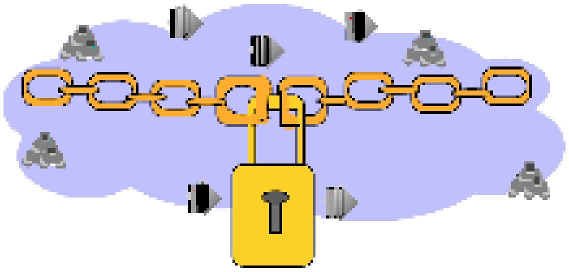
Quick Start to Implementing X.509 Certificates on z/OS Communications Server

RACF Users Group Meeting
November 10, 2009

Speaker: Linda Harrison,
IBM Advanced Technical Support,
Gaithersburg, MD



Presentation by: Gwen Dente, IBM Advanced Technical Support, Gaithersburg, MD

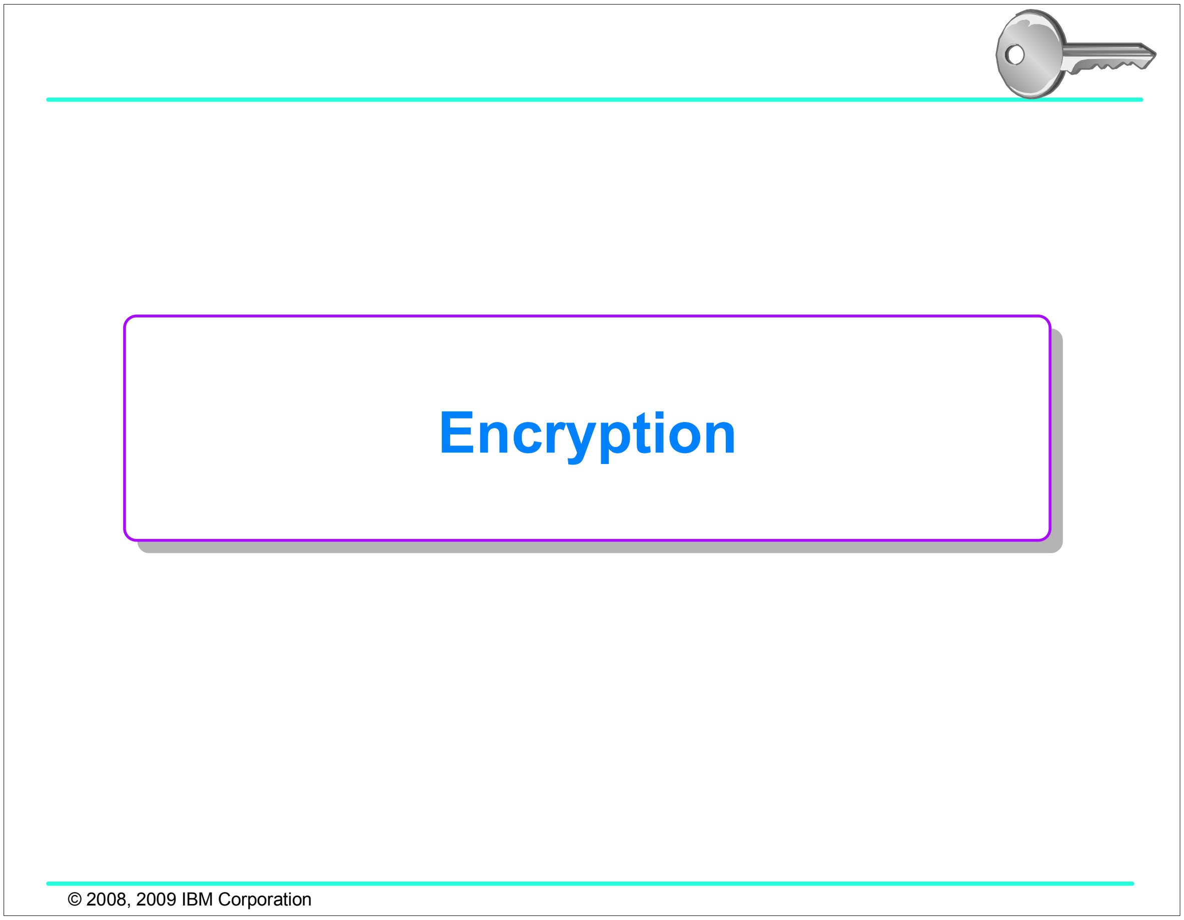
Abstract and Acknowledgments

- You know the theory of certificates and you have seen the sessions on setting up SSL/TLS or IPsec with certificates. But you don't know how to design this certificate and keying implementation for your environment.
- Sure, you have been to all the sessions on configuring certificate requests and certificates . You have been to the sessions on providing security with x.509 certificates. You have been to the sessions on Public Key Infrastructure (PKI). But you still don't know how to design the certificates for use with practical applications.
- When do you use SITE certificates? When do you use a user or personal certificate? When do you use a CA certificate? What exactly do you put on a keyring? Which certificates belong there?
- This session shows you how to design a certificate and keyring environment, focusing on the requirements for implementation on z/OS Communications Server applications.
- Although many of the materials in this section are original, other materials were derived from work produced by the following people, whom we wish to acknowledge:

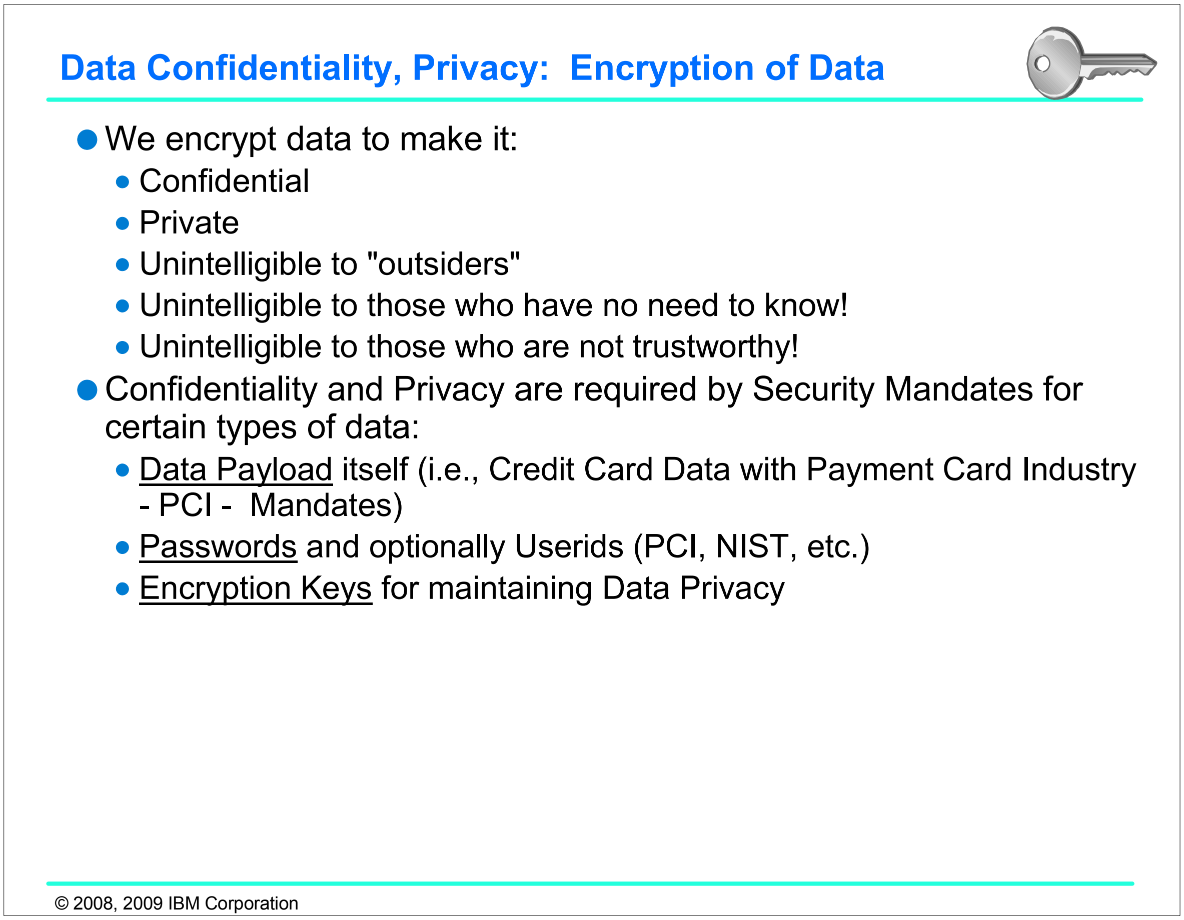
  - Wai Choi, IBM
    - zOS RACF Development
  - Alyson Comer, IBM
    - z/OS System SSL Development
  - Erin Farr, IBM
    - z/OS OpenSSH Development
  - Vicente Ranieri, IBM
    - Advanced Technical Support, System z Security
  - Christopher Meyer, IBM
    - z/OS Communications Server Development

Agenda

- **Encryption**
- **Keys**
- **X.509 Certificate**
- **Keyring / Key Database**
- **RACF Commands**
- **Certificate and Key Management**
- **OpenSSH Security**
- **Appendices:**
  - Protocol Comparisons (IPsec, SSL, SSH)
  - Advanced Certificate Concepts
- **References**

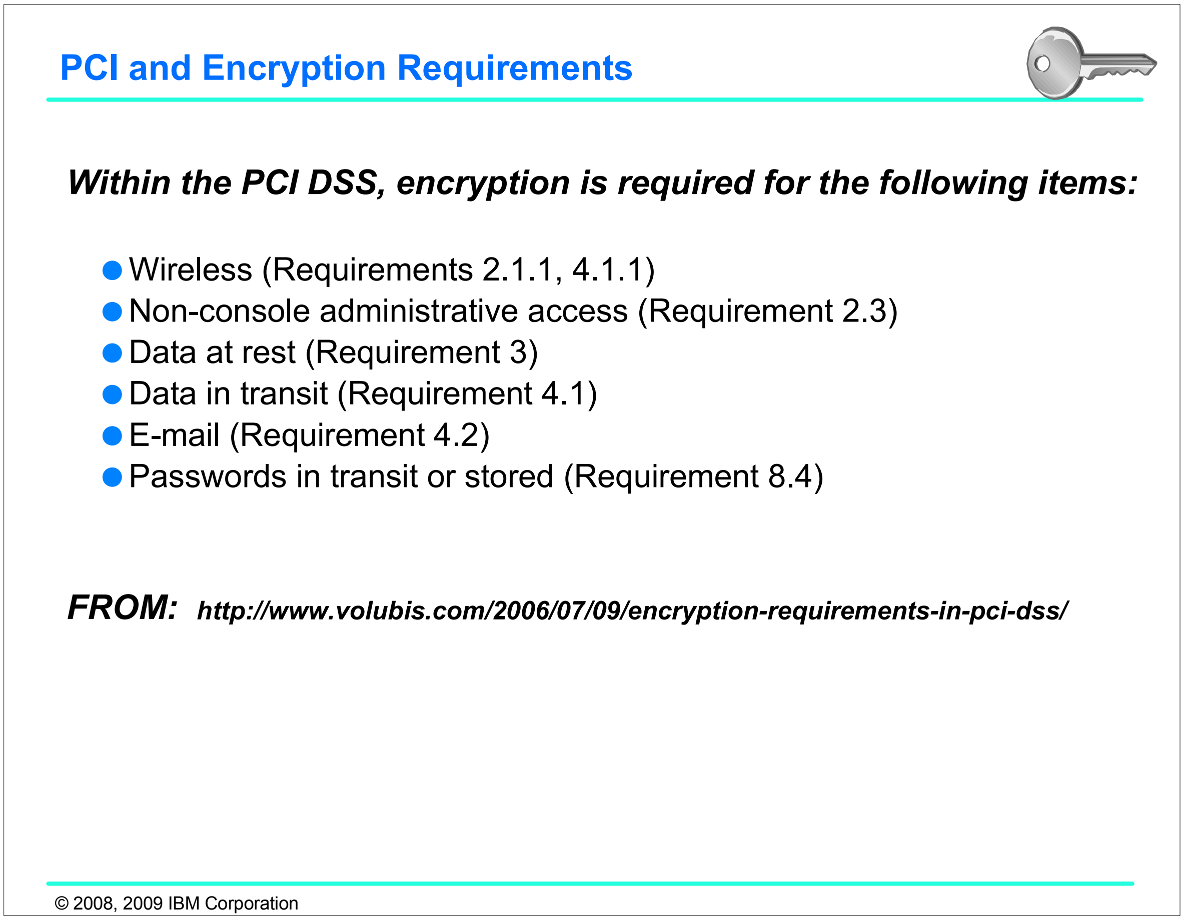


Encryption



Data Confidentiality, Privacy: Encryption of Data

- We encrypt data to make it:
  - Confidential
  - Private
  - Unintelligible to "outsiders"
  - Unintelligible to those who have no need to know!
  - Unintelligible to those who are not trustworthy!
- Confidentiality and Privacy are required by Security Mandates for certain types of data:
  - Data Payload itself (i.e., Credit Card Data with Payment Card Industry - PCI - Mandates)
  - Passwords and optionally Userids (PCI, NIST, etc.)
  - Encryption Keys for maintaining Data Privacy



PCI and Encryption Requirements

Within the PCI DSS, encryption is required for the following items:

- Wireless (Requirements 2.1.1, 4.1.1)
- Non-console administrative access (Requirement 2.3)
- Data at rest (Requirement 3)
- Data in transit (Requirement 4.1)
- E-mail (Requirement 4.2)
- Passwords in transit or stored (Requirement 8.4)

FROM: ***http://www.volubis.com/2006/07/09/encryption-requirements-in-pci-dss/***

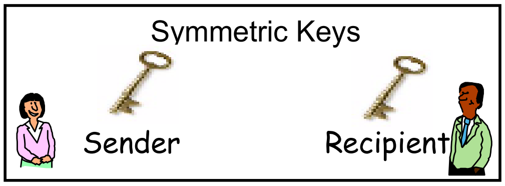
- WIRELESS:
  - Requirement 2.1.1: "For wireless environments connected to the cardholder data environment or transmitting cardholder data, change wireless vendor defaults, including but not limited to default wireless encryption keys, passwords, and SNMP community strings. Ensure wireless device security settings are enabled for strong encryption technology for authentication and transmission."
  - Requirement 4.1.1: "4.1.1 Ensure wireless networks transmitting cardholder data or connected to the cardholder data environment, use industry best practices (for example, IEEE 802.11i) to implement strong encryption for authentication and transmission.
    - For new wireless implementations, it is prohibited to implement WEP after March 31, 2009.
    - For current wireless implementations, it is prohibited to use WEP after June 30, 2010."
  - Requirement 2.3: "2.3 Encrypt all non-console administrative access. Use technologies such as SSH, VPN, or SSL/TLS for webbased management and other non-console administrative access."
  - Requirement 3: "Protect stored cardholder data. Protection methods such as encryption, truncation, masking, and hashing are critical components of cardholder data protection. If an intruder circumvents other network security controls and gains access to encrypted data, without the proper cryptographic keys, the data is unreadable and unusable to that person. Other effective methods of protecting stored data should be considered as potential risk mitigation opportunities. For example, methods for minimizing risk include not storing cardholder data unless absolutely necessary, truncating cardholder data if full PAN is not needed, and not sending PAN in unencrypted e-mails."
  - Requirement 4.1: "4.1 Use strong cryptography and security protocols such as SSL/TLS or IPSEC to safeguard sensitive cardholder data during transmission over open, public networks. Examples of open, public networks that are in scope of the PCI DSS are:
    - .. The Internet,
    - .. Wireless technologies,
    - .. Global System for Mobile communications (GSM), and
    - .. General Packet Radio Service (GPRS)."
  - Requirement 4.2: "Never send unencrypted PANs by end-user messaging technologies (for example, e-mail, instant messaging, chat)."
  - Requirement 8.4: "8.4 Render all passwords unreadable during transmission and storage on all system components using
  - strong cryptography (defined in PCI DSS Glossary of Terms, Abbreviations, and Acronyms)."

Keys

Focus on Encryption

- **Encryption Algorithm**
  - A set of mathematical instructions (a procedure) for encoding data to make the data unrecognizable.
  - The data encryption algorithms themselves are publicly known. Therefore, the algorithms by themselves are not secure!
  - But the algorithms use "keys" during the cryptographic execution and those keys do not all have to be publicly known!
  - The keys and the algorithm together are what make the encrypted output a secret!
  - To unlock or decode or "decrypt" the encoded data, you need to know the key or keys that were used to encrypt the data to begin with!
- **Keys are Essential to Privacy (Encryption Algorithms)**
  - Symmetric Keys
  - Asymmetric Keys
- **Keys are Essential to**
  - Authentication and
  - Non-Repudiation
- **Hashes are Essential to**
  - Data Integrity Verification
  - Data Privacy

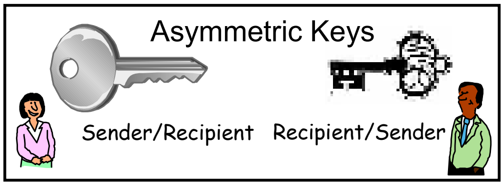
Encryption Algorithm



Symmetric Keys

Sender

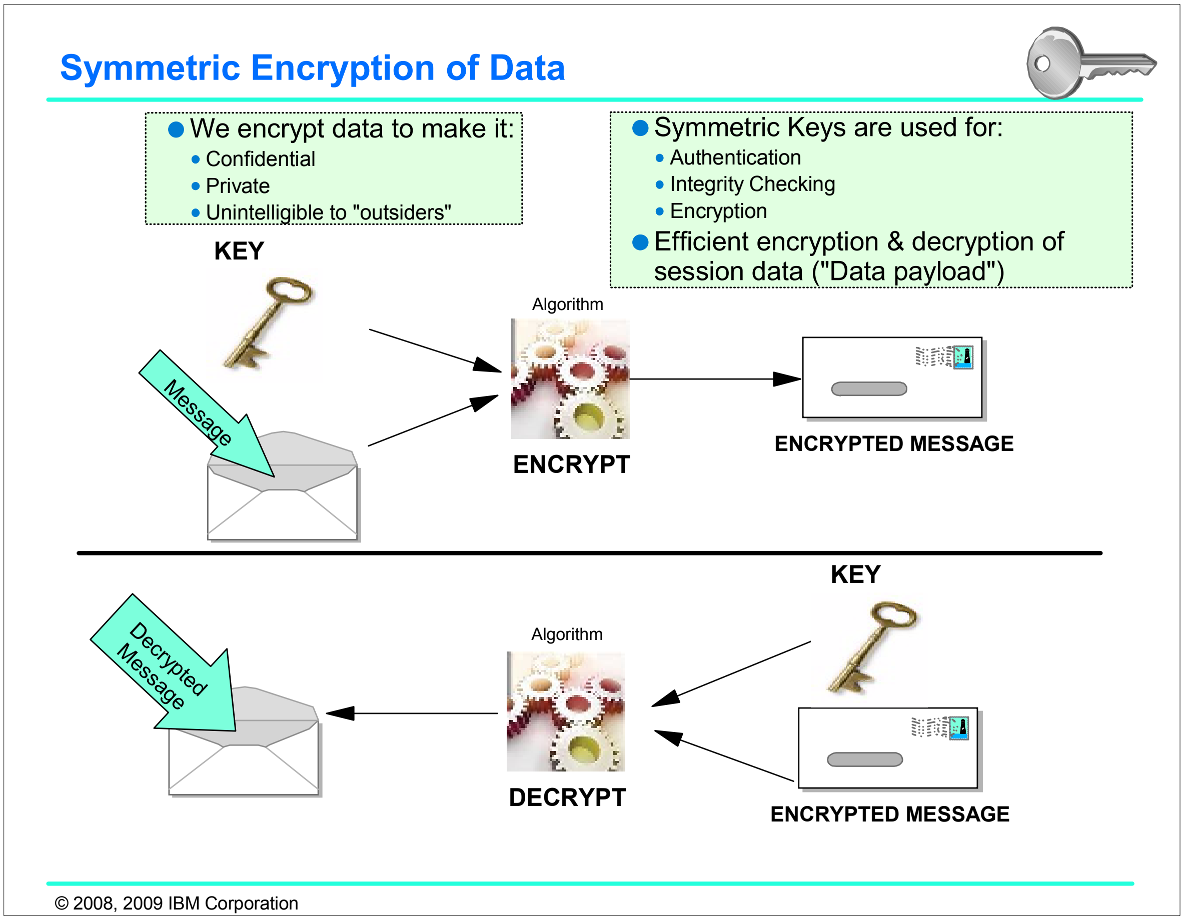
Recipient



Asymmetric Keys

Sender/Recipient

Recipient/Sender

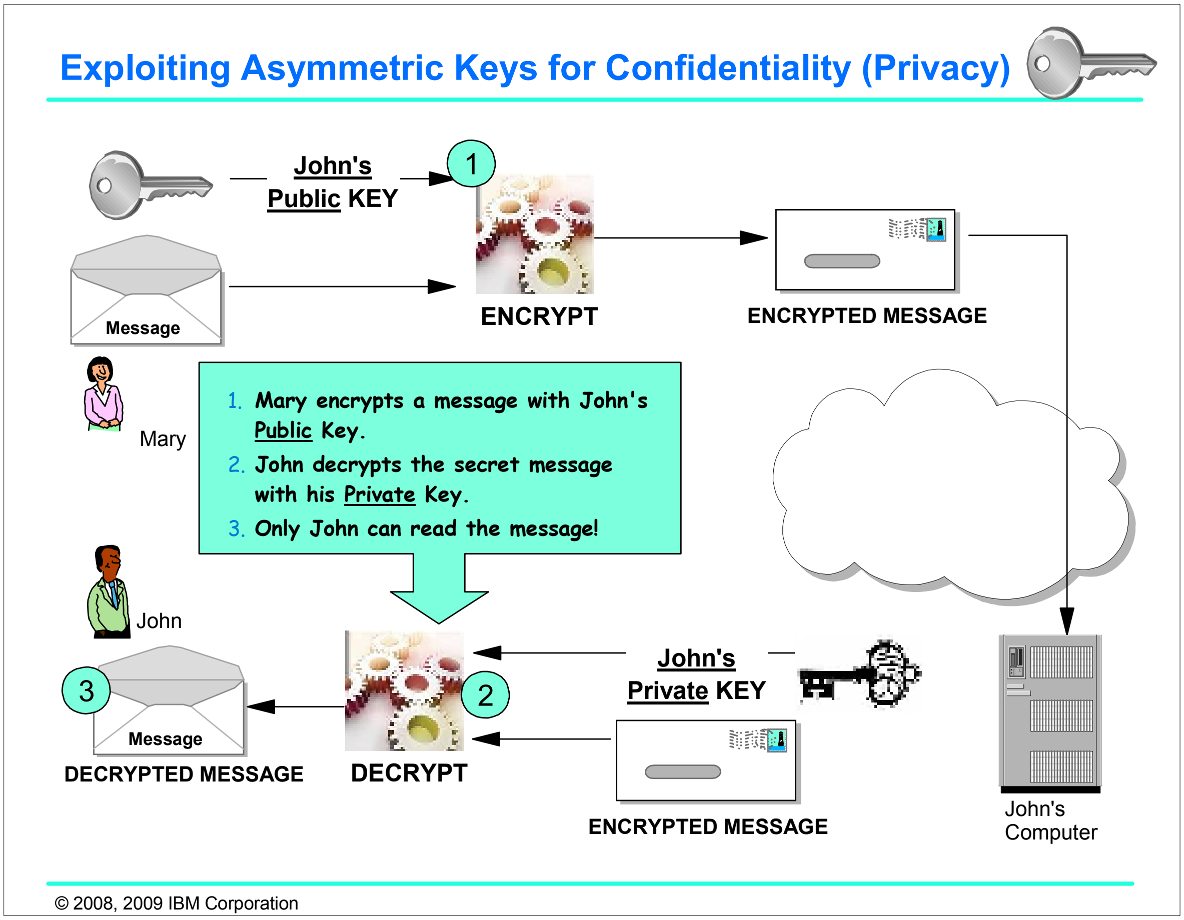


Symmetric Encryption of Data

- We encrypt data to make it:
  - Confidential
  - Private
  - Unintelligible to "outsiders"

- Symmetric Keys are used for:
  - Authentication
  - Integrity Checking
  - Encryption
- Efficient encryption & decryption of session data ("Data payload")

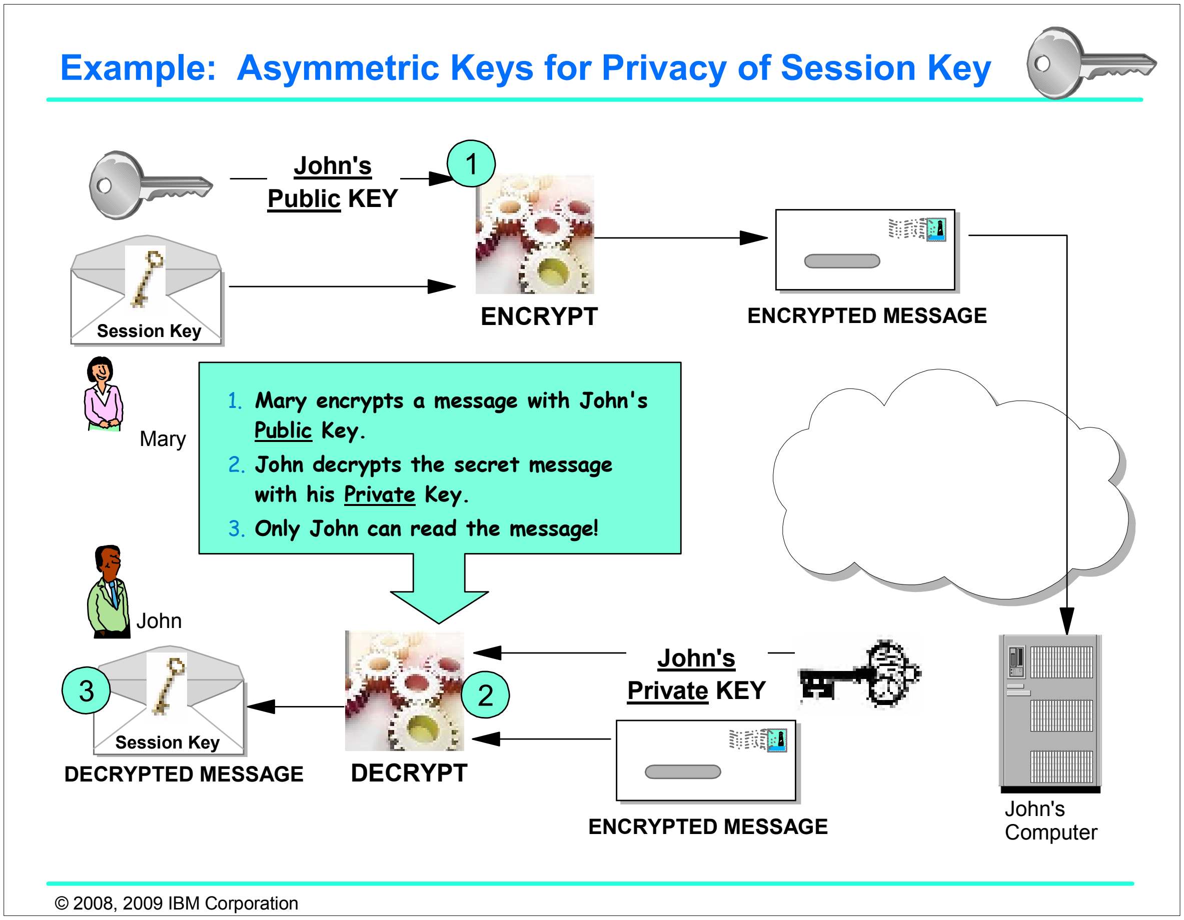
- Symmetric Keys are Defined keys; with symmetric keys the SAME KEY resides at both ends of the communication connection.
- They Provide data encryption capabilities
- The encryption Algorithm defines the strength of the encryption – DES, Triple DES, AES etc
- Symmetric encryption is typically more efficient -- i.e., faster and less CPU-intensive -- than asymmetric encryption. This is why they are used for the encryption and descryption of the payload data ... that is, the actual session data. Therefore we find that the keys used for session traffic, i.e., the Session Keys, are symmetric keys. "Efficiency" is a relative term when it comes to encryption. Encryption always carries a price in terms of response time, CPU, and throughput. But Symmetric encryption is more efficient than Asymmetric encryption, which can be 1000x more compute-intensive than symmetric.
- Remember that symmetric session keys must be transmitted in a confidential fashion to minimize their chance of compromise. The security technologies we use to provide data security tend to employ Asymmetric encryption algorithms in order to transmit the symmetric key in a confidential fashion.



Exploiting Asymmetric Keys for Confidentiality (Privacy)

1. Mary encrypts a message with John's Public Key.
2. John decrypts the secret message with his Private Key.
3. Only John can read the message!

- The private and public keys are mathematically related. The private key, owned by one entity (John in this case), can "unlock" the secret that was encrypted by John's Public key that is available to anyone.
- Note that the Public key can be made available to the world if necessary. In our example, Mary is using John's Publicly available key to send him an encrypted message that only his private key can decrypt. But the Private key is in the possession of only ONE entity (John in this case).



Example: Asymmetric Keys for Privacy of Session Key

John's Public KEY

1

Session Key

ENCRYPT

ENCRYPTED MESSAGE

Mary

1. Mary encrypts a message with John's Public Key.
2. John decrypts the secret message with his Private Key.
3. Only John can read the message!

John

3

Session Key

DECRYPTED MESSAGE

DECRYPT

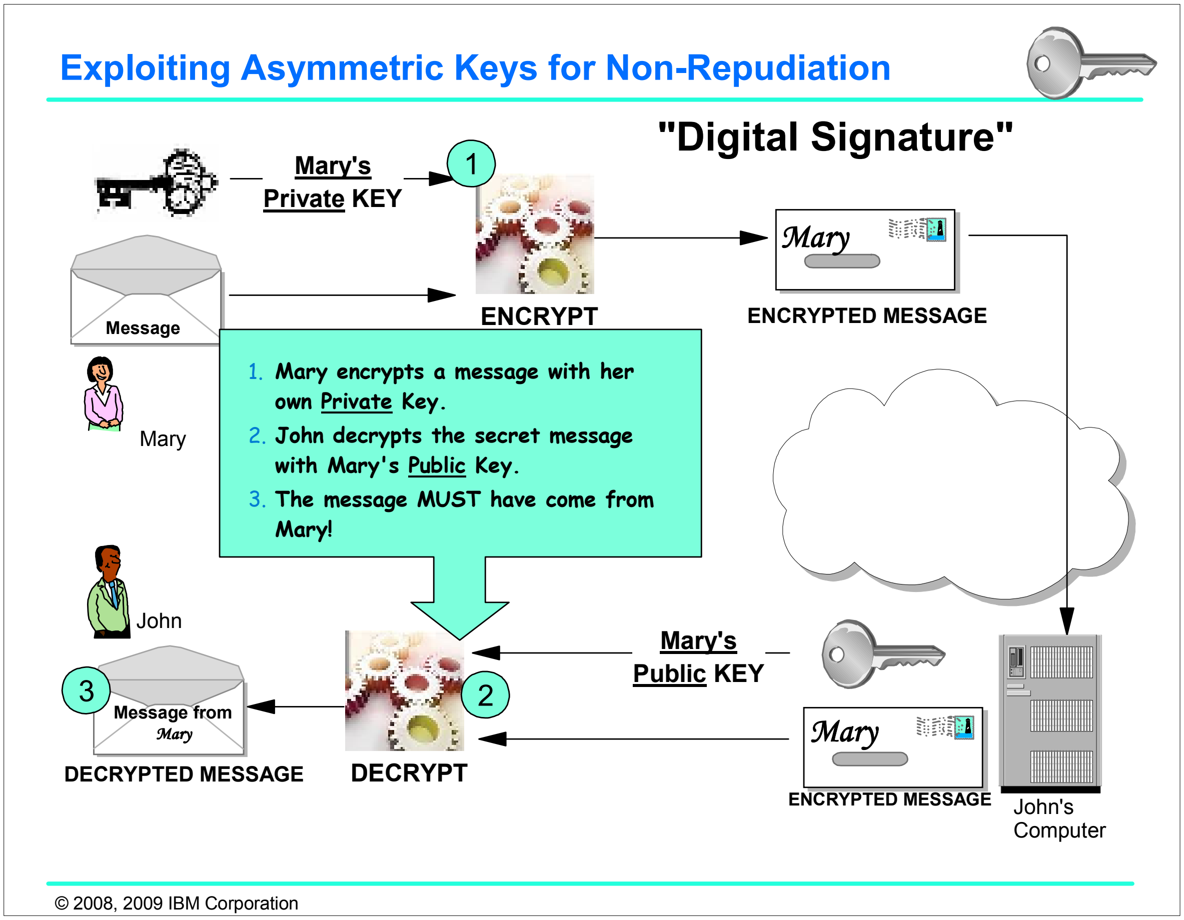
2

John's Private KEY

ENCRYPTED MESSAGE

John's Computer

- In this example you see how Asymmetric Public/Private Key Pair are used to maintain the confidentiality of the symmetric Session Key. Once the partners have authenticated, a Session Key is generated using a mathematical algorithm. Then that Symmetric Key is sent to the partner; but it is sent encrypted using a Public/Private key algorithm so that it remains confidential in transit.
- For SSL/TLS or AT-TLS, the authentication need not be bidirectional. That is, Server authentication is required. However, Client Authentication is optional. This is all configurable.
- For IPSec, authentication must be bidirectional: both peers need to authentication to each other.
- For OpenSSH, authentication can be bidirectional or unidirectional only. This is all configurable.

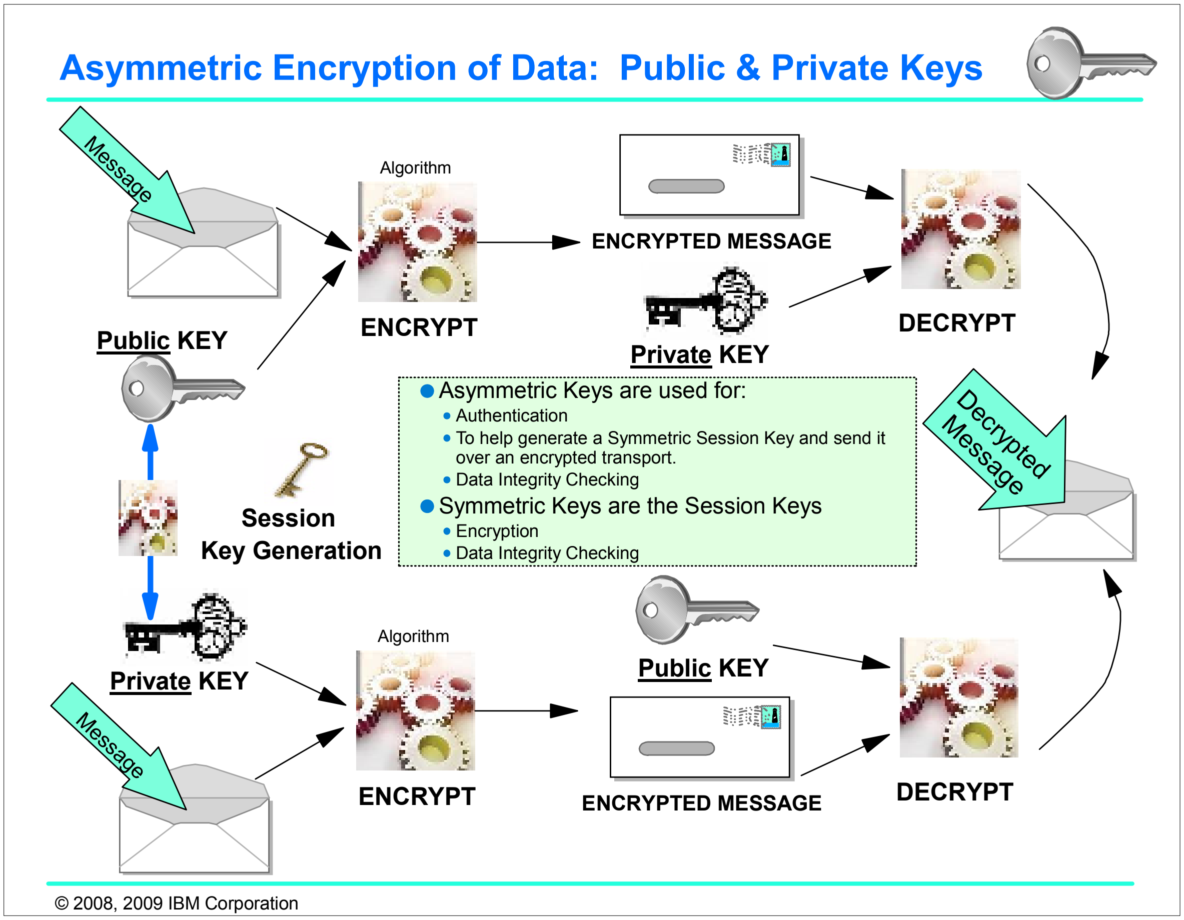


Exploiting Asymmetric Keys for Non-Repudiation

"Digital Signature"

1. Mary encrypts a message with her own Private Key.
2. John decrypts the secret message with Mary's Public Key.
3. The message MUST have come from Mary!

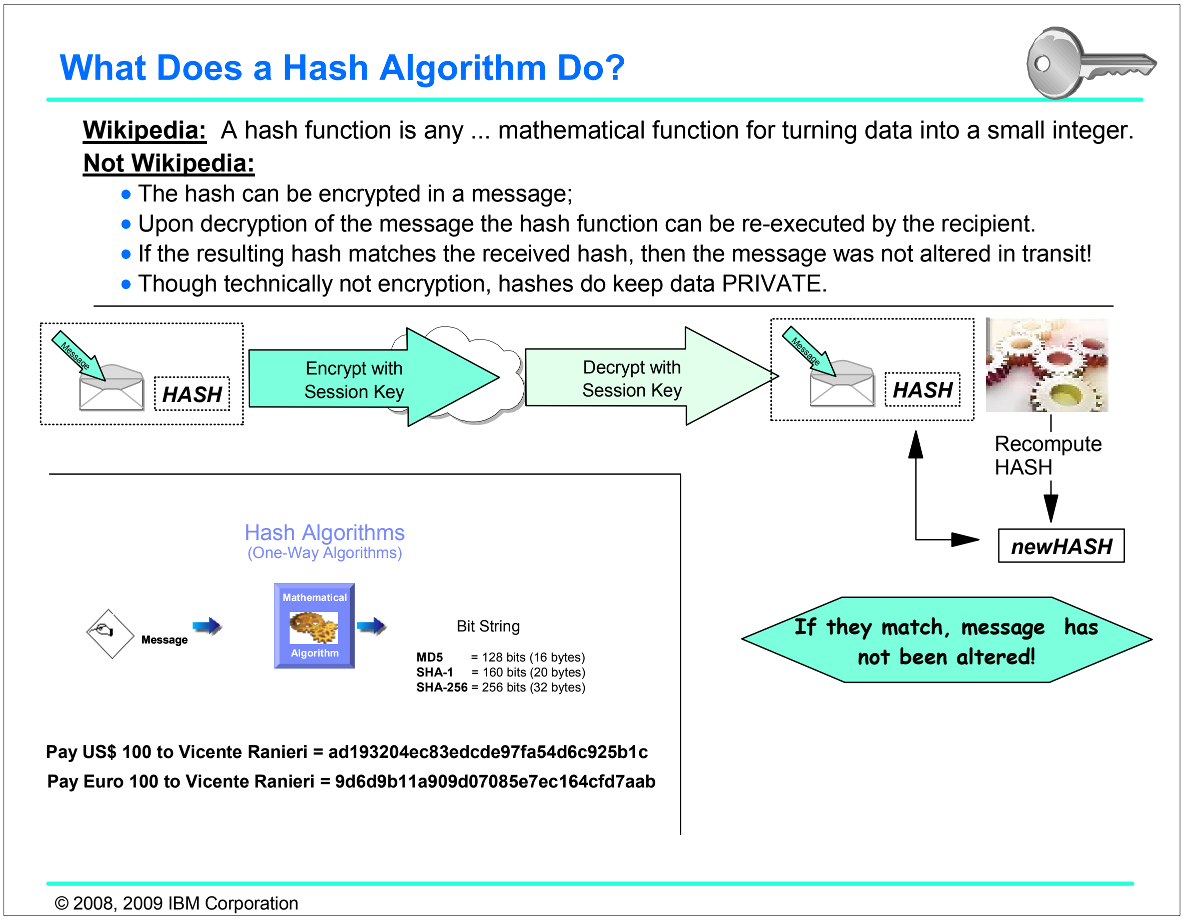
- The private and public keys are mathematically related. The private key, owned by one entity (Mary in this case), can "unlock" the secret that was encrypted by the public key that is available to anyone. If Mary's private key can unlock the secret, then Mary must have been the sender and she cannot refute or repudiate it -- that is, unless someone has STOLEN her private key. This fact explains why KEY VALUES MUST BE PROTECTED!
- Note that the Public key can be made available to the world if necessary.
- In our example of Non-Repudiation, we have used the Digital Signature feature, whereby we can prove that Mary sent the message. Only Mary's Public key can unlock the secret that was sent, thereby proving that Mary was the sender.
- Digital Signature: If it bears your signature, it came from you!
  - Certificate Digital Signature - Signature generated using the issuer’s private key .



Asymmetric Encryption of Data: Public & Private Keys

- Asymmetric Keys are used for:
  - Authentication
  - To help generate a Symmetric Session Key and send it over an encrypted transport.
  - Data Integrity Checking
- Symmetric Keys are the Session Keys
  - Encryption
  - Data Integrity Checking

- Asymmetric Keys
  - Public/private key pairs
  - A public key and a related private key are numerically associated with each other.
  - Data encrypted/signed using one of the keys may only be decrypted/verified using the other key.
  - Strength of the encryption or signature is defined by the size of the keys
    - e.g. RSA – 1024, 2048, 4096-bit keys
  - Public key is intended to be given freely
  - Private key needs to be treated very securely and not distributed. The private key can also be stored in Cryptographic Hardware.
- Asymmetric encryption is typically less efficient -- i.e., slower and more CPU-intensive -- than symmetric encryption.



What Does a Hash Algorithm Do?

Wikipedia: A hash function is any ... mathematical function for turning data into a small integer.

Not Wikipedia:

- The hash can be encrypted in a message;
- Upon decryption of the message the hash function can be re-executed by the recipient.
- If the resulting hash matches the received hash, then the message was not altered in transit!
- Though technically not encryption, hashes do keep data PRIVATE.

Message

HASH

Encrypt with Session Key

Decrypt with Session Key

Message

HASH

Recompute HASH

newHASH

If they match, message has not been altered!

Hash Algorithms
(One-Way Algorithms)

Message

Mathematical Algorithm

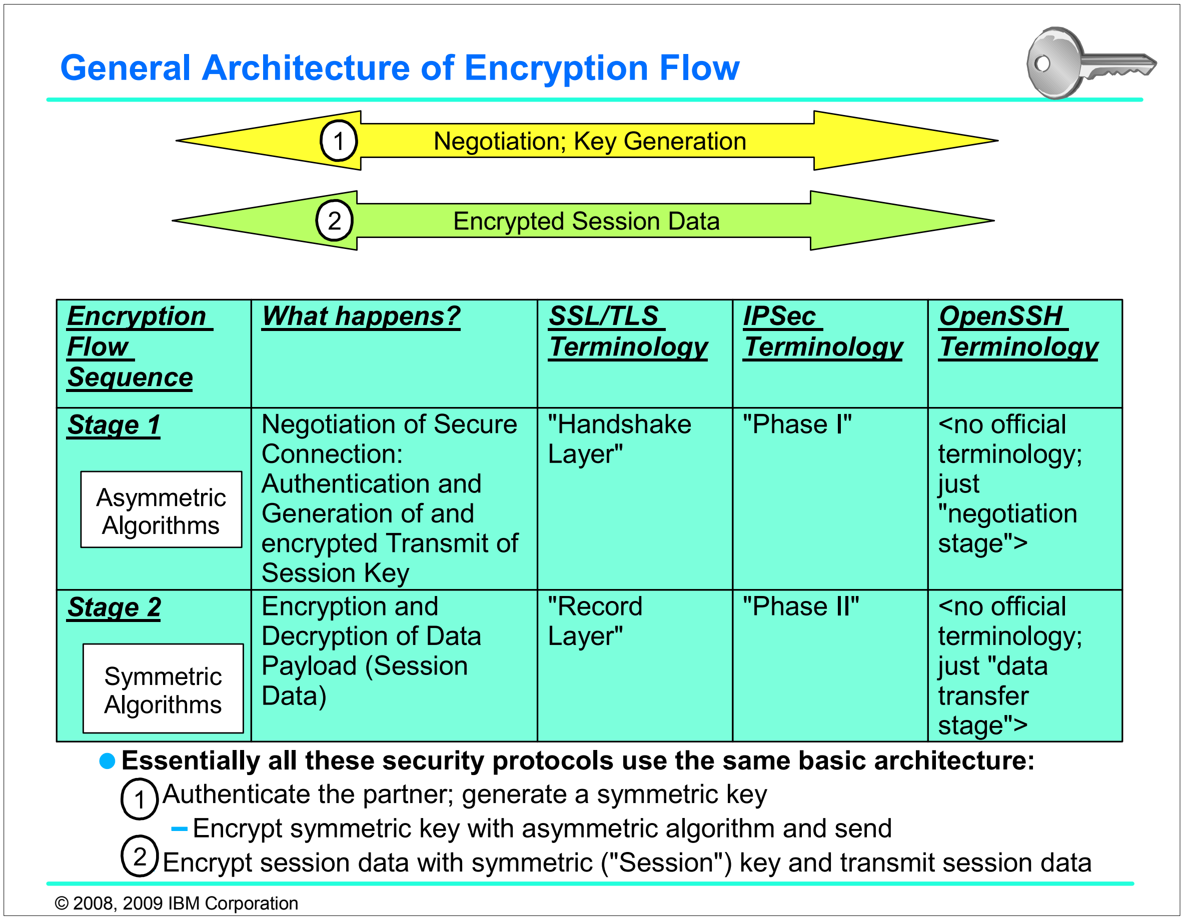
Bit String

MD5	= 128 bits (16 bytes)
SHA-1	= 160 bits (20 bytes)
SHA-256	= 256 bits (32 bytes)

Pay US$ 100 to Vicente Ranieri = ad193204ec83edcde97fa54d6c925b1c

Pay Euro 100 to Vicente Ranieri = 9d6d9b11a909d07085e7ec164cfd7aab

- When the hash value is used for providing data integrity for the data phase of a flow, the hash value is protected by the session (symmetric) key.
- In the negotiation phase of a secure connection establishment, the hash is generated against the session key to prove that it has not been altered in flight. When the hash value is used in certificates, it also provides data integrity for a subset of the certificate and becomes the signature in the certificate.
- A hash is a way of representing the original content in a compressed and unique form. Since it is one-way, it cannot be reversed to reveal the original value or form. But if the original value or form is hashed again, the new hash should match the hash that was received, thus proving that the original data was not changed.



General Architecture of Encryption Flow

1 Negotiation; Key Generation

2 Encrypted Session Data

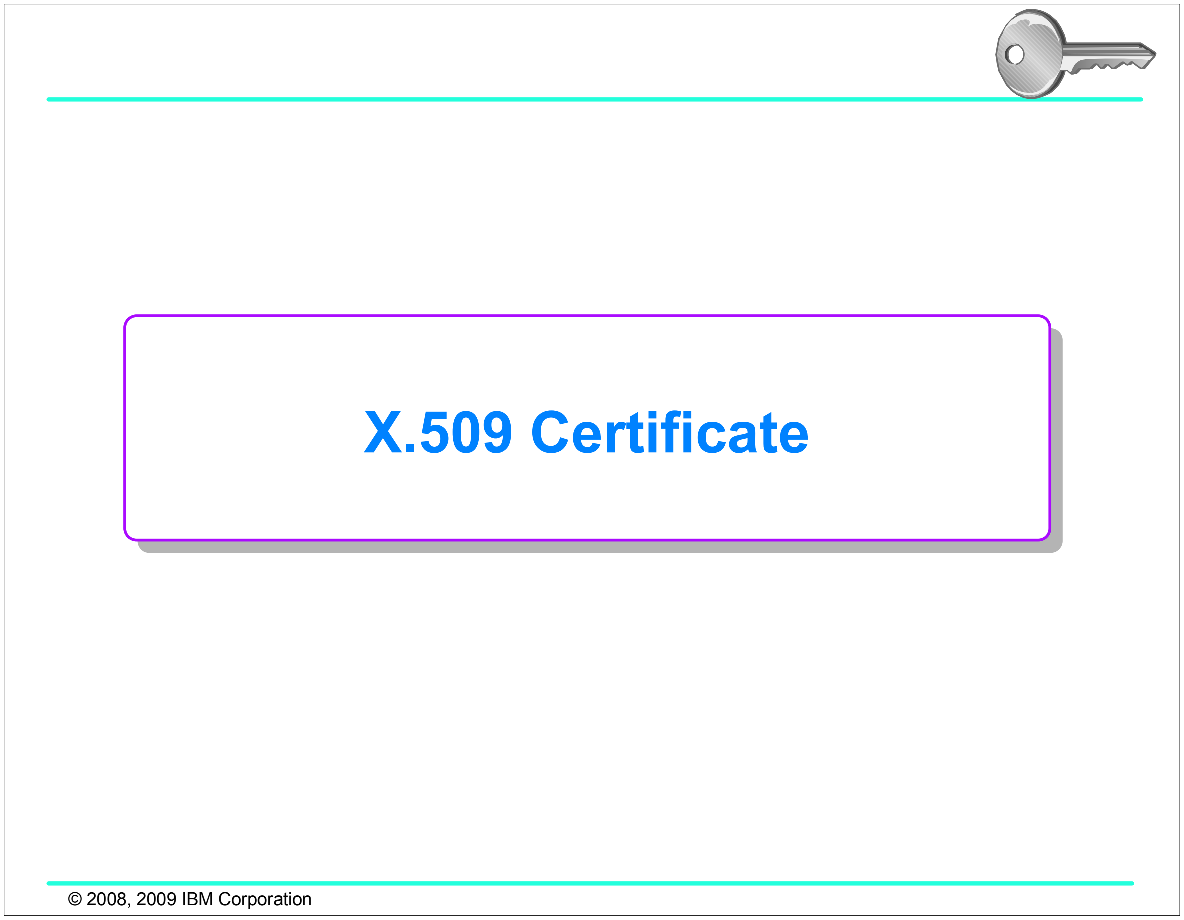
Encryption Flow Sequence	***What happens?***	***SSL/TLS Terminology***	***IPSec Terminology***	***OpenSSH Terminology***
Stage 1 Asymmetric Algorithms	Negotiation of Secure Connection: Authentication and Generation of and encrypted Transmit of Session Key	"Handshake Layer"	"Phase I"	<no official terminology; just "negotiation stage">
Stage 2 Symmetric Algorithms	Encryption and Decryption of Data Payload (Session Data)	"Record Layer"	"Phase II"	<no official terminology; just "data transfer stage">

- **Essentially all these security protocols use the same basic architecture:**
  1. Authenticate the partner; generate a symmetric key
     - Encrypt symmetric key with asymmetric algorithm and send
  2. Encrypt session data with symmetric ("Session") key and transmit session data

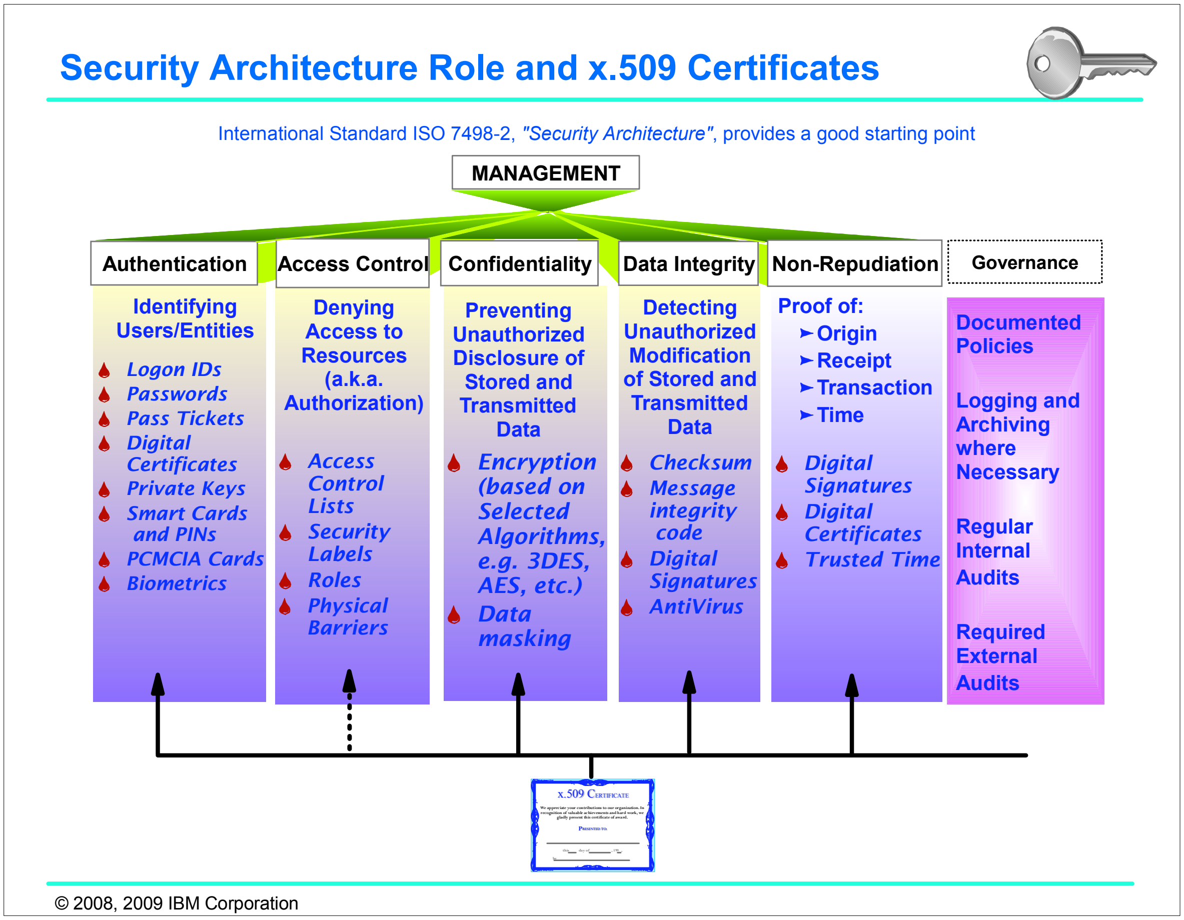
- From the z.OS Security Server RACF Security Administrator's Guide (SA22-7683-11), Chapter 21 "RACF and Digital Certificates"
- "Each party, both client and server, has its own certificate, a matching private key, and a list of trusted certificate-authority certificates. When the client needs to authenticate itself to the server to be able to perform a transaction, both the server and client need to verify one another. The protocol for a secure handshake for mutual verification begins with the parties exchanging certificates. Each party then separately validates the other’s certificate to make sure that its signature is valid, that the subject name in the certificate is correct, and that the certificate originated from a trusted certificate authority. If successful, each party must prove to the other that it owns the private key that matches its public key certificate. This step establishes proof of possession and can be accomplished by having each party sign a known unique value, such as a hash of the message traffic between the two parties. If each signature can be validated using the associated public key, the proofs are successful. The final step in this handshake is for one of the parties to generate a random symmetric key, encrypt it using the other party’s public key, and send it to the other party. This random symmetric key may then be used to encrypt the data for the remainder of the session. Once the secure handshake is complete, secure transactions can be safely handled in the z/OS environment between this client and server."

Digital Keys Are Integral to Data Security

- **Key Functions:**
  - Authentication and Non-Repudiation
  - Encryption
  - Data Integrity Verification through Hashing (not covered in this topic)
- **Key Types:**
  - Symmetric; Asymmetric
- **Common Networking Protocols that Use Keys**
  - SSL/TLS (or AT-TLS)
  - IPSec (Virtual Private Networks or "VPN")
  - Secured Shell ("SSH")
- **Where Can Keys Be Stored? How Are They Distributed?**
  - In a Configuration File (Definition File)
    - OSPF in OMPRoute (Dynamic Routing Protocol)
      - OSPF uses a "password" that is used to build an MD5 Hashing value -- not really a key, but used for authentication of partner.
    - Manual Tunnels for VPNs (Symmetric Keys)
    - Dynamic Tunnel with Pre-Shared Keys (Symmetric Keys)
  - In a Generated Key File and "known_hosts file"
    - SSH (Uses Asymmetric, Public/Private Keys to generate Session Key)
  - Bound to an x.509 Certificate: Public Asymmetric Key
    - Public / Private Key Pair work together to generate a Session Key
    - With x.509 Certificates, the implementation is called Public Key Infrastructure (PKI)
  - In a Keyring or Key Database: Private Asymmetric Key
  - In Cryptographic Hardware (Master Key)
    - Not recommended to store Private key in hardware, but rather in PKDS dataset for ICSF.



X.509 Certificate



Security Architecture Role and x.509 Certificates

International Standard ISO 7498-2, *"Security Architecture"*, provides a good starting point

MANAGEMENT

Authentication

Identifying Users/Entities

- *Logon IDs*
- *Passwords*
- *Pass Tickets*
- *Digital Certificates*
- *Private Keys*
- *Smart Cards and PINs*
- *PCMCIA Cards*
- *Biometrics*

Access Control

Denying Access to Resources (a.k.a. Authorization)

- *Access Control Lists*
- *Security Labels*
- *Roles*
- *Physical Barriers*

Confidentiality

Preventing Unauthorized Disclosure of Stored and Transmitted Data

- *Encryption (based on Selected Algorithms, e.g. 3DES, AES, etc.)*
- *Data masking*

Data Integrity

Detecting Unauthorized Modification of Stored and Transmitted Data

- *Checksum*
- *Message integrity code*
- *Digital Signatures*
- *AntiVirus*

Non-Repudiation

Proof of:

- ►Origin
- ►Receipt
- ►Transaction
- ►Time

- *Digital Signatures*
- *Digital Certificates*
- *Trusted Time*

Governance

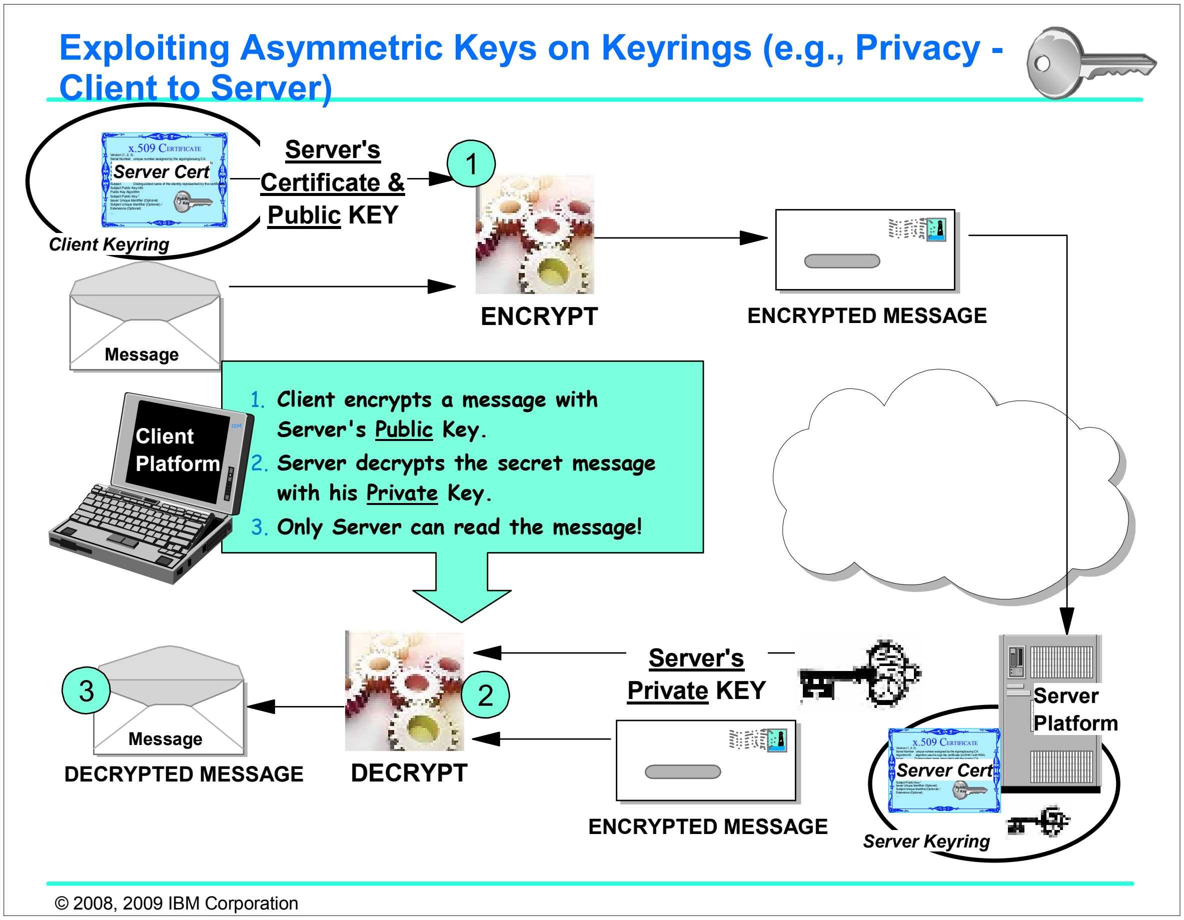
Documented Policies

Logging and Archiving where Necessary

Regular Internal Audits

Required External Audits

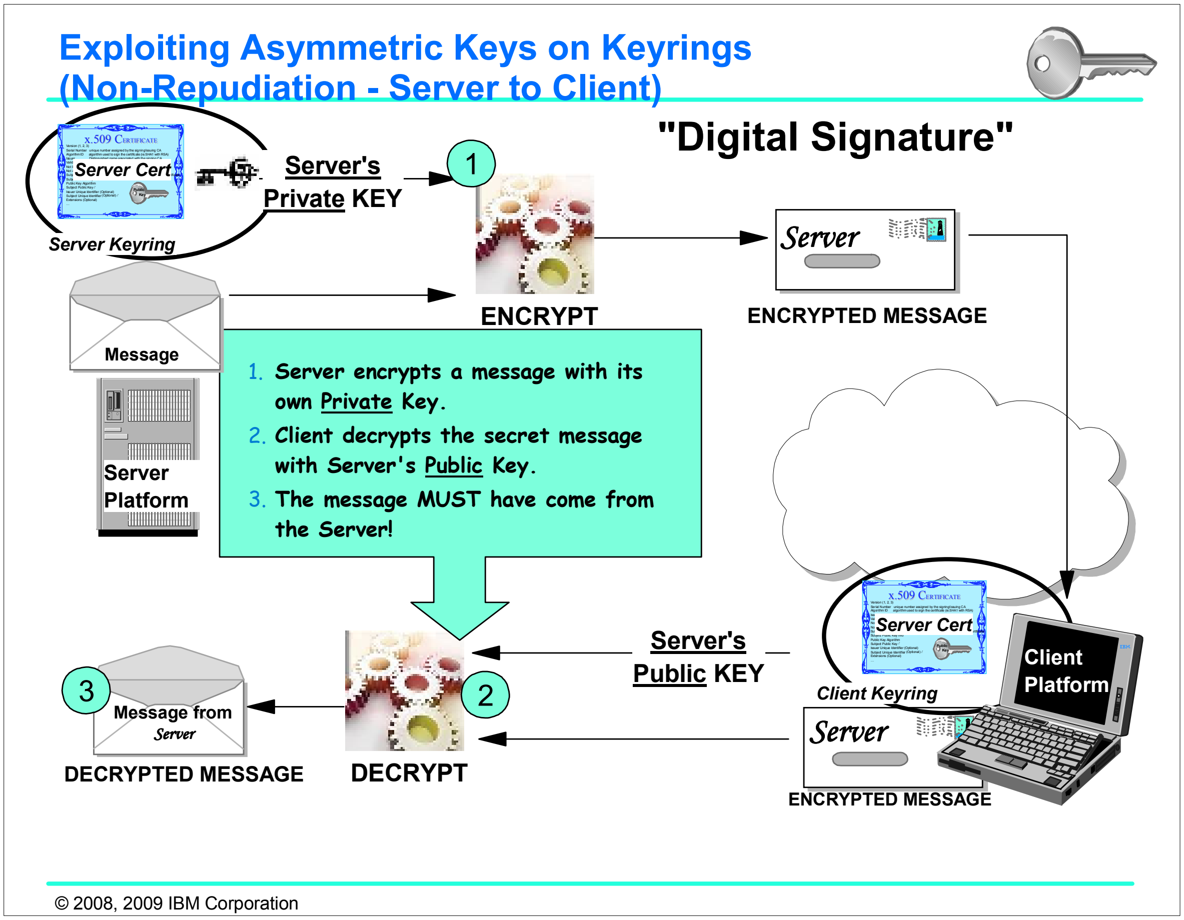
- This is an older version of the ISO security model. Note the entry for "Governance" and "Logging." This is not part of the ISO model, but it is nevertheless integral for any security implementation. We have added it here to show its importance.
- Certificates play security roles in four respects:
  - For Authentication of the Server and Optionally the Client
  - For Encrypting data that will flow between entities secured by x.509 certificates
  - For Ensuring that data has not been altered in flight
  - For providing proof of origin and receipt of data.
- In some ways, especially if one considers the use of Client Certificates in an SSL/TLS or AT-TLS implementation, the Client Certificate can be considered to provide access to a server. This is why the arrow pointing from "x.509 Certificate" to "Access Control" is represented with a dotted line.



Exploiting Asymmetric Keys on Keyrings (e.g., Privacy - Client to Server)

1. Client encrypts a message with Server's Public Key.
2. Server decrypts the secret message with his Private Key.
3. Only Server can read the message!

- The private and public keys are mathematically related. The private key, owned by one entity (the Server in this case), can "unlock" the secret that was encrypted by the public key that is available to anyone.
- Note that the Public key can be made available to the world if necessary. In our example, The Client is using the Server's Publicly available key to send the server an encrypted message that only the Server's Private key can decrypt. But the Private key is in the possession of only ONE entity (the Server in this case).
- Notice how the Client Keyring contains a copy of the Server Certificate. The Client's keyring does not have access to the Private key of the Server.
  - The Server's Certificate can be pre-installed on the Client Keyring. It is more typical to accept the Server Certificate and Install it on the Client Keyring or in the Client Key Database as part of the security negotiation or Handshake.
- Notice how the Server Keyring contains the Server's Certificate (which includes the Public Key) and how it contains the Server's private key as well.
- If Client Authentication is desired, then a Client may also make use of an x.509 Certificate.

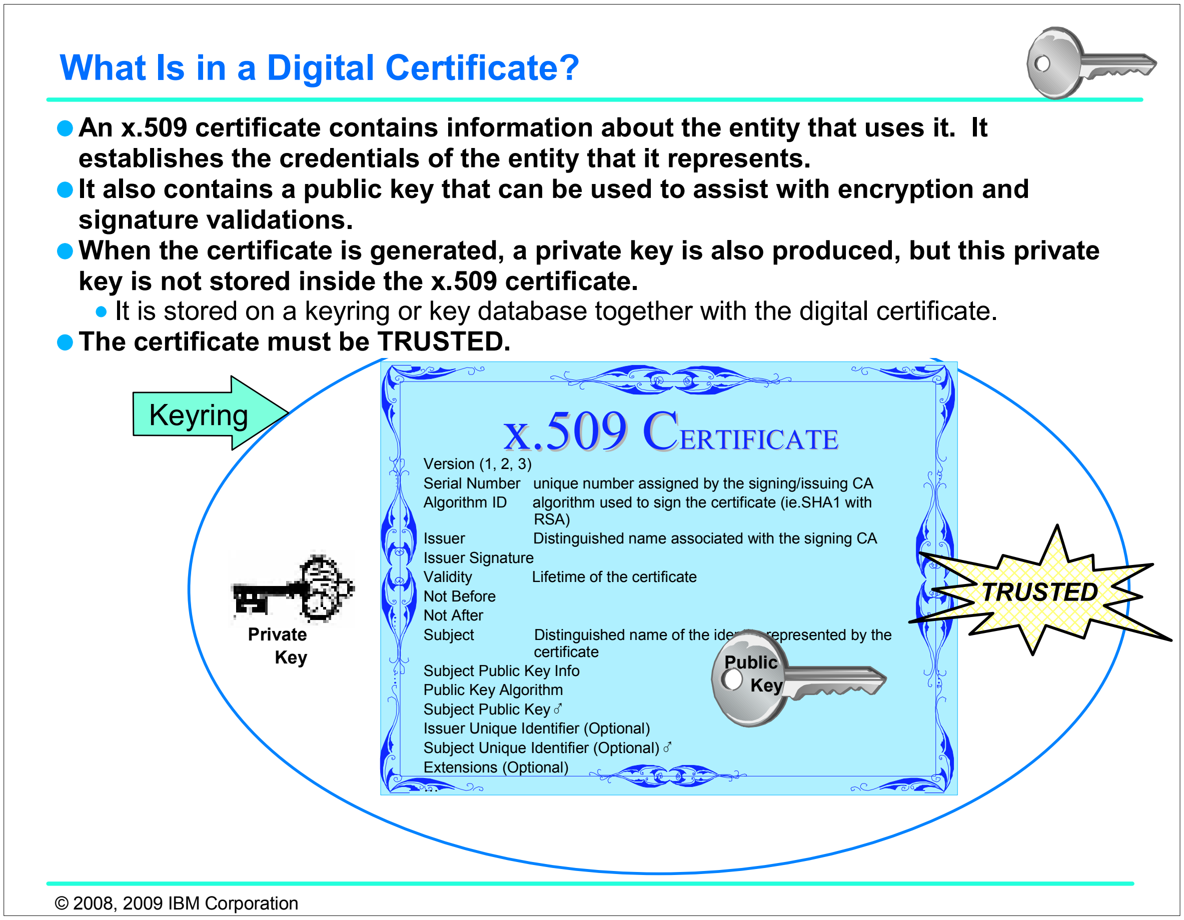


Exploiting Asymmetric Keys on Keyrings (Non-Repudiation - Server to Client)

"Digital Signature"

1. Server encrypts a message with its own Private Key.
2. Client decrypts the secret message with Server's Public Key.
3. The message MUST have come from the Server!

- The private and public keys are mathematically related. The private key, owned by one entity (the Server in this case), can "unlock" the secret that was encrypted by the public key that is available to anyone. If the Server's public key can unlock the secret, then the Server must have been the sender and that server cannot refute or repudiate it -- that is, unless someone has STOLEN the Server's private key. This fact explains why KEY VALUES MUST BE PROTECTED!
- Note that the Public key can be made available to the world if necessary.
- In our example of Non-Repudiation, we have used the Digital Signature feature, whereby we can prove that the Server sent the message. Only the Server's Public key can unlock the secret that was sent, thereby proving that this Server was the sender.
- Digital Signature: If it bears your signature, it came from you!
  - Certificate Digital Signature - Signature generated using the issuer’s private key .
- Notice how the Client Keyring contains a copy of the Server Certificate. The Client's keyring does not have access to the Private key of the Server.
  - The Server's Certificate can be pre-installed on the Client Keyring. It is more typical to accept the Server Keyring and Install it on the Client Keyring or in the Client Key Database as part of the security negotiation or Handshake.
- Notice how the Server Keyring contains the Server's Certificate (which includes the Public Key) and how it contains the Server's private key as well.
- If Client Authentication is desired, then a Client may also make use of an x.509 Certificate.



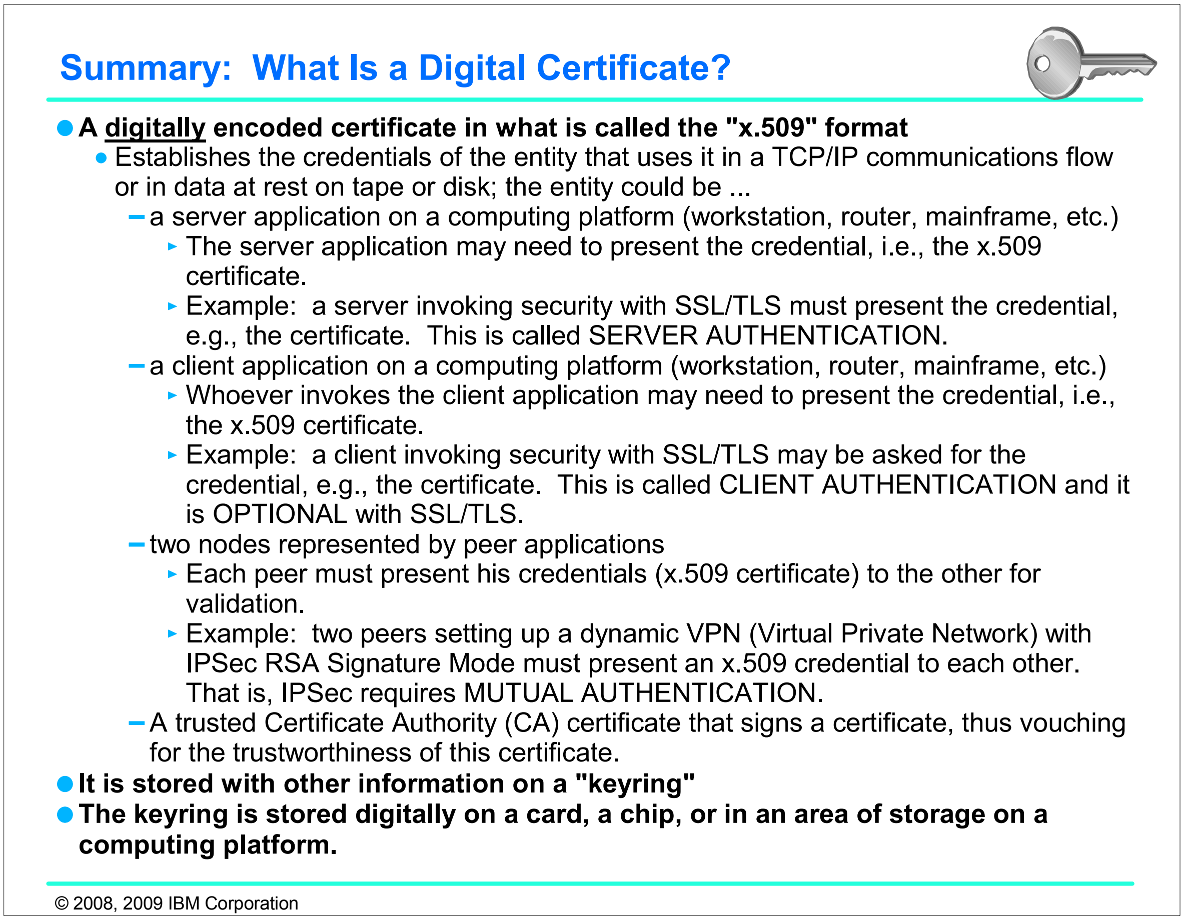
What Is in a Digital Certificate?

- **An x.509 certificate contains information about the entity that uses it. It establishes the credentials of the entity that it represents.**
- **It also contains a public key that can be used to assist with encryption and signature validations.**
- **When the certificate is generated, a private key is also produced, but this private key is not stored inside the x.509 certificate.**
  - It is stored on a keyring or key database together with the digital certificate.
- **The certificate must be TRUSTED.**

x.509 Certificate

Version (1, 2, 3)	
Serial Number	unique number assigned by the signing/issuing CA
Algorithm ID	algorithm used to sign the certificate (ie.SHA1 with RSA)
Issuer	Distinguished name associated with the signing CA
Issuer Signature	
Validity	Lifetime of the certificate
Not Before	
Not After	
Subject	Distinguished name of the identity represented by the certificate
Subject Public Key Info	
Public Key Algorithm	
Subject Public Key ♂	
Issuer Unique Identifier (Optional)	
Subject Unique Identifier (Optional) ♂	
Extensions (Optional)	

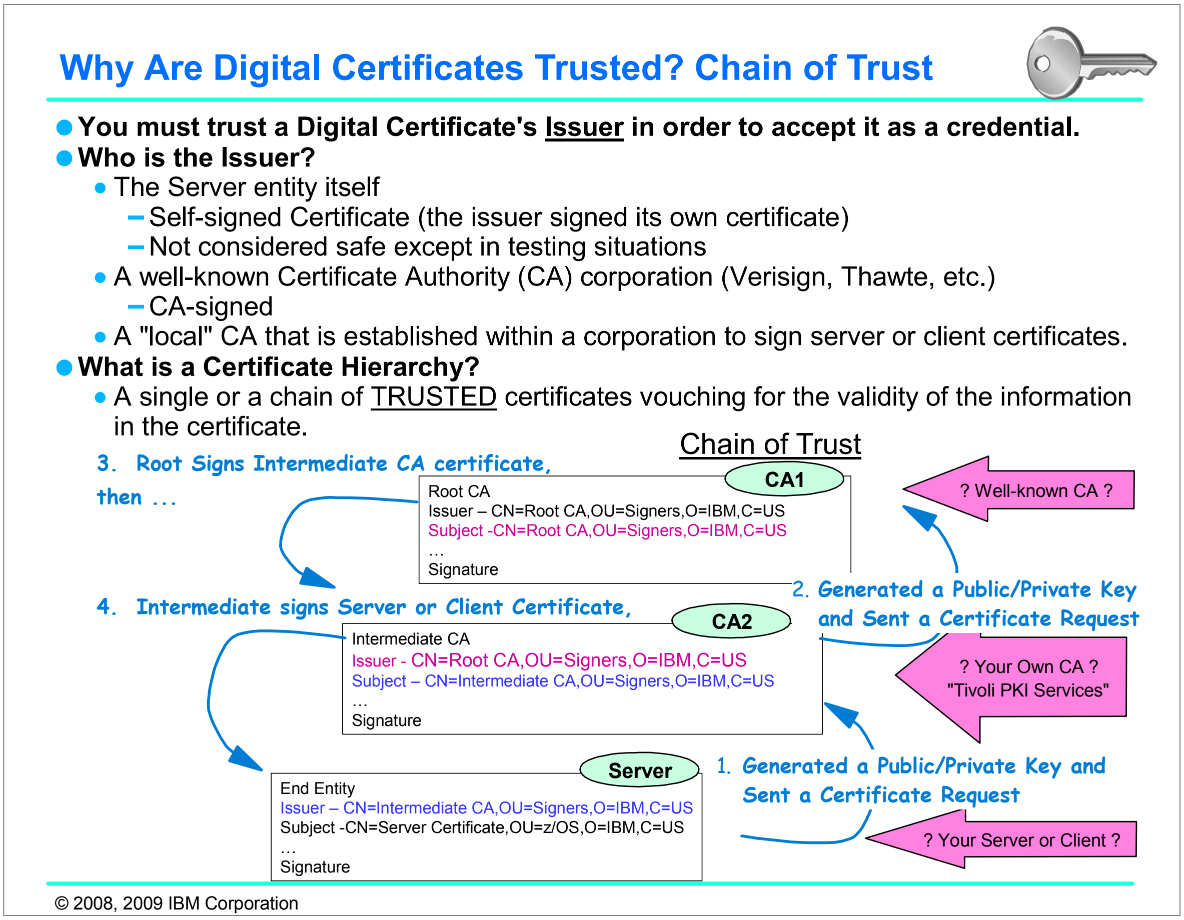
- Only the Public key resides in the x.509 certificate itself.
- The private key resides in the repository of the end-entity that is represented with the certificate.
  - The private key is used for signing certificates.
  - It is also used to decrypt data that has been encrypted with the corresponding public key.
  - It is also used to encrypt data that can be decrypted only by the corresponding public key.
- Any certificate in the "chain of trust" must be marked as TRUSTED in the repository where the keyring or key database resides.
- Digital signature: This is generated using the signing authority's (CA's) private key. To verify the digital signature, we need the signing authority's public key, which is found in the certificate of the signing authority.



Summary: What Is a Digital Certificate?

- **A digitally encoded certificate in what is called the "x.509" format**
  - Establishes the credentials of the entity that uses it in a TCP/IP communications flow or in data at rest on tape or disk; the entity could be ...
    - a server application on a computing platform (workstation, router, mainframe, etc.)
      - The server application may need to present the credential, i.e., the x.509 certificate.
      - Example: a server invoking security with SSL/TLS must present the credential, e.g., the certificate. This is called SERVER AUTHENTICATION.
    - a client application on a computing platform (workstation, router, mainframe, etc.)
      - Whoever invokes the client application may need to present the credential, i.e., the x.509 certificate.
      - Example: a client invoking security with SSL/TLS may be asked for the credential, e.g., the certificate. This is called CLIENT AUTHENTICATION and it is OPTIONAL with SSL/TLS.
    - two nodes represented by peer applications
      - Each peer must present his credentials (x.509 certificate) to the other for validation.
      - Example: two peers setting up a dynamic VPN (Virtual Private Network) with IPSec RSA Signature Mode must present an x.509 credential to each other. That is, IPSec requires MUTUAL AUTHENTICATION.
    - A trusted Certificate Authority (CA) certificate that signs a certificate, thus vouching for the trustworthiness of this certificate.
- **It is stored with other information on a "keyring"**
- **The keyring is stored digitally on a card, a chip, or in an area of storage on a computing platform.**

- Platforms are usually built with a common list of well-known Certificate Authority Certificates. On z/OS you must mark as "Trusted" the desired CA certificates that you wish to add to keyrings. Here is an example of the RACF racdcert command that can mark a well-known CA as Trusted.
  - RACDCERT CERTAUTH ALTER(LABEL('Verisign Class 3 Primary CA')) TRUST



Why Are Digital Certificates Trusted? Chain of Trust

- **You must trust a Digital Certificate's Issuer in order to accept it as a credential.**
- **Who is the Issuer?**
  - The Server entity itself
    - Self-signed Certificate (the issuer signed its own certificate)
    - Not considered safe except in testing situations
  - A well-known Certificate Authority (CA) corporation (Verisign, Thawte, etc.)
    - CA-signed
  - A "local" CA that is established within a corporation to sign server or client certificates.
- **What is a Certificate Hierarchy?**
  - A single or a chain of TRUSTED certificates vouching for the validity of the information in the certificate.

Chain of Trust

3. Root Signs Intermediate CA certificate, then ...

CA1

Root CA
Issuer – CN=Root CA,OU=Signers,O=IBM,C=US
Subject -CN=Root CA,OU=Signers,O=IBM,C=US
…
Signature

? Well-known CA ?

2. Generated a Public/Private Key and Sent a Certificate Request

4. Intermediate signs Server or Client Certificate,

CA2

Intermediate CA
Issuer - CN=Root CA,OU=Signers,O=IBM,C=US
Subject – CN=Intermediate CA,OU=Signers,O=IBM,C=US
…
Signature

? Your Own CA ?
"Tivoli PKI Services"

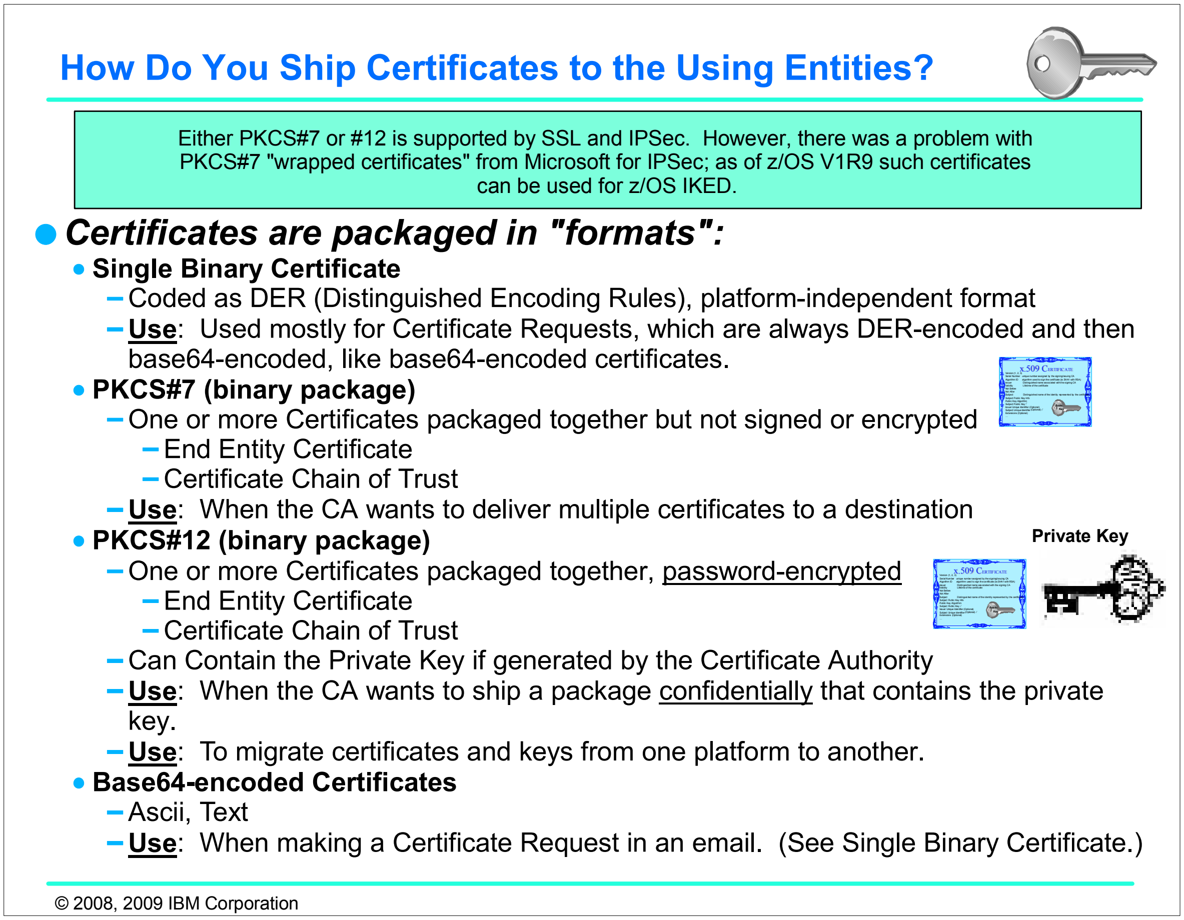
Server

End Entity
Issuer – CN=Intermediate CA,OU=Signers,O=IBM,C=US
Subject -CN=Server Certificate,OU=z/OS,O=IBM,C=US
…
Signature

1. Generated a Public/Private Key and Sent a Certificate Request

? Your Server or Client ?

- Self signed Certificates are Self-issued
  - Issuer and subject names identical
  - Signed by iServer or Owning Entity itself using associated private key
- Signed Certificates
  - Signed/issued by a trusted Certificate Authority Certificate using its private key.
  - By signing the certificate, the CA certifies the validity of the information. Can be a well-known commercial organization or local/internal organization.
- Hierarchy:
  - Single (self-signed) level in the Chain of Trust ("no chain")
    - Not recommended for end entity certificates in production
    - No trusted party involved; trusting subject
  - Multiple levels in the Chain of Trust
    - Consists of the end entity certificate and 1 or more certificate authority (CA) certificates
    - No defined limit on number of CA certificates
- You might use a publicly trusted corporation as your Certificate Authority or you might use IBM's TIVOLI PKI Services to be your own Certificate Authority.
- The Certificate hierarchy depicted here contains 3 entities. The end-entity certificate is issued by the Intermediate CA. The Intermediate CA is issued by the Root CA. The Root CA is self-issued (signed by its own private key).
  - You can use any depth CA chain. The key to getting this to work is what is sent in the certificate request payload. A CA chain Root->Sub1->Sub2->Sub3-Sub4-Sub5...SubN-EE will work as long as a Certificate Request sent identifies SubN or is empty.
  - WARNING: When using a multi-level CA hierarchy, the Windows 2003 server sent its certificate using PKCS #7 encoding, which z/OS Communications Server did not support **for IPSec** prior to V1R10. The support was rolled back to z/OS Communications Server V1R9. Different Windows servers behave different - the problem with PKCS#7 has only surfaced with the 2003 server and with IPSec -- not SSL/TLS or AT-TLS.
- When validating the certificate chain, validation starts with end entity certificate. The public key of the Intermediate CA certificates is used to verify the end-entity certificate. If valid, the public key of the Root CA certificate is used to verify the Intermediate CA certificate. If valid, the Root CA is validated using its own public key.
- Note: Signature validation of the self-signed root certificate ensures that the certificate has not been altered but does not guarantee the trust of the certificate. You must establish trust with the certificate authority prior to using the certificate.
- A certificate signing request (also CSR or certification request) is a message sent from the certificate requestor to a certificate authority to obtain a signed digital certificate
  - contains identifying information and public key for the requestor
  - Corresponding private key is not included in the CSR, but is used to digitally sign the request to ensure the request is actually coming from the requestor
  - CSR may be accompanied by other credentials or proofs of identity required by the certificate authority, and the certificate authority may contact the requestor for further information.
  - If the request is successful, the certificate authority will send back an identity certificate that has been digitally signed with the private key of the certificate authority.



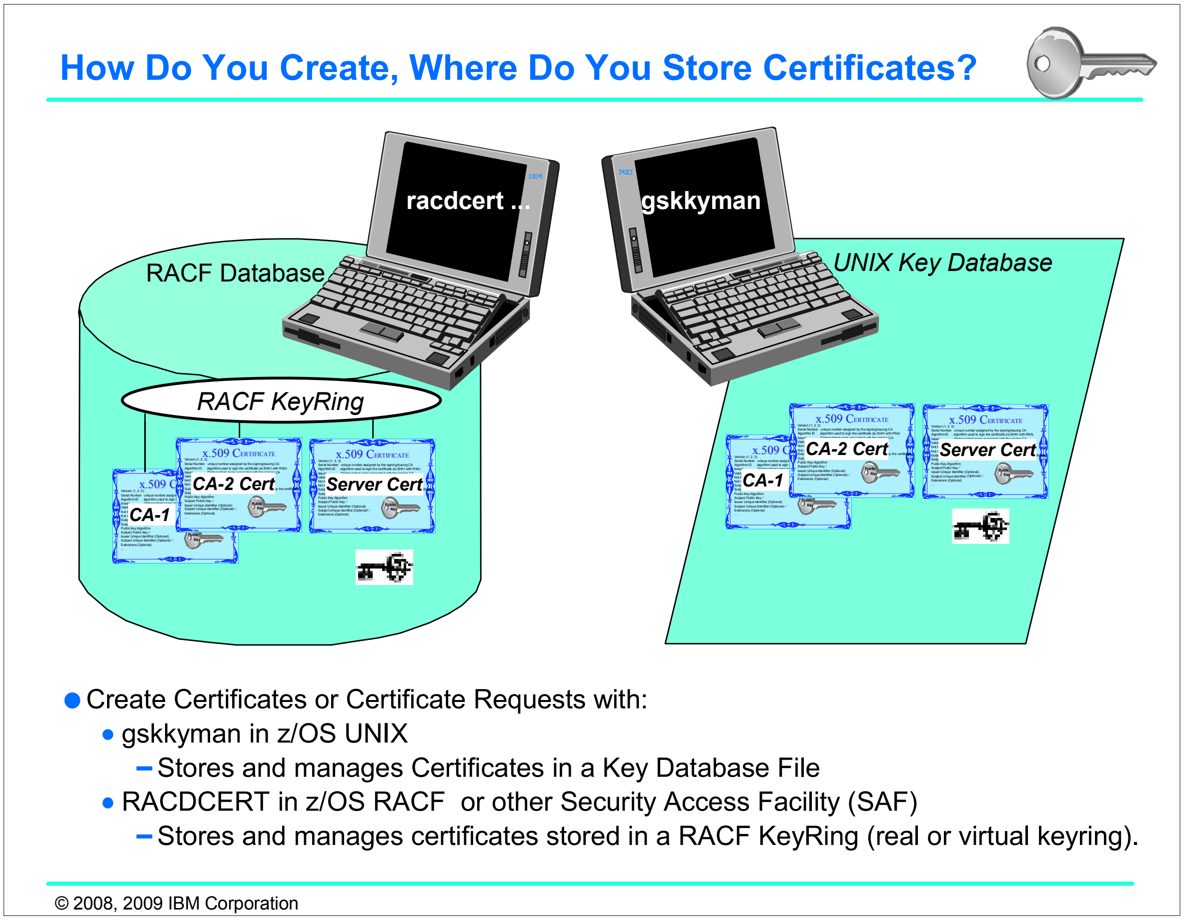
How Do You Ship Certificates to the Using Entities?

Either PKCS#7 or #12 is supported by SSL and IPSec. However, there was a problem with PKCS#7 "wrapped certificates" from Microsoft for IPSec; as of z/OS V1R9 such certificates can be used for z/OS IKED.

- ***Certificates are packaged in "formats":***
  - **Single Binary Certificate**
    - Coded as DER (Distinguished Encoding Rules), platform-independent format
    - **Use**: Used mostly for Certificate Requests, which are always DER-encoded and then base64-encoded, like base64-encoded certificates.
  - **PKCS#7 (binary package)**
    - One or more Certificates packaged together but not signed or encrypted
      - End Entity Certificate
      - Certificate Chain of Trust
    - **Use**: When the CA wants to deliver multiple certificates to a destination
  - **PKCS#12 (binary package)**
    - One or more Certificates packaged together, password-encrypted
      - End Entity Certificate
      - Certificate Chain of Trust
    - Can Contain the Private Key if generated by the Certificate Authority
    - **Use**: When the CA wants to ship a package confidentially that contains the private key.
    - **Use**: To migrate certificates and keys from one platform to another.
  - **Base64-encoded Certificates**
    - Ascii, Text
    - **Use**: When making a Certificate Request in an email. (See Single Binary Certificate.)

- PKCS = Public Key Cryptographic Standards
- If you want to create a backup copy of an existing certificate (and its non-ICSF private key) on a different system, use the RACDCERT EXPORT command to create a PKCS #12 format data set on the system where the certificate resides, and send the data set to the other system where you can use it as input with the RACDCERT ADD command to recreate the same certificate. Restriction: If the private key is stored in ICSF (key type ICSF or PCICC), a PKCS #12 data set cannot be created. (See z/OS Security Server RACF Command Language Reference for details about using the RACDCERT EXPORT command.)

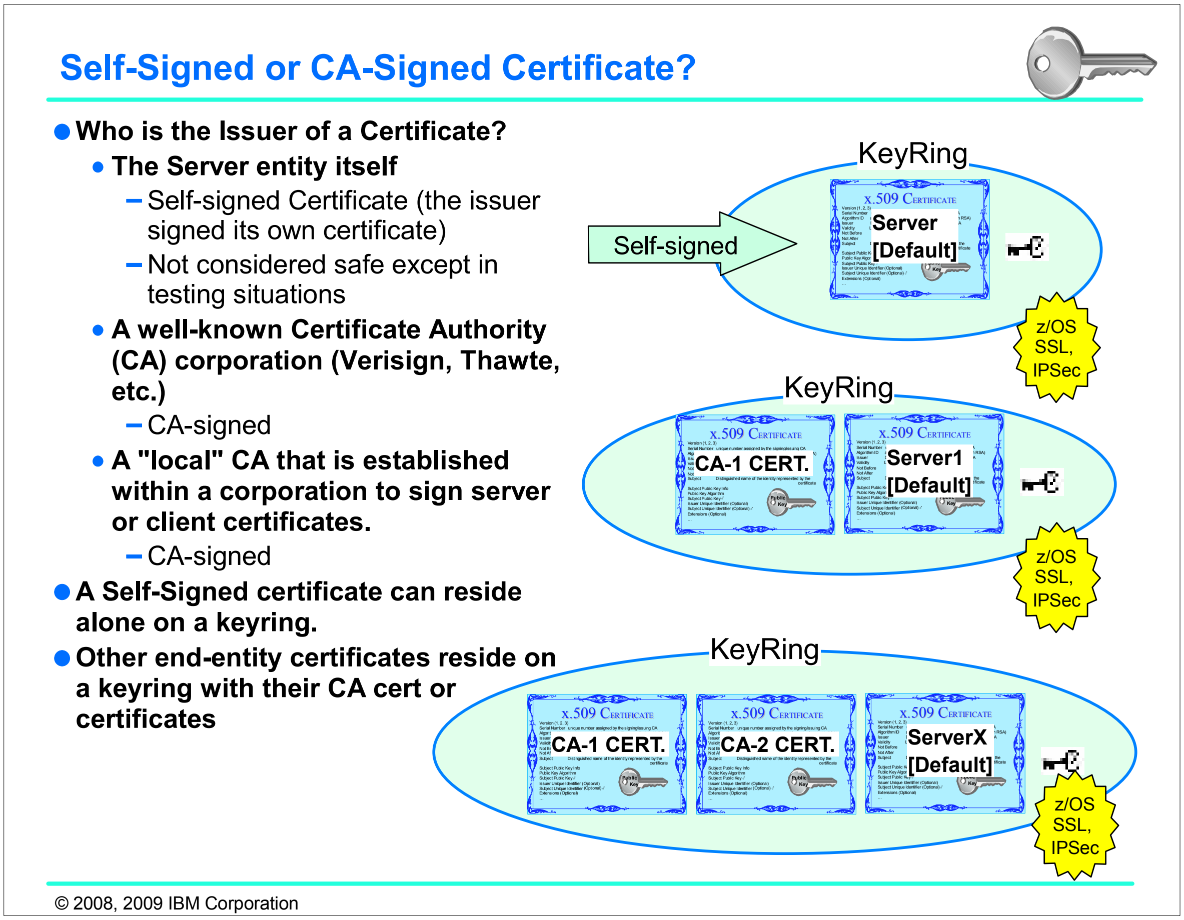
Keyring / Key Database



How Do You Create, Where Do You Store Certificates?

- Create Certificates or Certificate Requests with:
  - gskkyman in z/OS UNIX
    - Stores and manages Certificates in a Key Database File
  - RACDCERT in z/OS RACF or other Security Access Facility (SAF)
    - Stores and manages certificates stored in a RACF KeyRing (real or virtual keyring).

- The RACF RACDCERT Command can be used to generate a Certificate Request that can be sent to a Certificate Authority that will produce the x.509 certificate and send it back. RACDCERT is used to install and maintain digital certificates, key rings, and digital certificate mappings in RACF. RACDCERT should be used for all maintenance of the DIGTCERT, DIGTRING, and DIGTNMAP class profiles.
  - It also produces a keyring.
  - It can generate keys.
  - It can produce a self-signed certificate.
  - Imports/Exports certificates (with and without private keys)
  - Can renew and revoke a certificate.
- The RACDCERT command is a RACF TSO command used to:
  - List information about the certificates for a specified RACF-defined user ID, or your own user ID.
  - Add a certificate definition and associate it with a specified RACF-defined user ID, or your own user ID, and set the TRUST flag.
  - Alter the TRUST flag or the label name for a definition.
  - Delete a definition.
  - List a certificate contained in a data set and determine if it is associated with a RACF-defined user ID.
  - Add or remove a certificate from a key ring.
  - Create, delete, or list a key ring.
  - Generate a public/private key pair and certificate.
  - Write a certificate to a data set.
  - Create a certificate request.
  - Create, alter, delete, or list a user ID mapping.
  - Add, delete, or list a z/OS PKCS #11 token.
  - Bind a certificate to a z/OS PKCS #11 token.
  - Remove (unbind) a certificate from a z/OS PKCS #11 token.
  - Export a certificate (with its private key, if present) from a z/OS PKCS #11 token and add it to RACF.
- Alternatively, the UNIX System Services Command "gskkyman" can be used to generate the Certificate Request.
  - It also produces a keyring.
  - It can generate keys.
  - It can produce a self-signed certificate.
  - Imports/Exports certificates (with and without private keys)
  - Can renew and revoke a certificate.
- RACF Key Rings
  - RACF key rings are protected by resource profiles.
  - Users need read access to IRR.DIGTCERT.LISTRING to be able to read the contents of their key ring.
- gskkyman key database files
  - Protected by the file system’s permission bits and password
  - Upon creation, permission bits are 700 giving the issuer of gskkyman read and write to the file only.
  - Applications using these files need at least read to the file
- OTHER PLATFORMS: Other utilities on different platforms are commonly used to create certificates as well. You may run into a utility called "mkkf" or another one called "ikeyman."



Self-Signed or CA-Signed Certificate?

- Who is the Issuer of a Certificate?
  - The Server entity itself
    - Self-signed Certificate (the issuer signed its own certificate)
    - Not considered safe except in testing situations
  - A well-known Certificate Authority (CA) corporation (Verisign, Thawte, etc.)
    - CA-signed
  - A "local" CA that is established within a corporation to sign server or client certificates.
    - CA-signed
- A Self-Signed certificate can reside alone on a keyring.
- Other end-entity certificates reside on a keyring with their CA cert or certificates

KeyRing

Self-signed

Server [Default]

z/OS SSL, IPSec

KeyRing

CA-1 CERT.

Server1 [Default]

z/OS SSL, IPSec

KeyRing

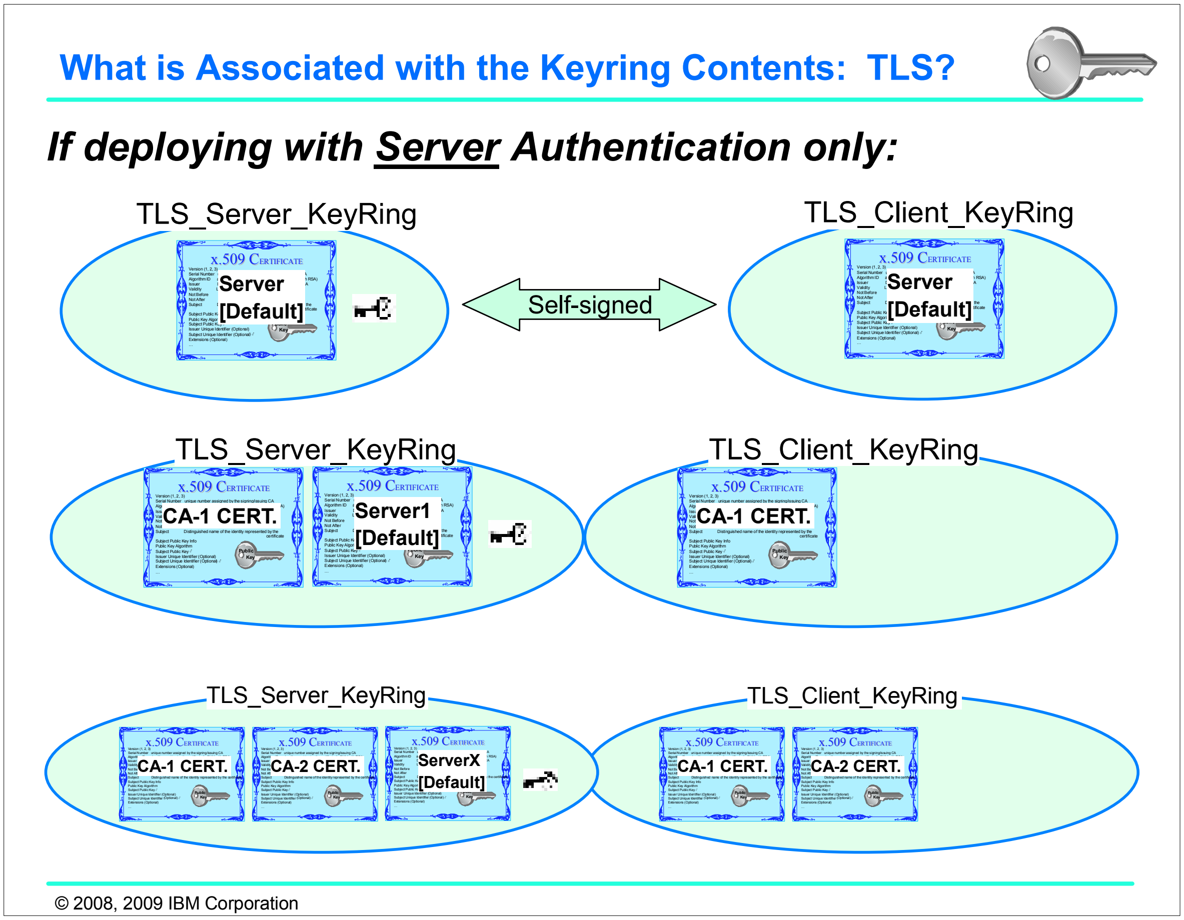
CA-1 CERT.

CA-2 CERT.

ServerX [Default]

z/OS SSL, IPSec

- System SSL and other security middleware use the R_datalib callable service (IRRSDL00 or IRRSDL64) to retrieve certificate information from RACF. In order for applications to retrieve certificates and private keys from RACF, the certificates must be connected to a RACF key ring (including a virtual key ring) or a z/OS PKCS #11 token.
- Note how the self-signed certificate resides alone on its keyring.
- Note how the keyring in the middle of the page shows a Server Certificate that is signed by a well-known CA named "CA1."
- Note how the keyring at the bottom of the page shows a Server Certificate that is signed by a corporate (intermediate CA) CA (CA2), which has then been signed by a well-known CA (CA1).
- Which certificates should be on a LOCAL keyring? In short, you should find the certificates that will authenticate both any local certificate that might be needed and any remote certificate that might be presented. For example:
  - the local end-entity certificate if it is to be presented to a peer in the secure exchange.
  - the Certificate that has signed the local end-entity certificate
  - The Certificate that has signed the signing certificate
  - at least one TRUSTED certificate that has signed the remote end-entity's certificate.
- NOTE: Although it is possible for a keyring to work successfully even if not all of the cited certificates reside on it, it is wiser to place all the certificates in the keyring. Otherwise certain negotiations may fail due to the protocol and platform differences that exist for the implementation of SSL/TLS (AT-TLS) and IPSec.
- The usage assigned to a certificate when it is connected to a key ring indicates its intended purpose.
  - Personal certificates are to be used by the local server application to identify itself.
  - Certificate-authority certificates are to be used to verify the peer entity’s certificate.
  - Peers with certificates issued by certificate authorities connected to the key ring are considered trusted network entities.
  - NOTE: Peers possessing certificates that can not be verified because the certificate-authority certificate is not available may also be considered trusted if their certificates are connected to the key ring as a trusted site certificate.
  - RESTRICTIONS:
    - 1. Use caution when connecting a peer’s certificate to a key ring as a trusted site certificate. The normal certificate verification tests performed by the server on the peer’s certificate are bypassed in this case. Hence, even expired certificates are considered trusted.
    - 2. Certificates marked NOTRUST cannot be retrieved using the R_datalib callable service even if they are connected to a key ring. RACF hides them from the calling application and does not indicate that they are connected to the key ring.



What is Associated with the Keyring Contents: TLS?

If deploying with __Server__ Authentication only:

TLS_Server_KeyRing

Server [Default]

Self-signed

TLS_Client_KeyRing

Server [Default]

TLS_Server_KeyRing

CA-1 CERT.

Server1 [Default]

TLS_Client_KeyRing

CA-1 CERT.

TLS_Server_KeyRing

CA-1 CERT.

CA-2 CERT.

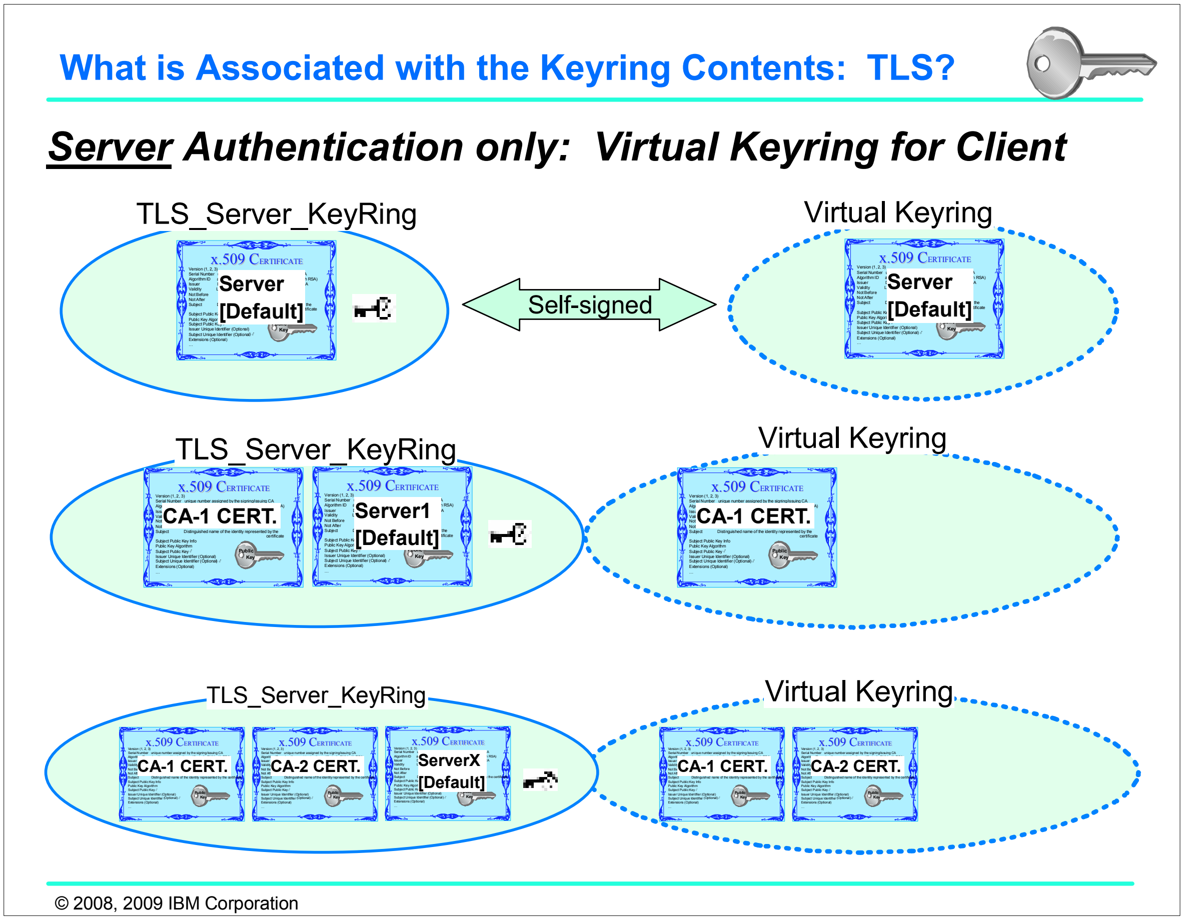
ServerX [Default]

TLS_Client_KeyRing

CA-1 CERT.

CA-2 CERT.

- We show you here three examples of keyring configurations when using <u>Server Authentication only for SSL/TLS</u> or AT-TLS. In general, a TRUSTED Certificate or Certificate Chain and the PRIVATE Key of the End-Entity that owns the certificate is required. But whether or not the PRIVATE KEY need be available depends on whether mutual authentication is required or not. You see in the three examples that the client keyring needs no PRIVATE key associated with it if only Server Authnetication is in use.
- In the first example, you see that we have deployed only a server certificate. In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate.
  - The Client's keyring needs a copy of the self-signed server certificate in order to validate the certificate that the Server sends to it during SSL/TLS or AT-TLS negotiation.
- In the second example, you see that we have deployed a server certificate that has been signed by a CA certificate (CA1). In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It must also have access to the CA certificate that has signed the server certificate.
  - The Client's keyring needs a copy of the CA certificate that has signed the Server certificate in order to validate the certificate that the Server sends to it during SSL/TLS or AT-TLS negotiation.
- In the third example, you see that we have deployed a server certificate that has been signed by a CA certificate (CA2), which itself has been signed by another root CA certificate (CA1). In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It must also have access to both CA certificates.
  - The Client's keyring needs a copy of both CA certificates that were used to sign the server certificate and the intermediate CA certificate.
- EXAMPLE:
  - A local keyring should contain:
  - The server certificate for the local End Entity.
  - The CA certificate that signed the local End Entity certificate.
  - The CA certificate that signed a remote End Entity certificate, if the negotiation requests Client Authentication.
  - The CA certificates that may have signed an Intermediate CA certificate that resides on the ring.



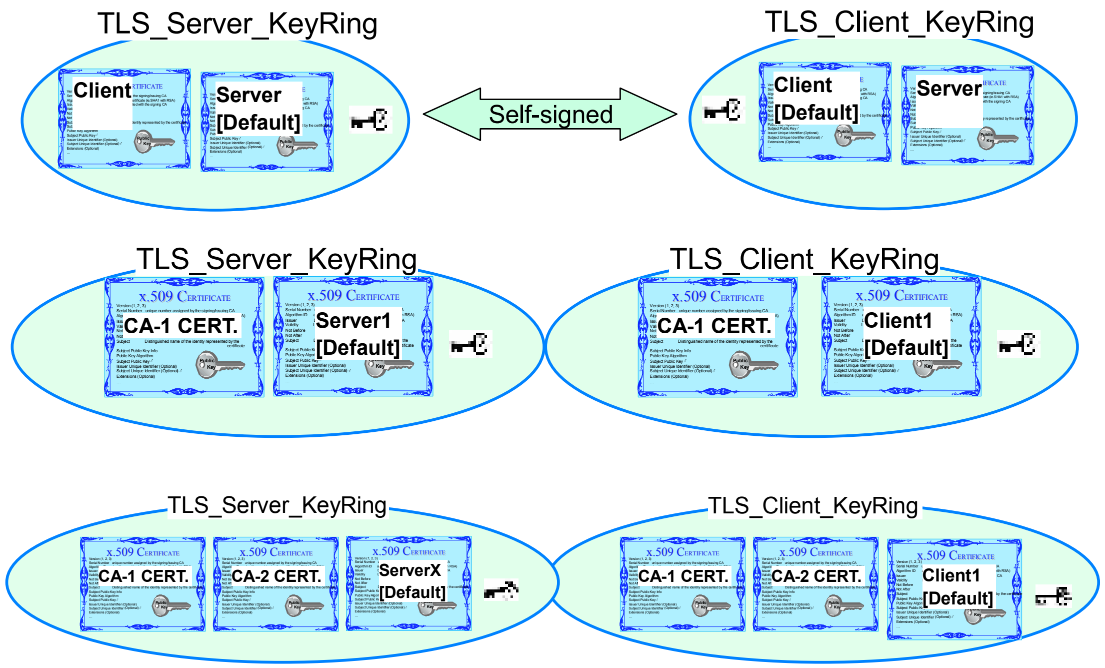
What is Associated with the Keyring Contents: TLS?

Server Authentication only: Virtual Keyring for Client

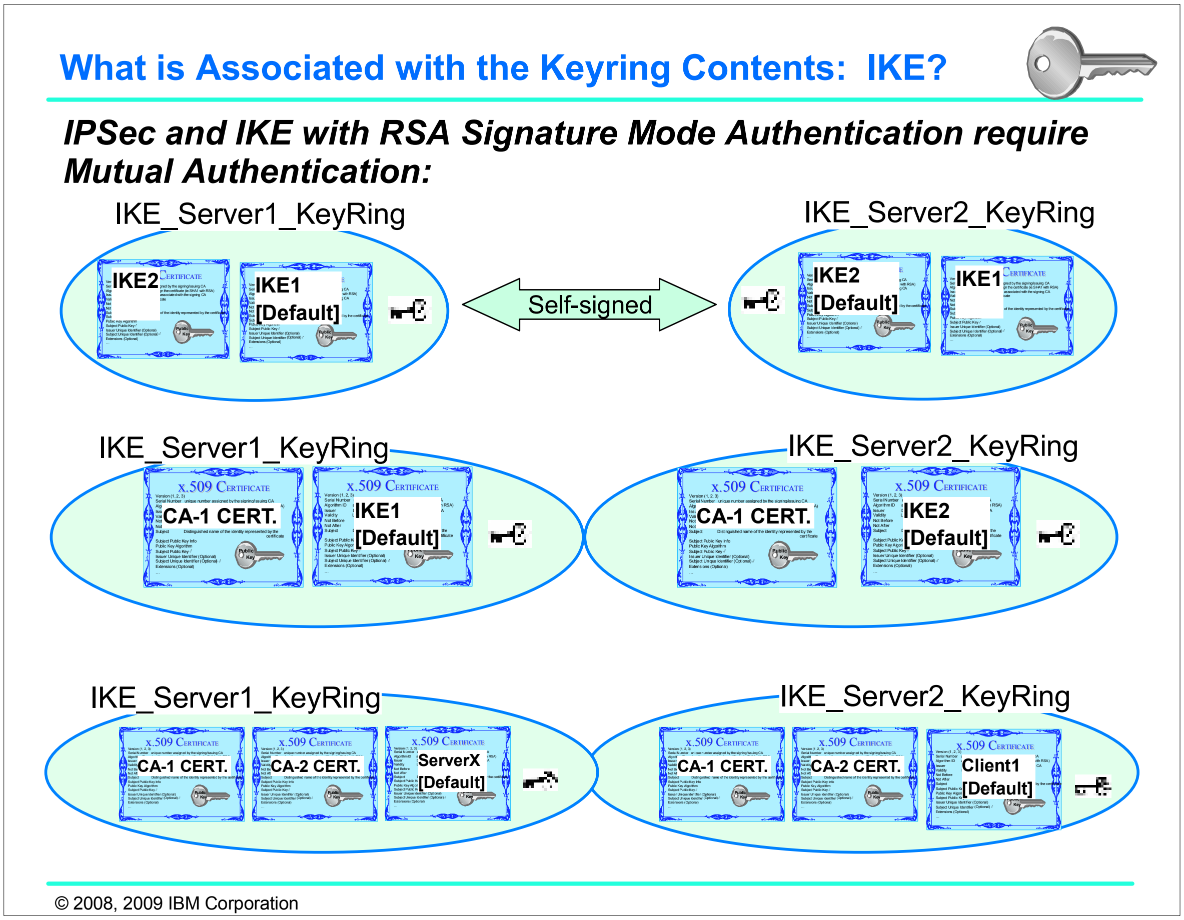
- This chart shows you that as of V1R8 you can build a RACF Virtual Keyring if you are not presenting a client certificate for verification.
- For applications using System SSL, such as z/OS FTP, or other middleware programs that read RACF key rings through the R_datalib callable service, a virtual key ring can be specified in place of a real key ring, whenever a real key ring is expected. To include virtual key rings, the application user specifies an asterisk (*) for the key ring name along with the ring owner's user ID using the form ring-owner/*
- We show you here three examples of keyring configurations when using Server Authentication only for SSL/TLS or AT-TLS. In general, a TRUSTED Certificate or Certificate Chain and the PRIVATE Key of the End-Entity that owns the certificate is required. But whether or not the PRIVATE KEY need be available depends on whether mutual authentication is required or not. You see in the three examples that the client keyring needs no PRIVATE key associated with it if only Server Authnetication is in use.
- In the first example, you see that we have deployed only a server certificate. In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate.
  - The Client's keyring needs a copy of the self-signed server certificate in order to validate the certificate that the Server sends to it during SSL/TLS or AT-TLS negotiation.
- In the second example, you see that we have deployed a server certificate that has been signed by a CA certificate (CA1). In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It must also have access to the CA certificate that has signed the server certificate.
  - The Client's keyring needs a copy of the CA certificate that has signed the Server certificate in order to validate the certificate that the Server sends to it during SSL/TLS or AT-TLS negotiation.
- In the third example, you see that we have deployed a server certificate that has been signed by a CA certificate (CA2), which itself has been signed by another root CA certificate (CA1). In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It must also have access to both CA certificates.
  - The Client's keyring needs a copy of both CA certificates that were used to sign the server certificate and the intermediate CA certificate.

What is Associated with the Keyring Contents: TLS?

If deploying with ***Server + Client*** *Authentication:*



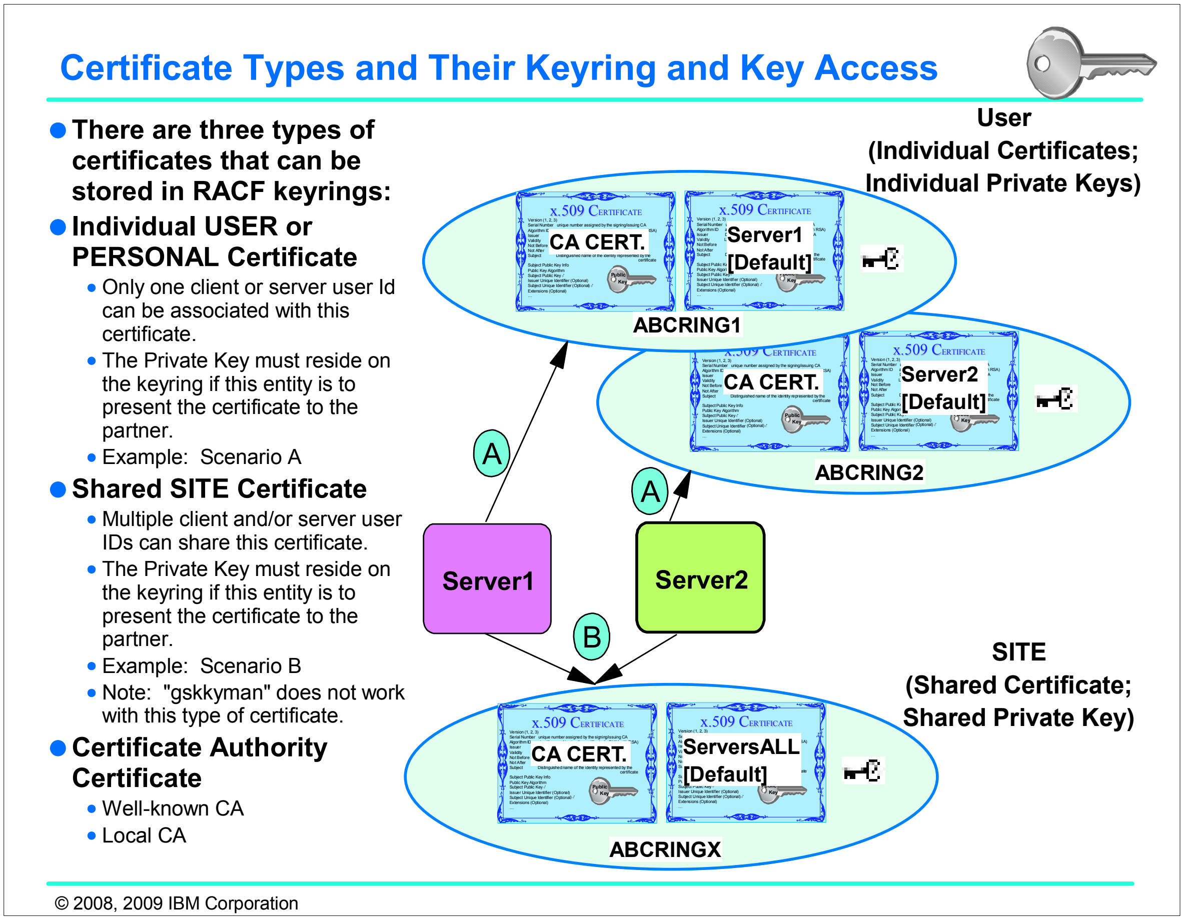
- We show you here three examples of keyring configurations when using <u>Server and Client (or MUTUAL) Authentication </u>only for SSL/TLS or AT-TLS. In general, a TRUSTED Certificate or Certificate Chain and the PRIVATE Key of the End-Entity that owns the certificate is required. But whether or not the PRIVATE KEY need be available depends on whether mutual authentication is required or not. You see in the three examples that the client keyring needs a copy of its own PRIVATE key, and of its own Certificate, and also of the signing CA for the server Certificate. Similar requirements exist for the Server Ring, as you saw on the previous page.
- In the first example, you see that we have deployed mutual authentication with self-signed certificates.
  - In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It also needs a copy of the Client certificate in order to validate the Client certificate sent to it during client authentication.
  - The Client's keyring needs a copy of the self-signed server certificate in order to validate the certificate that the Server sends to it during SSL/TLS or AT-TLS negotiation. It must also have a copy of its own self-signed Client Certificate.
- In the second example, you see that we have deployed a server certificate that has been signed by a CA certificate (CA1). The Client certificate has been signed by the same CA (CA1).
  - In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It must also have access to the CA certificate (CA1) that has signed the server certificate. It must also have access to the CA certificate (CA1) that has signed the client certificate.
  - The Client's keyring needs a copy of the Client Certificate and of the CA certificate (CA1) that has signed the Client certificate. The Private Key of the client must be available as well. The client's keyring must also have access to the CA certificate (CA1) that has signed the Server certificate. In this example, the signing CA was the same for both the client and the server certificates.
- In the third example, you see that we have deployed a server certificate that has been signed by a CA certificate (CA2), which itself has been signed by another root CA certificate (CA1).
  - In this case, the server's keyring or key repository must have access to the server's private key as well as to its own server certificate. It must also have access to both CA certificates (CA1 and CA2).
  - The Client's keyring needs a copy of both CA certificates that were used to sign the server certificate (CA2) and the intermediate CA certificate (CA1). The client keyring requires a copy of its own client certificate and access to the private key of the Client. It requires copies of the CA chain that has signed the Client certificate. However, since this is the same CA chain that was used for the Server Certificate, the presence of both CA1 and CA2 provides a trusted string to authenticate both the client and server certificates.



What is Associated with the Keyring Contents: IKE?

IPSec and IKE with RSA Signature Mode Authentication require Mutual Authentication:

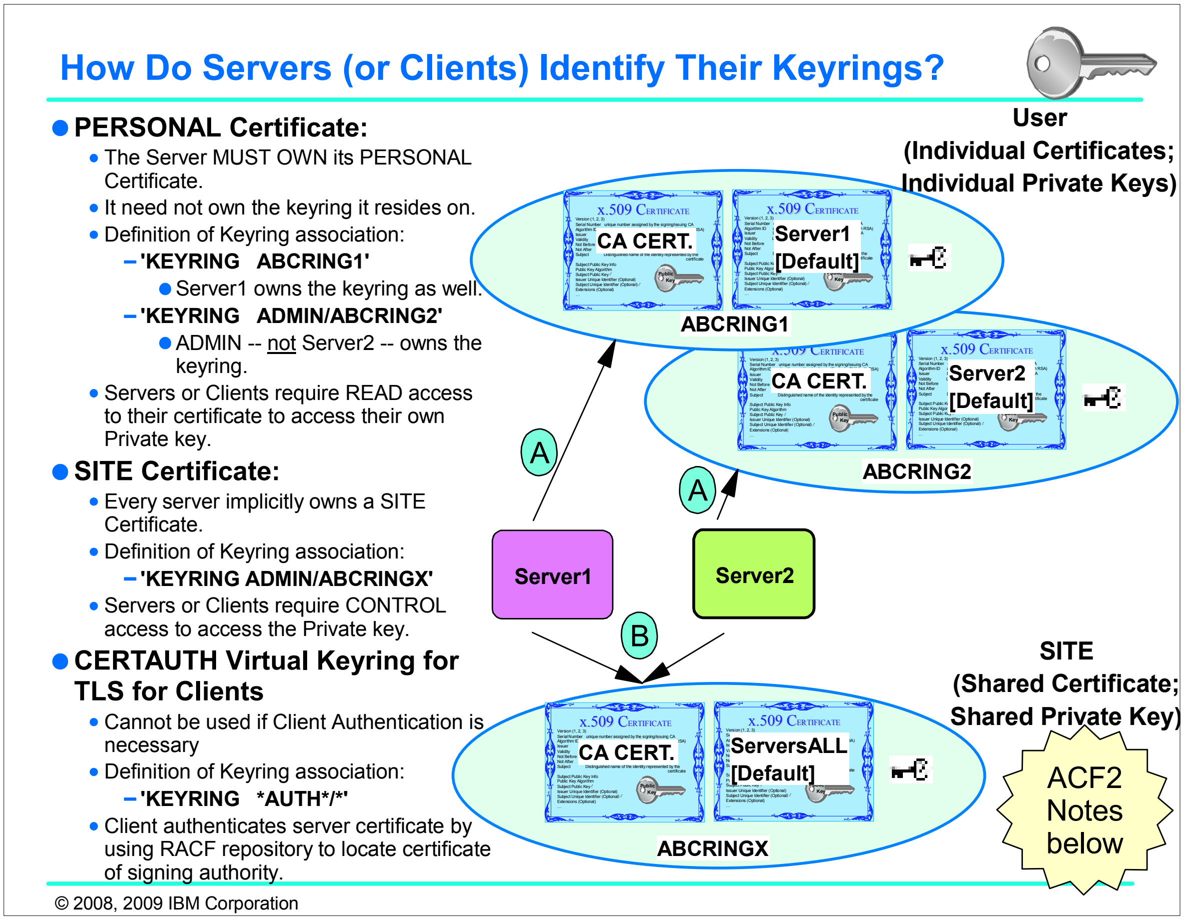
- We show you here three examples of keyring configurations when using MUTUAL Authentication only for an IPSec environment using RSA Signature Mode Authentication. In general, a TRUSTED Certificate or Certificate Chain and the PRIVATE Key of the End-Entity that owns the certificate is required.
- The configuration of the keyrings is similar to what you saw for AT-TLS, when Client/Server authentication is in use. In both cases mutual authentication is required.
- In the first example, you see that we have deployed mutual authentication with self-signed certificates.
  - In this case, the IKE1 keyring or key repository must have access to its own private key as well as to its IKE1 end-entitiy certificate. It also needs a copy of the IKE2 certificate in order to validate the IKE2 certificate sent to it during IKE authentication.
  - The IKE2 keyring must have access to its own private key as well as to its IKE2 end entity certificate. It also needs a copy of the IKE1 certificate in order to validate the IKE1 certificate sent to it during IKE authentication.
- In the second example, you see that we have deployed IKE certificates that have been signed by a CA certificate (CA1).
  - In this case, the IKE1 keyring or key repository must have access to the IKE1 private key as well as to its own IKE1 certificate. It must also have access to the CA certificate (CA1) that has signed the IKE1 certificate. It must also have access to the CA certificate (CA1) that has signed the IKE2 certificate. (In our example, the same CA signed both IKE1 and IKE2 certificates.
  - The IKE2 keyring needs a copy of the IKE2 Certificate and of the CA certificate (CA1) that has signed the IKE2 certificate. The Private Key of IKE2 must be available as well. The IKE2 keyring must also have access to the CA certificate (CA1) that has signed the IKE1 certificate. In this example, the signing CA was the same for both of the IKEs.
- In the third example, you see that we have deployed an IKE1 certificate that has been signed by a CA certificate (CA2), which itself has been signed by another root CA certificate (CA1). The IKE2 certificate has also been signed by CA2, which has been signed by the root CA certificate (CA1).
  - In this case, the IKE1 must have access to the IKE1 private key as well as to the IKE1 certificate. It must also have access to both CA certificates (CA1 and CA2).
  - In this case, the IKE2 must have access to the IKE2 private key as well as to the IKE2 certificate. It must also have access to both CA certificates (CA1 and CA2).



Certificate Types and Their Keyring and Key Access

- There are three types of certificates that can be stored in RACF keyrings:
- Individual USER or PERSONAL Certificate
  - Only one client or server user Id can be associated with this certificate.
  - The Private Key must reside on the keyring if this entity is to present the certificate to the partner.
  - Example: Scenario A
- Shared SITE Certificate
  - Multiple client and/or server user IDs can share this certificate.
  - The Private Key must reside on the keyring if this entity is to present the certificate to the partner.
  - Example: Scenario B
  - Note: "gskkyman" does not work with this type of certificate.
- Certificate Authority Certificate
  - Well-known CA
  - Local CA

- USER CERTIFICATE:
  - A certificate that is associated with a RACF user ID and is used to authenticate the user’s identity. The RACF user ID can represent a traditional user or be assigned to a server or started procedure.
- SHARED SITE CERTIFICATE:
  - You can share a certificate and the certificate’s private key among two or more servers (user IDs) when you add or generate the shared certificate and its private key as a SITE certificate, for example using the RACDCERT SITE GENCERT command. Sharing a certificate can save you the expense of purchasing a new certificate for each server and avoids the overhead of exporting and importing certificate copies. Sharing a private key requires a high degree of authority for each server involved. The key ring containing the shared certificate must be protected and each server must be configured to access the shared key ring and have sufficient access authority to read it. In addition, each server must have CONTROL authority for the IRR.DIGTCERT.GENCERT resource. This resource controls the server’s ability to retrieve private keys using the R_datalib callable service and is checked when you issue the RACDCERT GENCERT SIGNWITH command.
- CERTIFICATE-AUTHORITY CERTIFICATE:
  - A certificate that is associated with a certificate authority and is used to verify signatures in other certificates. Also called a "Signing Certificate."

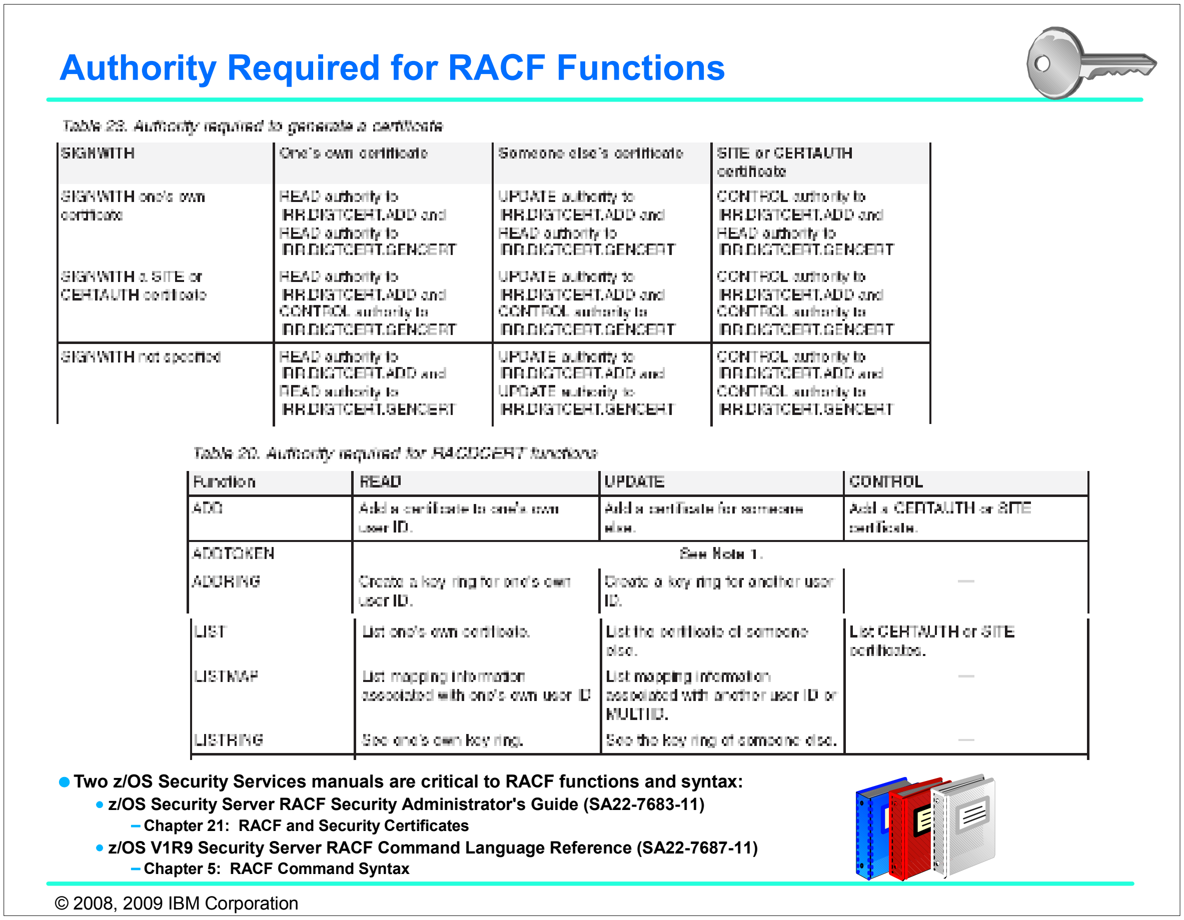


How Do Servers (or Clients) Identify Their Keyrings?

- **PERSONAL Certificate:**
  - The Server MUST OWN its PERSONAL Certificate.
  - It need not own the keyring it resides on.
  - Definition of Keyring association:
    - **'KEYRING ABCRING1'**
      - Server1 owns the keyring as well.
    - **'KEYRING ADMIN/ABCRING2'**
      - ADMIN -- not Server2 -- owns the keyring.
  - Servers or Clients require READ access to their certificate to access their own Private key.
- **SITE Certificate:**
  - Every server implicitly owns a SITE Certificate.
  - Definition of Keyring association:
    - **'KEYRING ADMIN/ABCRINGX'**
  - Servers or Clients require CONTROL access to access the Private key.
- **CERTAUTH Virtual Keyring for TLS for Clients**
  - Cannot be used if Client Authentication is necessary
  - Definition of Keyring association:
    - **'KEYRING *AUTH*/*'**
  - Client authenticates server certificate by using RACF repository to locate certificate of signing authority.

- You have learned that an individual certificate and individual private key provide the most granular security. You see such a scenario in Example A.
  - ABCRING1 is owned by Server1 and is populated with the CA Cert that signed the individual Server Certificate. The individual Server Certificate for Server1 resides on this keyring.
    - An individual Server Certificate MUST BE OWNED BY THE PROCESS that invokes it!
    - The keyring can be owned by any user. It just happens to be owned by Server1 in this example.
  - ABCRING2 is not owned by Server2, but this ring is populated with the CA Cert that signed the individual Server Certificate. The individual Server Certificate for Server2 resides on this keyring.
    - An individual Server Certificate MUST BE OWNED BY THE PROCESS that invokes it!
    - Now you see that ADMIN owns the keyring, but Server2 owns the Server Certificate. When Server2 needs to point to its keyring, it needs to specify that the owner is ADMIN:
      - KEYRING ADMIN/ABCRING2.
- In Scenario B both of the servers are sharing a certificate and a private key. This is not the most granular security implementation, but it may be safe enough within an Intranet as long as other compliance mandates -- like PCI -- do not prohibit you from setting up a scenario like this.
  - Either server -- Server1 or Server2 -- may use the SITE Certificate for authentication; either server may use the same private key. (This might have been a client ring, in which case, any client can use that key.)
  - When the servers point to their keyring in their implementation definitions, they can simply use a definition like:
    - KEYRING ADMIN/ABCRINGX (if the keyring is owned by ADMIN and by neither of the servers.)
    - Alternatives to this are as follows:
      - KEYRING ABCRINGX for Server1 (if Server1 owns the keyring), or
      - KEYRING ADMIN/ABCRINGX (if Server1 does not own the keyring).
- Each RACF user ID is associated with a virtual key ring. The most common type is the CERTAUTH virtual key ring, which is used when an application validates the certificates of others but has no need for its own certificate and private key.
- For applications using System SSL, such as z/OS FTP, or other middleware programs that read RACF key rings through the R_datalib callable service, a virtual key ring can be specified in place of a real key ring, whenever a real key ring is expected. To include virtual key rings, the application user specifies an asterisk (*) for the key ring name along with the ring owner's user ID using the form ring-owner/*. Client authentication is not required and the virtual key ring is used only to authenticate the FTP server.
- If using a SAF repository other than RACF, note that ACF2 appends a suffix of ".KEYRING" to the keyring name. Therefore, a keyring that you think is called "MYKEYRING" must be referenced as "MYKEYRING.KEYRING" for an ACF2 repository.

RACF Commands



Authority Required for RACF Functions

Table 23. Authority required to generate a certificate

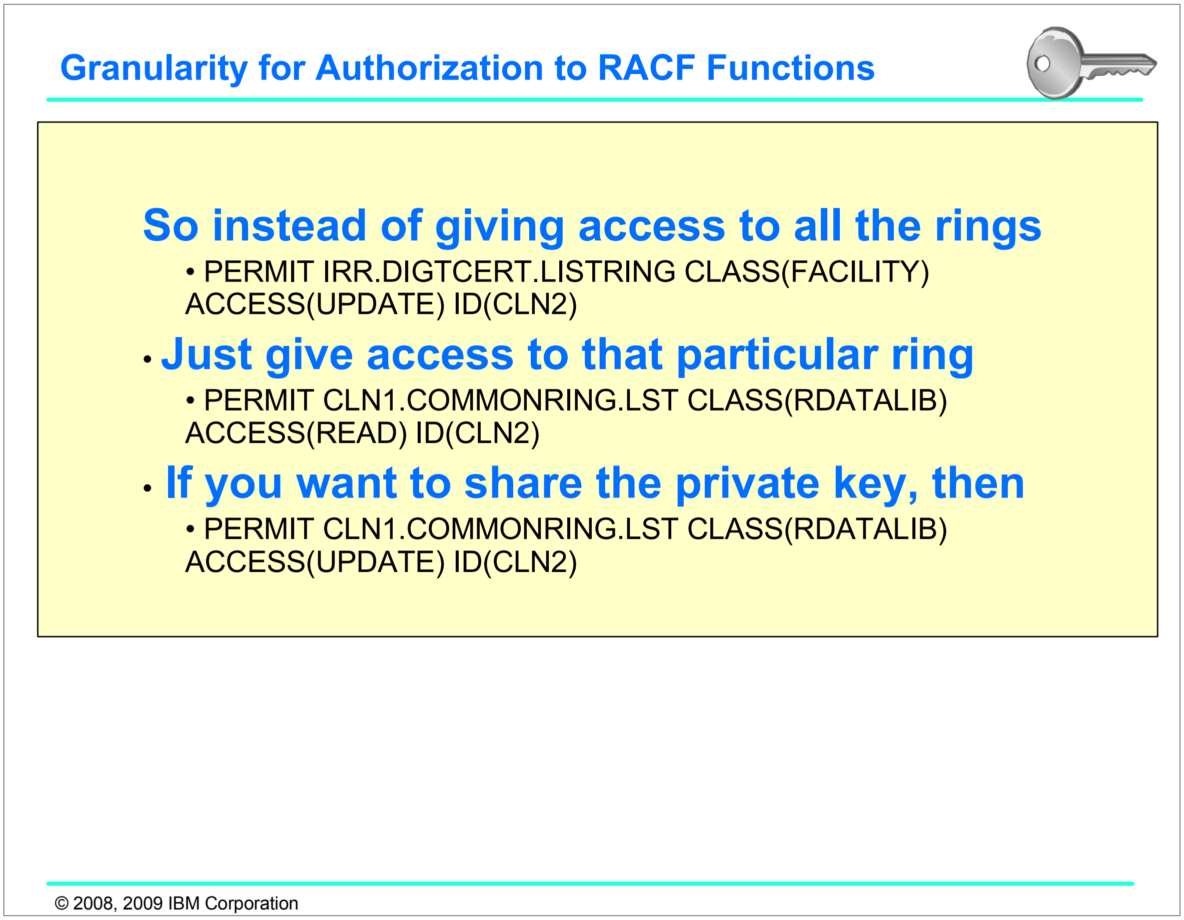
SIGNWITH	One's own certificate	Someone else's certificate	SITE or CERTAUTH certificate
SIGNWITH one's own certificate	READ authority to IRR.DIGTCERT.ADD and READ authority to IRR.DIGTCERT.GENCERT	UPDATE authority to IRR.DIGTCERT.ADD and READ authority to IRR.DIGTCERT.GENCERT	CONTROL authority to IRR.DIGTCERT.ADD and READ authority to IRR.DIGTCERT.GENCERT
SIGNWITH a SITE or CERTAUTH certificate	READ authority to IRR.DIGTCERT.ADD and CONTROL authority to IRR.DIGTCERT.GENCERT	UPDATE authority to IRR.DIGTCERT.ADD and CONTROL authority to IRR.DIGTCERT.GENCERT	CONTROL authority to IRR.DIGTCERT.ADD and CONTROL authority to IRR.DIGTCERT.GENCERT
SIGNWITH not specified	READ authority to IRR.DIGTCERT.ADD and READ authority to IRR.DIGTCERT.GENCERT	UPDATE authority to IRR.DIGTCERT.ADD and UPDATE authority to IRR.DIGTCERT.GENCERT	CONTROL authority to IRR.DIGTCERT.ADD and CONTROL authority to IRR.DIGTCERT.GENCERT

Table 20. Authority required for RACDCERT functions

Function	READ	UPDATE	CONTROL
ADD	Add a certificate to one's own user ID.	Add a certificate for someone else.	Add a CERTAUTH or SITE certificate.
ADDTOKEN	See Note 1.		
ADDRING	Create a key ring for one's own user ID.	Create a key ring for another user ID.	—
LIST	List one's own certificate.	List the certificate of someone else.	List CERTAUTH or SITE certificates.
LISTMAP	List mapping information associated with one's own user ID	List mapping information associated with another user ID or MULTIID.	—
LISTRING	See one's own key ring.	See the key ring of someone else.	—

- **Two z/OS Security Services manuals are critical to RACF functions and syntax:**
  - **z/OS Security Server RACF Security Administrator's Guide (SA22-7683-11)**
    - **Chapter 21: RACF and Security Certificates**
  - **z/OS V1R9 Security Server RACF Command Language Reference (SA22-7687-11)**
    - **Chapter 5: RACF Command Syntax**

- In Chapter 5 of the z/OS Security Server RACF Command Language Reference you will find many variations of the RACDCERT command. You will see examples of these commands in the JCL we show you with which we have produced keyrings and certificates. You will also find several Facility Classes for Certificates and Keyrings. The authorities required (READ, UPDATE, CONTROL) to access these classes or functions are described in the Command Language Reference.
- Effective use of RACDCERT requires that its privileges be carefully controlled. However, end-users and application administrators should be allowed some flexibility in defining their security characteristics. These guidelines might prove useful.
  - The ability to add certificate authorities and site certificates should be allowed to only a small set of trusted people.
  - End users should be permitted to add, delete, and modify the contents of their own key rings and add, delete, and alter their own certificates.
  - Help desk personnel should be allowed the ability to list certificates and rings.
- Key rings are associated with specific RACF user IDs. A RACF user ID can have more than one key ring. Key rings are managed using the RACDCERT command, and are maintained in the general resource class called DIGTRING.
- **
- Authority to the IRR.DIGTCERT.function resource in the FACILITY class allows a user to issue the RACDCERT command. To issue the RACDCERT command, users must have one of the following authorities:
- The SPECIAL attribute
- Sufficient authority to resource IRR.DIGTCERT.function in the FACILITY class.
  - – READ access to IRR.DIGTCERT.function to issue the RACDCERT command for themselves.
  - – UPDATE access to IRR.DIGTCERT.function to issue the RACDCERT command for others.
  - – CONTROL access to IRR.DIGTCERT.function to issue the RACDCERT command for SITE and CERTAUTH certificates. (This authority also has other uses.)
- ***
- Authority required for the GENCERT function [Table 23 above]: The GENCERT keyword allows a certificate to be generated and signed. Effective controls on the user ID that is being associated with the certificate and what certificate is being used to sign the generated certificate are essential.
- Notes [on Table 20 above]: With the following exceptions, the access levels listed (READ, UPDATE and CONTROL) in this table [Table 20 above] are based on authority to the resource IRR.DIGTCERT.function in the FACILITY class.
  - 1. ADDTOKEN—Controlled by ICSF using resources in the CRYPTOZ class. (No authority to FACILITY class resources is required.)
  - 2. BIND—Controlled by ICSF using resources in the CRYPTOZ class, and by both IRR.DIGTCERT.BIND and IRR.DIGTCERT.ADD.
  - 3. CHECKCERT—Controlled by IRR.DIGTCERT.LIST
  - 4. DELTOKEN—Controlled by ICSF using resources in the CRYPTOZ class. (No authority to FACILITY class resources is required.)
  - 5. EXPORT—Controlled by either IRR.DIGTCERT.EXPORT or IRR.DIGTCERT.EXPORTKEY.
  - 6. GENCERT—Controlled by both IRR.DIGTCERT.ADD and IRR.DIGTCERT.GENCERT.
  - 7. IMPORT—Controlled by ICSF using resources in the CRYPTOZ class, and by IRR.DIGTCERT.ADD.
  - 8. LISTTOKEN—Controlled by ICSF using resources in the CRYPTOZ class, and by IRR.DIGTCERT.LIST.
  - 9. UNBIND—Controlled by ICSF using resources in the CRYPTOZ class. (No authority to FACILITY class resources is required.)



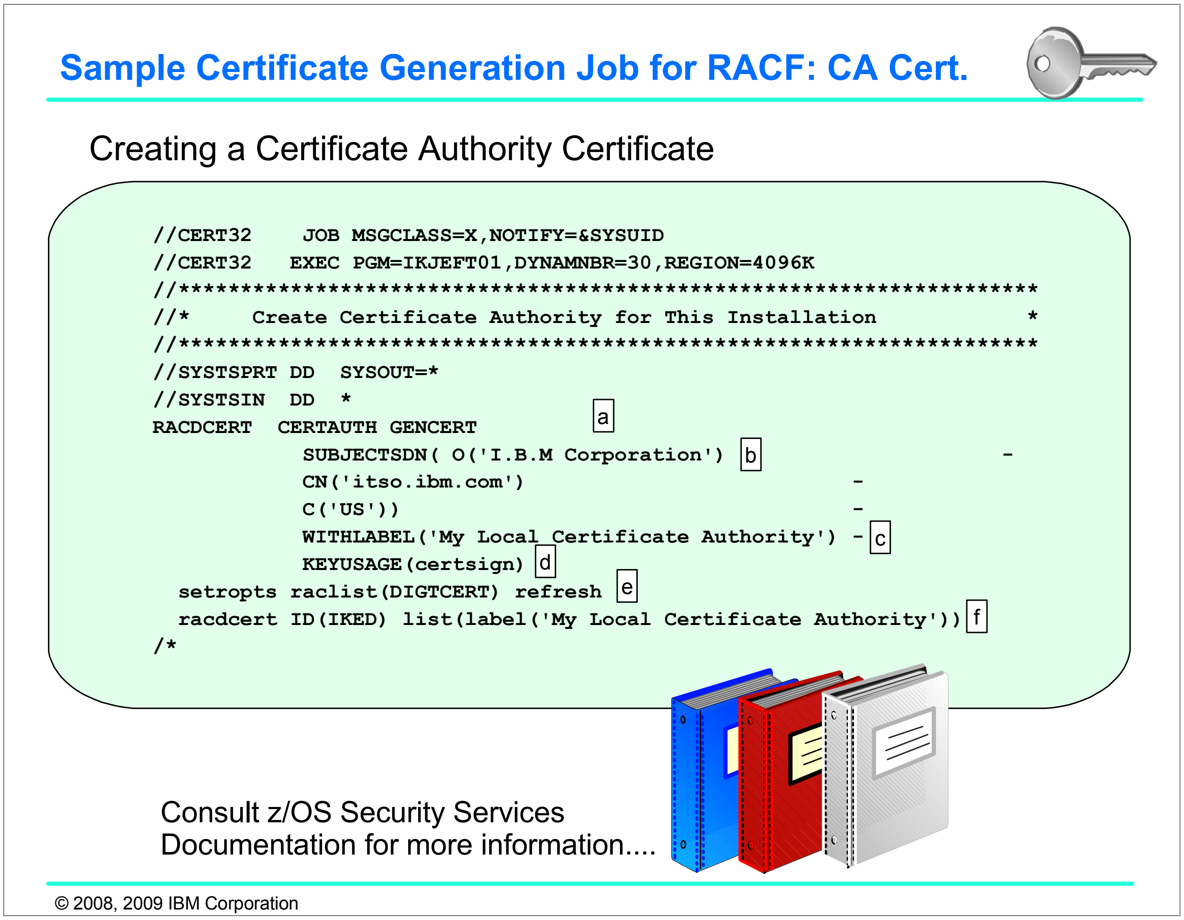
Granularity for Authorization to RACF Functions

So instead of giving access to all the rings

- PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ACCESS(UPDATE) ID(CLN2)

- **Just give access to that particular ring**
  - PERMIT CLN1.COMMONRING.LST CLASS(RDATALIB) ACCESS(READ) ID(CLN2)
- **If you want to share the private key, then**
  - PERMIT CLN1.COMMONRING.LST CLASS(RDATALIB) ACCESS(UPDATE) ID(CLN2)

- V1R9 provides another solution – Granular access control on Key Ring
  - Access is based on a profile of a specific key ring in a new class called RDATALIB
  - The class RDATALIB must be RACLISTed
  - A resource with the format <ringOwner>.<ringName>.LST is used to provide access control to a specific key ring on
- R_datalib READ functions
  - This new support also allows the retrieval of another person's private key
  - So instead of giving access to all the rings
    - PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ACCESS(UPDATE) ID(CLN2)
  - Just give access to that particular ring
    - PERMIT CLN1.COMMONRING.LST CLASS(RDATALIB) ACCESS(READ) ID(CLN2)
  - If you want to share the private key, then
    - PERMIT CLN1.COMMONRING.LST CLASS(RDATALIB) ACCESS(UPDATE) ID(CLN2)



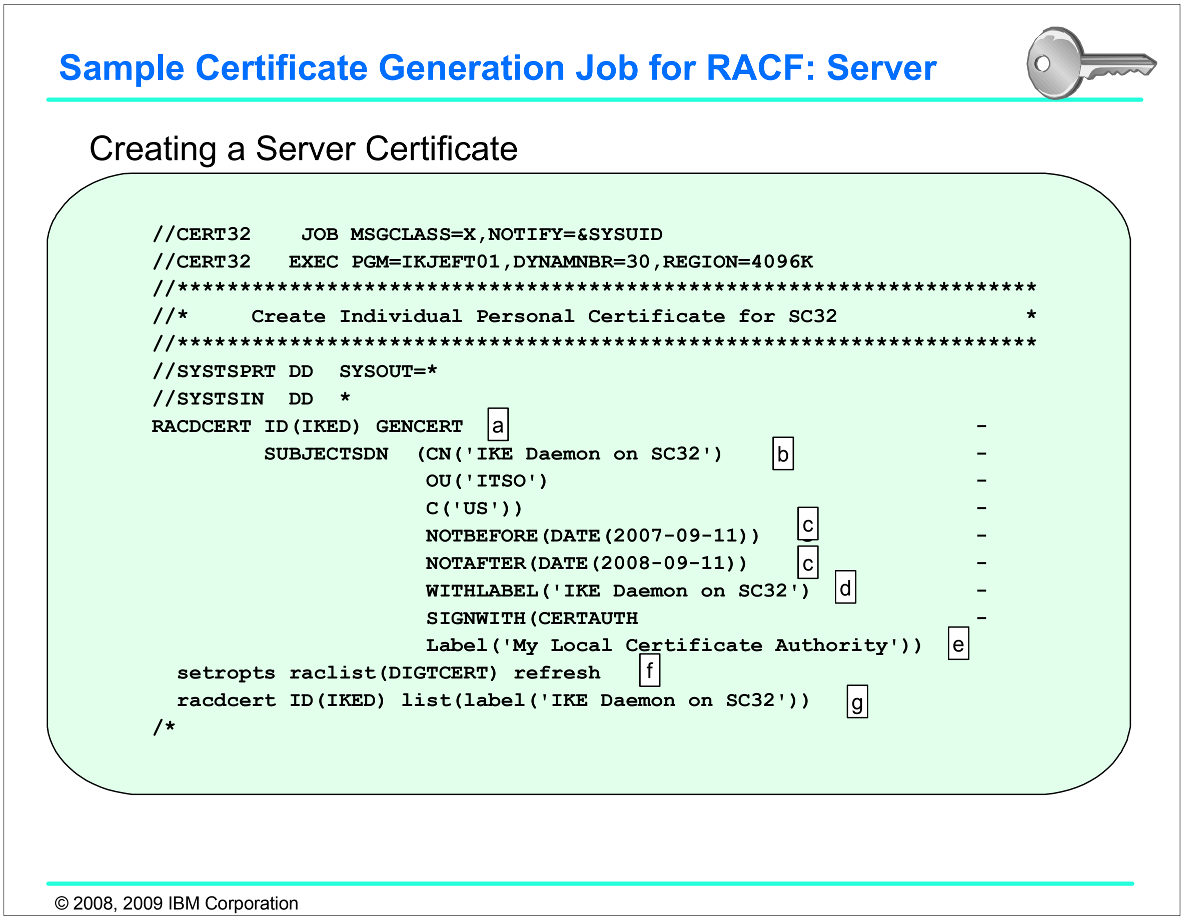
Sample Certificate Generation Job for RACF: CA Cert.

Creating a Certificate Authority Certificate

```
//CERT32    JOB MSGCLASS=X,NOTIFY=&SYSUID
//CERT32   EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//**********************************************************************
//*     Create Certificate Authority for This Installation            *
//**********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT  CERTAUTH GENCERT         a
            SUBJECTSDN( O('I.B.M Corporation') b                     -
            CN('itso.ibm.com')                           -
            C('US'))                                     -
            WITHLABEL('My Local Certificate Authority') - c
            KEYUSAGE(certsign) d
  setropts raclist(DIGTCERT) refresh e
  racdcert ID(IKED) list(label('My Local Certificate Authority')) f
/*
```

Consult z/OS Security Services
Documentation for more information....

A. The CERTAUTH parameter identifies this certificate that is being generated (GENCERT) as a Certificate Authority (CA) certificate.
B. SUBJECTSDN identifies several components that comprise the x.509 Distinguished Name (DN) of the certificate owner or holder. Each DN should be unique at least within a RACF database; it should also be unique across the world, since this DN is used to distinguish identities in many cases. We have utilized here only two components of the DN: CN, and C.
C. The RACF database requires a label for organizing the certificates within a RACF database. The label must be unique.
D. This definition tells the GENCERT process that this is a CA certificate for which the key will be created.
E. The setropts command refreshes the DIGTCERT class so that the changes are made immediately known in the running RACF environment and operating system.
F. The racdcert command allows us to verify that our certificate has been properly created; it displays the certificate with its attributes.

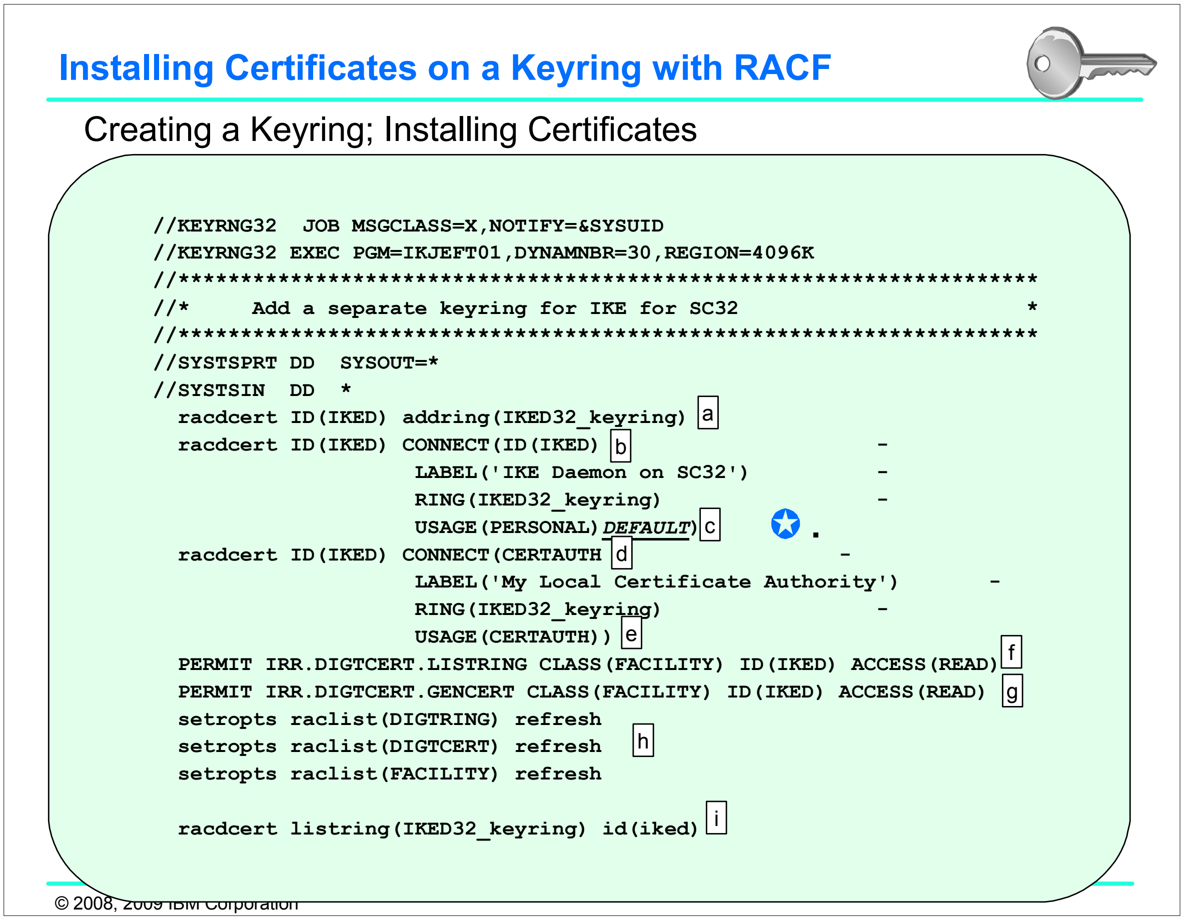


Sample Certificate Generation Job for RACF: Server

Creating a Server Certificate

```
//CERT32    JOB MSGCLASS=X,NOTIFY=&SYSUID
//CERT32   EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//*********************************************************************
//*     Create Individual Personal Certificate for SC32               *
//*********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT ID(IKED) GENCERT   a                                      -
         SUBJECTSDN  (CN('IKE Daemon on SC32')     b               -
                      OU('ITSO')                                   -
                      C('US'))                                     -
                      NOTBEFORE(DATE(2007-09-11))    c             -
                      NOTAFTER(DATE(2008-09-11))     c             -
                      WITHLABEL('IKE Daemon on SC32')   d          -
                      SIGNWITH(CERTAUTH                            -
                      Label('My Local Certificate Authority'))    e
  setropts raclist(DIGTCERT) refresh    f
  racdcert ID(IKED) list(label('IKE Daemon on SC32'))    g
/*
```

A. The ID parameter identifies this certificate that is being generated (GENCERT) as a personal user certificate. (It is not a CA or SITE certificate.)
B. SUBJECTSDN identifies several components that comprise the x.509 Distinguished Name (DN) of the certificate owner or holder. Each DN should be unique at least within a RACF database; it should also be unique across the world, since this DN is used to distinguish identities in many cases. We have utilized only three components of the DN for SC32: CN, OU, and C.
C. These parameters set a timeframe for the certificate’s validity. The default timeframe is only one year. It is common in a production environment to use a much longer timeframe than we have used.
D. The RACF database requires a label for organizing the certificates within a RACF database. The label must be unique.
E. This definition tells the GENCERT process which CA should be the signing authority for the user’s personal certificate.
F. The setropts command refreshes the DIGTCERT class so that the changes are made immediately known in the running RACF environment and operating system.
G. The racdcert command allows us to verify that our certificate has been properly created; it displays the certificate with its attributes.



Installing Certificates on a Keyring with RACF

Creating a Keyring; Installing Certificates

```
//KEYRNG32  JOB MSGCLASS=X,NOTIFY=&SYSUID
//KEYRNG32 EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//***********************************************************************
//*     Add a separate keyring for IKE for SC32                         *
//***********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  racdcert ID(IKED) addring(IKED32_keyring) a
  racdcert ID(IKED) CONNECT(ID(IKED) b                     -
                     LABEL('IKE Daemon on SC32')           -
                     RING(IKED32_keyring)                  -
                     USAGE(PERSONAL) DEFAULT) c
  racdcert ID(IKED) CONNECT(CERTAUTH d                 -
                     LABEL('My Local Certificate Authority')       -
                     RING(IKED32_keyring)                  -
                     USAGE(CERTAUTH)) e
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(IKED) ACCESS(READ) f
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(IKED) ACCESS(READ) g
  setropts raclist(DIGTRING) refresh
  setropts raclist(DIGTCERT) refresh   h
  setropts raclist(FACILITY) refresh

  racdcert listring(IKED32_keyring) id(iked) i
```

© 2008, 2009 IBM Corporation

A. This adds a keyring to the RACF database.
B. This connects a certificate that is already in the RACF database to the keyring that was just created.
C. USAGE(PERSONAL) indicates that this certificate may be used by a single entity (e.g., server) only. It cannot be shared. If an application program cannot or does not indicate a specific PERSONAL certificate, then it must use the DEFAULT certificate on the ring. (More about this later.)
D. This connects the installation's CA certificate to the keyring.
E. USAGE(CERTAUTH) indicates again that this is a CA certificate. CA certificates are always found by means of their labels and they do not need to assume a DEFAULT role the way a PERSONAL certificate does.
F. This protects the keyring with RACF access control commands.
G. Protect the private key and permit the user IDs of each server to access it. The private key is represented by the facility named IRR.DIGTCERT.GENCERT. In this case, userid of IKED owns the certificate and therefore an ACCESS(READ) is sufficient for GENCERT. If this had been a Site Certificate (which has no specific owner), then IKED would have needed ACCESS(CONTROL).
H. This refreshes classes in MVS storage, again protecting the contents of the various classes of objects.
I. This command lists the contents of the keyring with the label identified.

Certificate and Key Management

Key Management in PCI

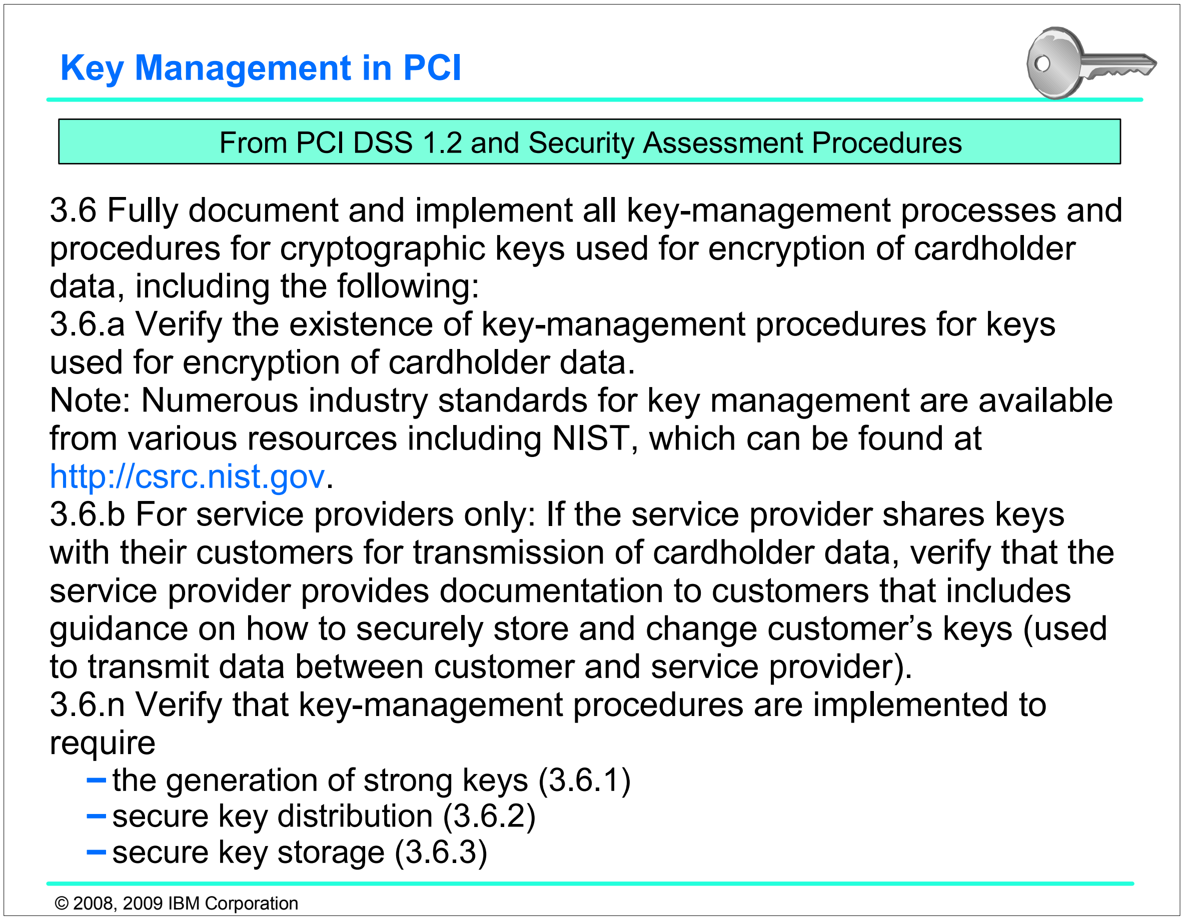
From PCI DSS 1.2 and Security Assessment Procedures

3.4.1.b Verify that cryptographic keys are stored securely (for example, stored on removable media that is adequately protected with strong access controls).

3.5 Verify processes to protect keys used for encryption of cardholder data against disclosure and misuse by performing the following:

3.5.1 Examine user access lists to verify that access to keys is restricted to very few custodians.

3.5.2 Examine system configuration files to verify that keys are stored in encrypted format and that key-encrypting keys are stored separately from data-encrypting keys.

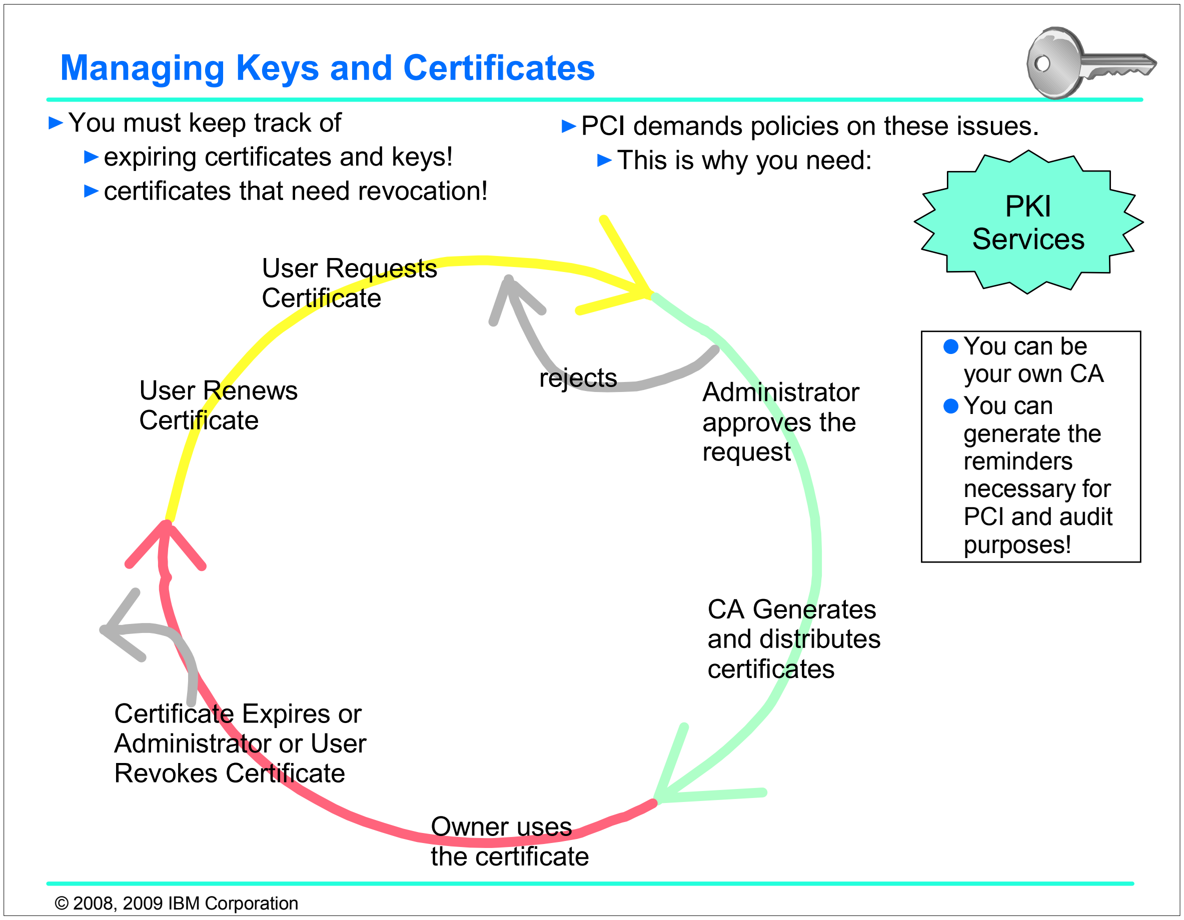


Key Management in PCI

From PCI DSS 1.2 and Security Assessment Procedures

3.6 Fully document and implement all key-management processes and procedures for cryptographic keys used for encryption of cardholder data, including the following:
3.6.a Verify the existence of key-management procedures for keys used for encryption of cardholder data.
Note: Numerous industry standards for key management are available from various resources including NIST, which can be found at http://csrc.nist.gov.
3.6.b For service providers only: If the service provider shares keys with their customers for transmission of cardholder data, verify that the service provider provides documentation to customers that includes guidance on how to securely store and change customer's keys (used to transmit data between customer and service provider).
3.6.n Verify that key-management procedures are implemented to require

- the generation of strong keys (3.6.1)
- secure key distribution (3.6.2)
- secure key storage (3.6.3)



Managing Keys and Certificates

- ▶ You must keep track of
  - ▶ expiring certificates and keys!
  - ▶ certificates that need revocation!
- ▶ PCI demands policies on these issues.
  - ▶ This is why you need:

PKI Services

- ● You can be your own CA
- ● You can generate the reminders necessary for PCI and audit purposes!

User Requests Certificate

Administrator approves the request

rejects

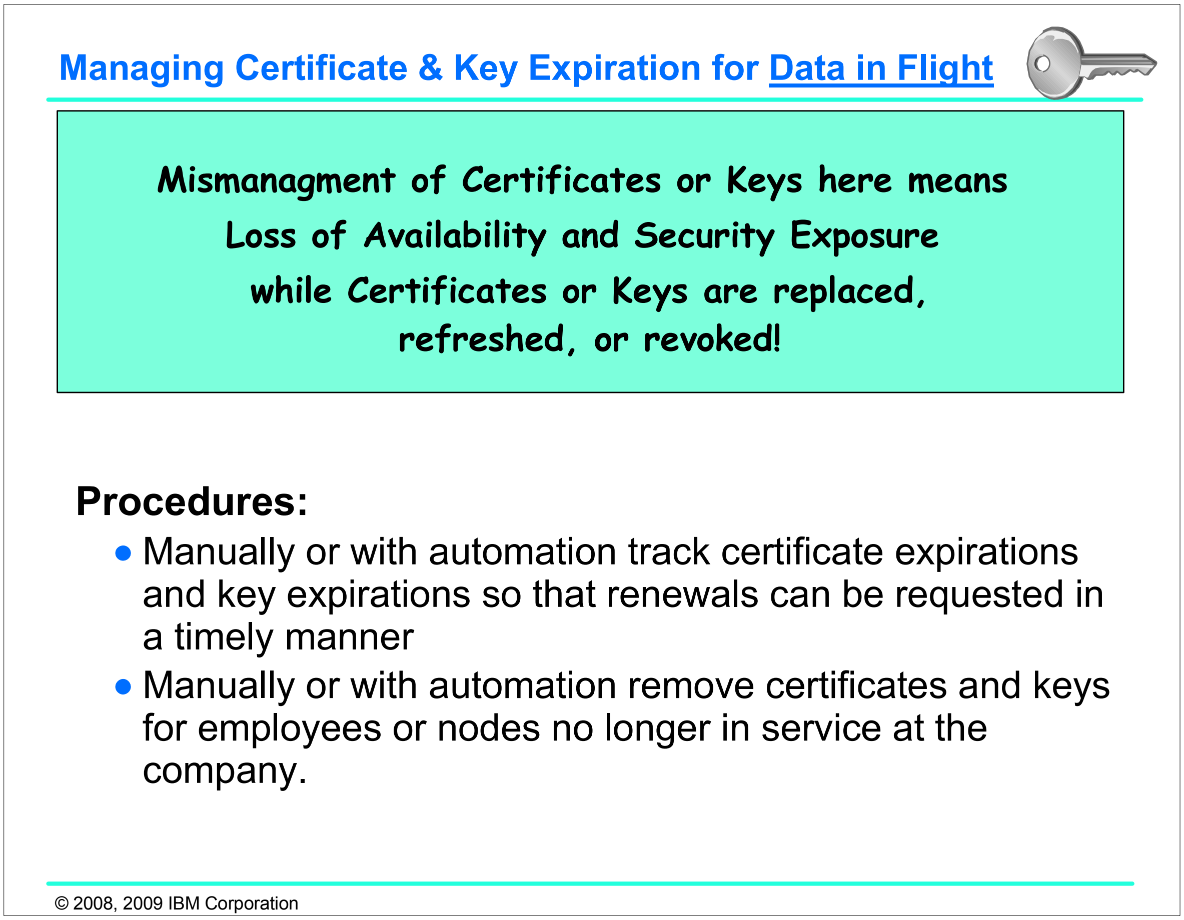
CA Generates and distributes certificates

Owner uses the certificate

Certificate Expires or Administrator or User Revokes Certificate

User Renews Certificate

- PKI Services
  - A component on z/OS since V1R3
  - Closely tied to RACF
  - The CA cert must be installed in RACF’s key ring
  - Authority checking goes through RACF’s callable service
  - Supports more functions than RACDCERT
  - Full certificate life cycle management: request, create, renew, revoke
  - Generation and administration of certificates via customizable web pages
  - Support automatic or administrator approval process
  - Support multiple revocation checking mechanisms
  - Certificate Revocation List (CRL)
  - Online Certificate Status Protocol (OCSP)
  - Certificates and CRLs can be posted to LDAP
  - Provides email notification
    - to notify end user for completed certificate request and expiration warnings
    - to notify administrator for pending requests
  - Provides Trust Policy Plug-in for certificate validation
  - Manual - "PKI Services Guide and Reference"
- Advantages:
  - Not a priced product. Licensed with z/OS. An alternative to purchasing third party certificates
  - IdenTrust™ compliant
    - ensures adherence to a common standard to provide a solid foundation for trust between financial institutions and their customers
  - Relatively low mips to drive thousands of certificates
  - Leverage existing z/OS skills and resources
  - Cost efficient for banks, government agencies to host Digital Certificate management
  - Run in separate z/OS partitions (integrity of zSeries® LPARs)
  - Scalable (Sysplex exploitation)
  - Secure the CA private key with zSeries cryptography
- PREREQUISITE PRODUCTS:
  - RACF (or equivalent) For storing PKI CA certificate and for authorization
  - IBM z/OS HTTP Server For web page interface
  - LDAP Directory (z/OS or other platforms) For publishing issued certificates and CRLs and for email notification
  - ICSF (optional) For more secure CA private key
- z /OS Communications Server (optional) For email notification



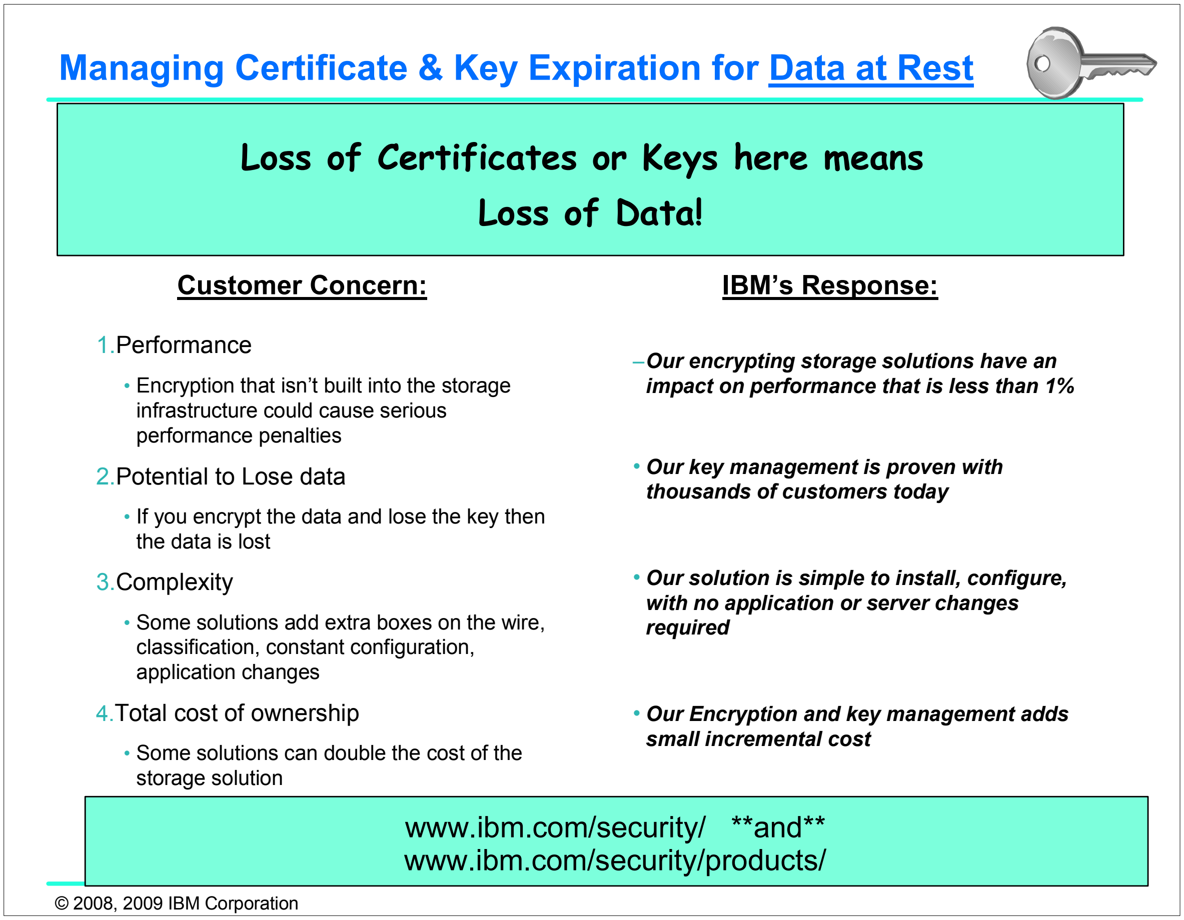
Managing Certificate & Key Expiration for Data in Flight

Mismanagment of Certificates or Keys here means Loss of Availability and Security Exposure while Certificates or Keys are replaced, refreshed, or revoked!

Procedures:

- Manually or with automation track certificate expirations and key expirations so that renewals can be requested in a timely manner
- Manually or with automation remove certificates and keys for employees or nodes no longer in service at the company.

- For data in flight, you don't lose access to your data if you lose or mismanage your keys. However, key expiration or certificate expiration mean that you lose some availability while you refresh keys or certificates so that you can continue sending data in flight. So, with regard to data in flight, key and certificate management only applies from the perspective of keeping your certificates up to date.



Managing Certificate & Key Expiration for Data at Rest

Loss of Certificates or Keys here means Loss of Data!

Customer Concern:

1. Performance
   - Encryption that isn't built into the storage infrastructure could cause serious performance penalties
2. Potential to Lose data
   - If you encrypt the data and lose the key then the data is lost
3. Complexity
   - Some solutions add extra boxes on the wire, classification, constant configuration, application changes
4. Total cost of ownership
   - Some solutions can double the cost of the storage solution

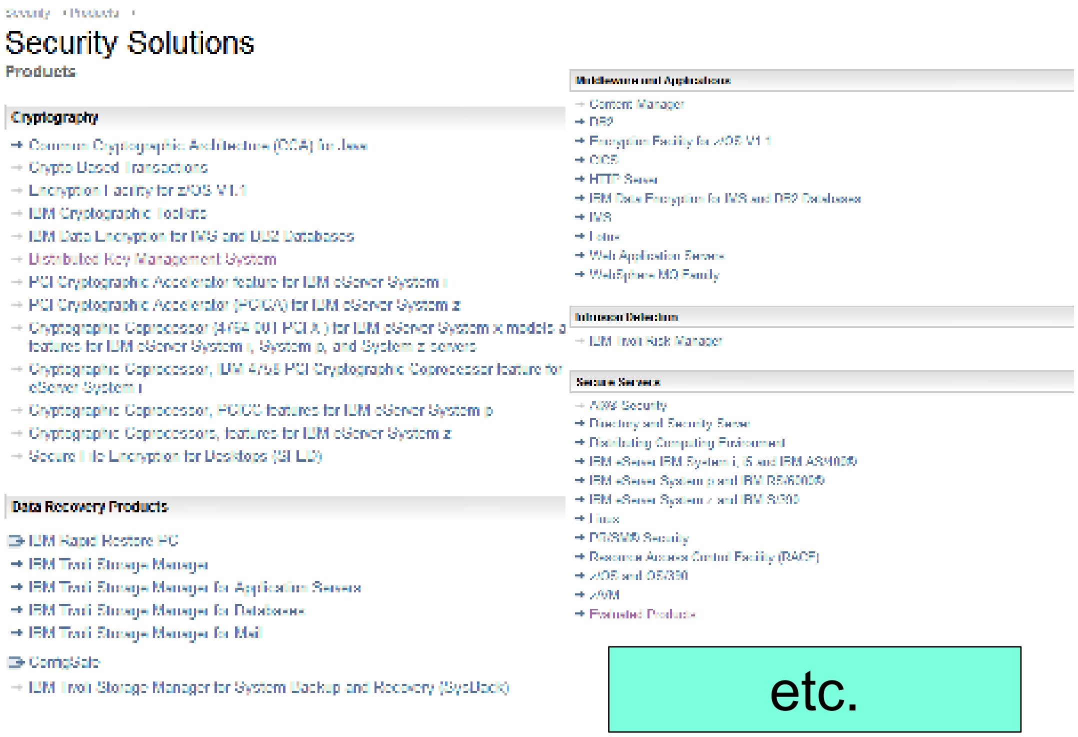
IBM's Response:

- *Our encrypting storage solutions have an impact on performance that is less than 1%*
- *Our key management is proven with thousands of customers today*
- *Our solution is simple to install, configure, with no application or server changes required*
- *Our Encryption and key management adds small incremental cost*

www.ibm.com/security/ **and**
www.ibm.com/security/products/

- Because expired or inactive data accounts for a large percentage of total data storage, encryption and key management solutions need to support data that an organization might use in the future as well as securely dispose of data that is no longer needed. Furthermore, organizations may have data retention policies, which, if implemented using encryption and key lifecycle management, can simplify and make more predictable the erasure of data.
- “I can’t afford to lose data because of poorly managed encryption. Can you assure me that my data will be available?”
- One does not have to be an expert in cryptography to understand that lost keys required to decrypt the data leads to unrecoverable data. Cryptographic key management will therefore be one of the storage manager’s top concerns. Beyond this fundamental issue, however, are requirements for assuring availability and performance according to service levels and policy demands.

List of IBM Products for Security



Security > Products >

Security Solutions

Products

Cryptography

- → Common Cryptographic Architecture (CCA) for Java
- → Crypto Based Transactions
- → Encryption Facility for z/OS V1.1
- → IBM Cryptographic Toolkits
- → IBM Data Encryption for IMS and DB2 Databases
- → Distributed Key Management System
- → PCI Cryptographic Accelerator feature for IBM eServer System i
- → PCI Cryptographic Accelerator (PCICA) for IBM eServer System z
- → Cryptographic Coprocessor (4764-001 PCI-X) for IBM eServer System x models a features for IBM eServer System i, System p, and System z servers
- → Cryptographic Coprocessor, IBM 4758 PCI Cryptographic Coprocessor feature for eServer System i
- → Cryptographic Coprocessor, PCICC features for IBM eServer System p
- → Cryptographic Coprocessors, features for IBM eServer System z
- → Secure File Encryption for Desktops (SFED)

Data Recovery Products

- ⇒ IBM Rapid Restore PC
- → IBM Tivoli Storage Manager
- → IBM Tivoli Storage Manager for Application Servers
- → IBM Tivoli Storage Manager for Databases
- → IBM Tivoli Storage Manager for Mail
- ⇒ CompuSafe
- → IBM Tivoli Storage Manager for System Backup and Recovery (SysBack)

Middleware and Applications

- → Content Manager
- → DB2
- → Encryption Facility for z/OS V1.1
- → CICS
- → HTTP Server
- → IBM Data Encryption for IMS and DB2 Databases
- → IMS
- → Lotus
- → Web Application Servers
- → WebSphere MQ Family

Intrusion Detection

- → IBM Tivoli Risk Manager

Secure Servers

- → AIX Security
- → Directory and Security Server
- → Distributing Computing Environment
- → IBM eServer IBM System i, i5 and IBM AS/400®
- → IBM eServer System p and IBM RS/6000®
- → IBM eServer System z and IBM S/390
- → Linux
- → PR/SM® Security
- → Resource Access Control Facility (RACF)
- → z/OS and OS/390
- → z/VM
- → Evaluated Products

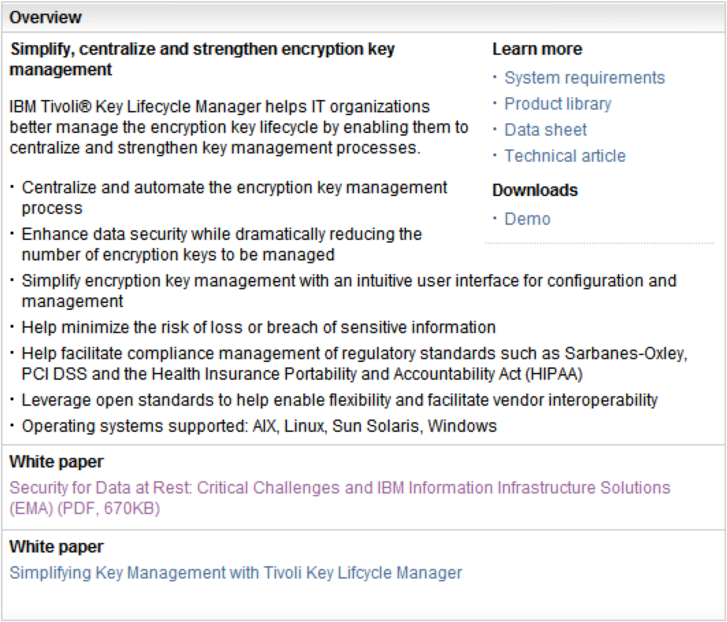
etc.

- This is a sample of the products you can find links to from URL .http://www..ibm.com/security/products/

Managing Keys for Data at Rest: TKLM

Software > Tivoli > Products > IBM Tivoli Key Lifecycle Manager >

Tivoli Key Lifecycle Manager



Overview

Simplify, centralize and strengthen encryption key management

IBM Tivoli® Key Lifecycle Manager helps IT organizations better manage the encryption key lifecycle by enabling them to centralize and strengthen key management processes.

- Centralize and automate the encryption key management process
- Enhance data security while dramatically reducing the number of encryption keys to be managed
- Simplify encryption key management with an intuitive user interface for configuration and management
- Help minimize the risk of loss or breach of sensitive information
- Help facilitate compliance management of regulatory standards such as Sarbanes-Oxley, PCI DSS and the Health Insurance Portability and Accountability Act (HIPAA)
- Leverage open standards to help enable flexibility and facilitate vendor interoperability
- Operating systems supported: AIX, Linux, Sun Solaris, Windows

Learn more

- System requirements
- Product library
- Data sheet
- Technical article

Downloads

- Demo

White paper

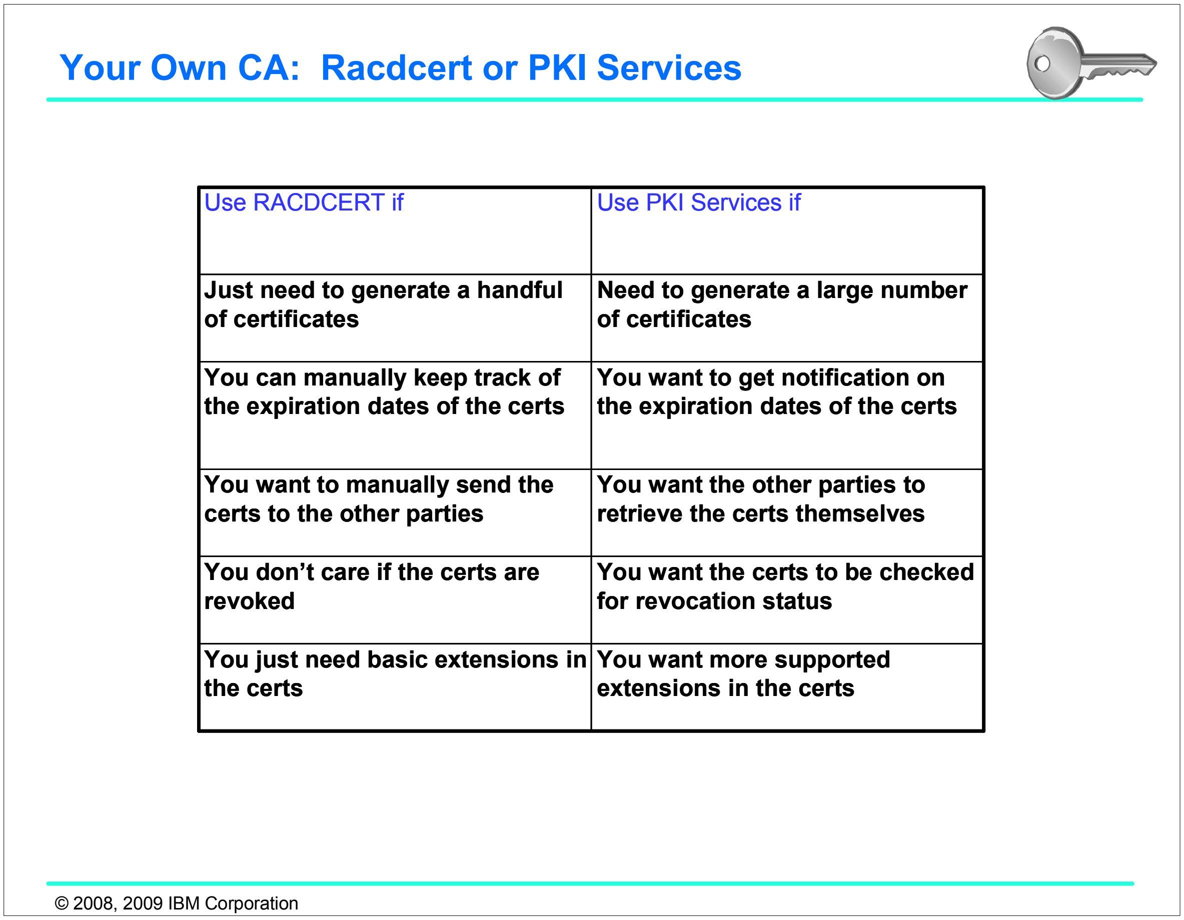
Security for Data at Rest: Critical Challenges and IBM Information Infrastructure Solutions (EMA) (PDF, 670KB)

White paper

Simplifying Key Management with Tivoli Key Lifcycle Manager

Evolved from
IBM Encryption Key Management (EKM)

- This is a description of Tivoli Key Lifecycle Manager, which points to two white papers: one about data at rest in general and one specifically about TKLM. The URL for this page is: http://www-01.ibm.com/software/tivoli/products/key-lifecycle-mgr/
- The URL for the white paper on Keys for Data at Rest is: ftp://submit.boulder.ibm.com/sales/ssi/sk/p0/5/r246447k43524l55/EMA_IBMSTORAGEDARSECURITY_WP.PDF
- The URL for the white paper on TKLM is: http://www.servicemanagementcenter.com/main/pages/IBMRBMS/SMRC/ShowCollateral.aspx?oid=39509

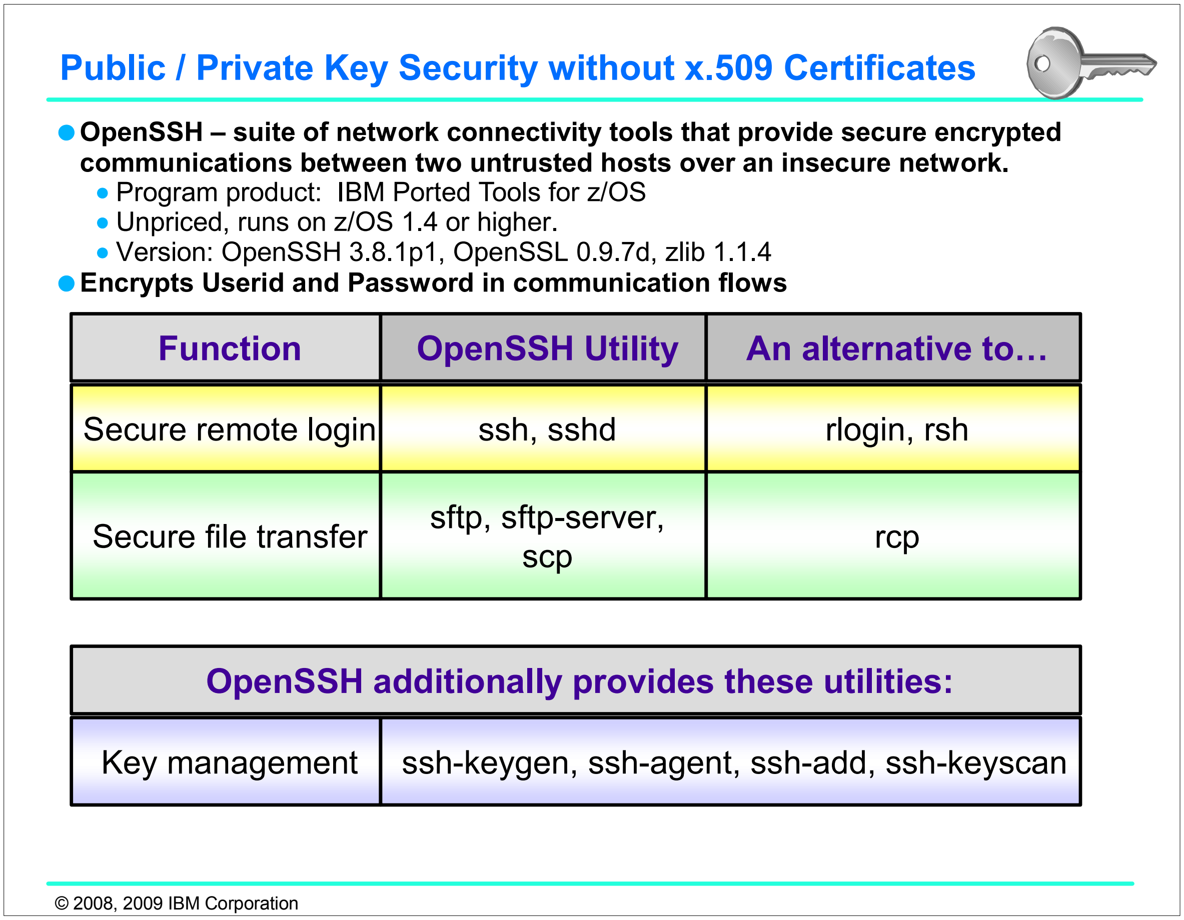


Your Own CA: Racdcert or PKI Services

Use RACDCERT if	Use PKI Services if
Just need to generate a handful of certificates	**Need to generate a large number of certificates**
You can manually keep track of the expiration dates of the certs	**You want to get notification on the expiration dates of the certs**
You want to manually send the certs to the other parties	**You want the other parties to retrieve the certs themselves**
You don’t care if the certs are revoked	**You want the certs to be checked for revocation status**
You just need basic extensions in the certs	**You want more supported extensions in the certs**

- Note: PKI Services does not have any function to manage the key ring. Ring management is provided by RACF.

OpenSSH Security



Public / Private Key Security without x.509 Certificates

- **OpenSSH – suite of network connectivity tools that provide secure encrypted communications between two untrusted hosts over an insecure network.**
  - Program product: IBM Ported Tools for z/OS
  - Unpriced, runs on z/OS 1.4 or higher.
  - Version: OpenSSH 3.8.1p1, OpenSSL 0.9.7d, zlib 1.1.4
- **Encrypts Userid and Password in communication flows**

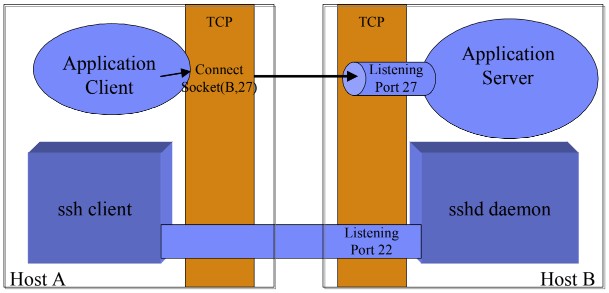
Function	OpenSSH Utility	An alternative to…
Secure remote login	ssh, sshd	rlogin, rsh
Secure file transfer	sftp, sftp-server, scp	rcp

OpenSSH additionally provides these utilities:	
Key management	ssh-keygen, ssh-agent, ssh-add, ssh-keyscan

OpenSSH Authentication & Encryption

OpenSSH – Without TCP/IP Port Forwarding

Direct client/server connection (no forwarding)



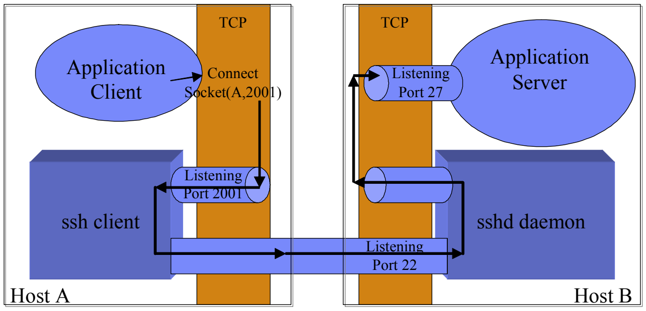
OpenSSH's key generation and management is separate from other key management provided by IBM.

- ssh-keygen
  - creates public/private key pairs
- ssh-agent
  - holds private keys in memory, saving you from retyping your passphrase repeatedly
- ssh-add
  - loads private keys into the agent
- ssh-keyscan
  - gathers SSH public host keys

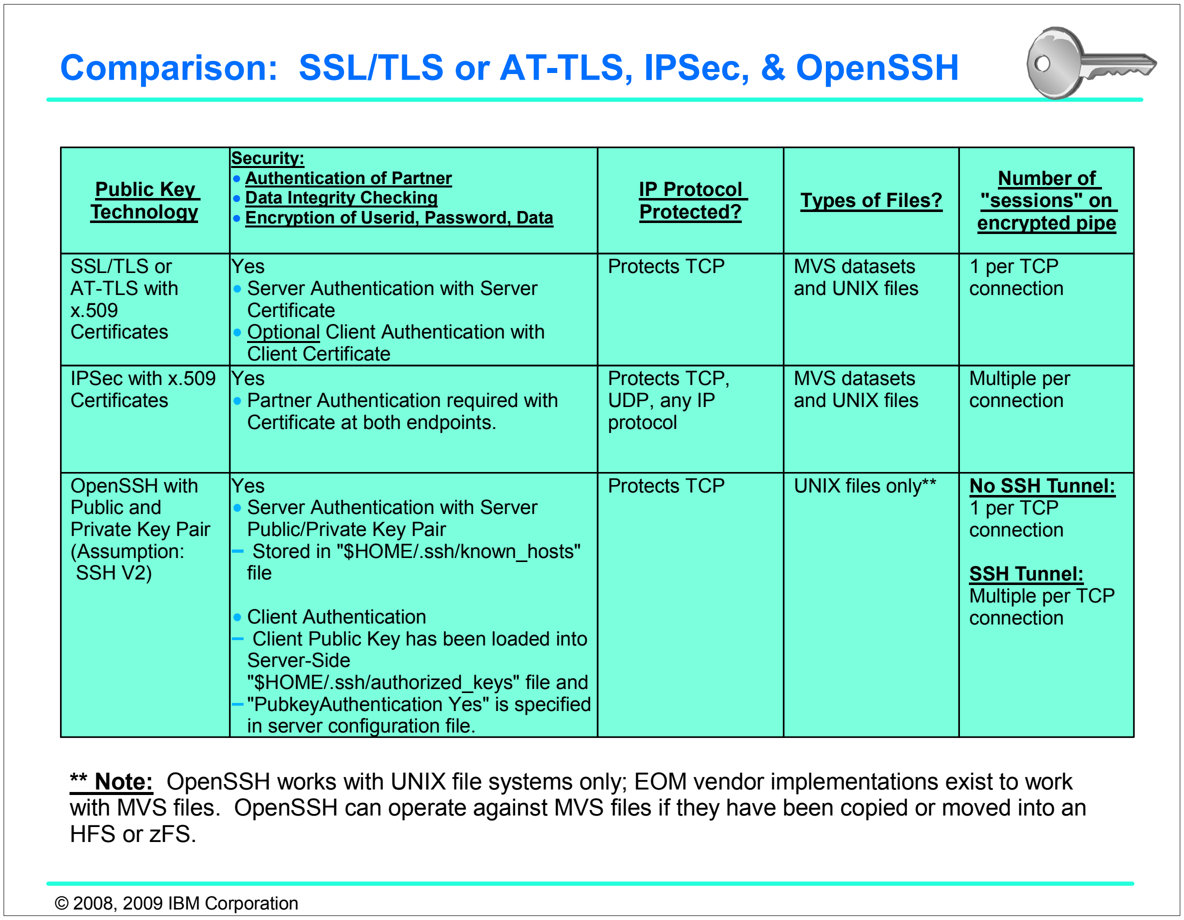
OpenSSH – TCP/IP Port Forwarding

ssh forwards the data through an SSH tunnel, sshd delivers to server

- Encrypts Userid and Password in communication flows
  - Directly, or
  - Through Port Forwarding or "Tunneling"



Appendix A: Protocol Comparisons (IPSec, SSL, SSH)

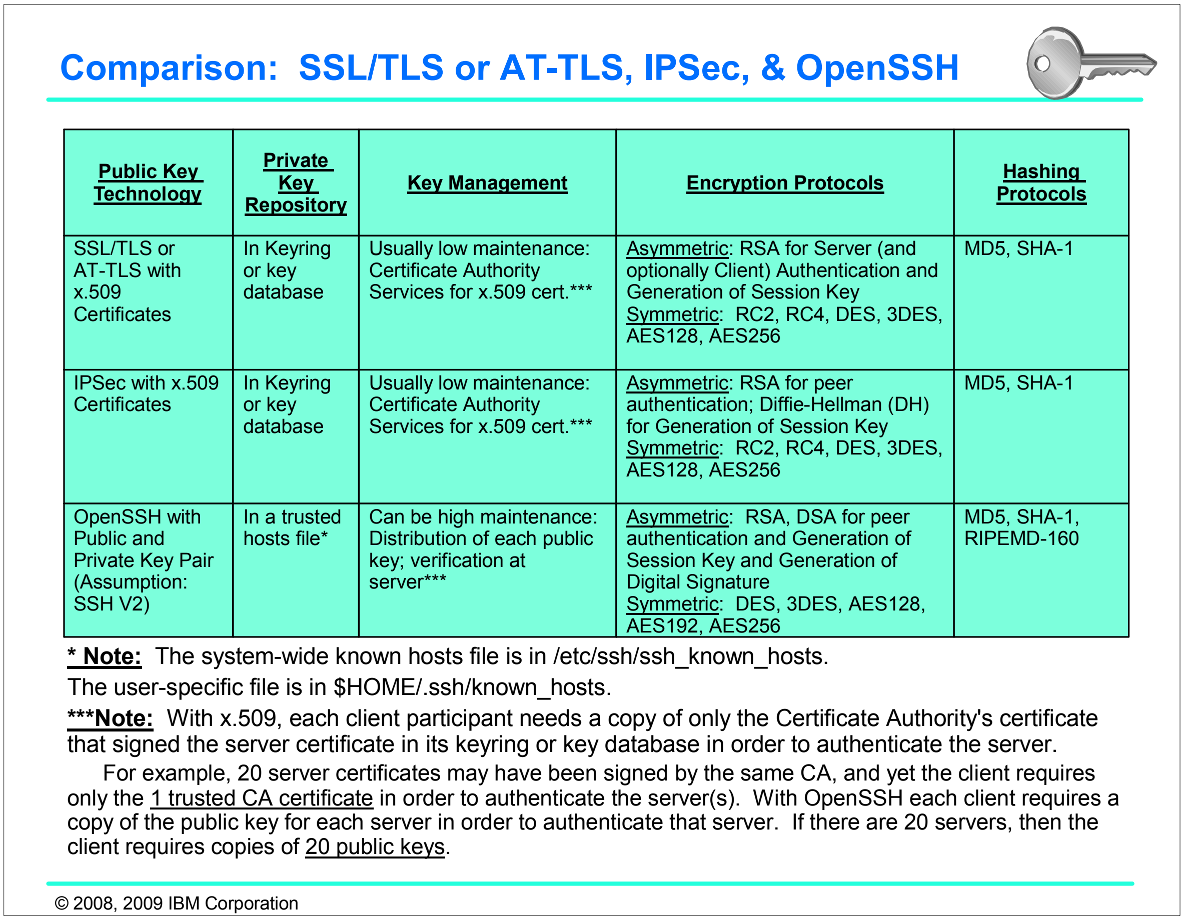


Comparison: SSL/TLS or AT-TLS, IPSec, & OpenSSH

Public Key Technology	**Security:** • **Authentication of Partner** • **Data Integrity Checking** • **Encryption of Userid, Password, Data**	**IP Protocol Protected?**	**Types of Files?**	**Number of "sessions" on encrypted pipe**
SSL/TLS or AT-TLS with x.509 Certificates	Yes • Server Authentication with Server Certificate • Optional Client Authentication with Client Certificate	Protects TCP	MVS datasets and UNIX files	1 per TCP connection
IPSec with x.509 Certificates	Yes • Partner Authentication required with Certificate at both endpoints.	Protects TCP, UDP, any IP protocol	MVS datasets and UNIX files	Multiple per connection
OpenSSH with Public and Private Key Pair (Assumption: SSH V2)	Yes • Server Authentication with Server Public/Private Key Pair – Stored in "$HOME/.ssh/known_hosts" file • Client Authentication – Client Public Key has been loaded into Server-Side "$HOME/.ssh/authorized_keys" file and – "PubkeyAuthentication Yes" is specified in server configuration file.	Protects TCP	UNIX files only**	**No SSH Tunnel:** 1 per TCP connection **SSH Tunnel:** Multiple per TCP connection

**** Note:** OpenSSH works with UNIX file systems only; EOM vendor implementations exist to work with MVS files. OpenSSH can operate against MVS files if they have been copied or moved into an HFS or zFS.

- OpenSSH from IBM z/OS: Program Product: IBM Ported Tools for z/OS
- `•` unpriced, runs on z/OS 1.4 and higher
- `• order from ShopzSeries, under “MVS: System Mgmt. and Security.”
- `• GA Version info: OpenSSH 3.5p1, OpenSSL 0.9.7b, zlib 1.1.4
- `• OA10315 version is: OpenSSH 3.8.1p1, OpenSSL 0.9.7d, zlib 1.1.4
- Base SSH: Uses Public Key Infrastructure for authentication and encryption,
- Authentication (both client and server) through:
- – Public key cryptography
- – Existing login passwords
- – Trusted hosts authentication
- • Data Privacy - through encryption
- • Data Integrity - guarantees data traveling over the network is unaltered
- • Authorization – regulates access control to accounts
- • Forwarding (a.k.a. tunneling) – encryption of other TCP/IP-based sessions
- BUT key management and distribution for large numbers of users are difficult because there is no concept of a Certificate Authority. As a result, the keys themselves or the trusted hosts file itself for each server needs to be distributed to the participants. In addition, the SSH option for eliminating
- Only for UNIX files with SFTP; only for UNIX shell with SSH (Tectia extensions allow usage on MVS files); only uses crypto card for generation of the keys
-
- Base SSL or TLS: Either Unix or MVS files with either TN3270 or FTPS;
- also uses Public Keys, but in addition relies on x.509 Certificates for authentication thus simplifying key management and distribution, especially if you use a well-known Certificate Authority;
- can store master key in hardware and can take advantage of crypto cards for handshakes and sometimes data encryption
-
-



Comparison: SSL/TLS or AT-TLS, IPSec, & OpenSSH

Public Key Technology	Private Key Repository	Key Management	Encryption Protocols	Hashing Protocols
SSL/TLS or AT-TLS with x.509 Certificates	In Keyring or key database	Usually low maintenance: Certificate Authority Services for x.509 cert.***	Asymmetric: RSA for Server (and optionally Client) Authentication and Generation of Session Key Symmetric: RC2, RC4, DES, 3DES, AES128, AES256	MD5, SHA-1
IPSec with x.509 Certificates	In Keyring or key database	Usually low maintenance: Certificate Authority Services for x.509 cert.***	Asymmetric: RSA for peer authentication; Diffie-Hellman (DH) for Generation of Session Key Symmetric: RC2, RC4, DES, 3DES, AES128, AES256	MD5, SHA-1
OpenSSH with Public and Private Key Pair (Assumption: SSH V2)	In a trusted hosts file*	Can be high maintenance: Distribution of each public key; verification at server***	Asymmetric: RSA, DSA for peer authentication and Generation of Session Key and Generation of Digital Signature Symmetric: DES, 3DES, AES128, AES192, AES256	MD5, SHA-1, RIPEMD-160

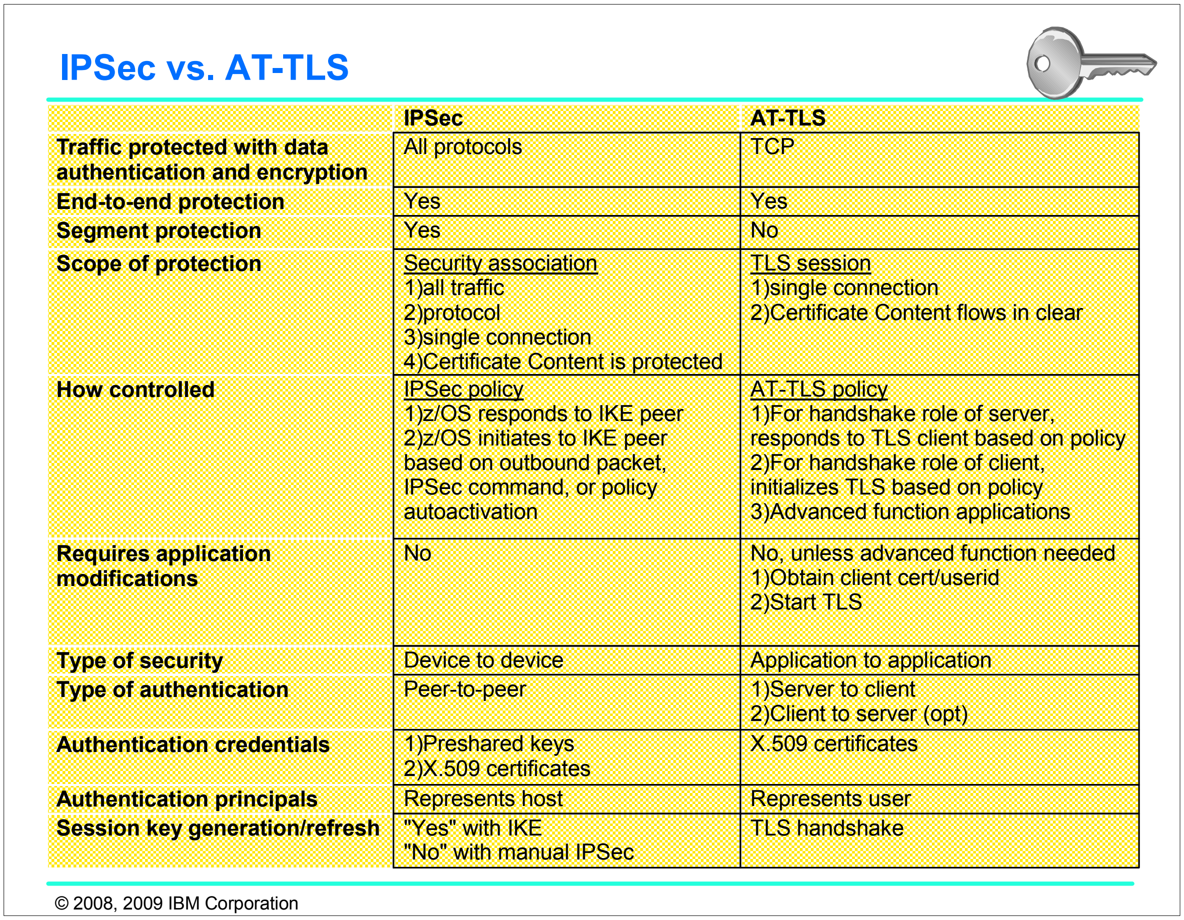
*** Note:** The system-wide known hosts file is in /etc/ssh/ssh_known_hosts.

The user-specific file is in $HOME/.ssh/known_hosts.

*****Note:** With x.509, each client participant needs a copy of only the Certificate Authority's certificate that signed the server certificate in its keyring or key database in order to authenticate the server.

For example, 20 server certificates may have been signed by the same CA, and yet the client requires only the 1 trusted CA certificate in order to authenticate the server(s). With OpenSSH each client requires a copy of the public key for each server in order to authenticate that server. If there are 20 servers, then the client requires copies of 20 public keys.

▸ Diffie-Hellman key exchange (D-H) is a cryptographic protocol that allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.
▸
▸
▸ Perform setup for server authentication:
  ▸ – Generate host keys for server
  ▸ • allows a client to verify the identity of the server.
▸ • Use ssh-keygen to create host keys:
  ▸ ssh keygen t dsa f /etc/ssh/ssh_host_dsa_key N ""
  ▸ ssh keygen t rsa f /etc/ssh/ssh_host_rsa_key N ""
▸ – Create local and remote ssh_known_hosts files
  ▸ • Contains host public keys for all hosts you know about
  ▸ • Copy local host’s public keys to the remote hosts
  ▸ • Gather public keys of remote hosts
▸
▸ OpenSSH from IBM z/OS: Program Product: IBM Ported Tools for z/OS
▸ `•` unpriced, runs on z/OS 1.4 and higher
▸ `• order from ShopzSeries, under “MVS: System Mgmt. and Security.”
▸ `• GA Version info: OpenSSH 3.5p1, OpenSSL 0.9.7b, zlib 1.1.4
▸ `• OA10315 version is: OpenSSH 3.8.1p1, OpenSSL 0.9.7d, zlib 1.1.4
▸ Base SSH: Uses Public Key Infrastructure for authentication and encryption,
▸ Authentication (both client and server) through:
▸ – Public key cryptography
▸ – Existing login passwords
▸ – Trusted hosts authentication
▸ • Data Privacy - through encryption
▸ • Data Integrity - guarantees data traveling over the network is unaltered
▸ • Authorization – regulates access control to accounts
▸ • Forwarding (a.k.a. tunneling) – encryption of other TCP/IP-based sessions
▸ BUT key management and distribution for large numbers of users are difficult because there is no concept of a Certificate Authority. As a result, the keys themselves or the trusted hosts file itself for each server needs to be distributed to the participants.
▸ Only for UNIX files with SFTP; only for UNIX shell with SSH (Tectia extensions allow usage on MVS files); only uses crypto card for generation of the keys
▸
▸ Base SSL or TLS: Either Unix or MVS files with either TN3270 or FTPS;
▸ also uses Public Keys, but in addition relies on x.509 Certificates for authentication thus simplifying key management and distribution, especially if you use a well-known Certificate Authority;
▸ can store master key in hardware and can take advantage of crypto cards for handshakes and sometimes data encryption
▸
▸

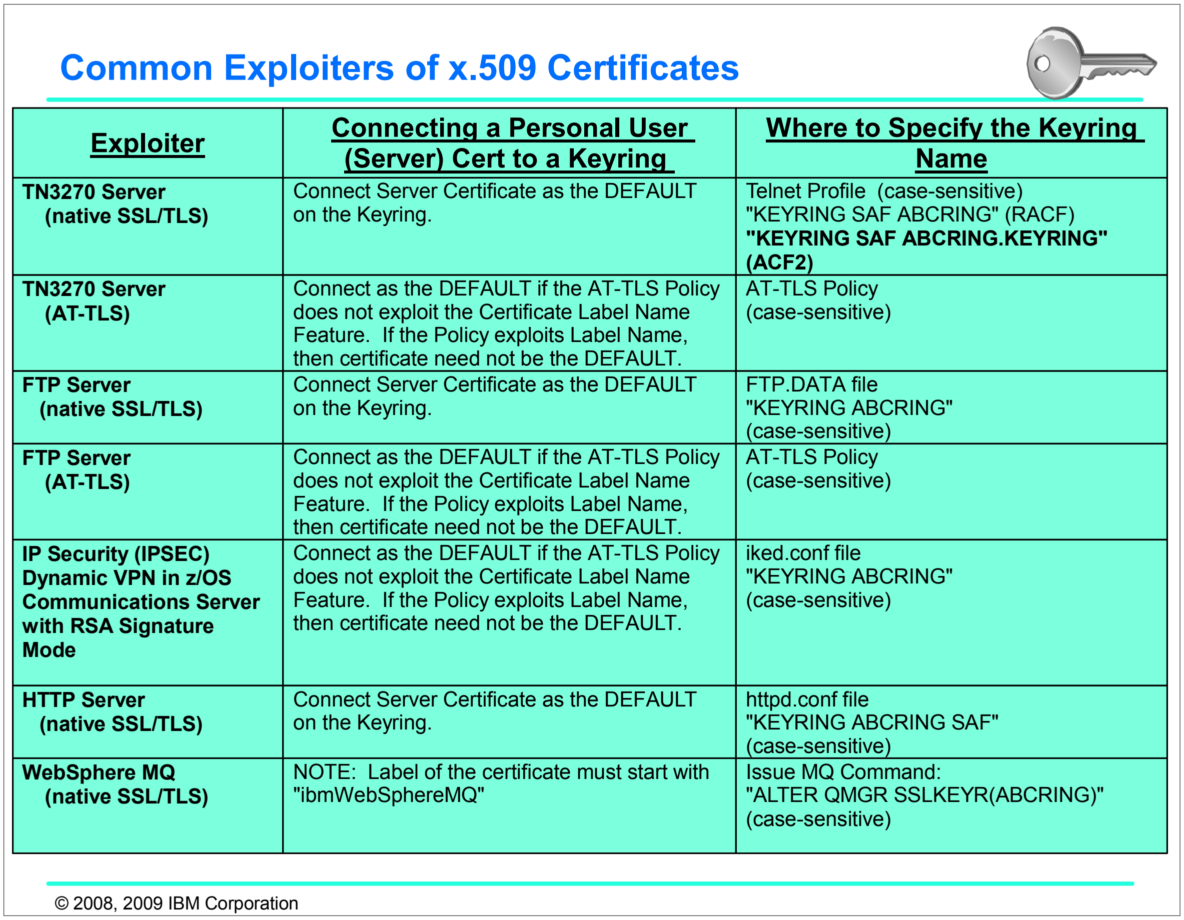


IPSec vs. AT-TLS

	IPSec	AT-TLS
Traffic protected with data authentication and encryption	All protocols	TCP
End-to-end protection	Yes	Yes
Segment protection	Yes	No
Scope of protection	Security association 1)all traffic 2)protocol 3)single connection 4)Certificate Content is protected	TLS session 1)single connection 2)Certificate Content flows in clear
How controlled	IPSec policy 1)z/OS responds to IKE peer 2)z/OS initiates to IKE peer based on outbound packet, IPSec command, or policy autoactivation	AT-TLS policy 1)For handshake role of server, responds to TLS client based on policy 2)For handshake role of client, initializes TLS based on policy 3)Advanced function applications
Requires application modifications	No	No, unless advanced function needed 1)Obtain client cert/userid 2)Start TLS
Type of security	Device to device	Application to application
Type of authentication	Peer-to-peer	1)Server to client 2)Client to server (opt)
Authentication credentials	1)Preshared keys 2)X.509 certificates	X.509 certificates
Authentication principals	Represents host	Represents user
Session key generation/refresh	"Yes" with IKE "No" with manual IPSec	TLS handshake

- There is one more difference between the two protocols.
  - If the SSL Private key is compromised, then any negotiation of a new Session Key is compromised
  - If the IPSec Private key is compromised, then there is a bit more protection, because the Session Key is independently negotiated with Diffie-Hellman.
- How likely is this? Not very.

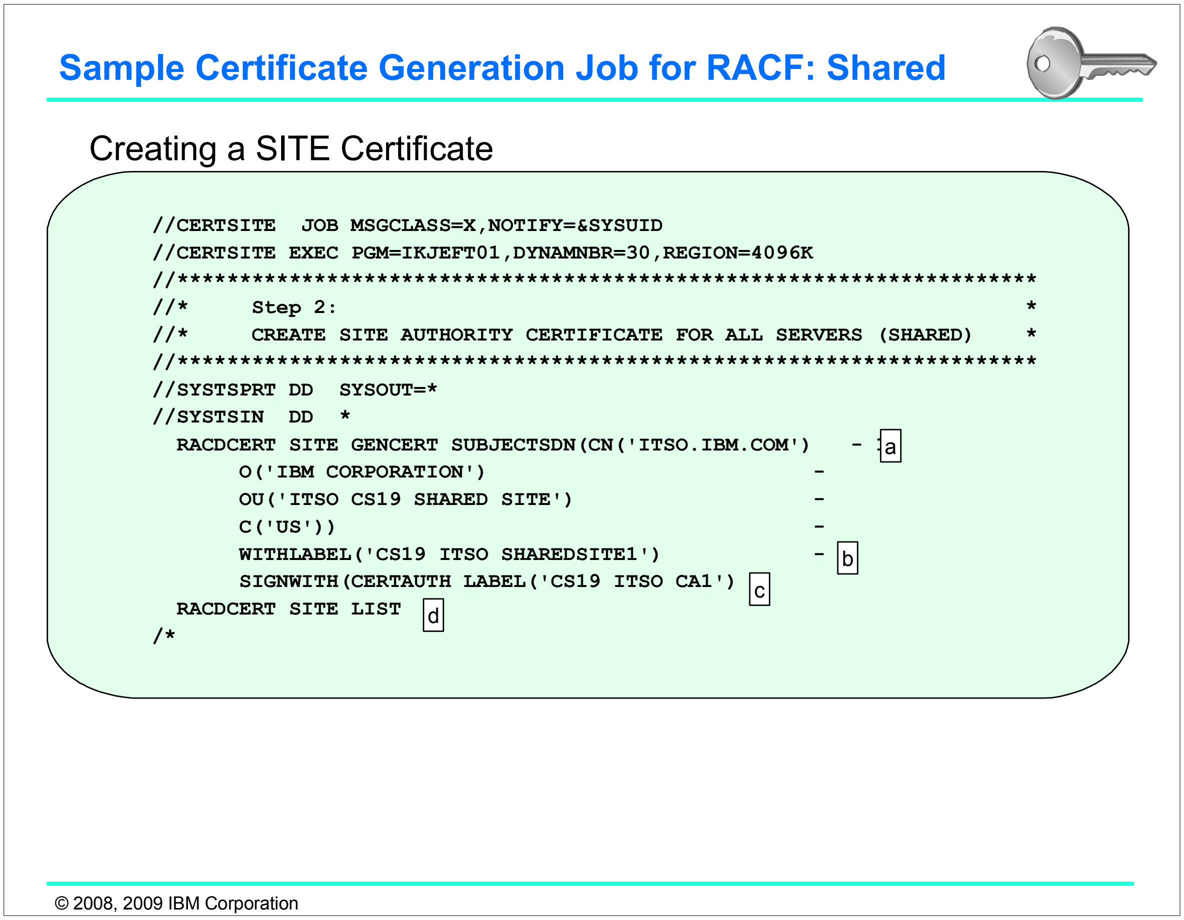
Appendix B:
Advanced Certificate Concepts



Common Exploiters of x.509 Certificates

Exploiter	Connecting a Personal User (Server) Cert to a Keyring	Where to Specify the Keyring Name
TN3270 Server (native SSL/TLS)	Connect Server Certificate as the DEFAULT on the Keyring.	Telnet Profile (case-sensitive) "KEYRING SAF ABCRING" (RACF) **"KEYRING SAF ABCRING.KEYRING" (ACF2)**
TN3270 Server (AT-TLS)	Connect as the DEFAULT if the AT-TLS Policy does not exploit the Certificate Label Name Feature. If the Policy exploits Label Name, then certificate need not be the DEFAULT.	AT-TLS Policy (case-sensitive)
FTP Server (native SSL/TLS)	Connect Server Certificate as the DEFAULT on the Keyring.	FTP.DATA file "KEYRING ABCRING" (case-sensitive)
FTP Server (AT-TLS)	Connect as the DEFAULT if the AT-TLS Policy does not exploit the Certificate Label Name Feature. If the Policy exploits Label Name, then certificate need not be the DEFAULT.	AT-TLS Policy (case-sensitive)
IP Security (IPSEC) Dynamic VPN in z/OS Communications Server with RSA Signature Mode	Connect as the DEFAULT if the AT-TLS Policy does not exploit the Certificate Label Name Feature. If the Policy exploits Label Name, then certificate need not be the DEFAULT.	iked.conf file "KEYRING ABCRING" (case-sensitive)
HTTP Server (native SSL/TLS)	Connect Server Certificate as the DEFAULT on the Keyring.	httpd.conf file "KEYRING ABCRING SAF" (case-sensitive)
WebSphere MQ (native SSL/TLS)	NOTE: Label of the certificate must start with "ibmWebSphereMQ"	Issue MQ Command: "ALTER QMGR SSLKEYR(ABCRING)" (case-sensitive)

- Consult the application Configuration Guide for details on case sensitivity, parameters, and so on.
- If using a SAF repository other than RACF, note that ACF2 appends a suffix of ".KEYRING" to the keyring name. Therefore, a keyring that you think is called "MYKEYRING" must be referenced as "MYKEYRING.KEYRING" for an ACF2 repository.



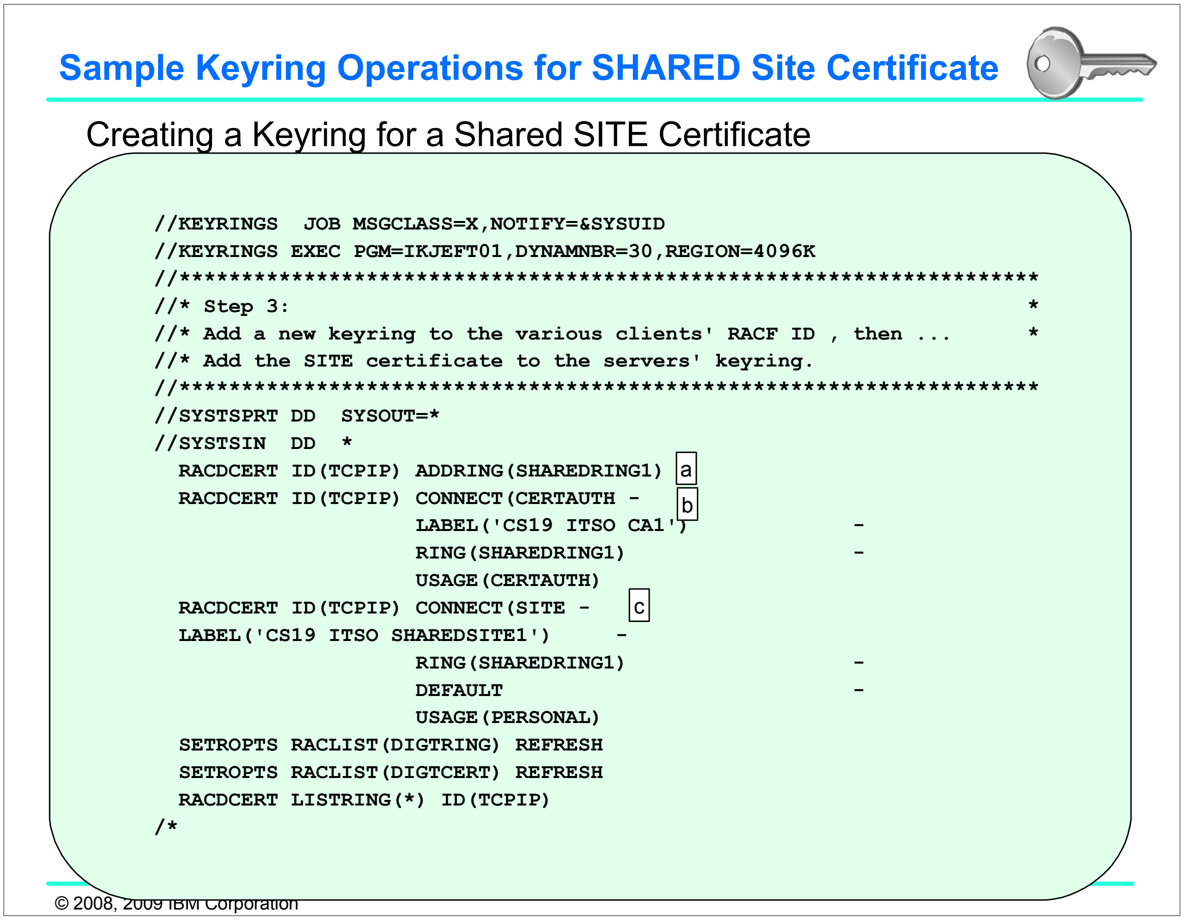
Sample Certificate Generation Job for RACF: Shared

Creating a SITE Certificate

```
//CERTSITE  JOB MSGCLASS=X,NOTIFY=&SYSUID
//CERTSITE EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//*********************************************************************
//*     Step 2:                                                       *
//*     CREATE SITE AUTHORITY CERTIFICATE FOR ALL SERVERS (SHARED)    *
//*********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  RACDCERT SITE GENCERT SUBJECTSDN(CN('ITSO.IBM.COM')   - a
       O('IBM CORPORATION')                          -
       OU('ITSO CS19 SHARED SITE')                   -
       C('US'))                                      -
       WITHLABEL('CS19 ITSO SHAREDSITE1')            - b
       SIGNWITH(CERTAUTH LABEL('CS19 ITSO CA1') c
  RACDCERT SITE LIST d
/*
```

A. Instead of a user ID, the SITE ID is used to indicate that this certificate is to be used as a site certificate, and is not associated with a specific user. The SITE parameter is used in this example because the private key of the certificate being generated is to be shared by multiple servers. It’s not a bad idea (but not required) to make sure that the common name (CN) is the same as the domain name of the site.
B. The LABEL name implies that the certificate is a shared site certificate
C. The SIGNWITH parameter indicates that the internally signed CA certificate that we created previously is used to sign this site certificate. The label of the CA certificate is specified to identify the CA certificate. This indicates that the site certificate should be digitally signed with the internal CA’s private key.
D. This command lists the names of all known site certificates in the RACF Database.

Site certificates are SHARED certificates -- multiple servers can use the same certificate. A site certificate should be used only when it is not necessary to have granular control over the SSL/TLS operations of specific servers. For PCI purposes, it is recommended to use individual PERSONAL (or USER) certificates: one for each server that is subject to PCI compliance mandates.



Sample Keyring Operations for SHARED Site Certificate

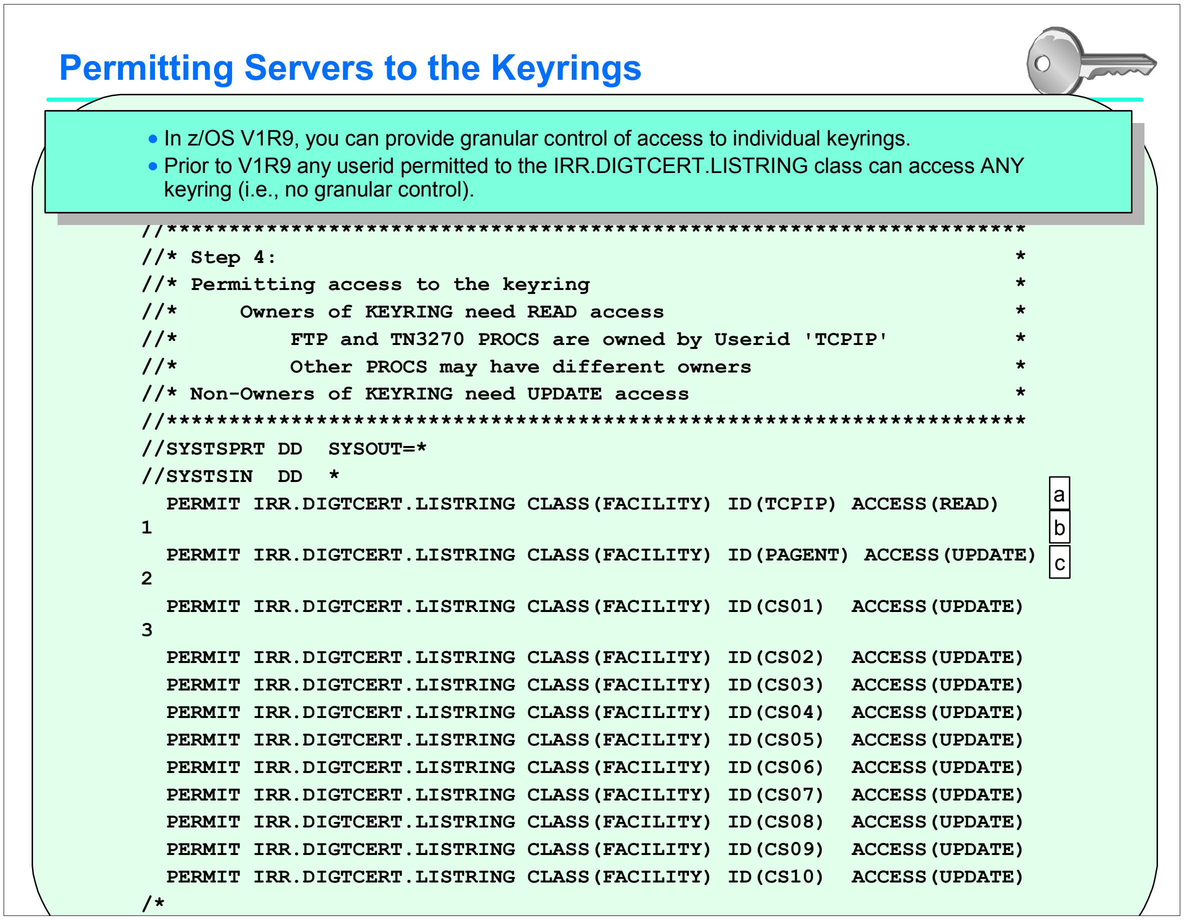
Creating a Keyring for a Shared SITE Certificate

```
//KEYRINGS  JOB MSGCLASS=X,NOTIFY=&SYSUID
//KEYRINGS EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//*********************************************************************
//* Step 3:                                                           *
//* Add a new keyring to the various clients' RACF ID , then ...      *
//* Add the SITE certificate to the servers' keyring.
//*********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  RACDCERT ID(TCPIP) ADDRING(SHAREDRING1) a
  RACDCERT ID(TCPIP) CONNECT(CERTAUTH -    b
                     LABEL('CS19 ITSO CA1')            -
                     RING(SHAREDRING1)                 -
                     USAGE(CERTAUTH)
  RACDCERT ID(TCPIP) CONNECT(SITE -    c
  LABEL('CS19 ITSO SHAREDSITE1')     -
                     RING(SHAREDRING1)                 -
                     DEFAULT                           -
                     USAGE(PERSONAL)
  SETROPTS RACLIST(DIGTRING) REFRESH
  SETROPTS RACLIST(DIGTCERT) REFRESH
  RACDCERT LISTRING(*) ID(TCPIP)
/*
```

A. Create a new RACF shared key ring using the RACDCERT ADDRING command.

B. Connect the internal CA certificate to the new key ring using the RACDCERT CONNECT command.

C. Connect the site certificate (which was signed by the internal CA certificate) to the new key ring using the RACDCERT CONNECT command. Even though the certificate was created as a site certificate, the USAGE must be specified as PERSONAL because the servers use it to authenticate themselves to the clients. It is this certificate that the servers send to the client during server authentication.



Permitting Servers to the Keyrings

- In z/OS V1R9, you can provide granular control of access to individual keyrings.
- Prior to V1R9 any userid permitted to the IRR.DIGTCERT.LISTRING class can access ANY keyring (i.e., no granular control).

```
//*********************************************************************
//* Step 4:                                                           *
//* Permitting access to the keyring                                  *
//*     Owners of KEYRING need READ access                            *
//*         FTP and TN3270 PROCS are owned by Userid 'TCPIP'          *
//*         Other PROCS may have different owners                     *
//* Non-Owners of KEYRING need UPDATE access                          *
//*********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(TCPIP) ACCESS(READ)     a
1                                                                         b
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(PAGENT) ACCESS(UPDATE)  c
2
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS01)  ACCESS(UPDATE)
3
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS02)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS03)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS04)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS05)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS06)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS07)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS08)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS09)  ACCESS(UPDATE)
  PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(CS10)  ACCESS(UPDATE)
/*
```

A. The JCL shows the RACF PERMIT commands that are necessary to grant key ring access to the servers sharing the key ring. Because the key ring is associated with the FTPD and TN3270 user ID, the user ID for the servers needs only READ access.
B. But any other started task IDs (B) or user IDs (C) need UPDATE access because the key ring belongs to a different user ID.

Prior to V1R9 any userid permitted to the IRR.DIGTCERT.LISTRING class can access ANY keyring (i.e., no granular control).

In z/OS V1R9, you can provide granular control of access to individual keyrings.
Instead of giving access to all the rings

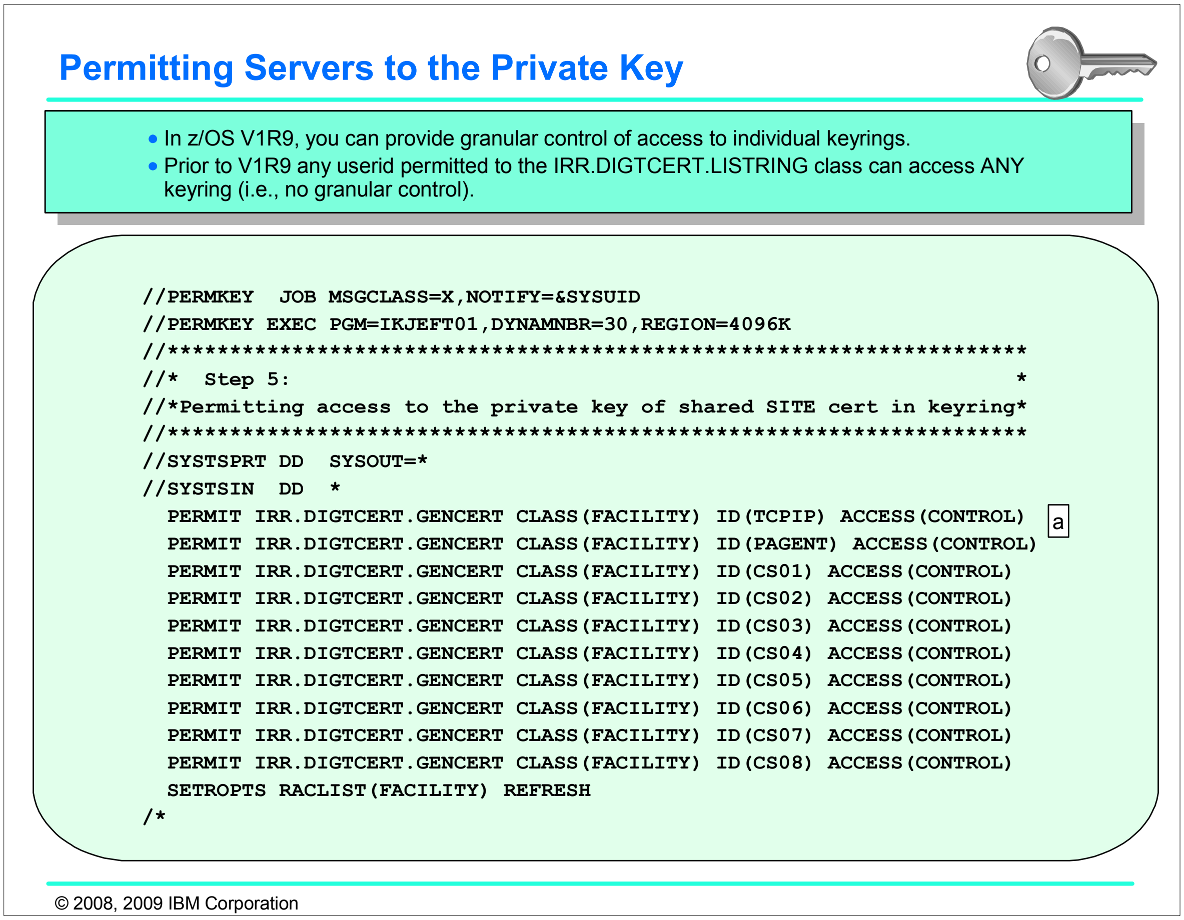
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ACCESS(UPDATE) ID(OTHER)

Just give access to that particular ring with a PERMIT to <ringowner.ringname.LST>:

PERMIT TCPIP.SHAREDRING1.COMMONRING.LST CLASS(RDATALIB) ACCESS(READ) ID(OTHER)

If you want to share the private key, then

PERMIT TCPIP.SHAREDRING1.COMMONRING.LST CLASS(RDATALIB) ACCESS(UPDATE) ID(OTHER)



Permitting Servers to the Private Key

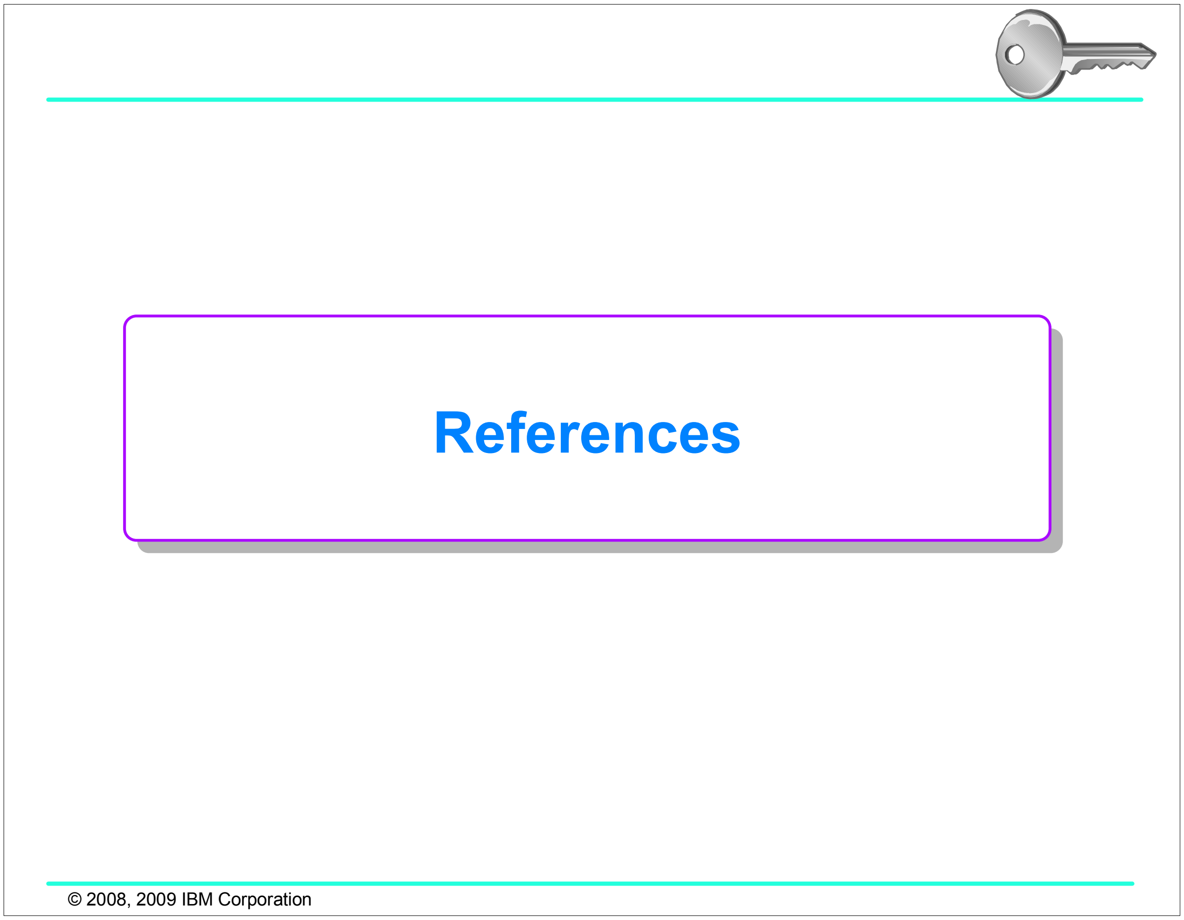
- In z/OS V1R9, you can provide granular control of access to individual keyrings.
- Prior to V1R9 any userid permitted to the IRR.DIGTCERT.LISTRING class can access ANY keyring (i.e., no granular control).

```
//PERMKEY  JOB MSGCLASS=X,NOTIFY=&SYSUID
//PERMKEY EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//*********************************************************************
//*  Step 5:                                                          *
//*Permitting access to the private key of shared SITE cert in keyring*
//*********************************************************************
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(TCPIP) ACCESS(CONTROL)  a
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(PAGENT) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS01) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS02) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS03) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS04) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS05) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS06) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS07) ACCESS(CONTROL)
  PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CS08) ACCESS(CONTROL)
  SETROPTS RACLIST(FACILITY) REFRESH
/*
```

A. Protect the private key and permit the user IDs of each server to access it. The private key is represented by the facility named IRR.DIGTCERT.GENCERT. They all need CONTROL access.

Prior to V1R9 any userid permitted to the IRR.DIGTCERT.LISTRING class can access ANY keyring (i.e., no granular control).

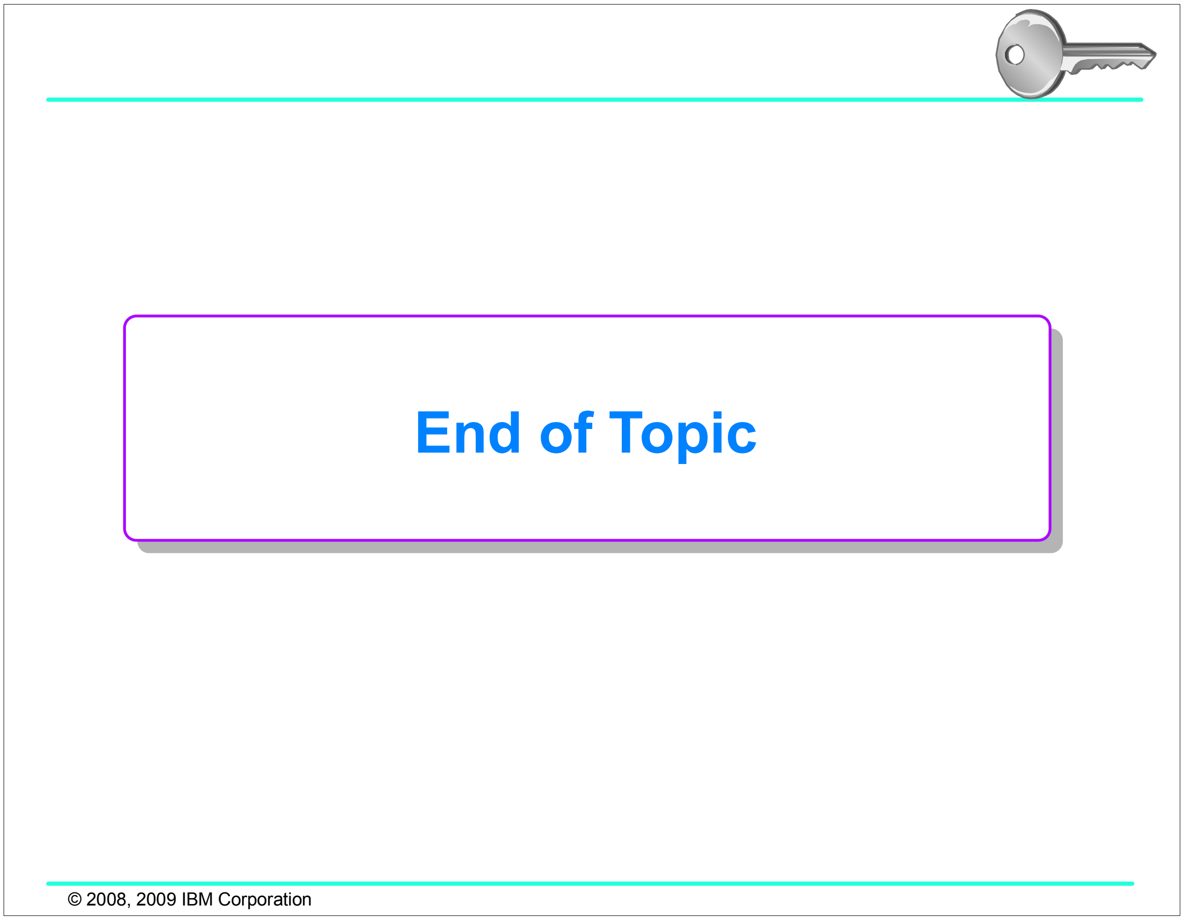
In z/OS V1R9, you can provide granular control of access to individual keyrings.
Instead of giving access to all the rings
PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ACCESS(UPDATE) ID(OTHER)
Just give access to that particular ring
PERMIT TCPIP.SHAREDRING1.LST CLASS(RDATALIB) ACCESS(READ) ID(OTHER)
If you want to share the private key, then
PERMIT TCPIP.SHAREDRING1.COMMONRING.LST CLASS(RDATALIB) ACCESS(UPDATE) ID(OTHER)



References

Web Sites, Manuals, TCP/IP Samples

- **RACF Command Samples for TCP/IP on z/OS**
  - SYS1.TCPIP.SEZAINST(EZARACF)
- **RACF web site:**
  - http://www.ibm.com/servers/eserver/zseries/zos/racf
- **IBM Redbooks**
  - z/OS V1 R8 RACF Implementation (SG24-7248)
  - Communications Server for z/OS V1R9 TCP/IP Implementation Volume 4: Security and Policy-Based Networking (SG24-7535)
- **Security Server Manuals:**
  - RACF Command Language Reference (SC28-1919)
  - RACF Security Administrator's Guide (SC28-1915)
  - RACF Callable Services Guide (SC28-1921)
  - LDAP Administration and Use (SC24-5923)
- **PKI Services web site:**
  - http://www.ibm.com/servers/eserver/zseries/zos/pki
- **PKI Services Red Book:**
  - http://www.redbooks.ibm.com/abstracts/sg246968.html
- **Cryptographic Services**
  - PKI Services Guide and Reference (SA22-7693)
  - OCSF Service Provider Developer's Guide and Reference (SC24-5900)
  - ICSF Administrator's Guide (SA22-7521)
  - System SSL Programming (SC24-5901)



End of Topic

End of Topic