SSL: Put Your Data Under Wraps

Mary Sweat
IBM Corporation
Washington System Center

sweatm@us.ibm.com

SSL: Introduction

- What is SSL
  - SSL stands for Secure Socket Layer
  - A "de facto" standard protocol developed by Netscape, Inc.(TM)
- Goal
  - provide communication privacy and reliability between two communicating applications
    - between client/server applications
    - prevent eavesdropping by supplying encryption
    - prevent tampering by using message integrity
    - prevent message forgery by using digital certificates

- The most common use today for SSL is between a web browser and a a web server.

- The SSL 3.0 Spec is located at http://home.netscape.com/eng/ssl3/3-SPEC.HTM

SSL: Why Use It

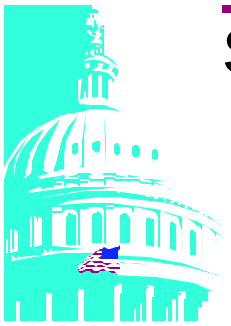
- Any application communicating via a client/server can use the SSL protocol
  - application protocol independent
- Increase the security level of communicated data (beyond what the communication method already provides)
- Ensure who is communicating
  - client and server authenticate each other
- Provide secure/private connections
  - client/server negotiate with each other regarding what encryption algorithm and cryptographic keys will be used
  - no knowledge of each other's code
  - encryption is used after an initial contact

- Application Data is protected while in transit

- A password and user ID could flow in the clear before the handshake is completed.
  Depending on how the application has been written, a URL may require a userid and password before it has started an HTTPS session.

SSL: Why Use It ....

- Efficiency
  - cryptographic operations can be highly CPU intensive
  - SSL incorporates an optional caching scheme
    - reduce the number of connections that need to be established from scratch
    - network activity has been reduced



SSL: Functions

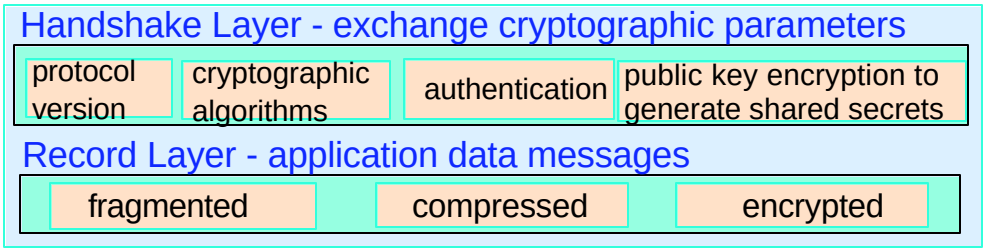
Server

1. provides information and data to the client at the client's request
2. decides what data should be protected
3. is usually an application written to provide data services outbound
4. has the responsibility to protect its identity (will prove its identity via a certificate)

Client

1. initiates the communications
2. generally selects the data to be provided by the Server
3. most are browsers but not necessarily
4. most applications do not care who the clients are
5. can prove its identity by also having a certificate

SSL: Protocol

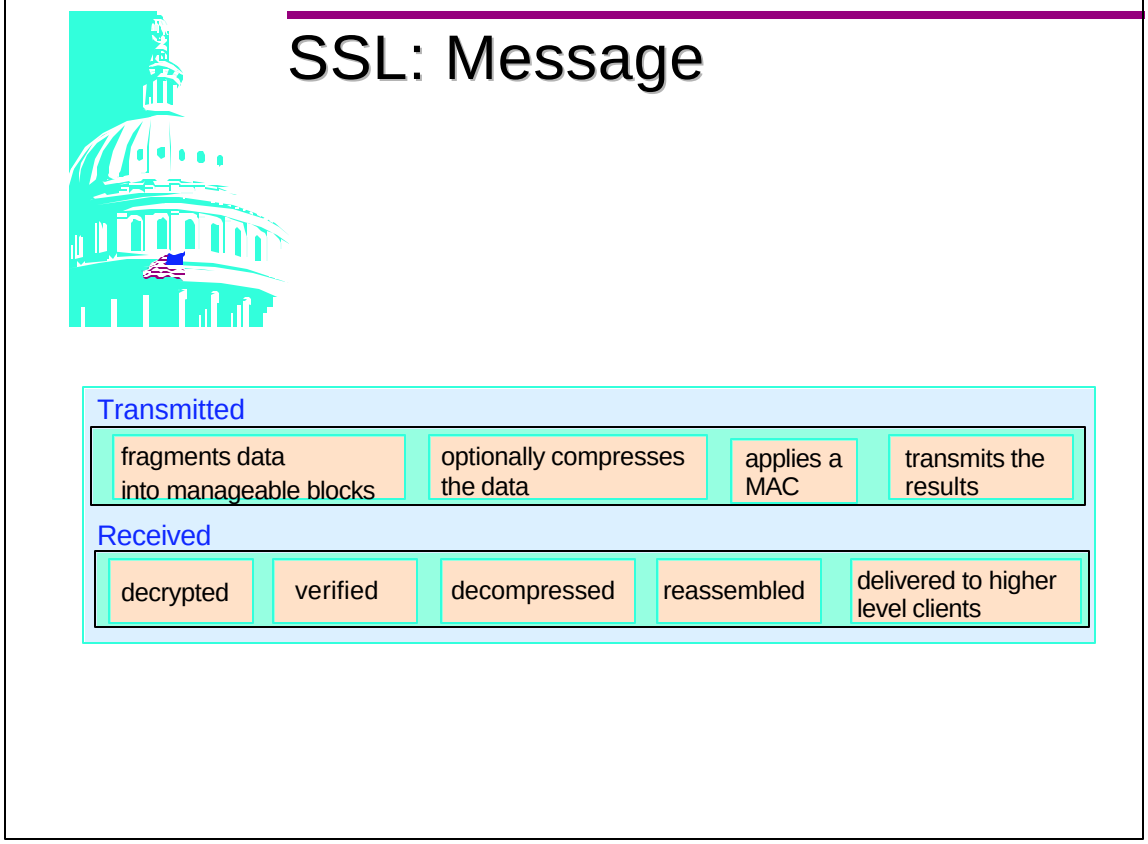


- SSL is a layered protocol. At each layer, messages may include fields for length, description, content.

- Handshake Layer operates on top of the SSL Record Layer. In the Record Layer the data is processed based on the functions selected in the Handshake Layer.

- An SSL session goes through a set of states, starting with an initial, clear-text socket connection.
  1. during the handshake process, the server and client can authenticate each other, using certificates (certificate will usually contain the certificate format, algorithms used to sign the certificate, name of the certificate authority, validity period of certificate, name of user certificate is issued for, user's public key, algorithms (such as RSA or DSS). used to generate the key and CA's signature)
  2 .the client and server negotiate what encryption algorithms and cryptographic keys will be used
  3. this is done before any application data is transmitted

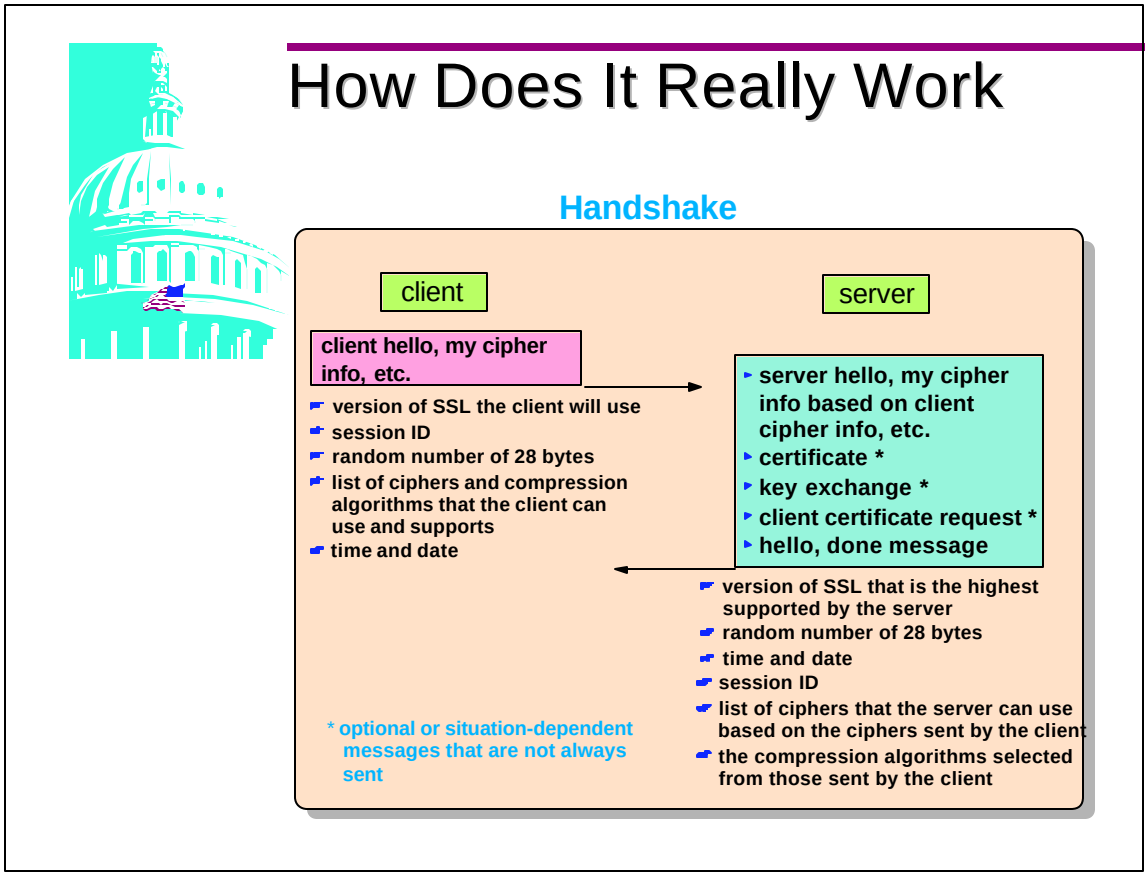
- Definitions:
  1. Algorithm is the procedure the defines the encryption/decryption process
  2. Encryption - encoding the contents of the message in such a way that hides its contents from outsiders (SSL encryption algorithms choices; RC2, RC4, DES or T-DES)



- Definitions:

1. Hashing - a hash algorithm is processed against the data to be transmitted and a numeric value is generated. This value along with packaging information (algorithm used etc.) and the data is sent to the receiver. The receiver will run the data through the same algorithm used by the sender. If the numeric value is the same as the value sent, the receiver knows that the data was not changed or modified during transit. Generating the same numeric value for different data is not feasible.
2. MAC - Message Authentication Codes, hash functions with a private key. To create or verify the MAC, one must have the key. This is useful for verifying that hashes have not been tampered with during transmission.

3 .Digital Signature - is the result of hash data that is encrypted by the senders private key. This provides non-repudiation, the proof that only the sender could have sent the data because only the sender has access to their private key.

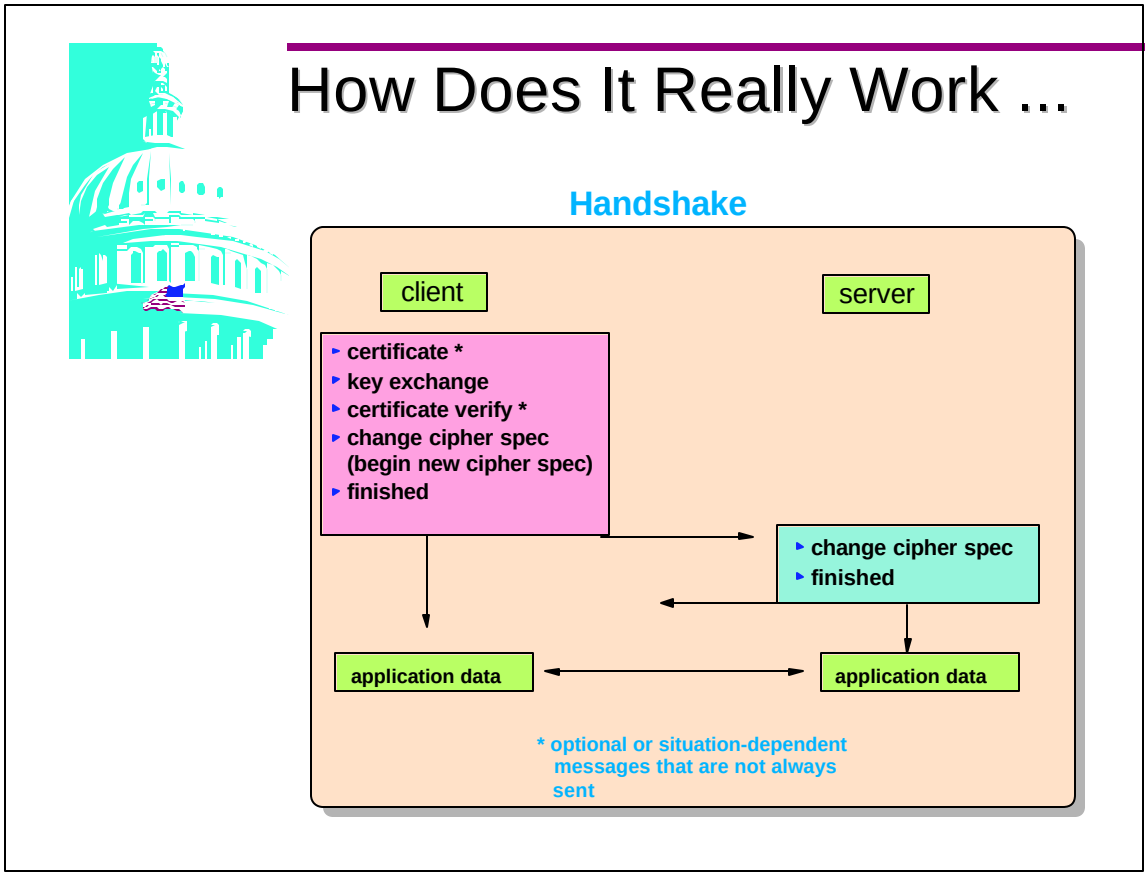


- The client and server hello messages generate random numbers created by a secure random number generator. These numbers must be different and will be used later when creating a master key which is used for encrypting the application data. The client also passes a list of algorithms used to compress data prior to encryption and a CipherSuite list. Each CipherSuite defines both a key exchange algorithm and a CipherSpec (bulk data encryption algorithm and MAC algorithm, hash size, etc.)

- During the Handshake process, the server processes the client's hello and responds with either a server hello message or a handshake_failure alert message. A failure occurs if no acceptable cryptographic algorithms are available to the server. The server selects a CipherSuite and a Compression algorithm from the list supplied by the client.

- If the server is to be authenticated, the server sends its certificate immediately following the server hello message. Certificates are in a sequence chain, ordered by the senders certificate first and the root certificate authority last. The certificate contains the certificate format info, the server's distinguished name, its public key, algorithms used to generate the key, validity dates, name of the CA, CA signature and algorithms used to sign the certificate. If the server has no certificate or the certificate is only used for signing (meaning the certificate can not be used for key distribution) the server will send it's **server key exchange message**. This message contains a Key Exchange algorithm (RSA, Diffie_Hellman or Fortezza), key exchange parameters (public encryption key) , the hash value of these parameters and the Signature algorithm used. The server may request a certificate from the client at this time.

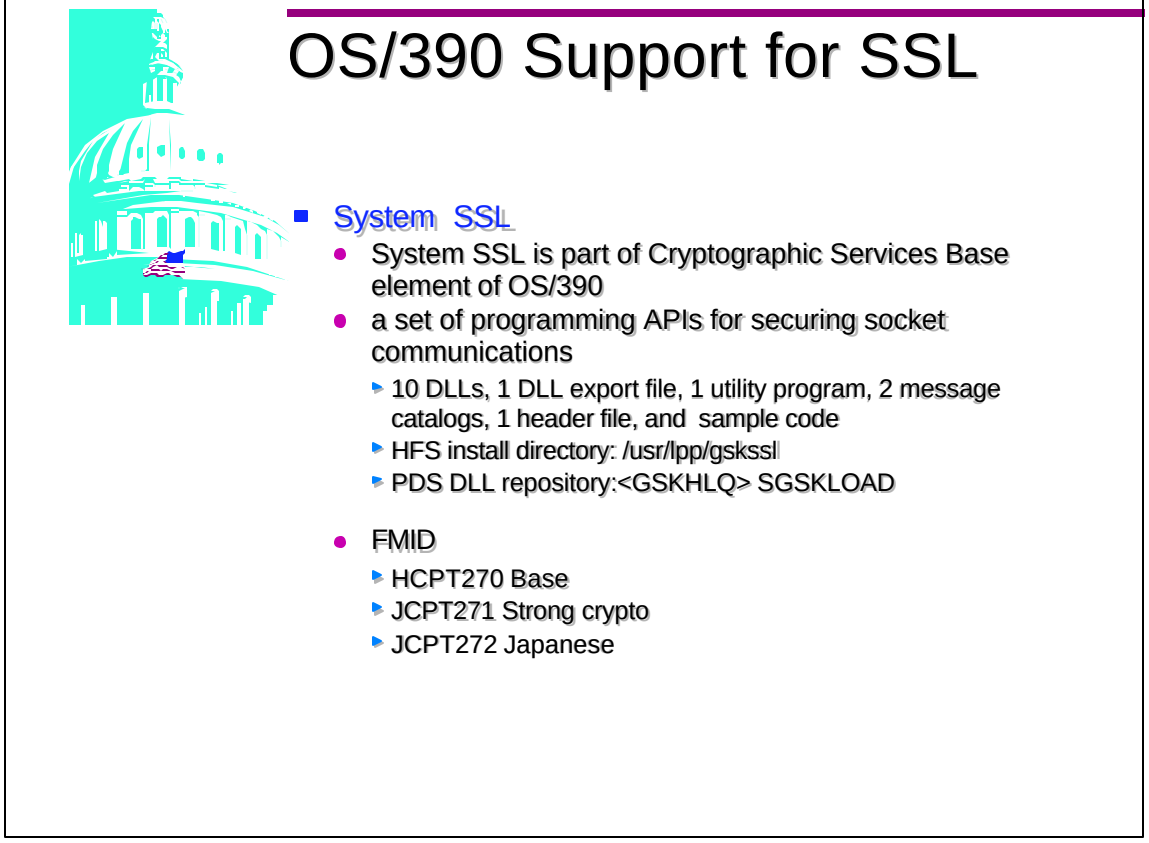
- You can generate a certificate request which you must then send to a certification authority to be certified. The utilities available on S/390 are;
    1. use gskkyman utility (provided with base S/390)
    2. use the RACF component
    3. use a certificate from the IBM Vault Registry (AIX, details located at www-4.ibm.com/software/security/registry)



- The client verifies that the server provided a valid certificate, if one was required, and checks that the parameters passed by the server are acceptable. If a client certificate was requested and the client has one, it is sent at this point. If it has none a , a 'no certificate' alert is sent. If the client has a certificate it sends a list of acceptable certificates ordered with the client certificate first and ending with the root .

- The client key exchange message uses the public key exchange algorithm specified in the server's certificate or key exchange message. If using RSA or Fortezza the client generates a pre_master_secret key and encrypts it with the server's public key. This public key was supplied in the server certificate or during the server key exchange. For Diffie-Hellman the pre_master_key is created from a Diffie_Hellman computation. This encrypted pre_master_key is sent to the server and the server decrypts. Both the client and the server convert the pre_master_key into the master_secret key using the random generate numbers created by the client and server earlier in the handshake. This master_secret key is the one use by the client and server to encrypt application data.

- A change cipher spec is sent from client to server to indicate that the new key and algorithms should be used for future communication. This is followed by a Finished message. This Finished message is the first message encrypted under the new algorithm so both the client and the server must decrypt this message and verify that the contents are correct.

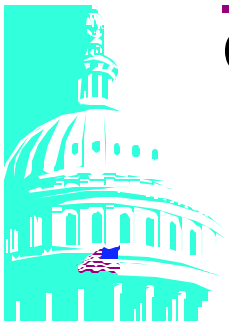


- The DLLs which implement the APIs are shipped in PDS form only. This allows them to be called from PDS-based programs or HFS-based programs.

- The purpose of System SSL is to allow you to code your application with APIs provided rather then you having to write your application from scratch and code every piece to conform to the SSL protocol.

- The utility program allows you to build your key database file for storing certificates

- Any IBM product that uses SSL may be using System SSL under the covers.

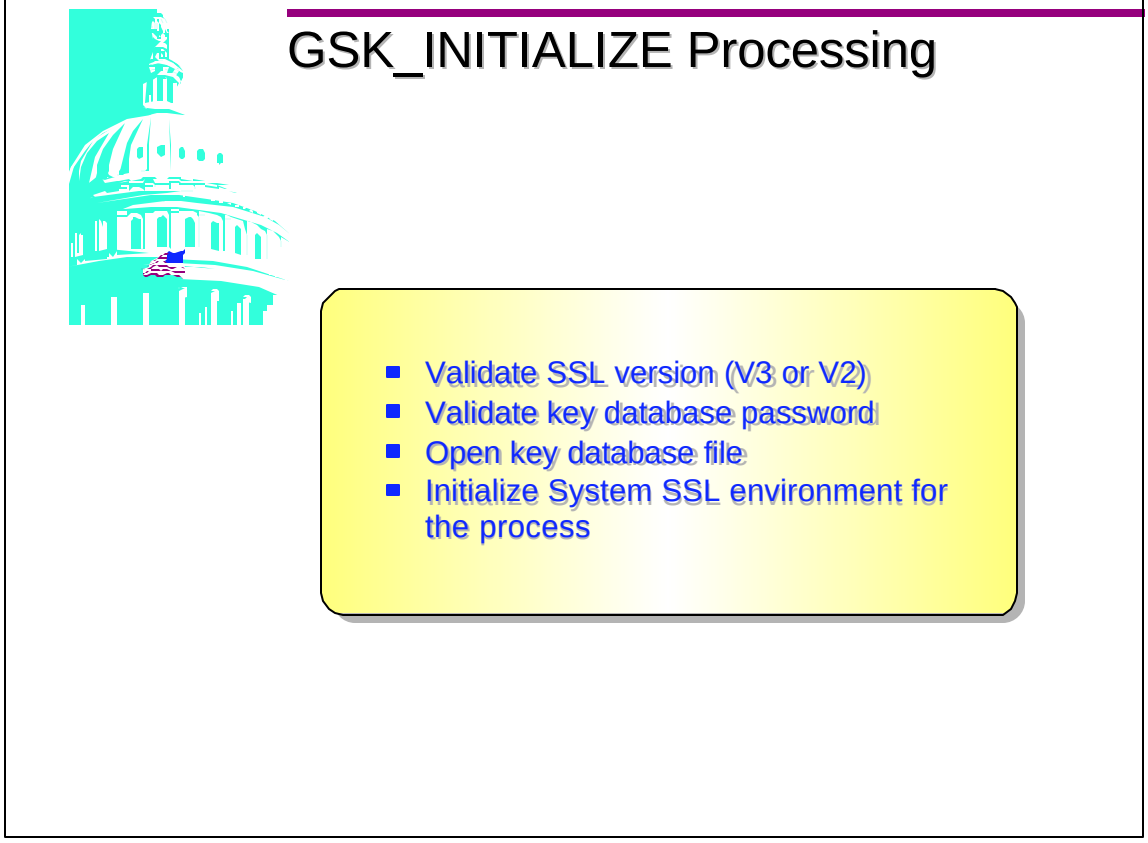


OS/390 Support for SSL ....

- Support consists of five C/C++ callable functions for establishing and using SSL socket connections
  - gsk_initialize
  - gsk_secure_soc_init
  - gsk_secure_soc_read
  - gsk_secure_soc_write
  - gsk_secure_soc_close
- All functions are exported from the GSKSSL DLL, other DLLs are loaded on-demand
- Other utility functions provided:
  - gsk_get_dn_by_label
  - gsk_get_cipher_info
  - gsk_free_memory

- The SSL programming interfaces used most often are the 5 concerning setting up and using the secure socket connection. These 5 APIs are the common used for establishing connections.

- The other utilities listed here can be used to get some limited information about certificates that are transferred during the SSL handshake:
  - gsk_get_dn_by_label will get the distinguished name of a certificate identified by the given label in the key database
  - gsk_get_cipher_info will return selected information from a client certificate
  - gsk_free_memory should be used to deallocate data that was allocated by the SSL APIs and returned to the caller

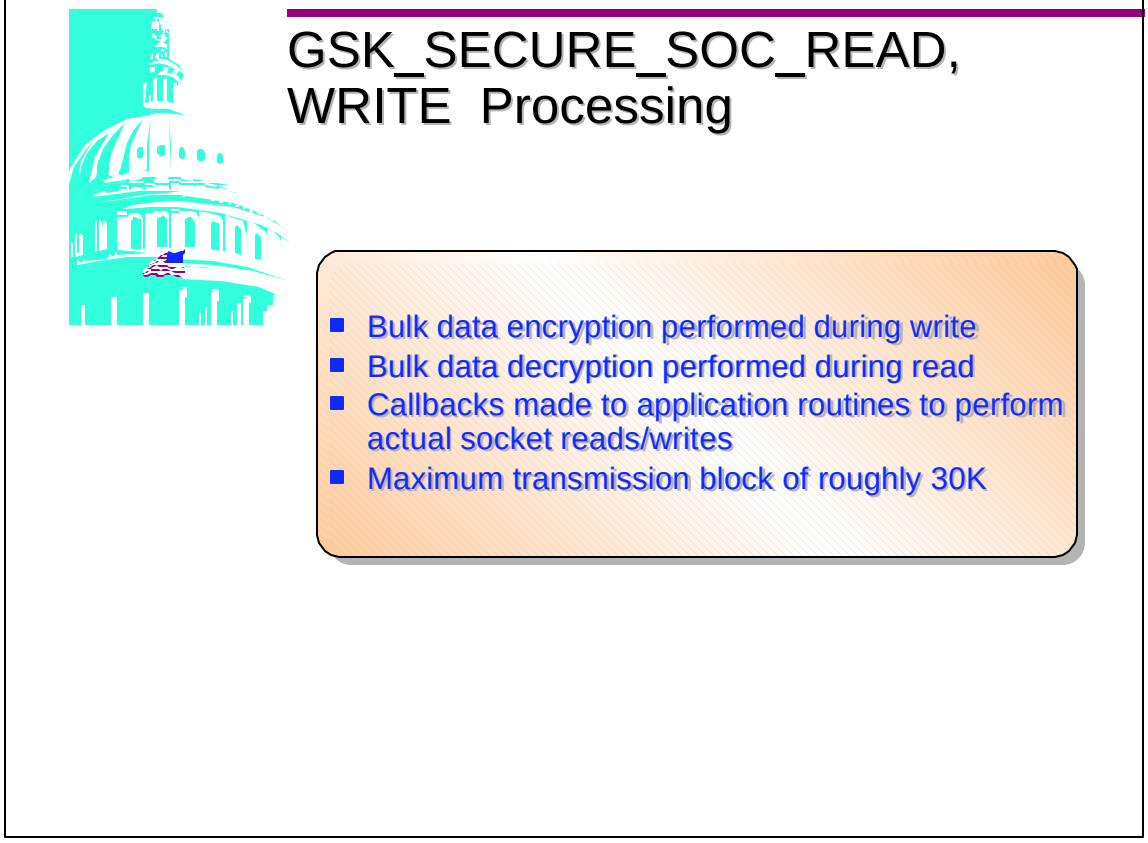


- This API must only be used once per process. If called more than once unpredictable errors could occur.
- Sets up the overall System SSL for initializing the environment for the current process.

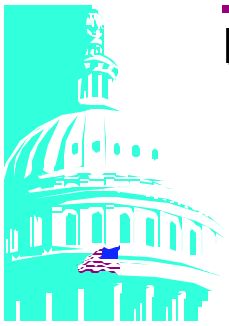
GSK_SECURE_SOC_INIT Processing

- Validate key database entry to be used
- Validate cipher spec preference
- Determine type (client/server) of handshake to perform
- Use the descriptor, *skread* and *skwrite* routines to wait and perform the SSL handshake
- Cipher spec used for SSL session returned
- SSL protocol version for session returned
- Public/private key operations performed during handshake

- This process is very CPU Intensive and some caching is involved.
- Initializes the data areas necessary for System SSL to either iniate or accepts a secure socket connection.
- This one API covers all the functions performed in the handshake.



- Applications must supply the read and write routines.
- The Web maximum transmission block is 8K.
- This API will receive or send data on a secure socket connection using the application specified read/write routines.

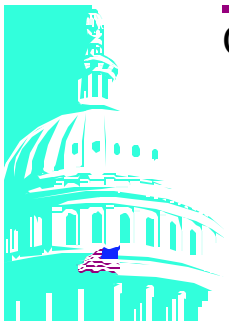


Key and Certificate Management

- X.509 Certificates are used by both Client and Server when securing communications using the SSL protocol
- Client must verify the server's certificate
  - based on the certificate of the Certificate Authority (CA) that signed the certificate

    OR
  - based on a self-signed certificate from the server
- Server must verify the client's certificate (if requested) is using the certificate of the CA that signed the certificate

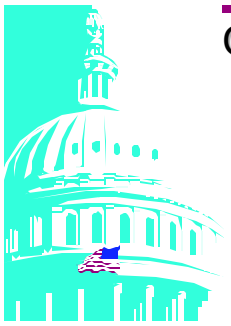
- System SSL requires that the server have a certificate - it can, however, be a self-signed certificate which GSKKYMAN can create. The client must be able to verify the server's certificate.
- If the certificate is from a CA that is known to the base key database created by gskkyman, the client's key database need only be created.
- If the server's certificate is a self-signed certificate, the server's certificate must be imported into the client's key database (after being exported to a file from the server's key database).



GSKKYMAN Utility

- CA certificates, client, and server certificates are stored in a key database managed by a UNIX shell command-line utility called gskkyman
- Key database must be an HFS file
- GSKKYMAN is also used to formulate certificate requests and handle certificate request responses
- GSKKYMAN is used by both client and server side of System SSL communications (when running OS/390)

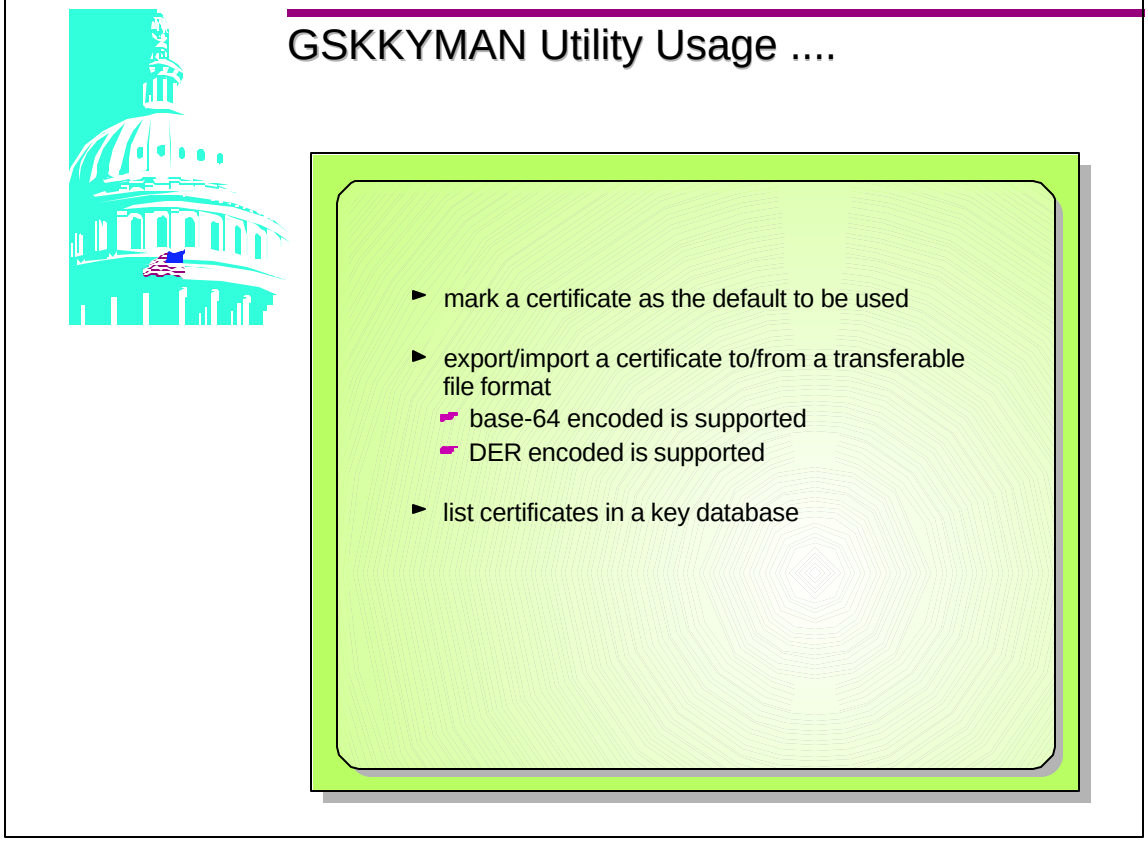
- To run GSKKYMAN, STEPLIB must be set.
- NLSPATH must point to a directory that contains the message catalogs for the GSKKYMAN utility. These appear in /usr/lib/nls/msg/C as well as /usr/lib/nls/msg/En_US.IBM-1047 (and the corresponding Japanese locale when JCPT272 is installed).



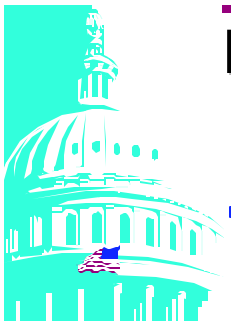
GSKKYMAN Utility Usage

- Use gskkyman to;
  - create a key database file and associate a password with it
    - this will add a number of well-known CA certificates to the key database (e.g. Verisign, Thawte)
    - if these CAs are used for obtaining server certificates, then nothing else is required on the client side
  - create a self-signed certificate

    OR
  - create a certificate request
  - import a certificate that was sent by a CA
    - cut and paste into a file
    - transfer the file to OS/390
    - use import function to import the certificate
    - either DER encoded or Base-64 encoded formats can be used



- DER - Data Encryption Rules
- BER - Binary Encoding Rules



Points to Remember

- SSL is a protocol
  - divided into two layers; handshake and record
  - in the handshake layer authentication occurs for the server and optionally the client
    - negotiations of sessions options also take place
- SSL protocol must be implemented by the application code wishing to use it
- S/390 products needing SSL, use SSL via a common code interface
  - an internal toolkit
  - OS/390 Cryptographic Services System SSL component
- S/390 servers using SSL determine the priority of the algorithm used at the record layer.

- System SSL Publication is SC24-5877 OS/390 Cryptographic Services System SSL Programming Guide and Reference