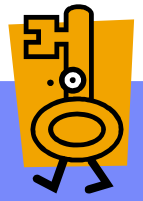
IBM Software Group

z/OS Cryptography demystified

Presented on

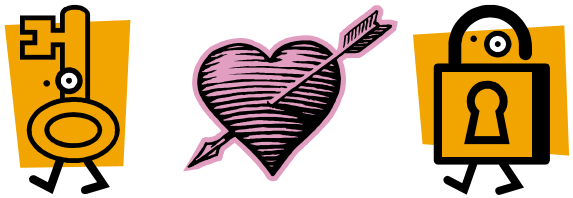
5th November at GSE UK Conference

Lennie Dymoke-Bradshaw
IBM, SWG



Contents

- **Why do we use cryptography?**
- **What is cryptography?**
- **What types of cryptography are there?**
- **How can cryptography be performed?**
- **What about cryptography on z/OS?**

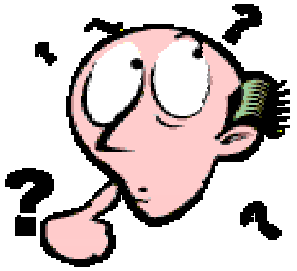


Why cryptography?

- **Regulation and Compliance**
  - PCI – Payment Card Industry
    - Standards for all organisations
    - Debit Cards and Credit Cards
  - Sarbanes-Oxley (SOX), etc. etc.
- **Offsite backups**
  - Once data is offsite, RACF cannot protect it.
  - Do tapes fall off your lorries?
- **Secure messaging and Non-repudiation**
  - Needs certificate management system.
  - Public Key Infrastructure (PKI).
- **Money management**
  - ATMs, PINs, Chip & PIN cards.
- **Banking and eCommerce on the internet**
  - Secure signon.

What is cryptography?

- **Oxford English Dictionary says**
  - –cryptography is,
    - "The art of writing or solving ciphers"
      - From the Greek *kryptós* meaning hidden
  - –A cipher is
    - "a secret or disguised way of writing"



What is cryptography?

- **In computing terms we associate cryptography with the use of KEYs.**
- **Data is hidden by using**
  - A **KEY**
  - An algorithm
- **A KEY is a string of binary data used to modify the original data via the algorithm.**
- **The aim is to disguise data so that only the chosen few can access it.**
- **The algorithm is public, the key is private**

Crypto example diagram

X'4C656E6E69652020' ***DES*** X'0102030405060708' = X'71AC9B8B4BBBCE13'

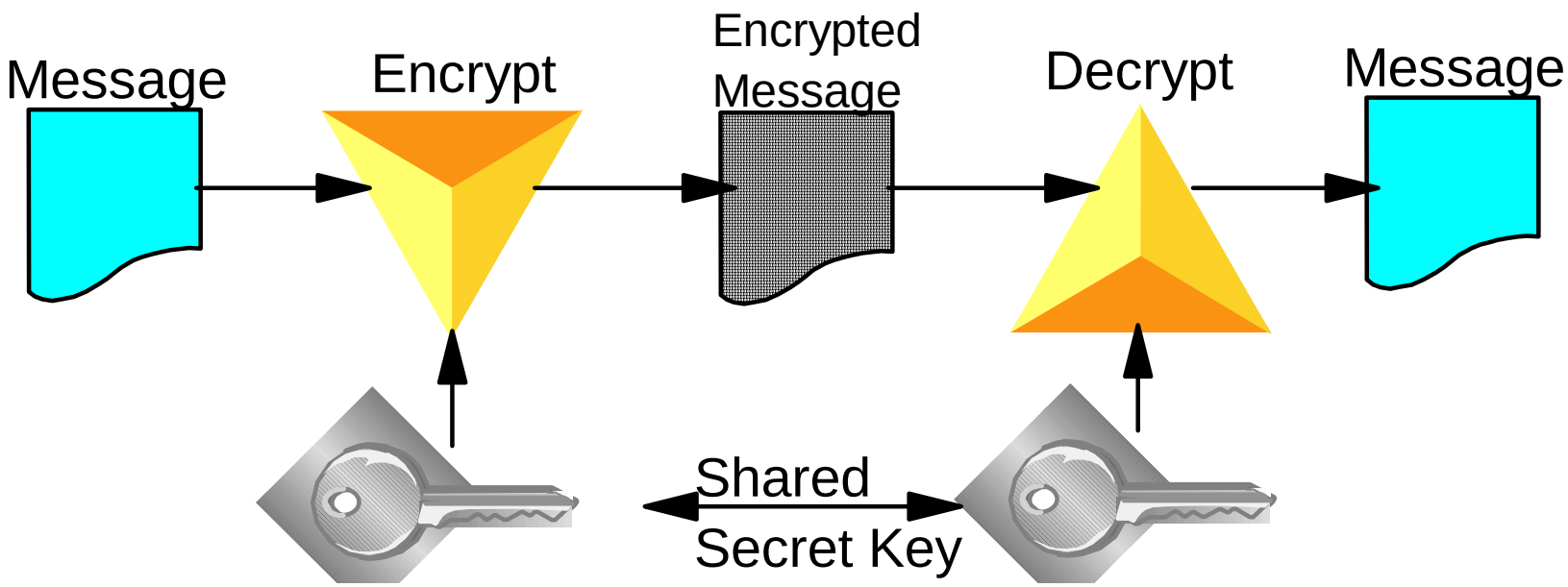
- **In this case**
  - Message is "Lennie " which is X'4C656E6E69652020'
  - Algorithm is DES
  - Key is X'0102030405060708'
  - Encrypted message is X'71AC9B8B4BBBCE13'

Note: Text in this example is in ASCII, and is padded with 2 blanks

Types of cryptography

- **Symmetric cryptography**
  - Such as DES, TDES, RC4, AES
- **Asymmetric cryptography**
  - Such as DSA, RSA, Diffie-Hellman, ECDSA (Elliptic Curve Digital Signature Algorithm)
- **Can use both combined**
  - Used in Secure Sockets Layer(SSL)

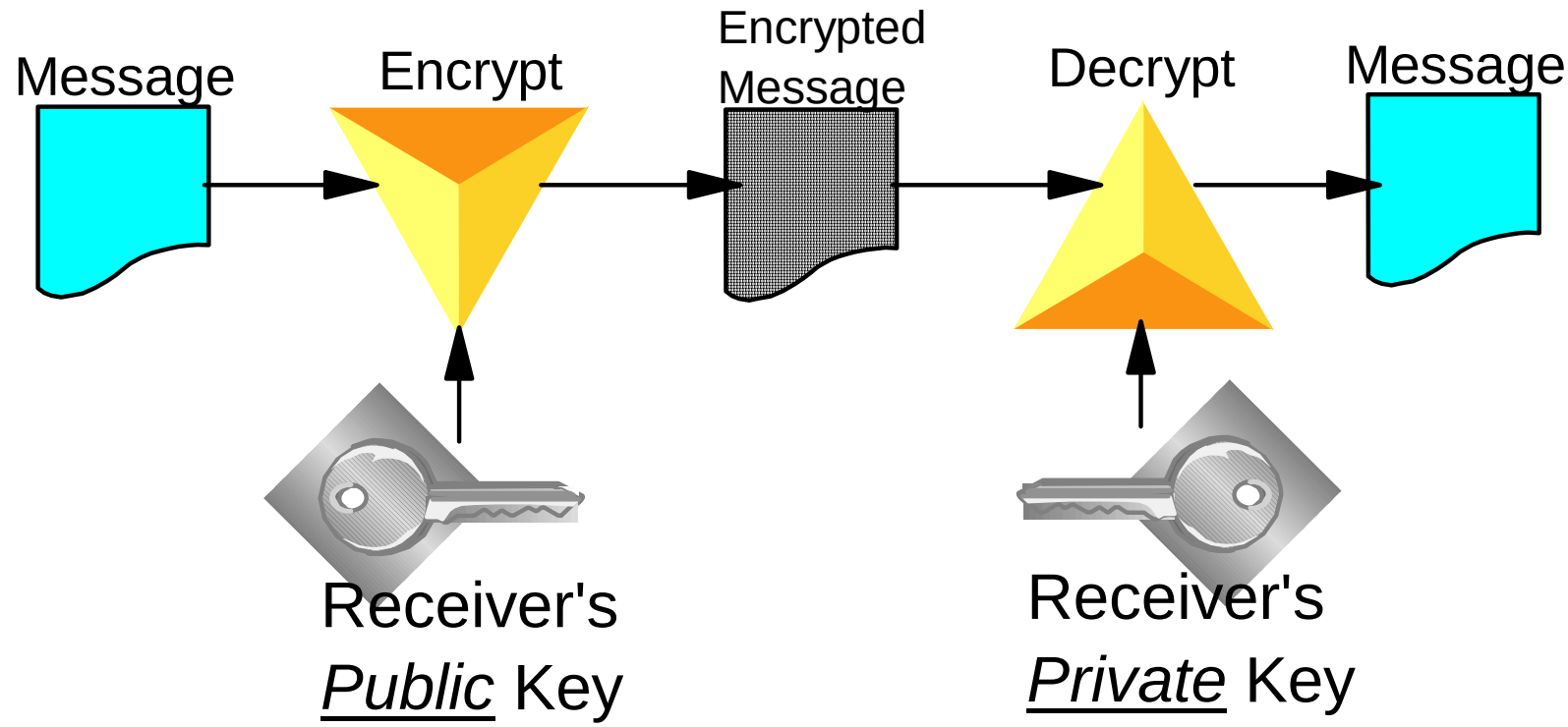
Symmetric Cryptography



Examples:

Data Encryption Algorithm (DEA) also known as Data Encryption Standard (DES)
Triple DES (TDES), which uses DES three times.
Advanced Encryption Standard (AES)

Asymmetric Cryptography



Examples:
Rivest Shamir and Adelman (RSA)
Diffie-Hellman

*Note: Asymmetric cryptography is **much** slower that symmetric cryptography*

SSL

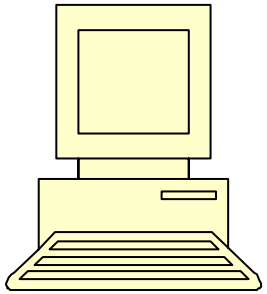
- Secure Sockets Layer
  - Uses Asymmetric encryption to agree a Symmetric key
    - Handshake process
  - Can negotiate Asymmetric algorithms
  - Can negotiate Symmetric algorithms
  - Can have an “abbreviated” handshake for performance
  - Once handshake is complete, the symmetric key is used to encrypt all data that flows.
  - Symmetric key is discarded after use

SSL

- Following slides are simplified
  - Do not use client authentication
  - Many different mechanisms supported by SSL
- SSL supports
  - Authentication of partners
  - Message privacy
  - Message integrity
- Lots of options and encryption algorithms supported
- The following diagrams are vastly simplified

SSL flow - 01

Client



ClientHello

I want a secure connection.

This is a list of the Hash functions and Ciphers I can support,

……

……

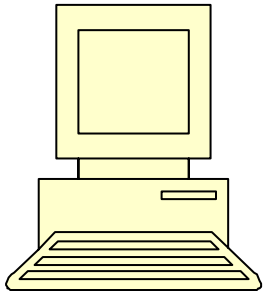
Server

Clients box of Goodies
…. Empty ….

Servers box of Goodies
1.Certificate with Public key
2.Private key

SSL flow - 02

Client



ServerHello

Server

Sounds reasonable….

Here is,

1.My name

2.The CA for my certificate

3.My certificate

4.A random number

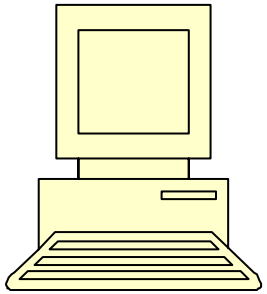
5.A session id

Clients box of Goodies
…. Empty ….

Servers box of Goodies
1.Certificate with Public key
2.Private key
3.Random Number
4.Session id

SSL flow - 03

Client



Client *may* validate the certificate

Clients box of Goodies

1.Servers Certificate with public key

2.Random Number

3.Session id

4.Ciphers and hash method

ServerHelloDone

Sounds reasonable….

Here is,

1.My name

2.The CA for my certificate

3.My certificate

4.A random number

5.A session id

6.The ciphers and hash method I chose

Server

Servers box of Goodies

1.Certificate with Public key

2.Private key

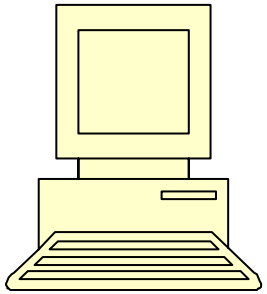
3.Random Number

4.Session id

5.Ciphers and hash method

SSL flow - 04

Client



ClientKeyExchange

Thanks….

Here is a random number I thought of, encrypted under your public key

Lets call this the “Pre-Master Secret”.

Server

Clients box of Goodies

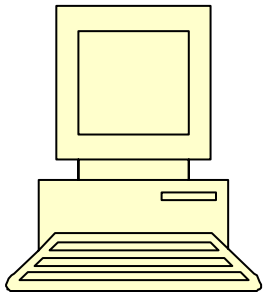
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**

Servers box of Goodies

1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**

SSL flow - 05

Client



Server

Actions take place simultaneously

Client Actions

Generate a Master Secret using

- Random Number
- Pre-Master Secret

Generate an encryption key from the Master Secret

Server Actions

Generate a Master Secret using

- Random Number
- Pre-Master Secret

Generate an encryption key from the Master Secret

Clients box of Goodies

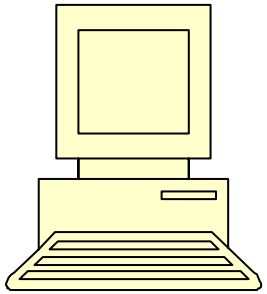
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**
6. **Master Secret**
7. **Encryption Key**

Servers box of Goodies

1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**
6. **Pre-Master Secret**
7. **Master Secret**
8. **Encryption Key**

SSL flow - 06

Client



ChangeCipherSpec

Everything I send after this message will be encrypted with the encryption key

Server

Clients box of Goodies

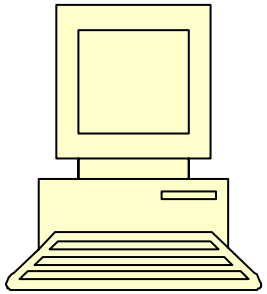
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**
6. **Master Secret**
7. **Encryption Key**

Servers box of Goodies

1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**
6. **Pre-Master Secret**
7. **Master Secret**
8. **Encryption Key**

SSL flow - 07

Client



ChangeCipherSpec

Everything I send after this message will be encrypted with the encryption key

Server

Clients box of Goodies

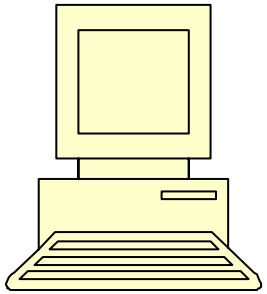
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**
6. **Master Secret**
7. **Encryption Key**

Servers box of Goodies

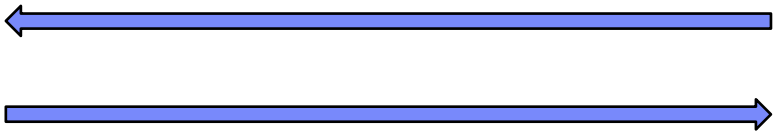
1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**
6. **Pre-Master Secret**
7. **Master Secret**
8. **Encryption Key**

SSL flow - 08

Client



Data flow



Server

Everything sent in each direction is now encrypted under the encryption key which both client and server know.

Clients box of Goodies

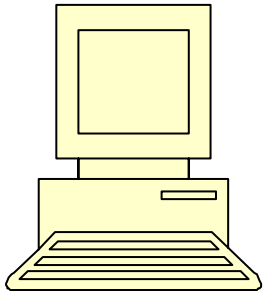
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**
6. **Master Secret**
7. **Encryption Key**

Servers box of Goodies

1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**
6. **Pre-Master Secret**
7. **Master Secret**
8. **Encryption Key**

SSL flow – Session id reuse - 01

Client



ClientHello

Server

I want a secure connection.

This is a list of the Hash functions and Ciphers I can support,

……

……

…. and I have this Session id from our last communication….

Clients box of Goodies

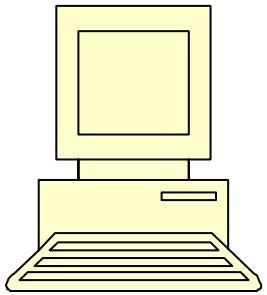
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**
6. **Master Secret**
7. **Encryption Key**

Servers box of Goodies

1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**
6. **Pre-Master Secret**
7. **Master Secret**
8. **Encryption Key**

SSL flow – Session id reuse - 02

Client



Server

Actions take place simultaneously

Client Actions

Generate a Master Secret using

- Random Number
- Pre-Master Secret

Generate a new encryption key from the Master Secret

Server Actions

Generate a Master Secret using

- Random Number
- Pre-Master Secret

Generate a new encryption key from the Master Secret

Clients box of Goodies

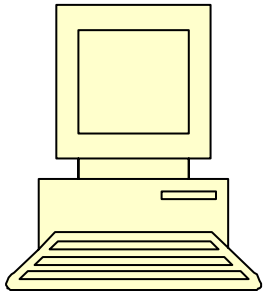
1. **Servers Certificate with public key**
2. **Random Number**
3. **Session id**
4. **Ciphers and hash method**
5. **Pre-Master Secret**
6. **Master Secret**
7. **New encryption Key**

Servers box of Goodies

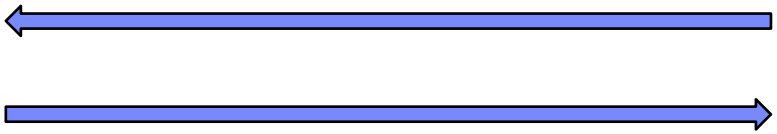
1. **Certificate with Public key**
2. **Private key**
3. **Random Number**
4. **Session id**
5. **Ciphers and hash method**
6. **Pre-Master Secret**
7. **Master Secret**
8. **New encryption Key**

SSL flow – Session id reuse - 03

Client



Data flow



Server

Everything sent in each direction is now encrypted under the new encryption key which both client and server know.

Clients box of Goodies

1. Servers Certificate with public key
2. Random Number
3. Session id
4. Ciphers and hash method
5. Pre-Master Secret
6. Master Secret
7. New encryption Key

Servers box of Goodies

1. Certificate with Public key
2. Private key
3. Random Number
4. Session id
5. Ciphers and hash method
6. Pre-Master Secret
7. Master Secret
8. New encryption Key

IBM Software Group

Cryptography on z/OS

(including some new stuff!)

A basic primer

Cryptography on z/OS

1. **Can be performed in software**
   - SSL uses software crypto in some cases
   - Java crypto packages
2. **Can used specialised machine instructions**
   - CPACF assembler instructions
   - z990 processors onwards
3. **Can use specialised crypto processors**
   - PCIXCC or Crypto Express 2
   - Follow “Common Cryptographic Architecture”
   - Managed using **ICSF**
   - Can be used from JAVA
4. **Can be used in peripherals**
   - TS1120 and TS1130tape drives
   - DS8000 FDE
5. **Can use 3rd party devices**
   - Thales, Atalla, etc.

Cryptography on z/OS

z10 EC, Z9 EC, z10 BC and z9 BC use

- CPACF (CP Assist for Cryptographic Functions)
  - CPACF has more functions on later processors
- Crypto Express 2 configured as
  - Crypto Engine (CEX2C), or
  - Crypto Accelerator (CEX2A)
- Crypto Express 3 configured as
  - Crypto Engine (CEX3C), or
  - Crypto Accelerator (CEX3A)
- Software
  - System SSL
  - RACF
  - PKI Services

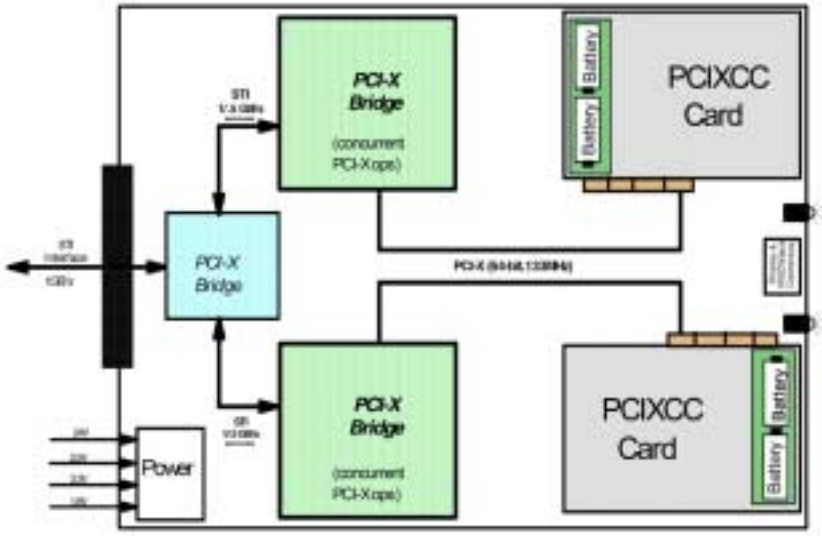
Cryptography on z/OS

- **CPACF**
  - Set of machine instructions available on every GP and speciality engine
  - Also known as “Message Security Assist” or MSA instructions
  - Perform
    - DES and AES encryption
    - Hashing
    - Random Number Generation
    - MAC generation
  - Clear key Operations

Cryptography on z/OS

Crypto Express 2

- **Feature Code 0863**
- **Uses 4764 processor**
  - Linux operating system
- **Top hardware rating**
  - FIPS 140-2 Level 4 certified
- **Each feature contains 2 crypto processors**
- **Connects in I/O cage**
- **Contains signed code with certificate**
- **Tamper-proof hardware**
  - Destroys keys if attacked
- **Conforms to IBM's CCA**
  - Common Cryptographic Architecture
- **Can be configured as crypto accelerator (CEX2A)**



Cryptography on z/OS – *NEW!*

Crypto Express 3

- **Feature Code 0864**
- **Uses 4765 processor**
  - Linux operating system
- **Top hardware rating**
  - FIPS 140-2 Level 4 certified
- **Each feature contains 2 crypto processors**
  - Also can get a Crypto Express 3 1-p
- **Connects in I/O cage**
- **Contains signed code with certificate**
- **Tamper-proof hardware**
  - Destroys keys if attacked
- **Conforms to IBM's CCA**
  - Common Cryptographic Architecture
- **Can be configured as crypto accelerator (CEX3A)**

Major Changes

- **32 domains (up from 16)**
- **Protected key support**
- **Enhanced temperature tolerance**
- **PCI-e interface (previously PCI-x)**
- **Improved RAS features**
- **Improved performance**
- **Concurrent code updates**

Cryptography on z/OS

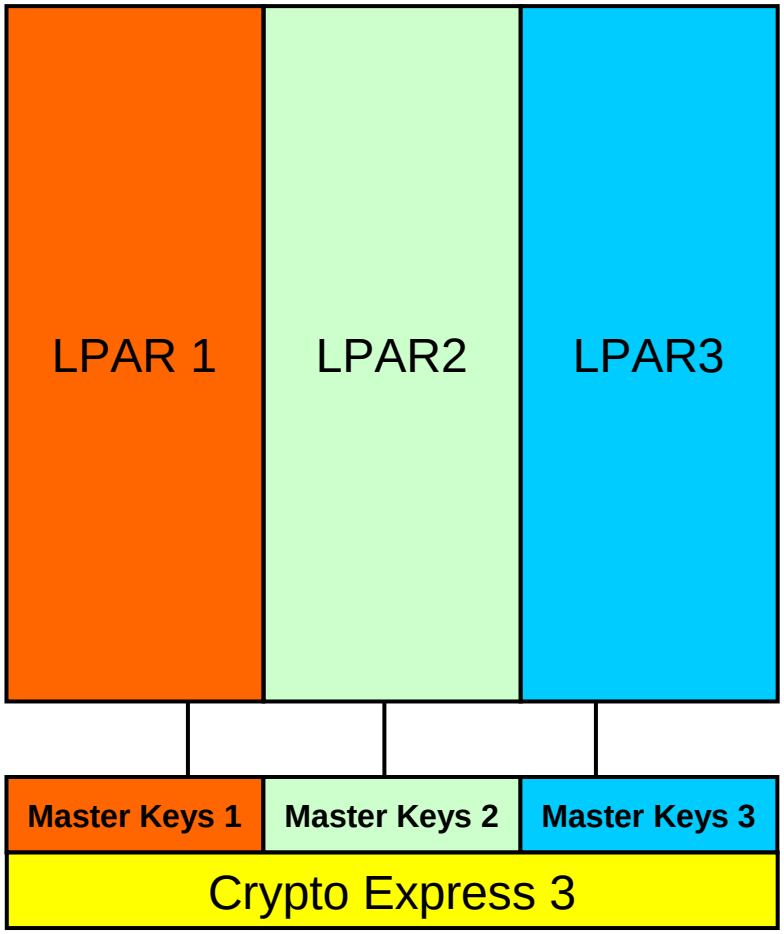
- **Clear Key**
  - Key is exposed in the storage of processor
  - Can be viewed in dump of storage
  - If correctly interpreted can expose data
  - Sometimes acceptable
    - for short-lived keys
    - with other constraints
- **Secure Key**
  - Key is only ever exposed in bounds of secure processor
  - Can never be seen in storage
  - Dump will not reveal key
  - Key is held encrypted under Master key

NEW – Protected keys (z10 only)

- **Clear Key**
  - Key is exposed in the storage of processor
  - Can be viewed in dump of storage
  - If correctly interpreted can expose data
  - Sometimes acceptable
    - for short-lived keys
    - with other constraints
- **Protected Key**
  - Key is not exposed in the storage of processor
  - Key is in clear outside of tamper-proof device
  - Can never be seen in dump of storage
  - Dump will not reveal key
  - Key is held in storage encrypted under a Wrapping Key
  - Hence called a Wrapped key.
- **Secure Key**
  - Key is only ever exposed in bounds of secure processor
  - Can never be seen in storage
  - Dump will not reveal key
  - Key is held encrypted under Master key

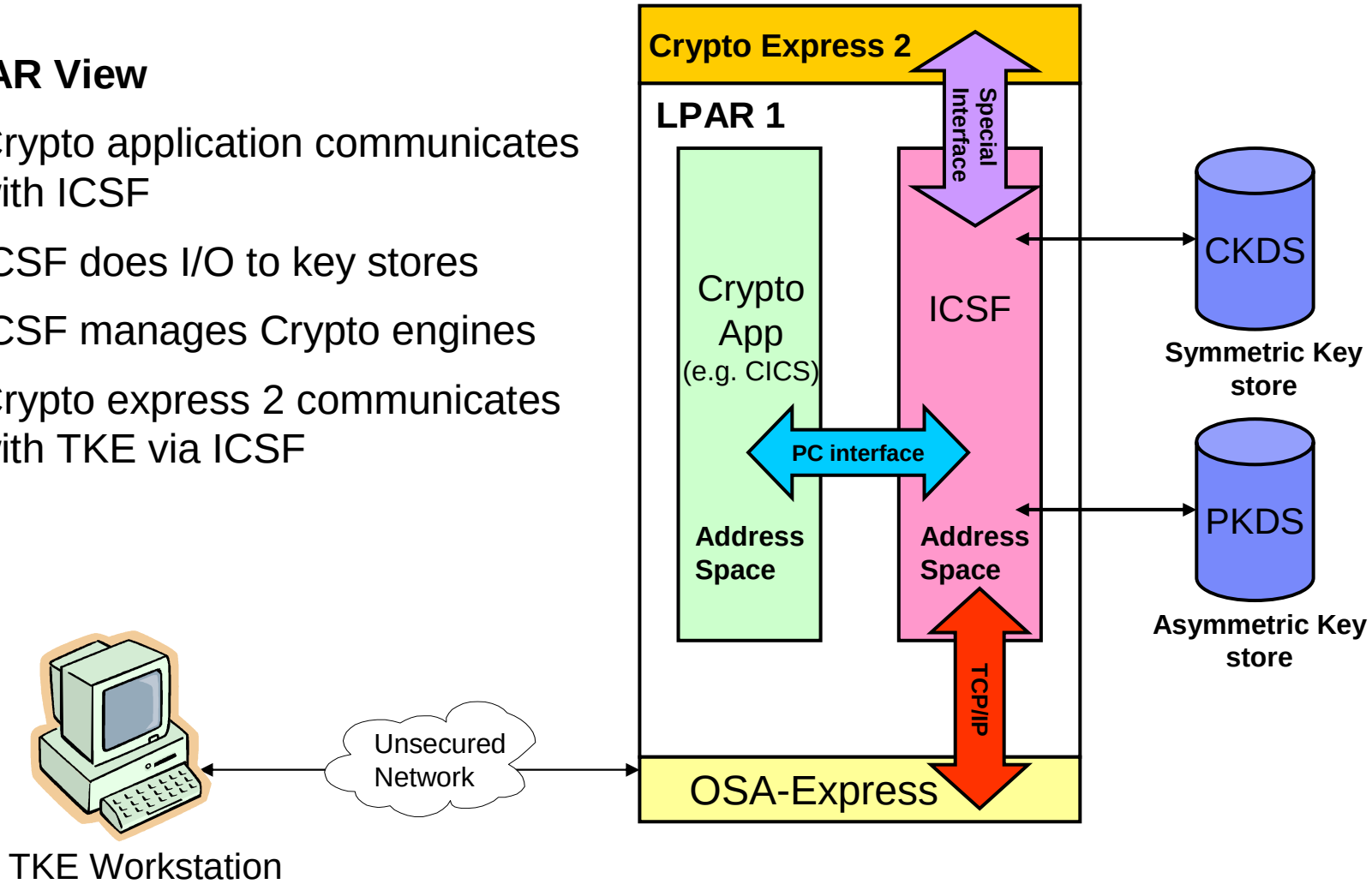
Cryptography on z/OS

- **z10 Processor view**
  - Each LPAR has own set of Master keys
  - Requests for Crypto processing queued into processor
  - Crypto Express services requests on FIFO basis.
  - Link to Crypto Express 3 is asynchronous
  - 32 domains per crypto engine
  - Can have multiple engines per LPAR



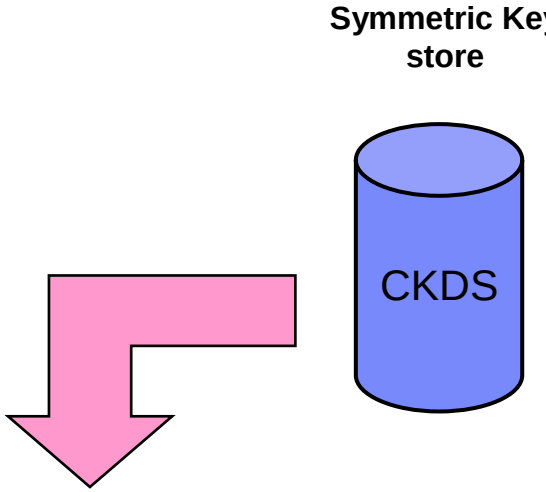
Cryptography on z/OS

- **LPAR View**
  - Crypto application communicates with ICSF
  - ICSF does I/O to key stores
  - ICSF manages Crypto engines
  - Crypto express 2 communicates with TKE via ICSF



Cryptography on z/OS

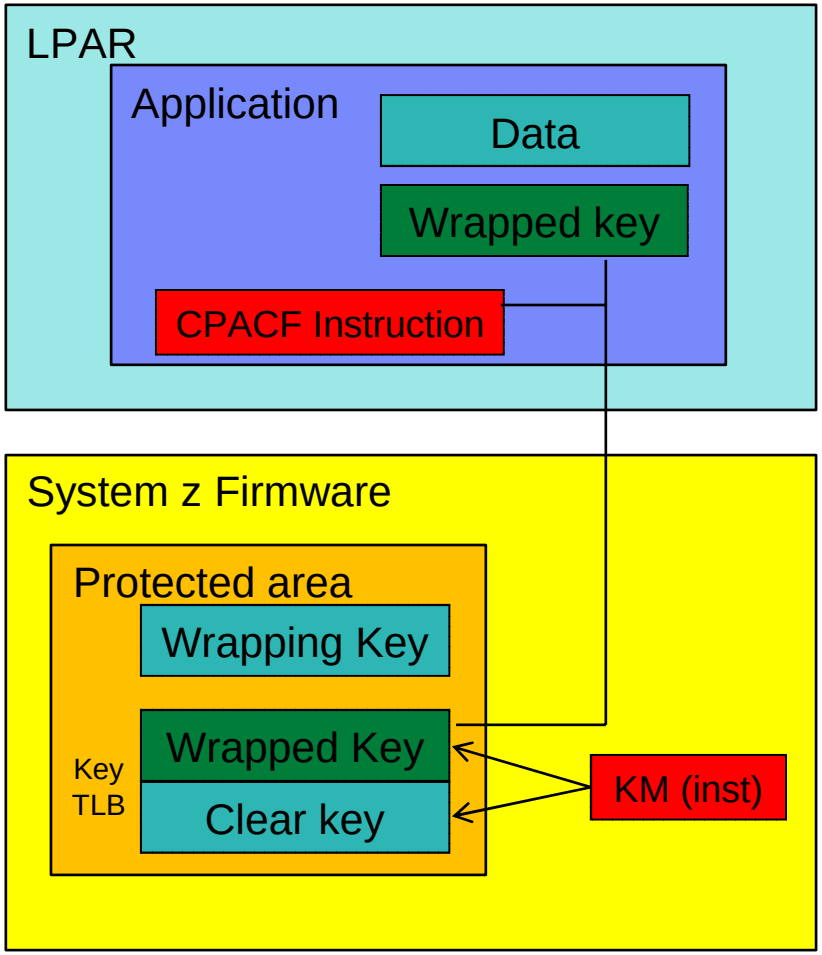
- **Master keys**
  - Symmetric master key
  - Asymmetric master key
  - Both are TDES
  - Used to encrypt keys held on CKDS and PKDS
  - CKDS and PKDS are VSAM datasets
  - Programs refer to keys by labels
  - Keys only ever exposed with Crypto Express 2



Label	Key
MYDESKEY01	Value of key – (Encrypted using Symmetric Master key)
YOURDESKEY01	Value of key – (Encrypted using Symmetric Master key)

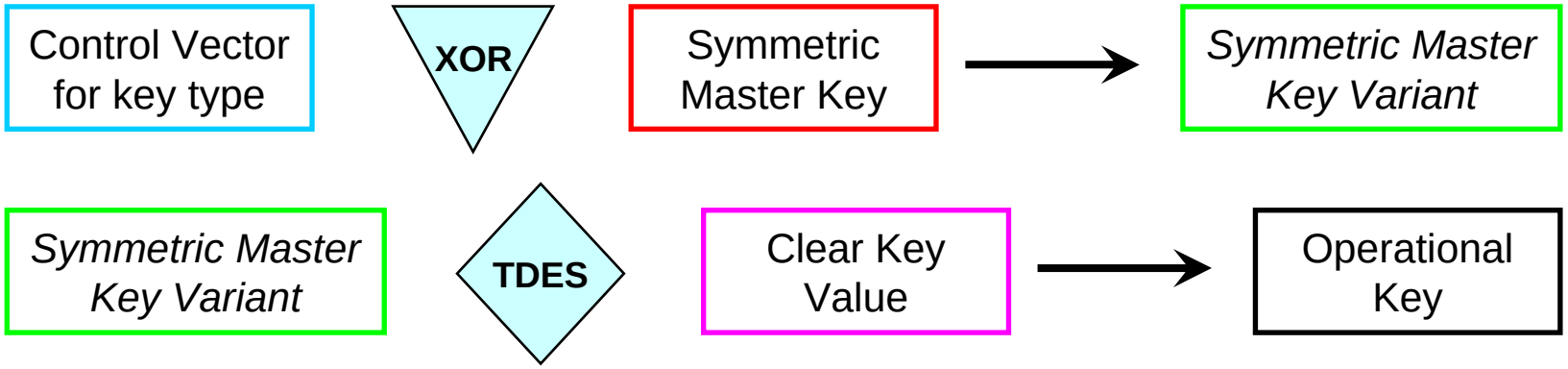
Protected keys – How do they work?

- **Wrapping key is derived**
  - At each LPAR activation
- **ICSF provides**
  - New API to extract a Secure key from CKDS, and
  - Converts to a Wrapped key
  - Needs Crypto Express 3
- **System z firmware**
  - Is part of HSA
  - Hold Wrapping key
  - Holds clear key value
  - Not visible to operating system or applications
- **CPACF**
  - New Function codes
  - New instruction PKCMO (privileged)



Cryptography on z/OS

- **Key Separation**
  - Each key in the CKDS and PKDS is encrypted by a master key
  - Before the master key is used to encrypt it a “Control Vector” is applied to the master key.
  - Control Vectors allow us to enforce specific “roles” for keys. e.g.
    - Data keys – for encrypting data
    - MAC keys – for producing Message Authentication Codes
    - Transport keys – used to exchange keys with other systems
    - PIN generation keys – used to generate PIN numbers



Cryptography on z/OS

- **ICSF does the following**
  - –manages Crypto engines
    - Online & Offline status
    - Reports status
    - Provides RMF data
  - –Provides access to the keys stores
    - CKDS and PKDS
  - –Provides the APIs for crypto services
    - Many and varied!
  - –Interfaces with Trusted Key Entry (TKE) workstation

ICSF runs as a started task on each MVS system in the sysplex

Cryptography on z/OS

- **ICSF APIs**
  - Symmetric Key management
    - Creating, exporting, importing, storing, etc.
  - Protecting data
    - Encipher, Decipher, etc.
  - Verifying data
    - Hash processing
    - MAC processing
  - Financial processing
    - PIN manipulation to industry standards
  - Digital Signatures
  - PKA (Asymmetric) Key management
    - Creating, exporting, importing, storing, etc.

Most ICSF APIs are callable services with names starting
CSNB…..
CSND…..

Note: There are over 100 APIs, each with many parameters. (Min 6, Max 21)

RACF checks in ICSF – basic checks

- **Check access to each service (API)**
  - Issues a RACF check in CSFSERV class
  - All APIs have a CSF…. resource name
  - Can protect sensitive APIs
- **Check access to each KEY**
  - Uses LABEL as a resource name in CSFKEYS class
  - Each label is a 64 byte name

RACF checks in ICSF – New with HCR7751

- **Key store policies**
  - –New resources in XFACILIT class to enable new function
  - –Access checked when token in place of label
  - –Duplicate token checking
    - Prevents storing of same key under duplicate labels
  - –Default token
    - Used for a token which has no matching label
  - –Granular key access
    - Uses an access level (READ, UPDATE, CONTROL) when checking access to a key
  - –Controls on key export
    - Uses XCSFKEY class

New ICSF for 2009 – HCR7770

- **Crypto Express 3 support**
- **Protected key support**
  - Also needs z10 GA3
- **Elliptical Curve cryptography**
  - designed to comply with NIST requirements to support a FIPS 140-2 mode of operation for IPSEC.
- **Extended PKCS#11 support**
  - new software cryptographic engine embedded in ICSF will allow PKCS11 processing even if no cryptographic coprocessors are available.
- **Performance improvements**
  - and ICSF now runs non-swappable and non-cancellable
- **New Query algorithm**
  - Supplies details on supported cryptography algorithms

Summary

- **Cryptography can “hide” data**
- **Cryptography uses “Keys”**
- **Keys can be Clear, Protected or Secure**
  - Secure is better, but slower
- **Symmetric and Asymmetric crypto**
  - Symmetric is faster
- **SSL uses both types of cryptography**
- **Crypto can be hardware or software driven**
- **Crypto Express 3 is the current secure key processor for System z**
- **ICSF performs 3 major functions**
  - manages crypto devices on z/OS
  - provides keys stores on z/OS
  - provides the cryptographic APIs on z/OS

Thank you!