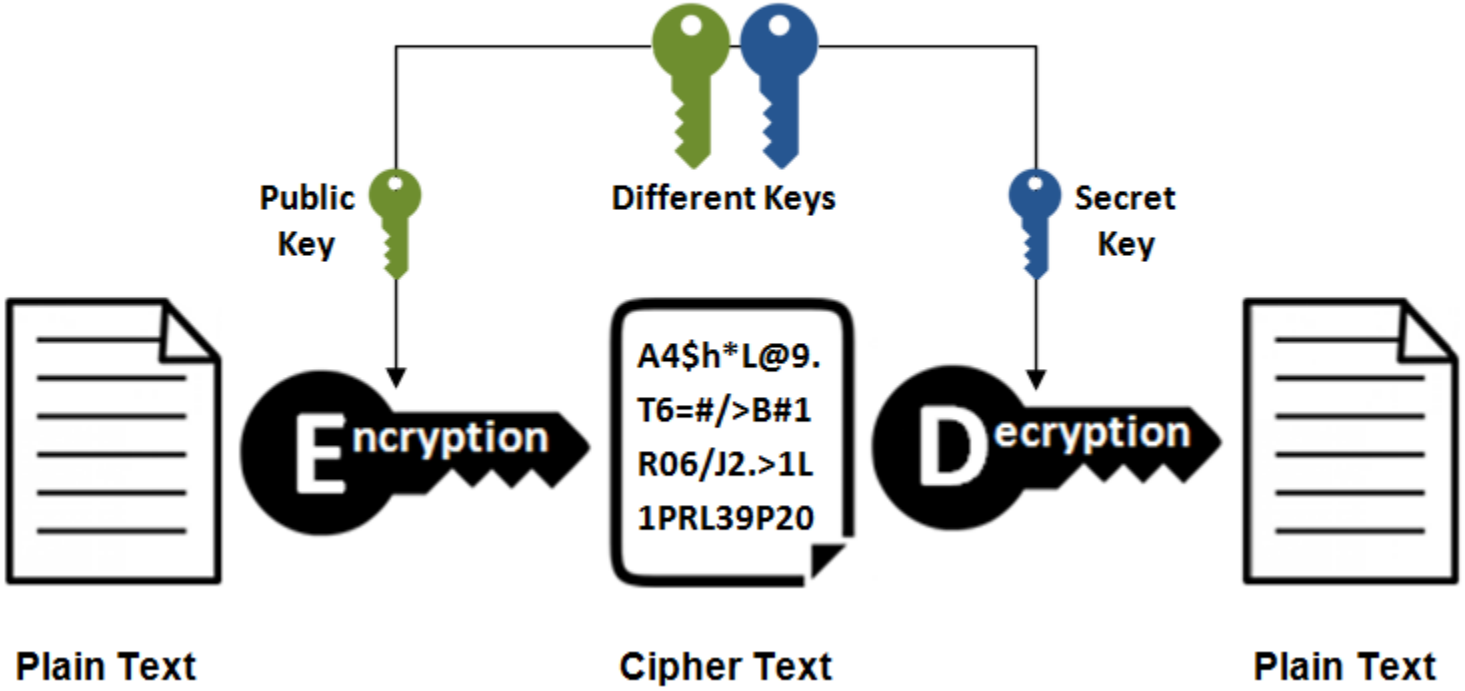



Diagram Credit: https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

Private key

Public key

RSA algorithm

Private key

Public key

Source: Asymmetric encryption - Simply explained Savjee

Private key

Public key

Bob’s public key

Private key

Public key

Alice’s public key

Source: Asymmetric encryption - Simply explained Savjee

Encrypted with
Bob's public key

Source: Asymmetric encryption - Simply explained Savjee

Decrypted with
Bob’s private key

Source: Asymmetric encryption - Simply explained Savjee

Source: Asymmetric encryption - Simply explained Savjee

Source: Asymmetric encryption - Simply explained Savjee

# Public Key Cryptography / Asymmetric Cryptography

- Type of encryption that operates using a public and private key pair.
  - Public key may be disseminated widely.
  - Security is reliant of the private key being known only to the owner.
- The public key is used to encrypt a message. Only the corresponding private key can decrypt that message.
- Keys are simply numbers.  Numbers with more digits are more secure.

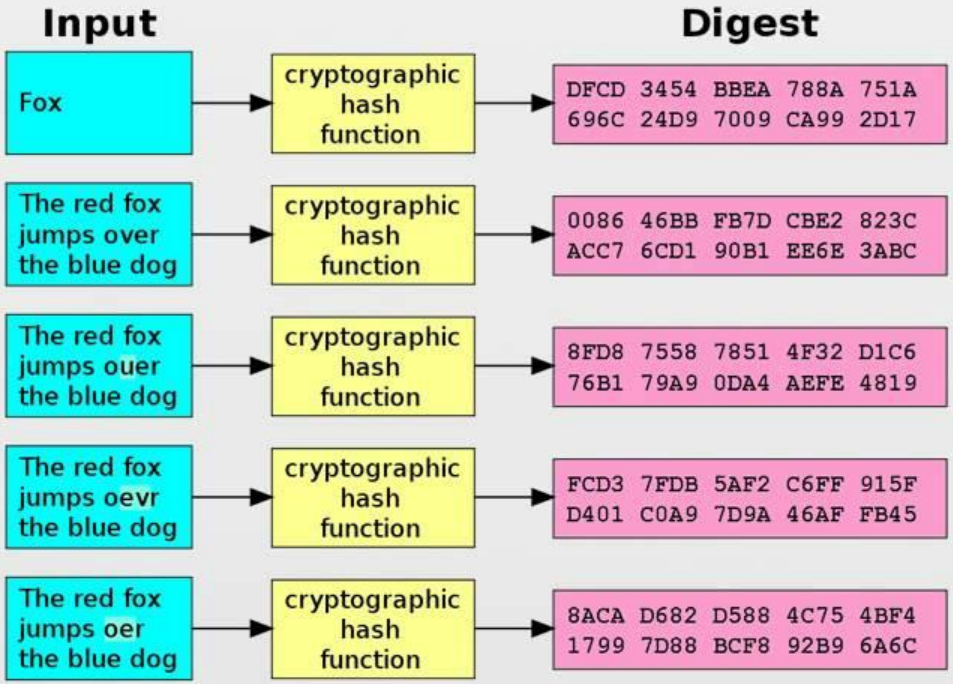
# Example Key (1024 bits, PCKS#1 Format)

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCqGKukO1De7zhZj6+H0qtjTkVxwTCpvKe4eCZ0FPqri0cb2JZfXJ/DgYSF6vUp
wmJG8wVQZKjeGcjDOL5UlsuusFncCzWBQ7RKNUSesmQRMSGkVb1/3j+skZ6UtW+5u09lHNsj6tQ5
1s1SPrCBkedbNf0Tp0GbMJDyR4e9T04ZZwIDAQABAoGAFijko56+qGyN8M0RVyaRAXz++xTqHBLh
3tx4VgMtrQ+WEgCjhoTwo23KMBAuJGSYnRmoBZM3lMfTKevIkAidPExvYCdm5dYq3XToLkkLv5L2
pIIVOFMDG+KESnAFV7l2c+cnzRMW0+b6f8mR1CJzZuxVLL6Q02fvLi55/mbSYxECQQDeAw6fiIQX
GukBI4eMZZt4nscy2o12KyYner3VpoeE+Np2q+Z3pvAMd/aNzQ/W9WaI+NRfcxUJrmfPwIGm63il
AkEAxCL5HQb2bQr4ByorcMWm/hEP2MZzROV73yF41hPsRC9m66KrheO9HPTJuo3/9s5p+sqGxOlF
L0NDt4SkosjgGwJAFklyR1uZ/wPJjj611cdBcztlPdqoxssQGnh85BzCj/u3WqBpE2vjvyyvyI5k
X6zk7S0ljKtt2jny2+00VsBerQJBAJGC1Mg5Oydo5NwD6BiROrPxGo2bpTbu/fhrT8ebHkTz2epl
U9VQQSQzY1oZMVX8i1m5WUTLPz2yLJIBQVdXqhMCQBGoiuSoSjafUhV7i1cEGpb88h5NBYZzWXGZ
37sJ5QsW+sJyoNde3xH8vdXhzU7eT82D6X/scw9RZz+/6rCJ4p0=
-----END RSA PRIVATE KEY-----
```

# Hashing

- Hash Function: A one-way function that converts any form of data into a unique sequence of bytes.
  - Input may be any data and have any size
  - The output hash is always the same size, regardless of input
  - One-Way meaning you cannot determine the input from the output
  - The same input always produces the same output
  - Changing even 1 bit of the input data will produce a completely different output

# Cryptographic hash function




https://en.wikipedia.org/wiki/File:Cryptographic_Hash_Function.svg

# How is this used in the real world?

# Digital Signature

- Not used to encrypt the message itself, but attaches a signature on the message to verify contents and sender.
- Signed with the sender's private key and can be verified by anyone with the sender's public key.
- Three Purposes
  - **Authentication** - The receiver has reason to believe the message was created and sent by a specific sender.
  - **Non-Repudiation** - The sender cannot deny having sent the message.
  - **Integrity** - Ensures the message was not altered in transit.
- Prevents forgery and tampering.

# Encryption VS Signing

- When encrypting, you use **their public key** to encrypt message and they use **their private key** to decrypt it.
- When signing, you use **your private key** to sign a message, and they use **your public key** to verify the signature.

- Encryption/decryption and signing/verifying are mathematically similar, but it's important to keep the terminology distinct.

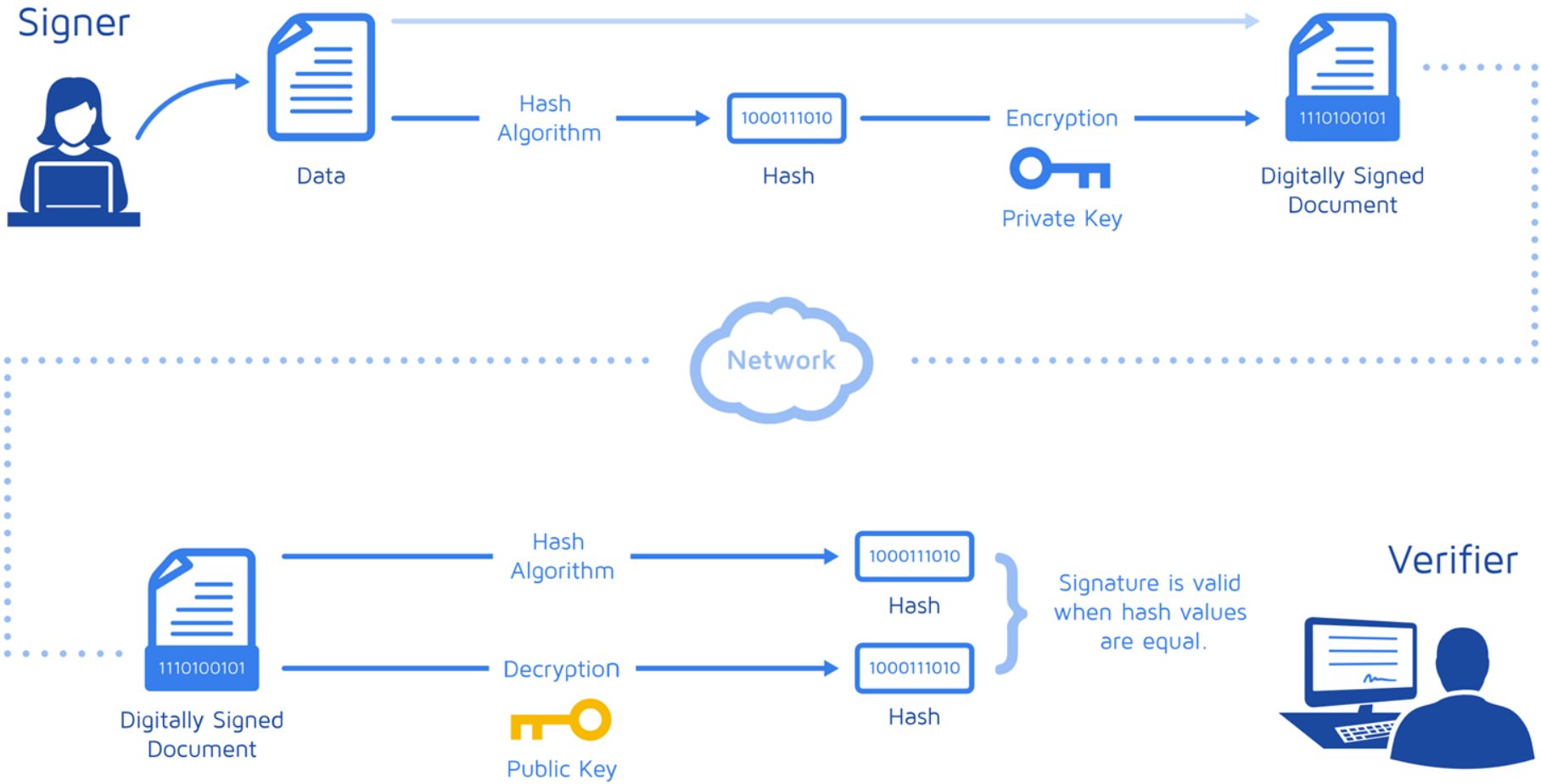
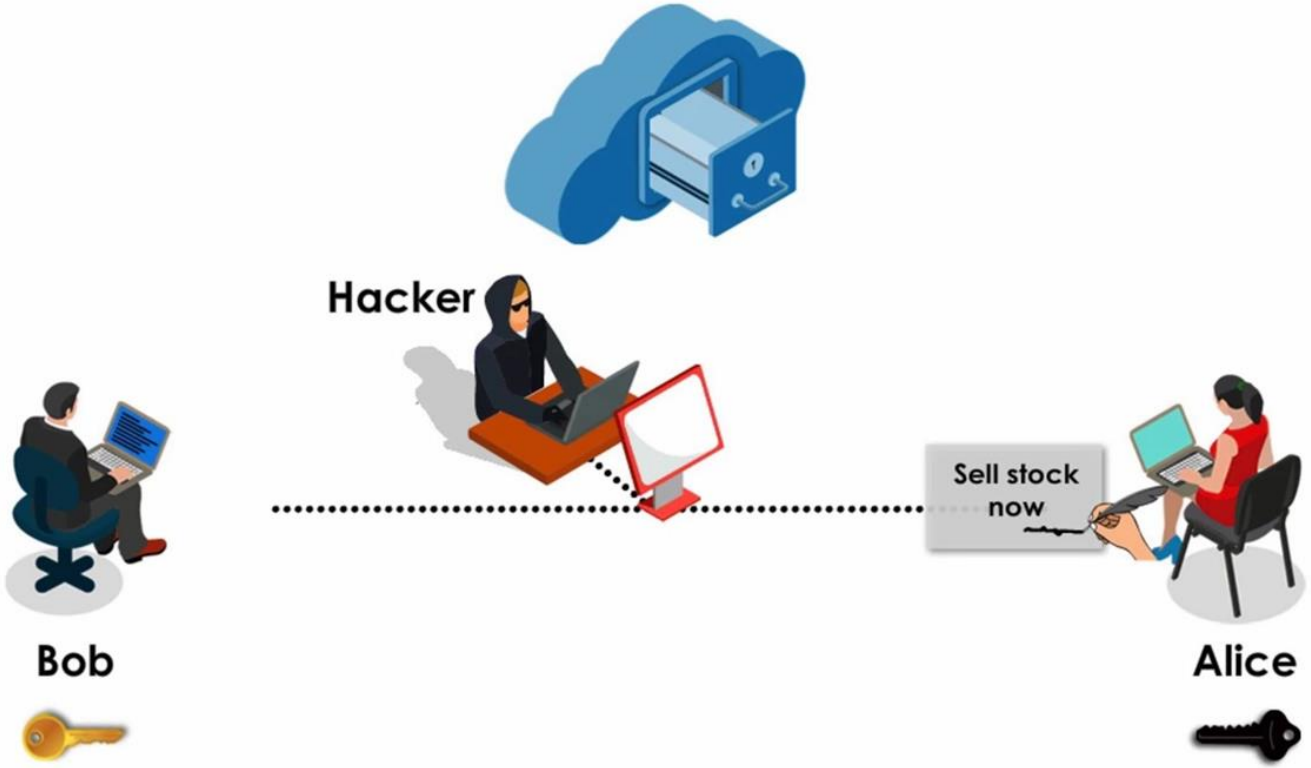



Diagram: https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq



# How we can be more confident in a sender's identity?

# Digital Certificate

- Electronic credentials issued by a trusted third party.  It not only verifies the identity of the owner, but also verifies the owner's public key.
- Prevents the man in the middle vulnerability of digital signatures.
- Based on trust of the certificate/signing authority.
- Certificate includes information such as:
  - Certificate owner's Name
  - Owner's public key and its expiration date
  - Certificate issuer's name
  - Certificate issuer's digital signature

# Certificate Authority (CA)

- IdenTrust, Comodo, DigiCert, GoDaddy, GlobalSign (Top 5 by Market Share)
- A Certificate Authority is responsible for verifying the identities of people or organizations applying for a certificate.
  - If verified, the CA will create a digital certificate using the information contained in the applicants Certificate Signing Request (CSR) .
- Web browsers and other applications come with certificate authorities certificates (including their public keys) pre-installed.

# Certificate Signing Request (CSR)

- Sent to a CA in order to have them generate signed certificate
- What is inside a CSR?
  - Public Key
  - Common Name ex: *.google.com
  - Organization ex: Google, Inc.
  - Division ex: Google Maps
  - City ex: Pittsburgh
  - State ex: Pennsylvania
  - Country ex: USA
  - Contact Info

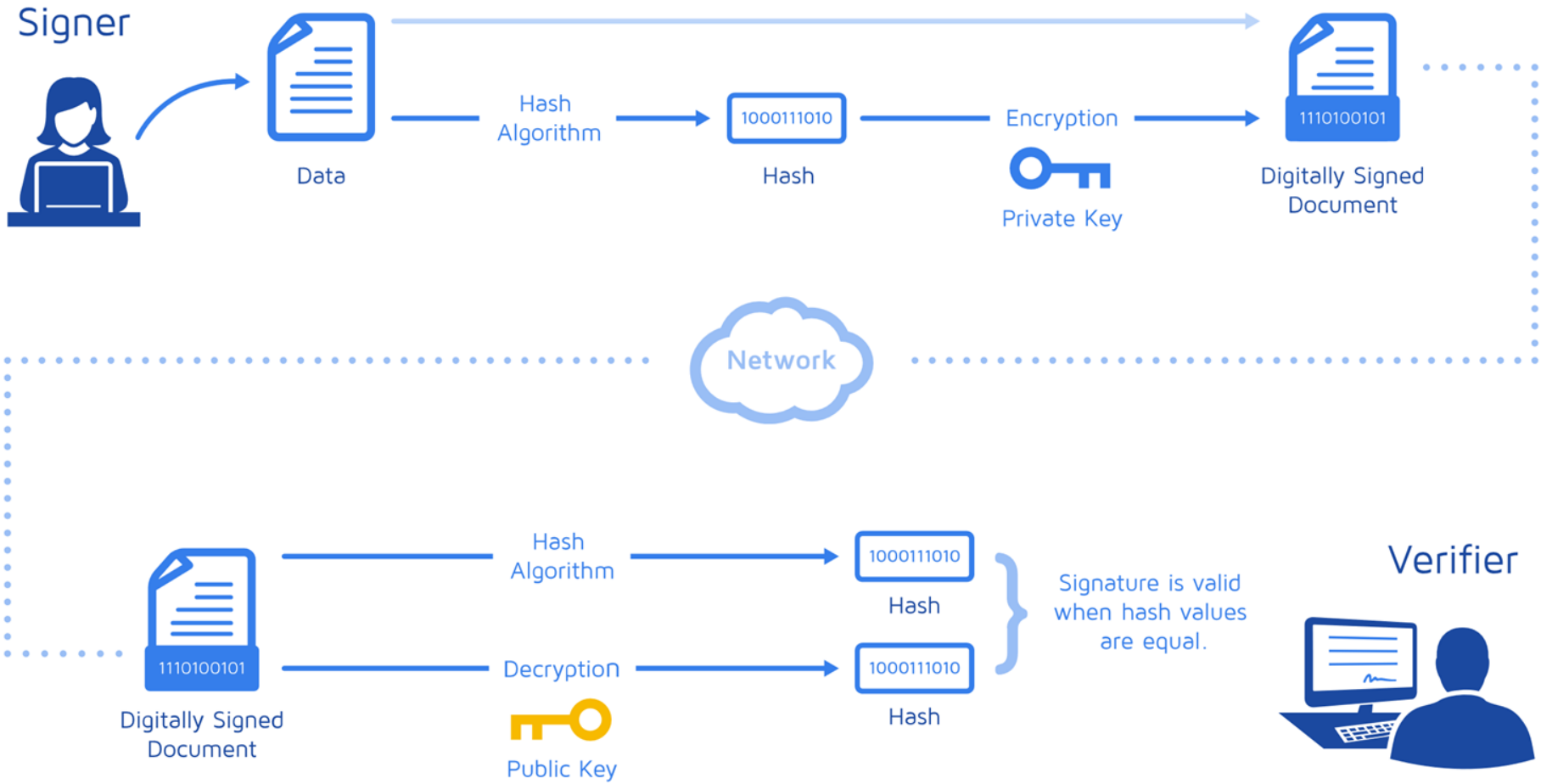
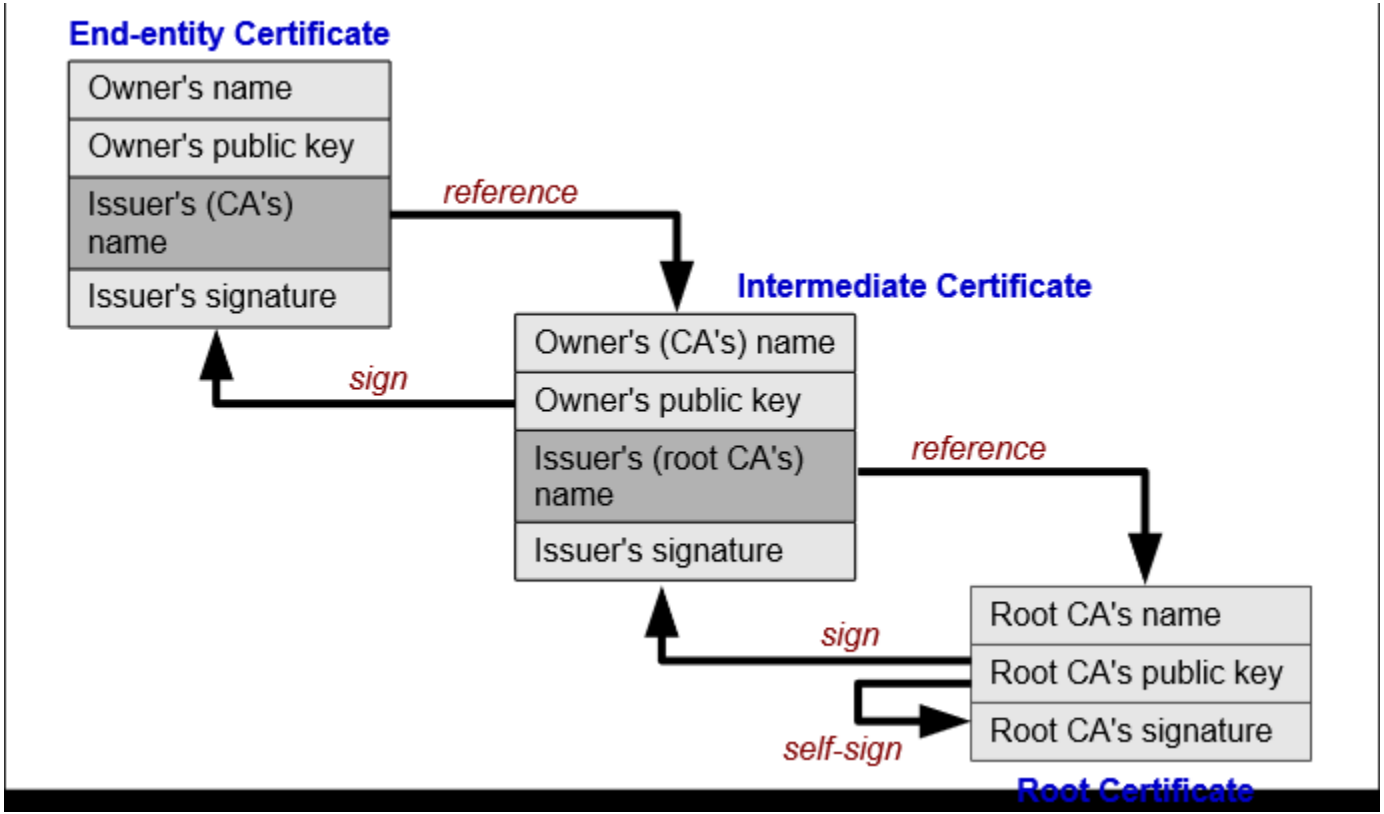



Diagram: https://www.docusign.com/how-it-works/electronic-signature/digital-signature/digital-signature-faq




Source: https://en.wikipedia.org/wiki/Chain_of_trust

# Public Key Infrastructure (PKI)

- A public key infrastructure is a system of digital certificates, Certificate Authorities, and other registration authorities used to verify and authenticate the identity of parties involved in an internet transaction.
  - Also known as **chain of trust** or trust hierarchy

# Key Store

- Typically a file containing self-identifying information such as certificates and private keys and their corresponding public keys.
  - E.g. keystore.jks
- Used to authenticate yourself to a remote party

# Trust Store

- Stores certificates from trusted parties such Certificate Authorities
- Used to verify remote certificates that you don’t already know and trust.
- Just another keystore, but used for a different purpose.

# Self-Signed Certificate

- A certificate signed by the same entity whose identity it certifies
  - Ex: Root Certificate owned by a Certificate Authority
- It is not advised to use your own self-signed certificates for anything other than testing purposes.

# Digital Certificates on z/OS

# What we have covered so far

- What is Cryptography
- Public/private key Cryptography (PKI)
- Real World Use

# Digital Certificates on the Mainframe

1. z/OS Terminology
2. z/OS Theory
3. z/OS Case Study: Tomcat Server on z/OS with Digicert Signed certificate

# z/OS Terminology

- ESM certificate repositories
- ICSF - Integrated Cryptographic Service Facility
- Keyring
- Virtual Keyring
- Special Users

# ESM Certificate Repositories

ESMs can store digital certificates and restrict access to them using the already familiar paradigm of Resource Rules.

- RACF database
- ACF2 INFOSTG database
- Top Secret security file database

Applications ask SAF for certificate data via an R_datalib call.  If allowed by the ESM: the certificate data will be returned to the requesting application.

Certificate files can be used on z/OS in USS but this has many shortcomings.

# ICSF - Integrated Cryptographic Service Facility

ICSF is a software component for z/OS.  That in conjunction with the hardware cryptographic features and an ESM(RACF) provides high speed cryptographic services.

What ICSF Provides:

- Added security
- Private key encryption
- Leverage hardware cryptographic coprocessor
- Protected by CSFKEYS and CSFSERV classes

# z/OS Keyring

A Keyring is a collection of digital certificates associated with a server or client task userID

- Keyrings are associated with a userid(owner)
- Certificates are CONNECTed to a Keyring
- The Client/Server Task's parameter file points to a Keyring
- The Client/Server task issues R_datalib calls when it starts to request the Keyring and all of the certificates CONNECTed to the Keyring
- Connected Certificate usage types:
  - PERSONAL - access public and private key
  - CERTAUTH - access public key
- Types of ESM Keyrings

1. Explicitly defined Keyring in the ESM database
2. Virtual Keyring

# z/OS Keyring - Virtual Keyrings

Virtual Keyrings

A virtual keyring is the set of all CERTAUTH or SITECERT certificates

- Virtual Keyring is specified with ‘*`AUTH*/*’ or ‘*SITE*/*’ in the parameter file of the client/server
- All valid, not expired, CERTAUTH/SITECERT will be returned by the ESM(R_datalib calls)

# Special Users(Certificate Owners)

Special Owners

- CERTAUTH
- SITE(SITECERT)

Regular Owner

- ID(USERID)

# z/OS Case Study

I have a tomcat server on z/OS but I want to secure connections to it

1. I would like to secure connections to the server with HTTPS
2. I must have access to administer certificates within the ESM
3. I must grant my tomcat server ID access to certificates it needs
4. I would like to use a certificate signed by DigiCert CA
5. I would like to store the server certificate in the ESM database
6. I would like to leverage digital keyrings

# z/OS Case Study - Permit Access via ESM

I am a security administrator on the system - I can do whatever I please

To complete this exercise I would need CONTROL access to the following FACILITY(ies)

1. IRR.DIGTCERT.ADD
2. IRR.DIGTCERT.ADDRING
3. IRR.DIGTCERT.CONNECT
4. IRR.DIGTCERT.GENCERT
5. IRR.DIGTCERT.GENREQ
6. IRR.DIGTCERT.LIST - Not required but we want to see what we have done
7. IRR.DIGTCERT.LISTRING - Not required but we want to see what we have done

# z/OS Case Study - Permit Access via ESM

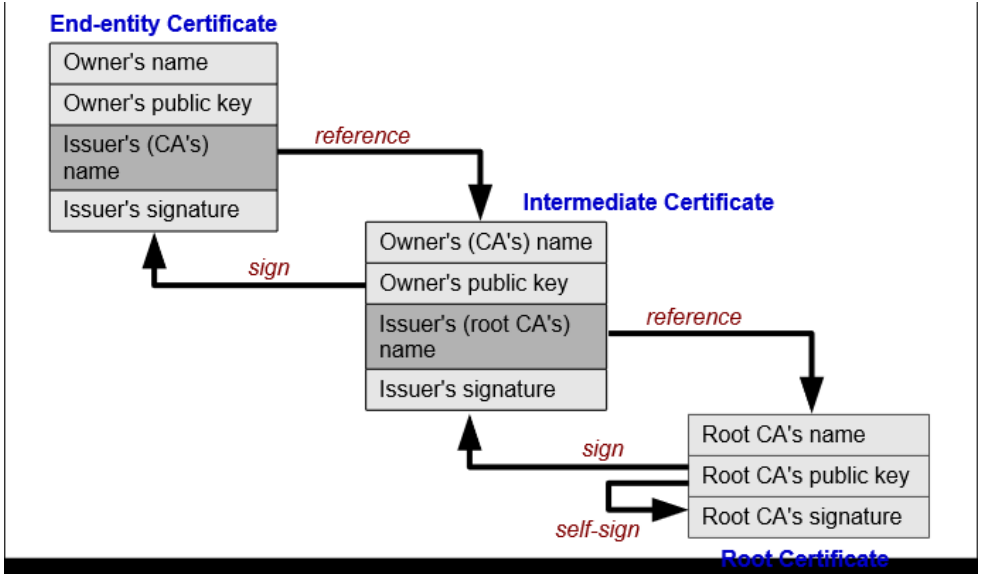
Assume the user running the tomcat STC is SHRSRV.

SHRSRV will need READ access to the following FACILITY:

1. IRR.DIGTCERT.LISTRING

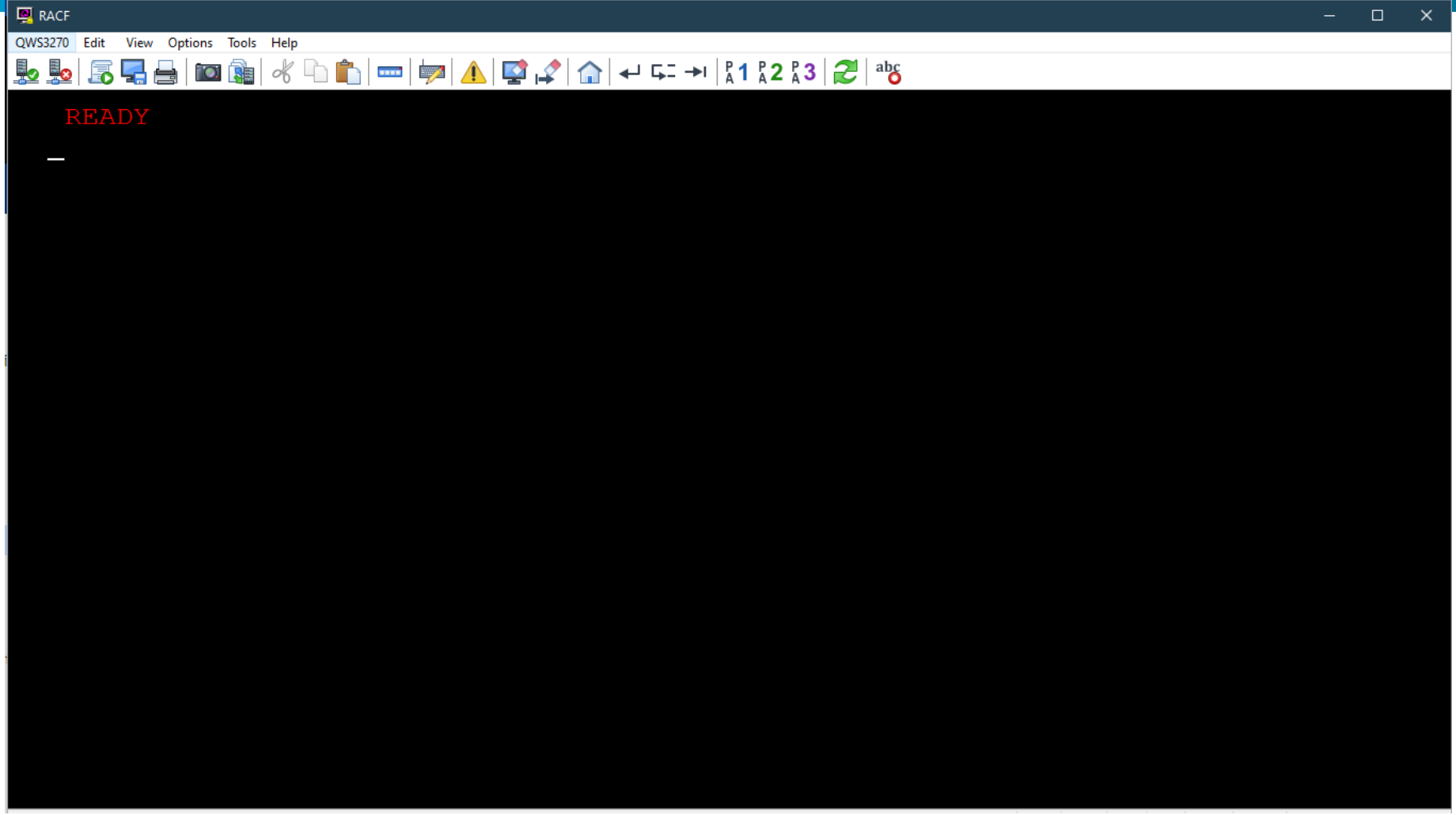
# z/OS Case Study - Secure connection

- I need  a server certificate
- It must be accessible to the server
- Its private key must be secure
- It should be publicly recognized






BROADCOM






BROADCOM

# z/OS Case Study - Certificate signed by DigiCert - RACF

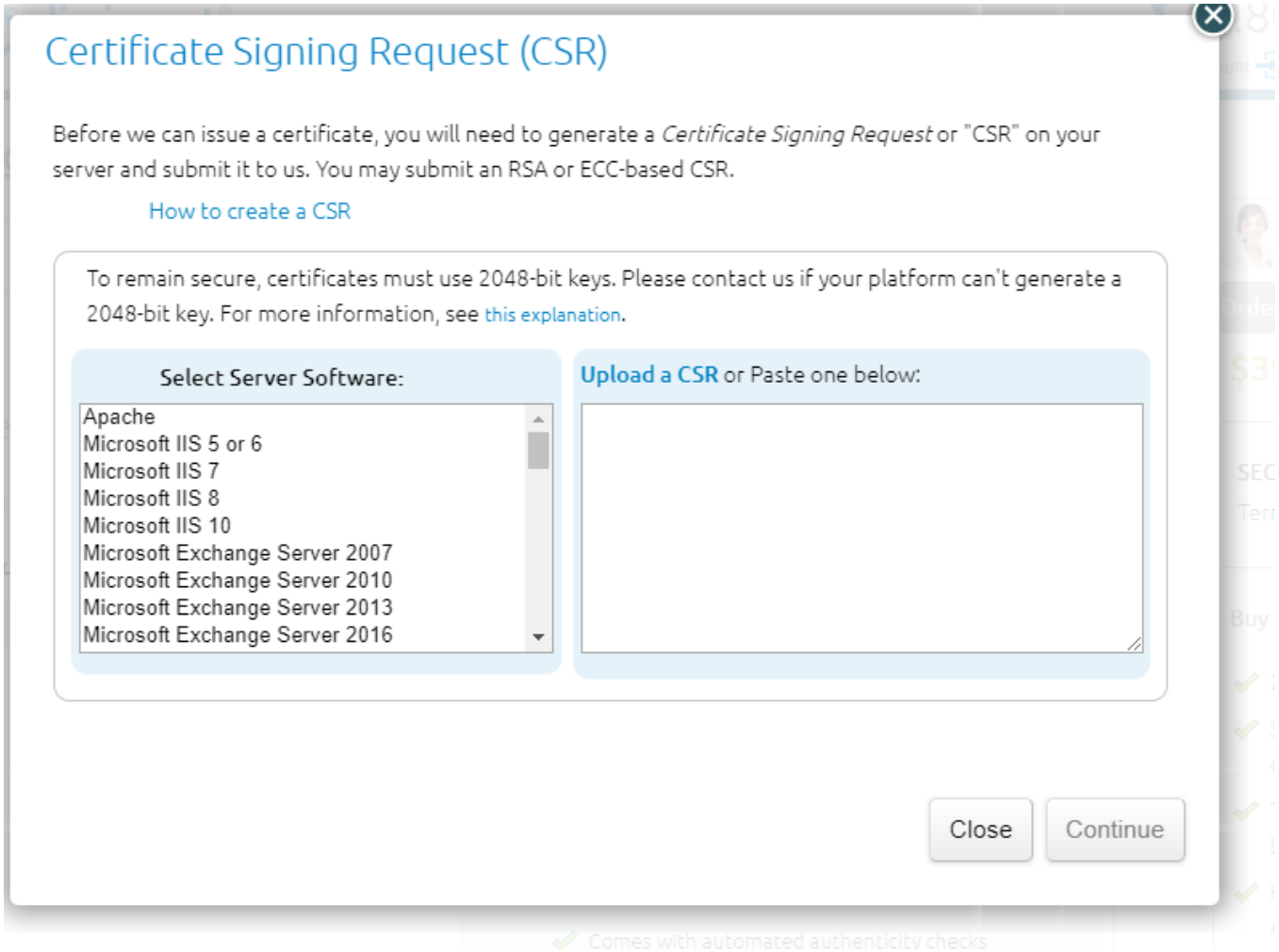
1. Create a placeholder certificate in our ESM

   *RACDCERT ID(**SHRSRV**) GENCERT SUBJECTSDN(CN('**share.org**') L('**Fort Worth**') SP('**Texas**')) NOTAFTER(DATE(**2021/02/20**)) WITHLABEL('**SHARE SRV SIGNREQ**') SIZE(**2048**)*

1. Create a Certificate Signing Request (CSR) in our ESM

   *RACDCERT ID(**SHRSRV**) GENREQ(LABEL('**SHARE SRV SIGNREQ**')) DSN('**SHRSRV.SERVER.SIGNREQ.OUT**')*

1. Provide CSR to DigiCert on the certificate signing application - FTP ASCII






BROADCOM

# z/OS Case Study - Certificate signed by DigiCert - RACF

4. Replace placeholder certificate with DigiCert signed certificate

```
RACDCERT ID(SHRSRV) ADD('SHRSRV.SERVER.SIGNEDCT.IN') TRUST
```

5. Import signing chain(trust chain)

```
RACDCERT CERTAUTH ADD('DIGICERT.ROOT001')
WITHLABEL('ROOT001') TRUST
```

# z/OS Case Study - Leverage Digital Keyrings - RACF

1. Create a RACF Keyring for tomcat server

   *RACDCERT ID(**SHRSRV**) ADDRING(**TOMRING**)*

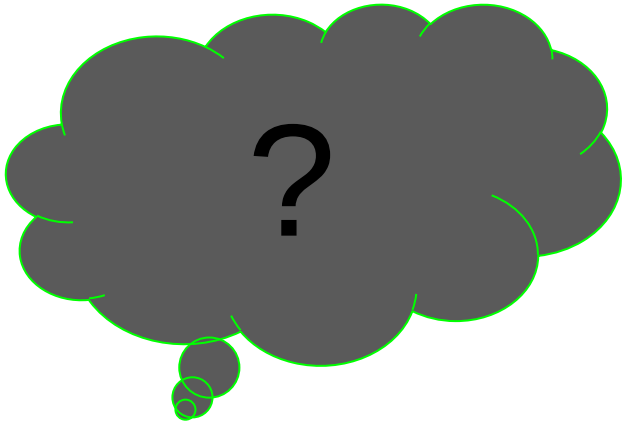
1. Connect DigiCert signed server certificate to keyring

   *RACDCERT ID(**SHRSRV**) CONNECT(ID(**SHRSRV**) LABEL('**SHARE SRV SIGNREQ**') RING(**TOMRING**) USAGE(**PERSONAL**))*

1. Connect signing chain to keyring - What about virtual keyring

   *RACDCERT ID(**SHRSRV**) CONNECT(**CERTAUTH** LABEL('**ROOT001**') RING(**TOMRING**) USAGE(**CERTAUTH**))*

# Questions



# Additional information and Resources

# ESM Digital Certificate Resources

RACF Digital Certificates doc:

https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.icha700/digcert.htm

ACF2 Digital Certificates doc:

https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/ca-mainframe-software/security/ca-acf2-for-z-os/16-0/administrating/digital-certificate-support.html

Top Secret Digital Certificates doc:

https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/ca-mainframe-software/security/ca-top-secret-for-z-os/16-0/using/digital-certificates.html

ICSF doc:

https://www.ibm.com/support/knowledgecenter/SSLTBW_2.3.0/com.ibm.zos.v2r3.csfb200/intro.htm

# z/OS Case Study - Certificate signed by DigiCert - ACF2

1. Create a placeholder certificate in our ESM

```
SET PROFILE(USER) DIV (CERTDATA)

    GENCERT SHRSRV.SERVER SUBJSDN(CN('share.org') L('Fort Worth')
    SP('Texas')) EXPIRE(2021/02/20) LABEL('SHARE SRV SIGNREQ')
    SIZE(2048)
```

1. Create a Certificate Signing Request (CSR) in our ESM

```
    GENREQ SHRSRV.SERVER DSN('SHRSRV.SERVER.SIGNREQ.OUT')
```

1. Provide CSR to DigiCert on the certificate signing application - FTP ASCII

# z/OS Case Study - Certificate signed by DigiCert - ACF2

4. Replace placeholder certificate with DigiCert signed certificate

*INSERT **SHRSRV.SERVER** DSN('SHRSRV.SERVER.SIGNEDCT.IN') LABEL('SHARE SRV SIGNREQ ') TRUST*

5. Import signing chain(trust chain)

*INSERT CERTAUTH.ROOT001 DSN('**DIGICERT.ROOT001**')*

*LABEL('ROOT001') TRUST*

# z/OS Case Study - Leverage Digital Keyrings - ACF2

1. Create a RACF Keyring for tomcat server

   *SET PROFILE(USER) DIV(KEYRING)*

   *INSERT SHRSRV RINGNAME(TOMRING)*

1. Connect DigiCert signed server certificate to keyring

   *CONNECT CERTDATA(SHRSRV.SERVER) KEYRING(SHRSRV.TOMRING) USAGE(PERSONAL)*

1. Connect signing chain to keyring - What about virtual keyring

*CONNECT CERTDATA(CERTAUTH.ROOT001) KEYRING(SHRSRV.TOMRING) USAGE(CERTAUTH)*

# z/OS Case Study - Certificate signed by DigiCert - TSS

1. Create a placeholder certificate in our ESM

   ***TSS GENCERT(SHRSRV) DIGICERT(SERVER) SUBJECTN('CN="share.org" L="Fort Worth") SP=\TX) KEYSIZE(2048) LABLCERT(SHARE SRV SIGNREQ) NADATE(02/20/21)***

1. Create a Certificate Signing Request (CSR) in our ESM

   ***TSS GENREQ(SHRSRV) DCDSN(SHRSRV.SERVER.SIGNREQ.OUT) DIGICERT(SERVER) LABLCERT('SHARE SRV SIGNREQ')***

1. Provide CSR to DigiCert on the certificate signing application - FTP ASCII

# z/OS Case Study - Certificate signed by DigiCert - TSS

4. Replace placeholder certificate with DigiCert signed certificate

```
TSS ADDTO(SHRSRV) DIGICERT(SERVER)
DCDSN(SHRSRV.SERVER.SIGNEDCT.IN) LABLCERT(SHARE SRV SIGNREQ)
```

5. Import signing chain(trust chain)

```
TSS ADDTO(CERTAUTH) DIGICERT(ROOT001)
DCDSN(DIGICERT.ROOT001) LABLCERT(ROOT001)
```

# z/OS Case Study - Leverage Digital Keyrings - TSS

1. Create a RACF Keyring for tomcat server

   *TSS ADD(**SHRSRV**) KEYRING(**TOMRING**) LABLRING(**TOMRING**)*

1. Connect DigiCert signed server certificate to keyring

   *TSS ADD(**SHRSRV**) KEYRING(**TOMRING**) RINGDATA(**SHRSRV**,**SERVER**) USAGE(**PERSONAL**)*

1. Connect signing chain to keyring - What about virtual keyring

   *TSS ADD(**SHRSRV**) KEYRING(**TOMRING**) RINGDATA(**CERTAUTH**,**ROOT001**) USAGE(**CERTAUTH**)*

# Creating and Managing A Java Key Store

- Keytool is a utility included with Java to create keys and key stores as well as manage certificates.
- Located in '$JAVA_HOME/bin'

- Create a key and keystore
  - If the keystore already exists, it will simply add the newly created key
  - You may be prompted for extra information such as your organization's name, division, and location.

```
keytool -genkeypair \
        -alias domain \
        -keyalg RSA \
        -keystore keystore.jks
```

# Creating and Managing A Java Key Store

- Creating a CSR
  - This command creates a CSR (`domain.csr`) signed by the private key identified by the alias (`domain`) in the (`keystore.jks`) keystore

```
keytool -certreq \
        -alias domain \
        -file domain.csr \
        -keystore keystore.jks
```

# Creating and Managing A Java Key Store

- Importing Trust Certificates
  - This command imports the certificate (`domain.crt`) into the keystore (`keystore.jks`), under the specified alias (`domain`). If you are importing a signed certificate, it must correspond to the private key in the specified alias:

```
keytool -importcert \
        -trustcacerts -file domain.crt \
        -alias domain \
        -keystore keystore.jks
```

  - Note: Your Java Trust Store is usually located at '`$JAVA_HOME/jre/lib/security/cacerts`'

# Creating and Managing A Java Key Store

- Export a Certificate
  - This command exports a binary DER-encoded certificate (`domain.der`), that is associated with the alias (`domain`), in the keystore (`keystore.jks`):

```
keytool -exportcert
        -alias domain
        -file domain.der
        -keystore keystore.jks
```

# Creating and Managing A Java Key Store

- List Keystore Contents
  - This command lists verbose information about the entries a keystore (keystore.jks) contains, including certificate chain length, fingerprint of certificates in the chain, distinguished names, serial number, and creation/expiration date, under their respective aliases:

```
keytool -list -v \
        -keystore keystore.jks
```

# Other Ways to Create Key Pairs

- OpenSSL
  - Unix/Linux systems
  - USS on z/OS

    Generate Key Pair: `openssl genrsa -des3 -out private.pem 2048`

- PuTTY
  - Windows

- gskkyman
  - IBM z/OS

# File Formats

PEM Format

- It is the most common format used for certificates
- Most servers (Ex: Apache) expects the certificates and private key to be in a separate files
- Usually they are Base64 encoded ASCII files
- Extensions used for PEM certificates are .cer, .crt, .pem, .key files
- **Apache** and similar server uses PEM format certificates

# File Formats

DER Format

- The DER format is the binary form of the certificate
- All types of certificates & private keys can be encoded in DER format
- DER formatted certificates do not contain the "BEGIN CERTIFICATE/END CERTIFICATE" statements
- DER formatted certificates most often use the ‘.cer’ and '.der' extensions
- DER is typically used in **Java Platforms**

# File Formats

P7B/PKCS#7 Format

- The PKCS#7 or P7B format is stored in Base64 ASCII format and has a file extension of .p7b or .p7c
- A P7B file only contains certificates and chain certificates (Intermediate CAs), not the private key
- The most common platforms that support P7B files are **Microsoft Windows** and **Java Tomcat**

# File Formats

PFX/P12/PKCS#12 Format

- The PKCS#12 or PFX/P12 format is a binary format for storing the server certificate, intermediate certificates, and the private key in one encryptable file
- These files usually have extensions such as .pfx and .p12
- They are typically used on **Windows machines** to import and export certificates and private keys