Encryption and Key Management

Jamie Farmer
z/TPF Development

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Encryption and Key Management
Securing Data on z/TPF

- The z/TPF system uses many methods to protect data, centering on data encryption, data integrity and authentication.
  - Data Encryption : Scrambles data making it unintelligible for unauthorized parties.
  - Data Integrity : Ensures the data has not been altered.
  - Authentication : Establishes that you are who you claim to be

- Protecting Data
  - Protecting Data In Flight : Data that is being transferred across the network
    - Transport Layer Security, or TLS (also known as SSL)

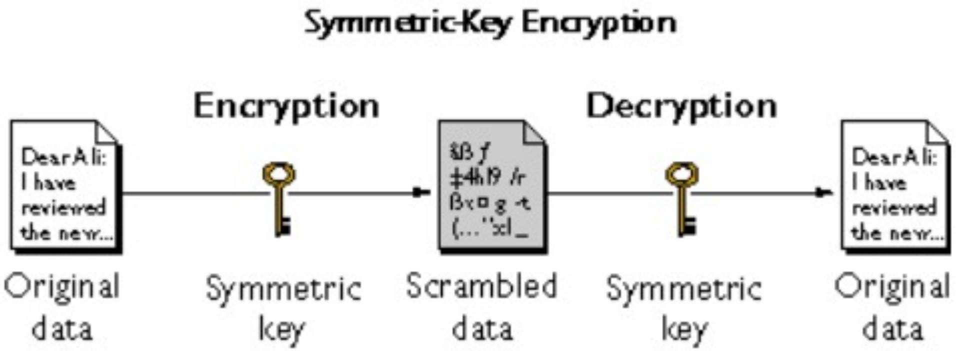
  - Protecting Data at Rest : Data that is being persisted (or stored)
    - Tape Encryption
    - z/TPFDF Encryption
    - Application encrypted using the secure keystore

  - Protecting Data in Use : Data residing in z/TPF memory during a transaction
    - Don’t want this sensitive data to be displayable (z-Commands, dumps, debugger, etc)
    - Can use non-displayable storage to mark storage areas as non-displayable

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

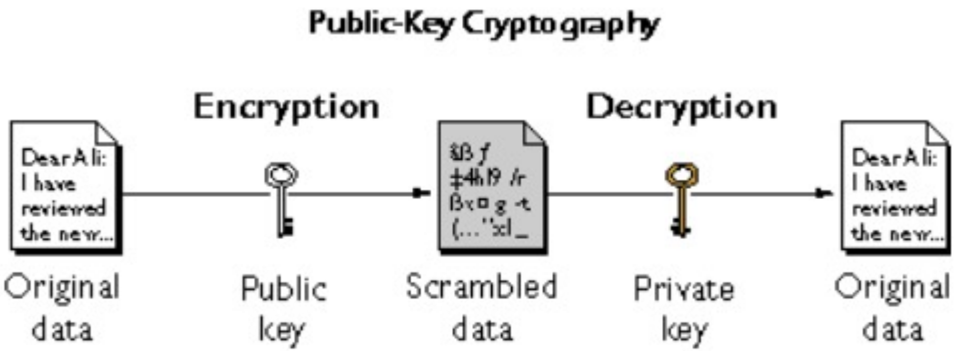
Symmetric vs Asymmetric Encryption
Symmetric Encryption



- Same key is used for encrypting and decrypting
- Used for encrypting / decrypting large amounts of data
  - For example, encrypting data across the network
  - Relatively inexpensive to do symmetric encryption
- The downfall is both sides need to exchange this secret key

- z/TPF supports
  - DES, TDES
  - **AES128, AES256 (recommended)**

Symmetric vs Asymmetric Encryption
Asymmetric Encryption



- Public / Private key pair is used for encrypting and decrypting
  - Data encrypted with the public key can only be decrypted with the private key
- Primarily used in OpenSSL to transfer secret symmetric key
- Extremely expensive operations to perform

- z/TPF supports
  - **RSA1024 and RSA2048**

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Cryptographic Hardware Acceleration

Cryptographic Hardware Acceleration

- There are two types of cryptographic hardware

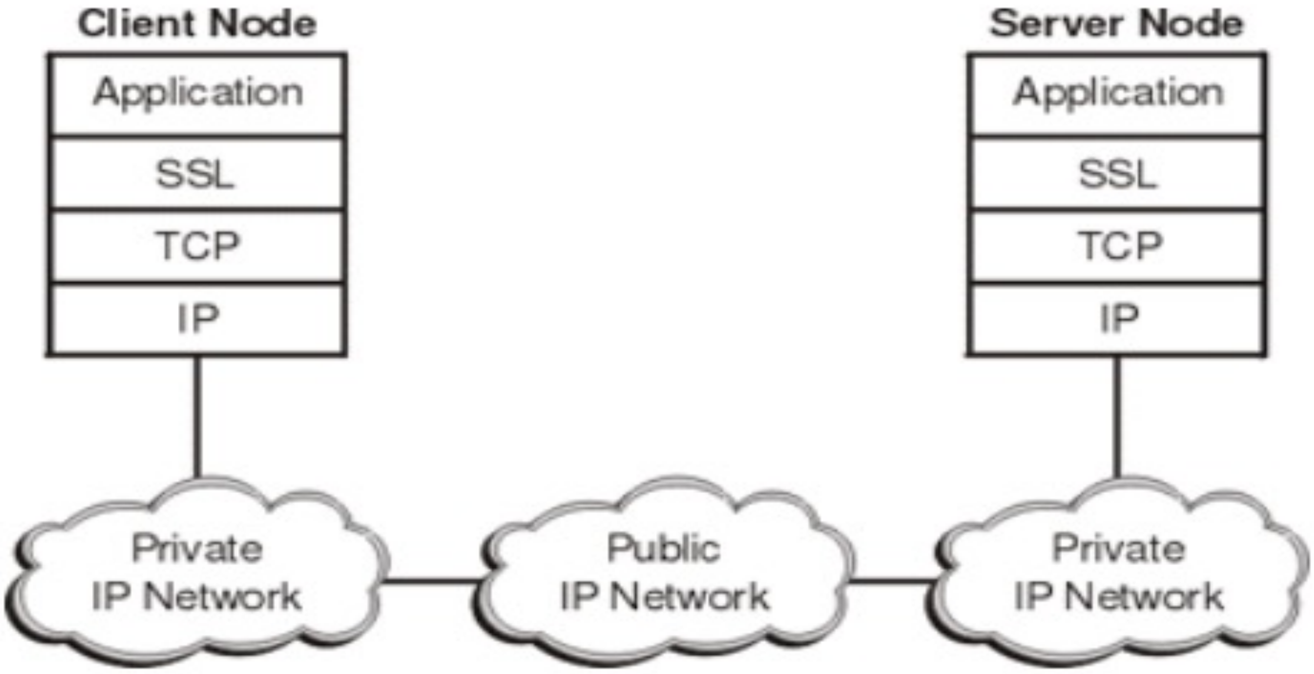
- Central Processor Assist for Cryptographic Function (CPACF)
  - Used for symmetric encryption and message digests (data integrity)
    - Encryption algorithms – DES, TDES, AES128 and AES256
    - Message Digests – SHA1, SHA256, SHA512
    - Random number generation - DRNG, TRNG
  - A co-processor that resides next to the main CPU
  - **Operations to CPACF are synchronous**
    - **Meaning CPU waits for operation to complete**

- CryptoExpress card
  - Used for asymmetric encryption
    - RSA1024 and RSA256
  - Separate physical card that gets put in the processor
    - Can add more cards as load increases
  - **Operations to CryptoExpress are asynchronous**
    - **Meaning CPU can do other work while operation is in progress**

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Transport Layer Security (TLS)
TLS Architecture



Transport Layer Security (TLS)
z/TPF Support for TLS

- The z/TPF system supports TLS (SSL) sessions by using standard OpenSSL (version 1.1.1)
  - Transport Layer Security (TLS) version 1.0, TLS version 1.1, and TLS version 1.2
  - Rivest-Shamir-Adelman (RSA) public key cryptography.
  - AES128, AES256, and (Triple-DES) ciphers
  - Message Digest Algorithm 5 (MD5), Secure Hash Algorithm (SHA-1, SHA-256) digest algorithms
  - Client and server authentication

- z/TPF OpenSSL Cryptography Usage:
  - Uses asymmetric encryption (RSA) for SSL session startup
    - Asymmetric encryption which is used to encrypt the symmetric key for this SSL session
      - A new symmetric key is created for every SSL session
  - z/TPF OpenSSL uses symmetric encryption (TDES or AES) for encrypting data across the wire.
  - z/TPF OpenSSL uses MD5, SHA1, SHA256 for data integrity

- Open source package has been updated to use the CPACF and CryptoExpress hardware acceleration when available.

Transport Layer Security (TLS)
TLS Handshake Flow

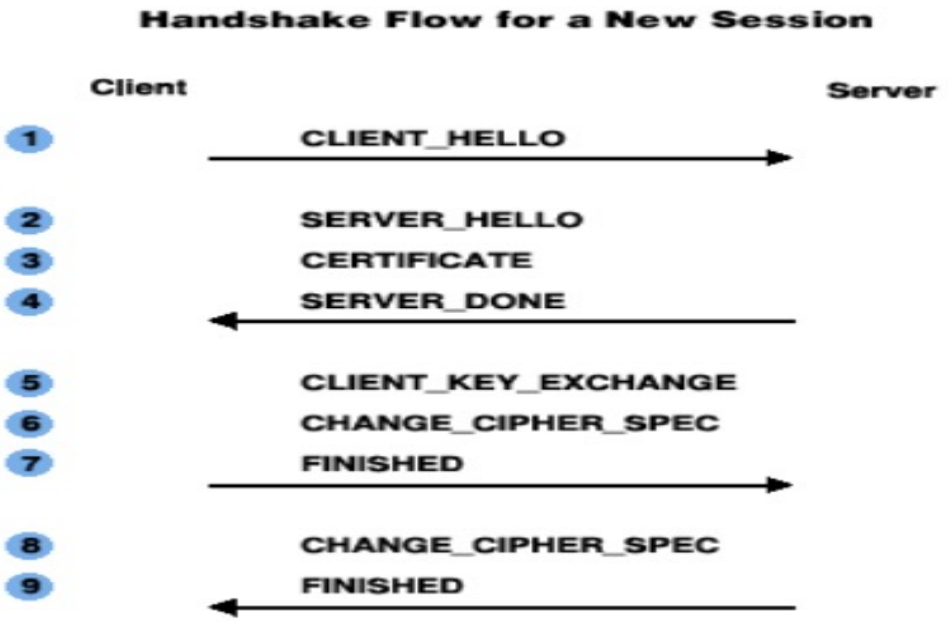
The following figure shows the handshake flow for a new session.

Figure 1. Handshake flow for a new session

Client

SSL Context (CTX)
- TLS Version
- TLS Ciphers
- CA Certificate

- ➢ Validates server certificate
- ➢ Uses public key in server certificate to encrypt secret symmetric key



Server

SSL Context (CTX)
- TLS Version
- TLS Ciphers
- Certificate
- Private Key

- ➢ Decrypts secret symmetric key using private key
- ➢ Negotiates cipher

Transport Layer Security (TLS)
z/TPF Support for TLS

- Specifying certificates and key files for applications is done programmatically in SSL

- Middleware that supports SSL in z/TPF generally uses SSL configuration files to define the SSL configuration for applications
  - Allows for administrative control over SSL configuration
  - For example, INETD model SSL inputs the SSL configuration in /etc/ssl/inetd/*servername*.conf

- SSL configuration generally includes
  - TLS Version (1.0, 1.1, 1.2)
  - Ciphers (ie. AES256-SHA256)
  - Certificates and Keys
    - Required for servers
    - Only required for clients when "Client Authentication" is enabled on the server
  - Certificate Authority
    - Generally required for clients (at times can be disabled for test)
    - Only required for server if "Client Authentication" is enabled.

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Transport Layer Security (TLS)
z/TPF Shared SSL Support

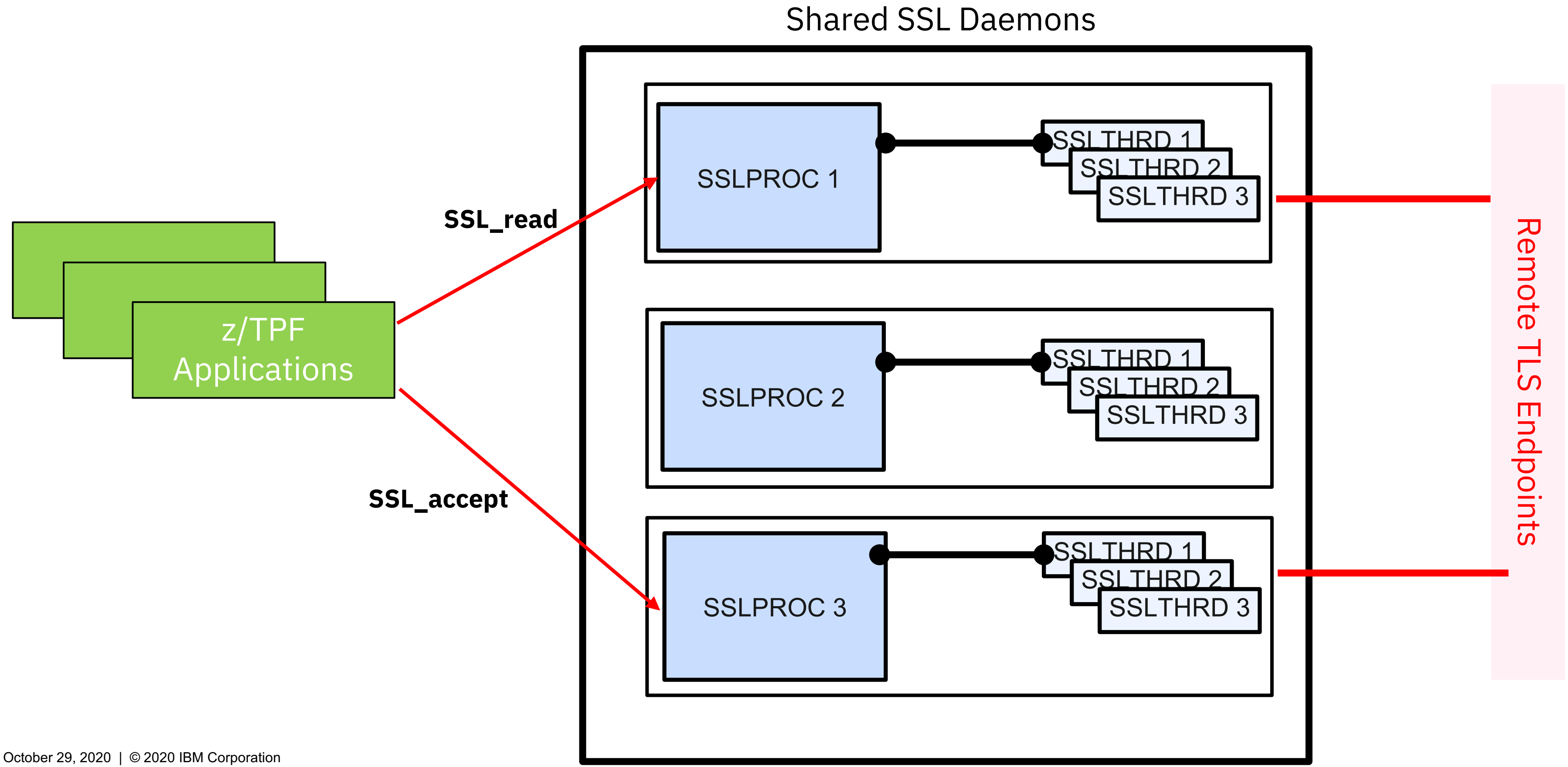
- Shared SSL increase the scalability and usability by allowing SSL sessions to be shared by ECBs on the z/TPF system

- The opensource OpenSSL library is heavily tied to a specific process (ECB)
  - When the process exits, the session is cleaned up (process scoped sockets)

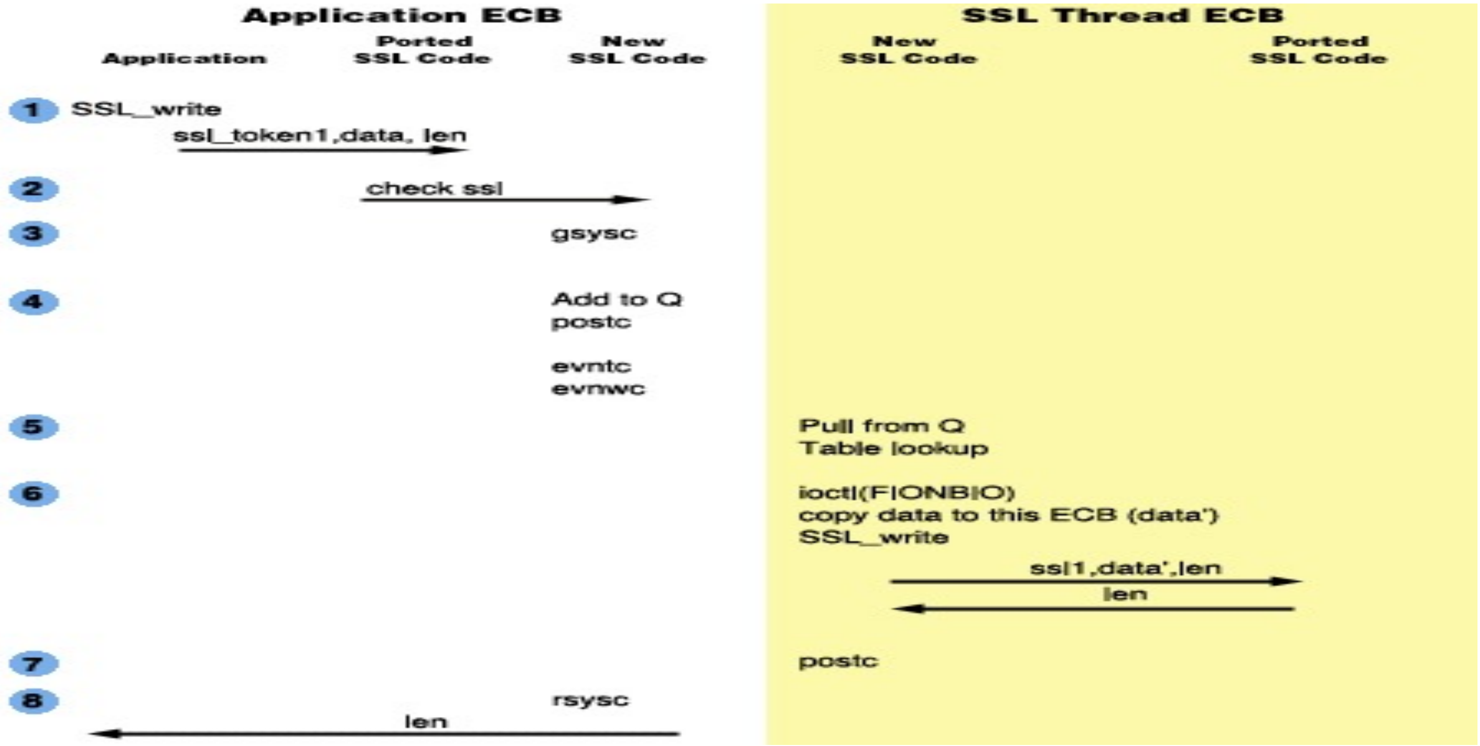
- z/TPF Shared SSL Support
  - A system managed configurable set of long running processes and threads that own the SSL sessions on behalf of z/TPF applications.

Transport Layer Security (TLS)

z/TPF Shared SSL Daemons



Transport Layer Security (TLS)
z/TPF Shared SSL Support



Transport Layer Security (TLS)
z/TPF Shared SSL - Application

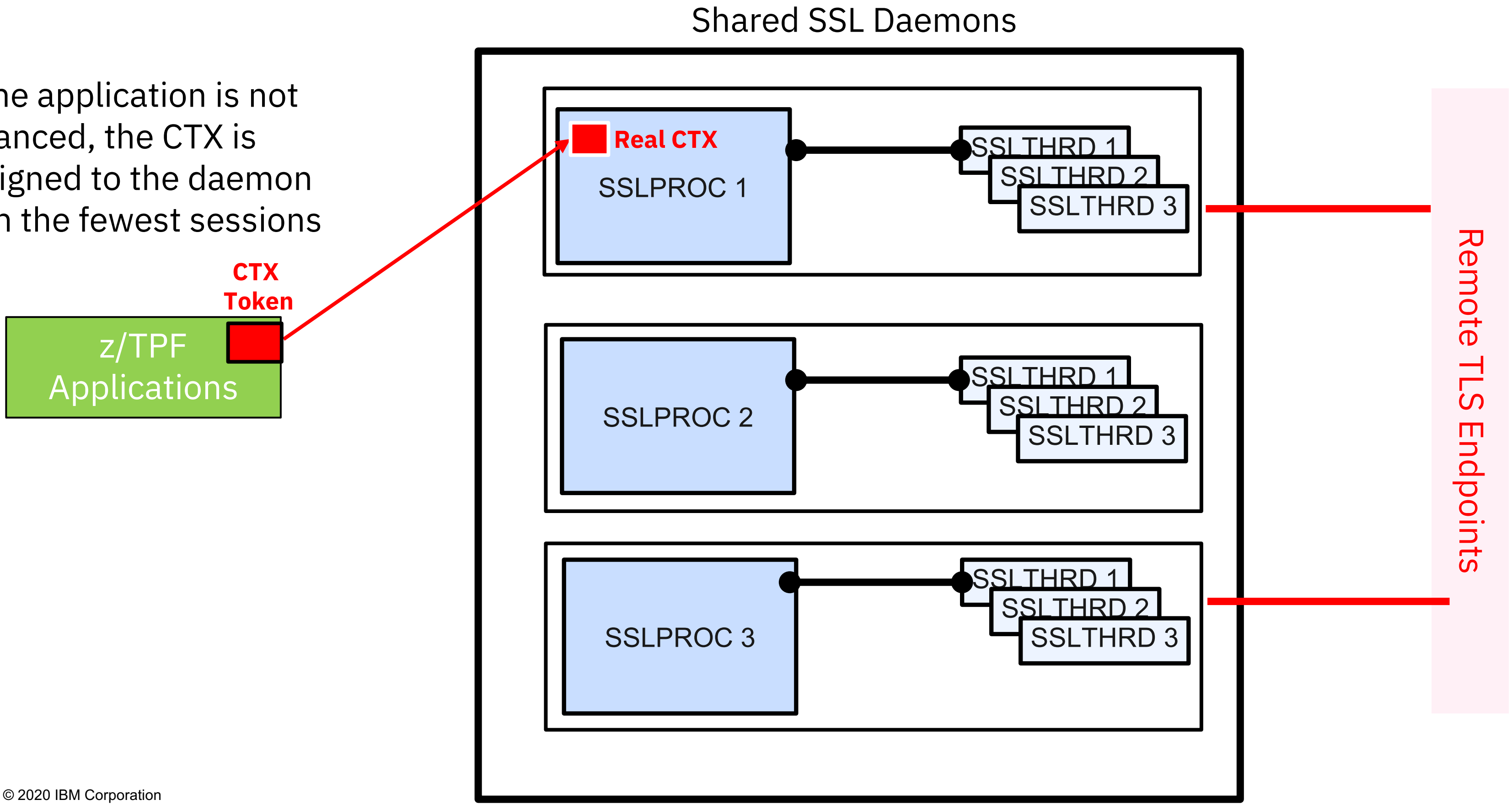
- An SSL application must first establish a context (CTX).
  - The CTX assigned holds things like certificates, session options, etc.
  - One or more SSL sessions can be created against the same CTX
  - SSL_CTX_new creates the CTX for an SSL process.

- For z/TPF Shared SSL, you issue SSL_CTX_new_shared
  - This assigns the CTX to one of the SSL daemon threads.
  - The application still gets a CTX, but its really acting as a token for TPF to assign requests to the shared SSL processes.
  - SSL sessions created against that CTX are also tokenized and assigned to a shared SSL process.

- You can load balance sessions for specific CTXs across multiple daemon processes:
  - Code and loading the /etc/sslshared.txt defines the balance for specific CTX structures
    - The SSL_CTX_new_shared takes in an optional name used in this file.

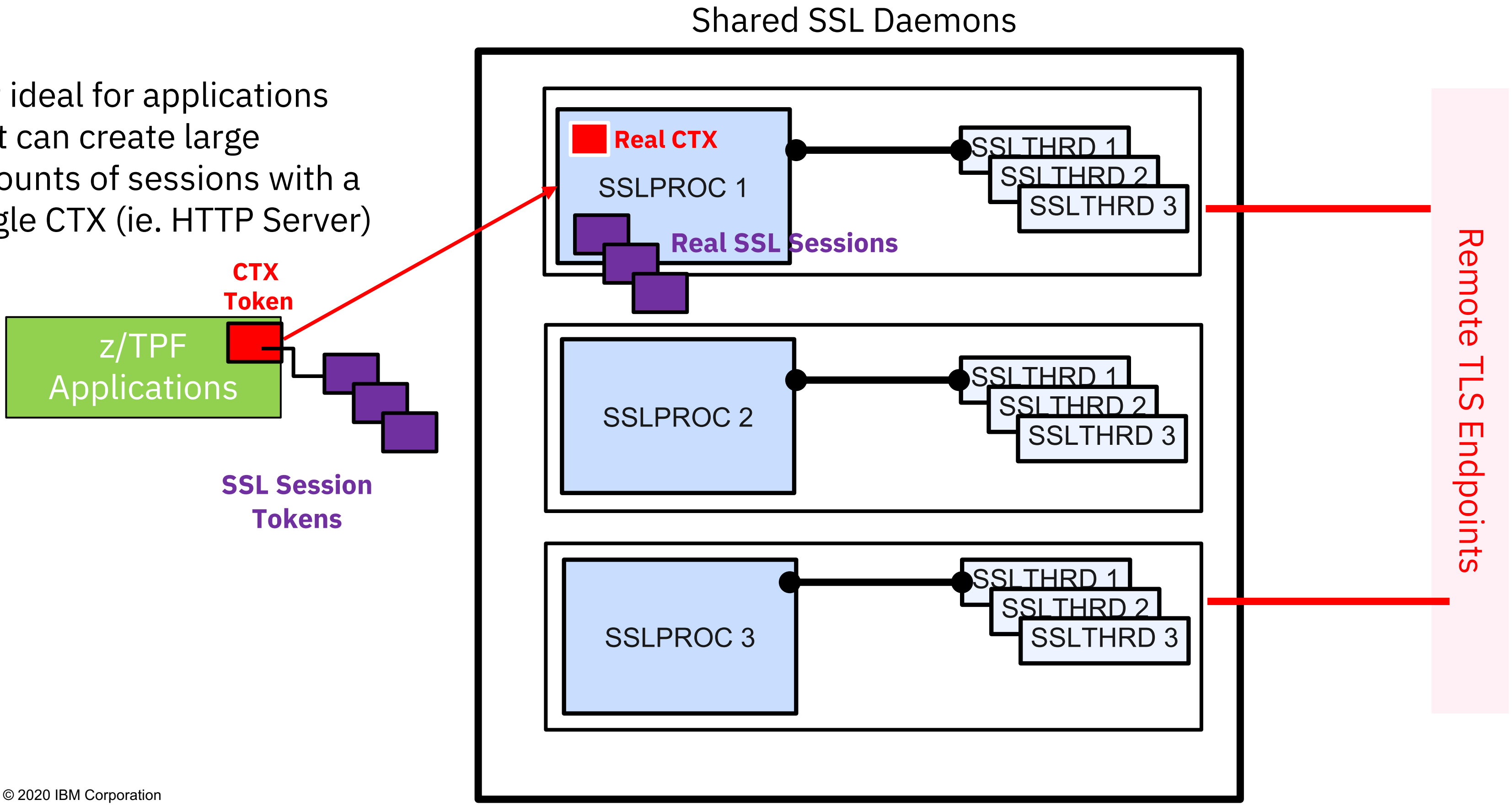
Transport Layer Security (TLS)
z/TPF Shared SSL CTX – Not Balanced

- If the application is not balanced, the CTX is assigned to the daemon with the fewest sessions



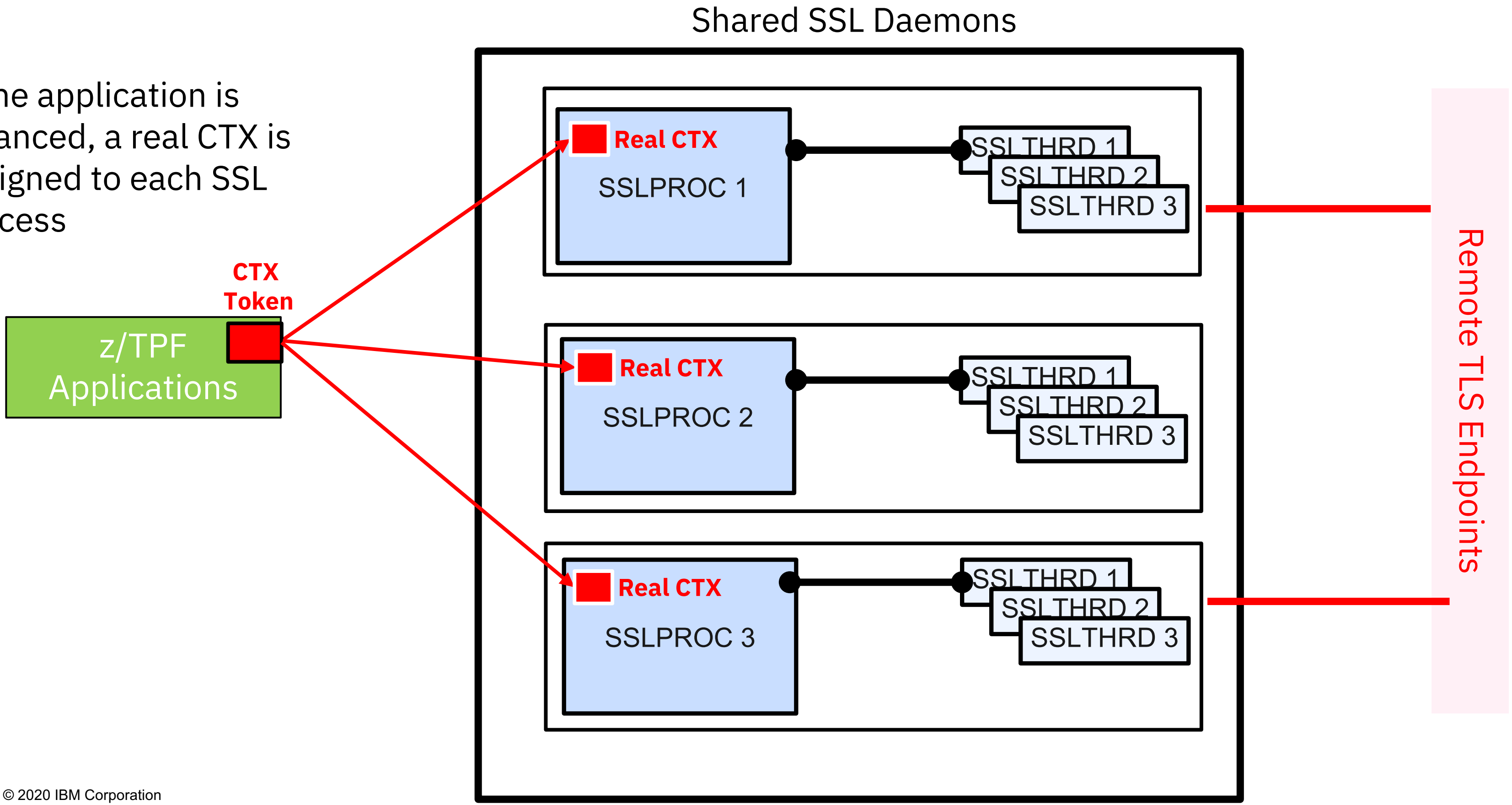
Transport Layer Security (TLS)
z/TPF Shared SSL Daemons – Not Balanced

- Not ideal for applications that can create large amounts of sessions with a single CTX (ie. HTTP Server)



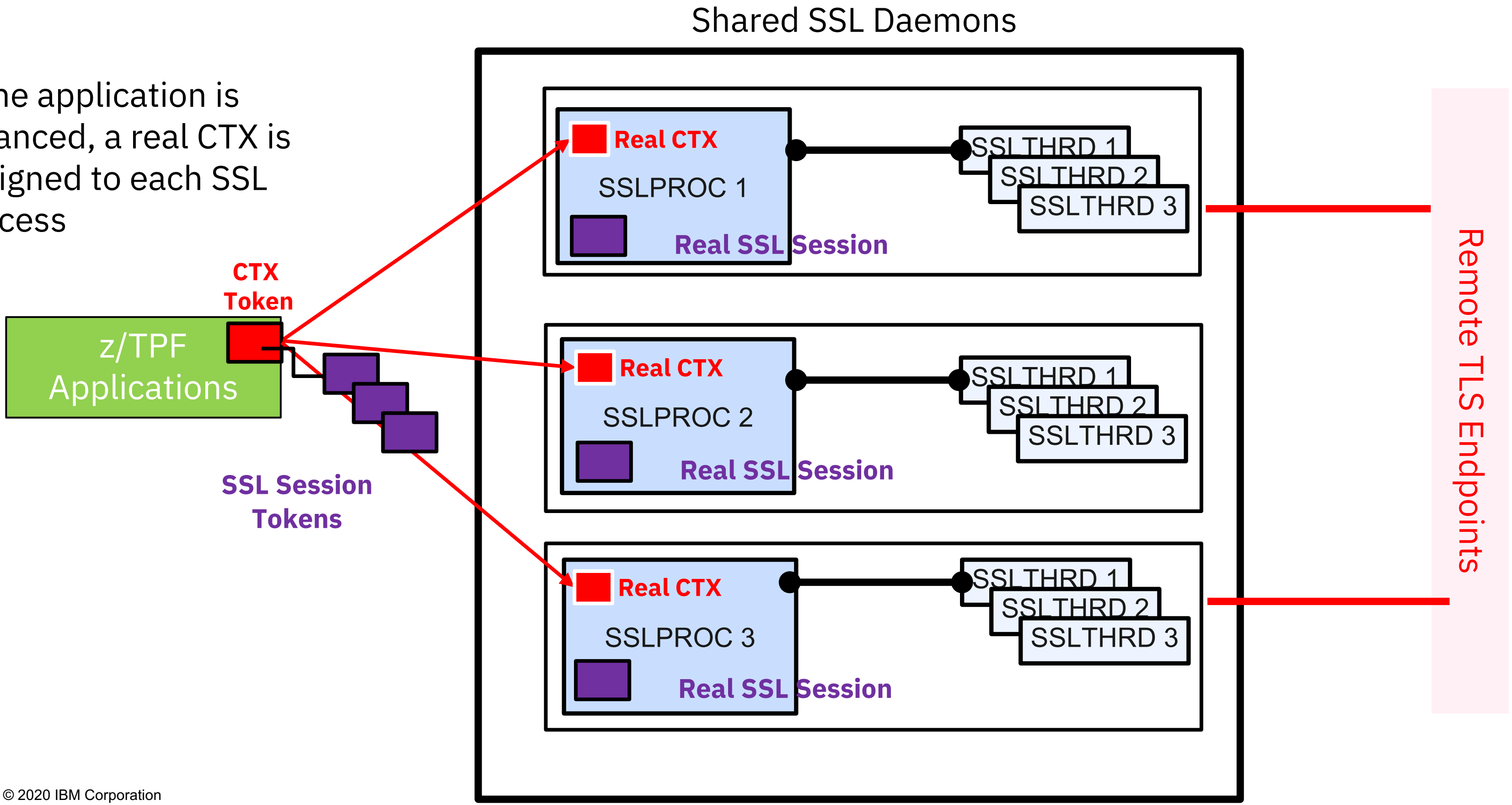
Transport Layer Security (TLS)
z/TPF Shared SSL CTX – Balanced Sessions

- If the application is balanced, a real CTX is assigned to each SSL process



Transport Layer Security (TLS)
z/TPF Shared SSL CTX – Balanced Sessions

- If the application is balanced, a real CTX is assigned to each SSL process



Transport Layer Security (TLS)
z/TPF Shared SSL – Balancing Sessions

- Balancing Sessions
  - The default action when balancing sessions is to assign the CTX to the daemon with the least sessions
    - Fine for a 1:1 CTX to session ratio
    - But when the CTX to session ratio is 1:many, balancing sessions is important
  - Why do you want to balance work across daemons?
    - Memory – each session takes up memory and you don’t want to run a shared SSL process out of memory.
    - Overloading worker threads on a particular process which could effect response time.

Transport Layer Security (TLS)
z/TPF Shared SSL – Daemon Processes and Threads

- The number of shared SSL processes and the threads per process is defined in keypoint 2
  - SSLPROC and SSLTHRD

- Tuning the number of SSLPROC
  - Dependent on memory usage for SSL sessions.

- Tuning the number of threads is based on the number of istreams
  - When large amount of connections are received (SSL_accept or SSL_connect)
    - Half the threads are used to service connection requests (and are blocked)

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
    - Shared SSL Support
- The z/TPF Keystore
    - Secure Symmetric Cryptography
    - Secure Public Key Cryptography

z/TPF Keystore
What is it?

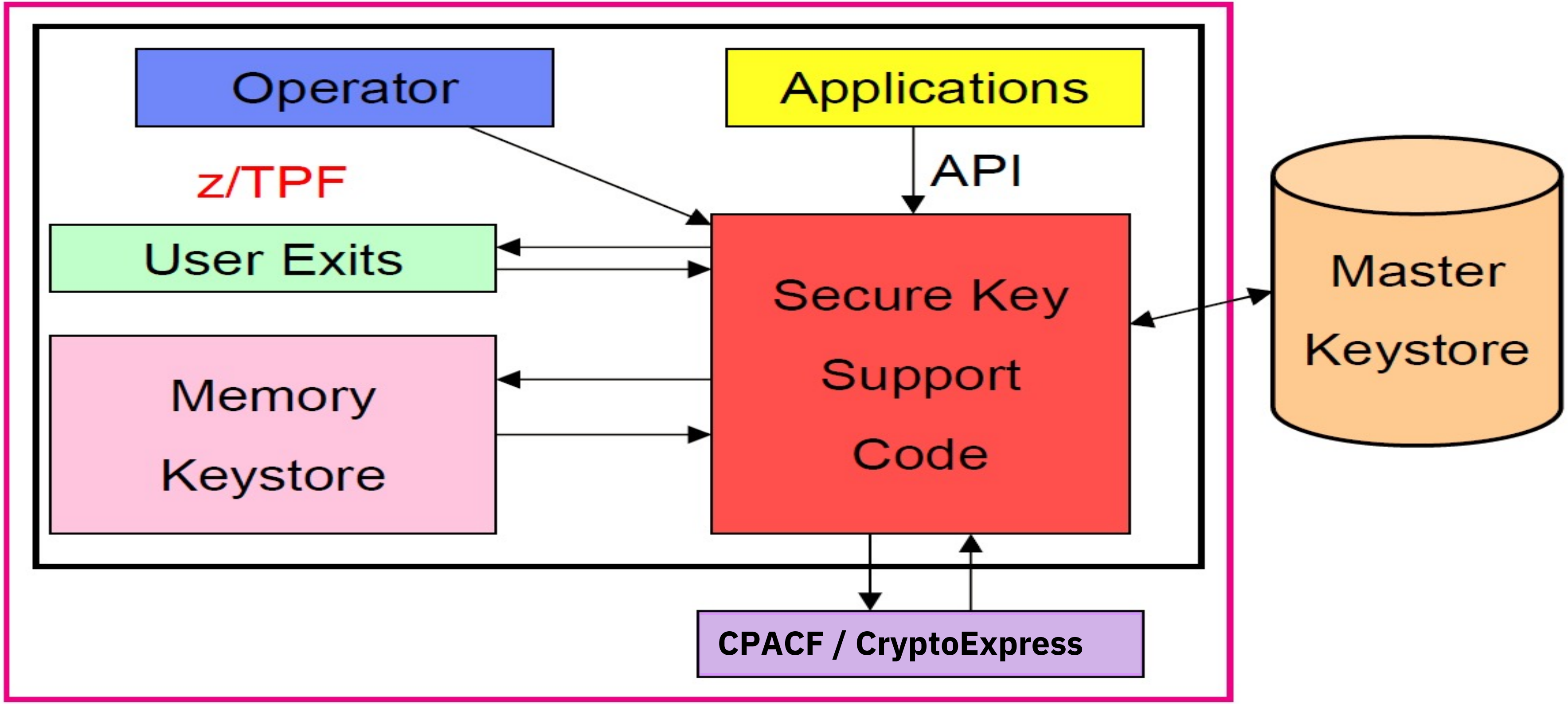
- Enables you to create and manage symmetric encryption keys in a secure manner (ie AES)
  - Applications can use the support to protect sensitive data
    - stored on tape or disk (data at rest)
    - flowing over the network (data in flight)

- Enables you to create and manage public-private keypairs in a secure manner (RSA Keys)
  - Applications can use private keys from the z/TPF keystore (never in the clear)

- High performance designed for mainline application use

- Secure backup and restore capability

z/TPF Keystore
Component View



z/TPF Keystore
Secure Key Components

- **Master Keystore**
  - Persistent copy of encryption/decryption keys on DASD
  - Shared by all processors in the complex

- **Memory Keystore**
  - Copy of the master keystore information in memory on each CPU
  - Exists for performance reasons

- **Operator Interface**
  - Commands to create/activate/change keys, display keystore information, backup/restore keystore information

- **Application Interface**
  - APIs to encrypt and decrypt data using secure keys
  - API to add a key to the keystore

- **User Exits**
  - Control and log key usage (encrypt/decrypt data APIs)
  - Control and log keystore adds (add key API)

Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Secure Symmetric Cryptography
What is it?

- **Create / Import Symmetric Keys**
  - AES128, AES256, TDES, DES

- **Secret keys are assigned names**
  - Encryption key name – used to encrypt data
  - Decryption key name
    - Returned from encrypt operations to be used for decrypt
    - Allows for changing keys without application changes

Secure Symmetric Cryptography
Defining Resources

- Determine how many secure keys you will need
- **Master Keystore**
  - Define #IKEYS fixed file records in the FACE table (FCTB)
    - Formula for how many records are needed based on the number of keys is in the z/TPF documentation
  - Load the new FCTB

- **Memory Keystore**
  - Define the size (number of entries) in Keypoint C (CTKC) KEYSENT parameter on the SKEYS macro in SIP CTKC is processor shared

- Defining the keystore enables secure key management support - sizes of the master keystore and memory keystore can be different

- Memory keystore must be large enough to hold all the keys defined in the master keystore

Secure Symmetric Cryptography

Creating and Activating Keys

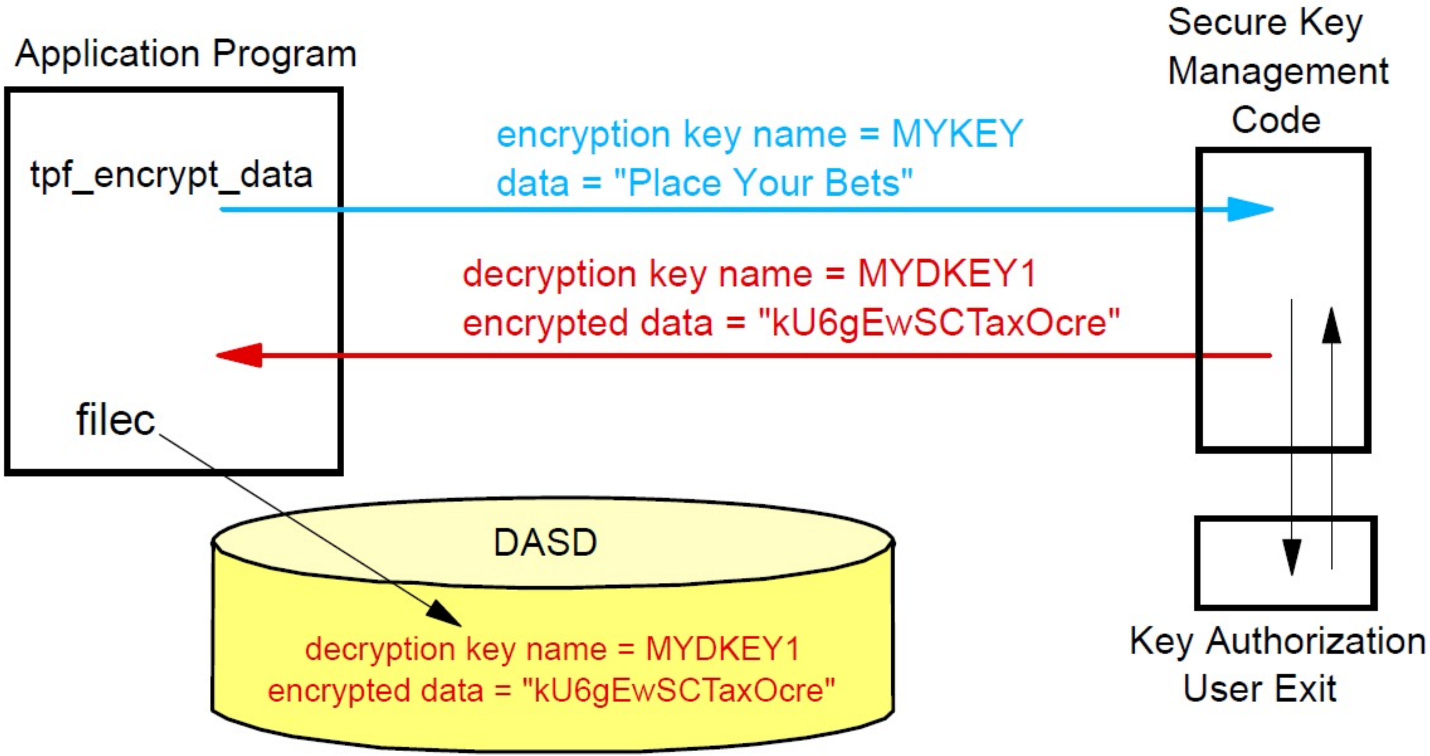
```
ZKEYS GENERATE ENC-MYKEY DEC-MYDKEY1 CIPHER-AES256 NEW
KEYS0002I 08:14:31 KEY ENC-MYKEY DEC-MYDKEY1 GENERATED AND ADDED TO MASTER KEYSTORE
KEYS0003I 08:14:31 KEY ENC-MYKEY DEC-MYDKEY1 ADDED TO MEMORY KEYSTORE ON ALL PROCESSORS
```

```
ZKEYS ACTIVATE ENC-MYKEY DEC-MYDKEY1
KEYS0006I 08:19:31 KEY ENC-MYKEY DEC-MYDKEY1 IS NOW ACTIVE IN MASTER KEYSTORE KEYS0007I
08:19:31 KEY ENC-MYKEY DEC-MYDKEY1 IS NOW ACTIVE IN MEMORY KEYSTORE ON ALL PROCESSORS
```

Keystore entry exists with the following information:

```
Encryption Key Name Decryption Key Name Active Cipher        Secret Key
------------------- ------------------- ------ ------        ----------
MYKEY               MYDKEY1                YES AES256        “KEY1”
```

Secure Symmetric Cryptography
Data Encryption Example



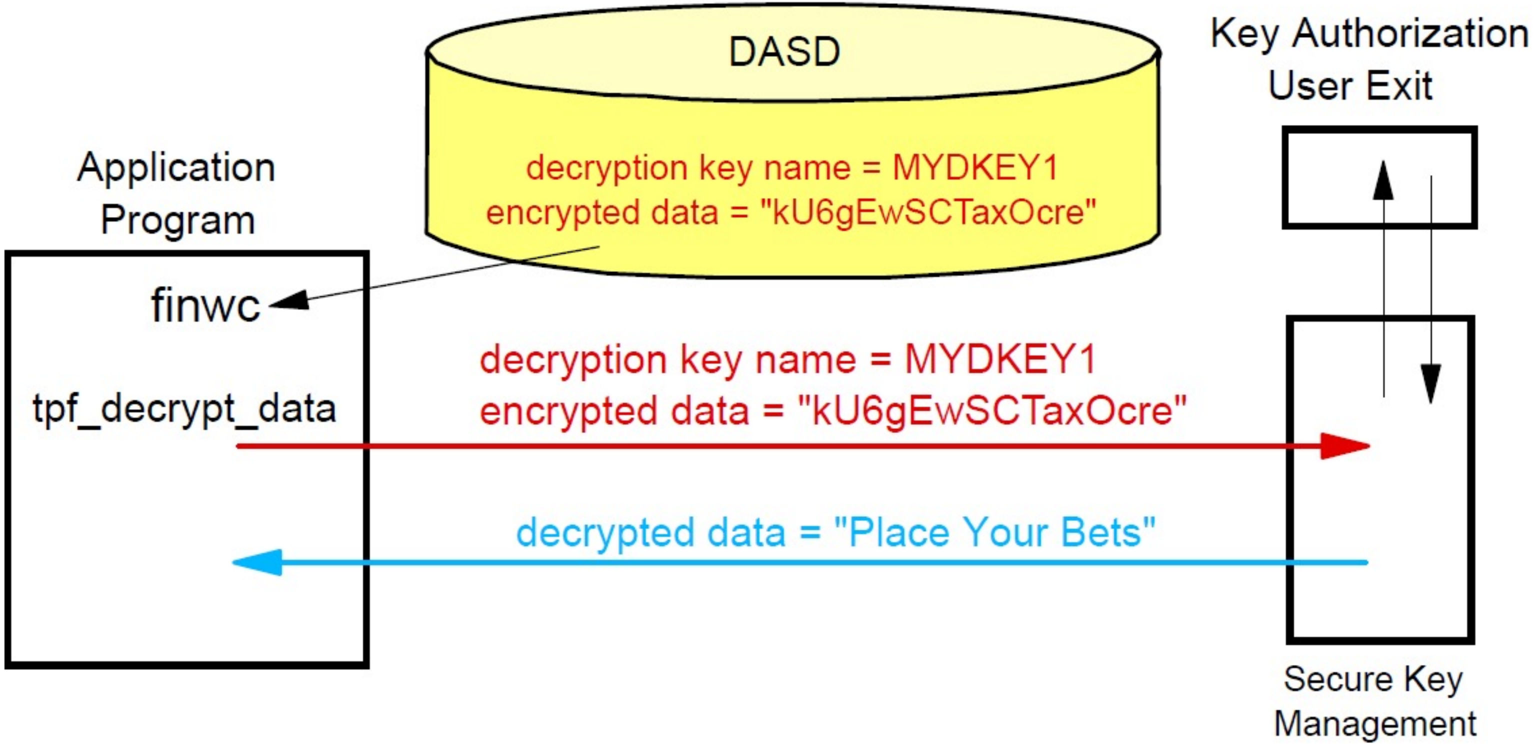
Secure Symmetric Cryptography
Secure Key Encrypt APIs

```
int tpf_encrypt_data(char *encrypt_key_name,
                     char *data, int data_length,
                     char *encrypt_buffer,
                      char *icv_ptr, char *decrypt_key_name);
```

```
int tpf_decrypt_data(char *decrypt_key_name,
                     char *data, int data_length,
                     char *decrypt_buffer,
                     char *icv_ptr);
```

```
char* encrypt_key_name = malloc(8);
char* data = malloc(sizeof ("Place Your Bets"));
int data_length = sizeof(data);
strcpy (encrypt_key_name,"MYKEY");
strcpy (data,"Place Your Bets");
rc = tpf_encrypt_data(encrypt_key_name, data,
                      data_length, data, NULL,
                      decrypt_key_name);
if (rc != 0)
    printf("Encrypt Failed\n");
```

Secure Symmetric Cryptography
Changing Keys Example – Decrypting Data Example

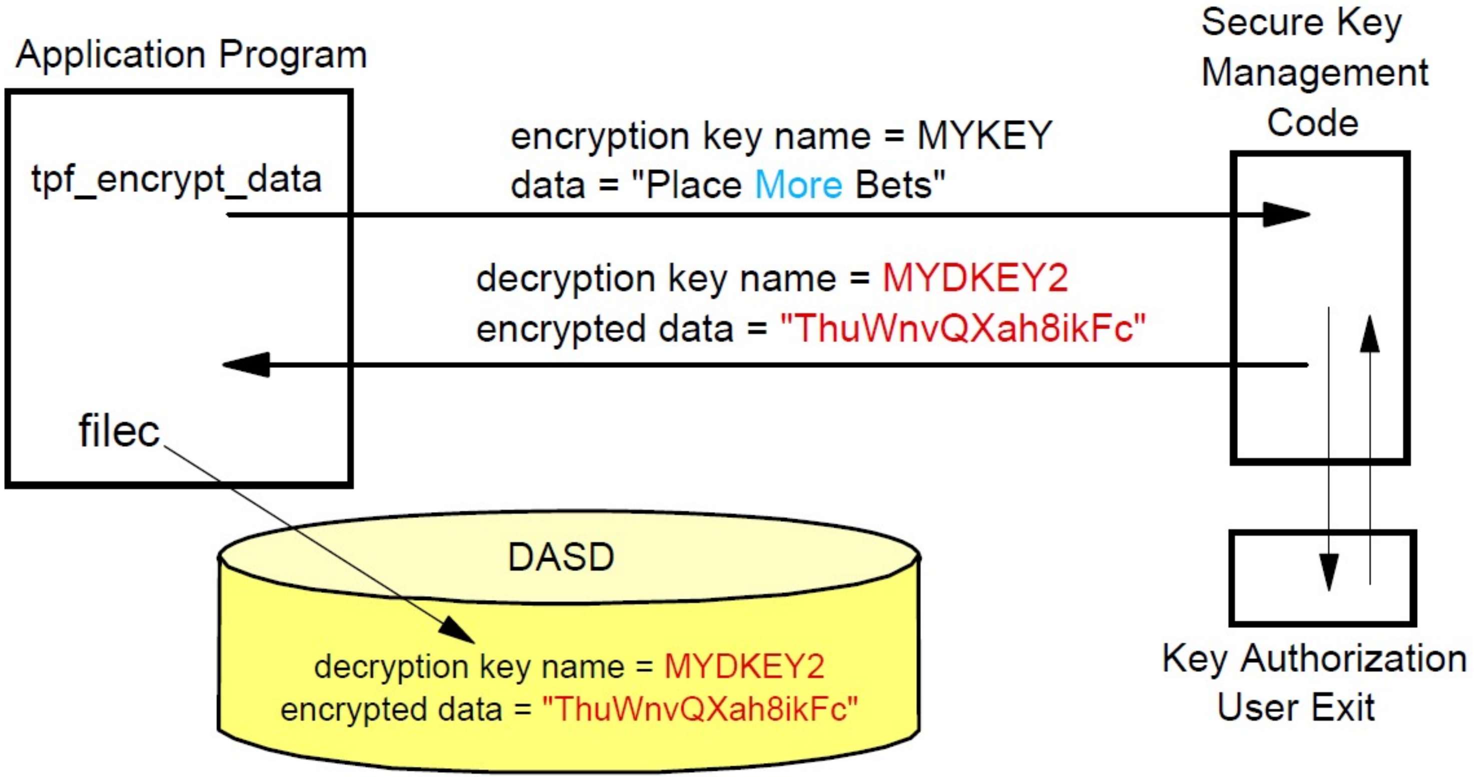


Secure Symmetric Cryptography
Changing Keys

- Data was encrypted using "KEY1" in the previous example
- A new key is created and activated that changes the key value used to encrypt data with encryption key name MYKEY from "KEY1" to "KEY2“
- The keystore now contains two entries:

Encryption Key Name	Decryption Key Name	Active	Cipher	Secret Key
MYKEY	MYDKEY1	NO	AES256	“KEY1”
MYKEY	MYDKEY2	YES	AES256	“KEY2”

Secure Symmetric Cryptography
Changing Keys Example



Agenda

- Securing Data on TPF
- Asymmetric vs Symmetric Encryption
- Cryptographic Hardware Acceleration
- Transport Layer Security (TLS)
  - Shared SSL Support
- The z/TPF Keystore
  - Secure Symmetric Cryptography
  - Secure Public Key Cryptography

Secure Public Key Cryptography
Secure PKI Keystore

- Allows you to create, manage, and use RSA key pairs in a secure manner
  - Extends z/TPF secure key management support
- Supports 1024-bit and 2048-bit RSA key pairs
- Key pair is referenced by name
- Private key value is secured – not visible to operators, applications, coverage, and so on
- Public key value is available to anyone

Secure Public Key Cryptography
Defining Resources

- Determine how many PKI keys you will need
- **Master Keystore**
  - Define #IPKI fixed file records in the FACE table (FCTB)
    - Formula for how many records are needed based on the number of keys is in the z/TPF documentation
  - Load the new FCTB

- **Memory Keystore**
  - Define the size (number of entries) in Keypoint C (CTKC) PUBKENT parameter on the PKEYS macro in SIP CTKC is processor shared

- Defining the PKI keystore requires the Secure Symmetric Keystore is enabled (KEYSENT)

- PKI key management supports Sizes of the master keystore and memory keystore can be different
  - Memory keystore must be large enough to hold all the keys defined in the master keystore

Secure Public Key Cryptography
Generating Secure PKI Keys

- Use the ZPUBK GENERATE command
  - Specify the name of the key pair (for example, KEYPAIR1) which is how subsequent operator commands and APIs will reference/use this key pair
  - Specify the key length (1024-bit or 2048-bit)
- RSA key pair is created and added to the PKI keystore
- Issue additional operator commands to backup the keystore and activate the key pair making it available for use

Secure Public Key Cryptography
Creating Certificates

1. z/TPF operator creates an RSA key pair called KEYPAIR1
2. Create a file (called myinfo.cfg in this example) containing the subject information needed to create a certificate request
3. Issue the ZPUBK REQCERT command. Input includes:

   Key pair name (KEYPAIR1) that says which public key to use

   Name of the file (myinfo.cfg) containing the subject information

   Name of the file (mycert.fil) into which to build the certificate request (PKCS #10 format)
4. Send (FTP) the certificate request to the CA who will create the digital certificate
5. From the CA, send (FTP) the certificate to your z/TPF system

***Ability to create self signed certificates for testing**

Secure Public Key Cryptography
Solving OpenSSL Private Key Concern

- **OpenSSL programming model cannot be changed**
  - Would break lots of middleware built on top of OpenSSL
- **SSL applications will be able to use an RSA key pair generated by z/TPF**
- **Application program still uses standard OpenSSL APIs to indicate the name of the file that contains the private key**
  - If the file name path starts with a special prefix (/tpfpubk), this tells z/TPF that the name that follows is really the name of the RSA key pair to use and not to try and open/use a file in the file system

Secure Public Key Cryptography

Using Secure PKI Keys In SSL

- **z/TPF operator creates an RSA key pair called KEYPAIR1**
- **To use this RSA key pair, an SSL application on z/TPF issues the *SSL_CTX_use_PrivateKey_file* API with the private key file name set to:**
  - /tpfpubk/keypair1.pem
- The private key value is not copied into the application program's memory space
- Only the key pair name (KEYPAIR1) is saved in the SSL structure in the application's memory space

Secure Public Key Cryptography
Secure PKI Summary

- **The Secure PKI Keystore allows for**
  - **Securely start OpenSSL Sessions**
  - **Import Symmetric Keys Securely**
    - tpf_secure_key_import
  - **User can encrypt / decrypt data using RSA keys**
    - tpf_RSA_encrypt_data
    - tpf_RSA_decrypt_data
  - **User can create / verify digital signatures**
    - tpf_RSA_sign
    - tpf_RSA_verify

Trademarks

Notes

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.