z/TPF EE V1.1
z/TPFDF V1.1
TPF Toolkit for WebSphere® Studio V3
TPF Operations Server V1.2

IBM Software Group

TPF Users Group Spring 2007

z/TPF Secure Key Management

Mark Gambino

Subcommittee Presentation

AIM Enterprise Platform Software

IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

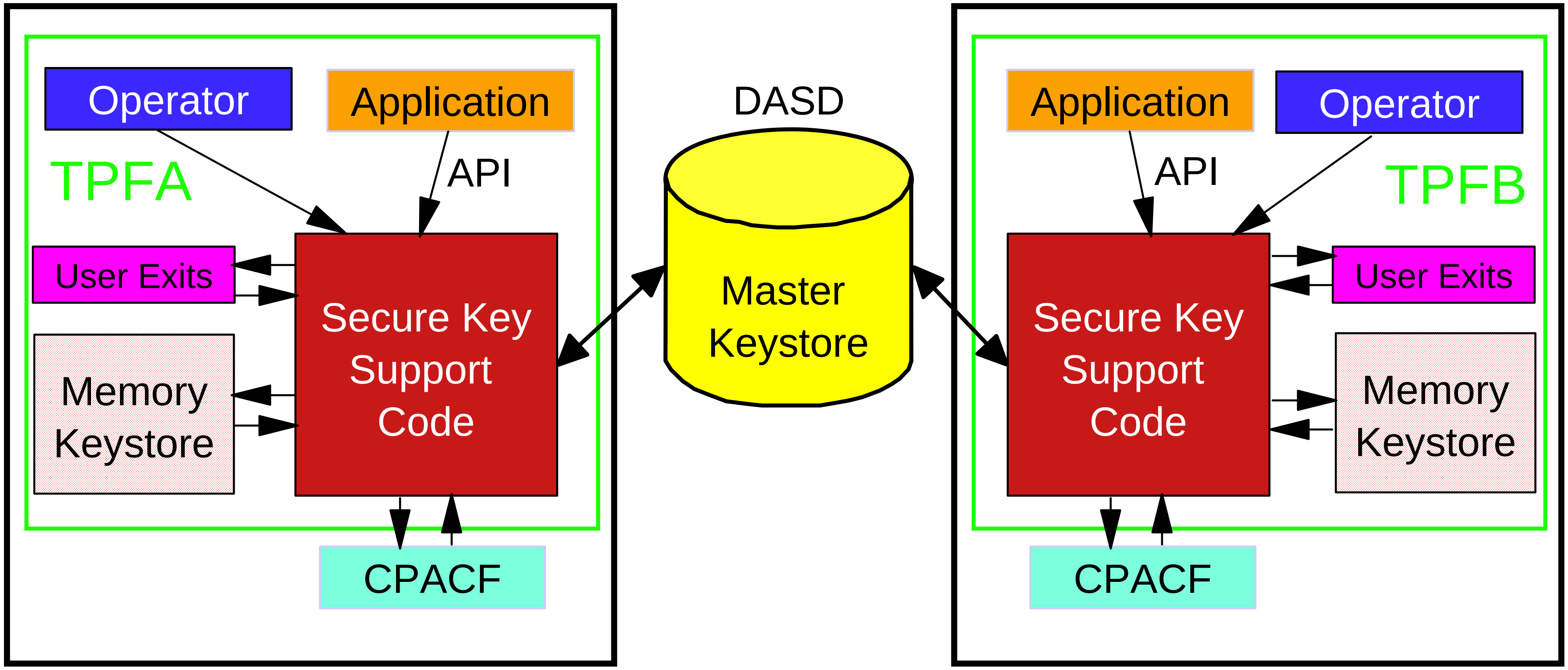
Level Set

- This presentation explains how secure key management support works, including information for application designers, application programmers, operators, and coverage staff on how to use the support
- Please refer to the "z/TPF Secure Key Management" TPFUG main tent presentation for information on why data security is important, what the basic principles of a secure key management solution are, and when data encryption at the application level is applicable

Agenda

- Secure key management support components
- Defining the keystore
- What are keys names and how applications use them to encrypt data
- Updating the keystore
- Operator procedures
- Detecting keystore corruption and how to recover
- Displaying keystore information
- Installation procedures and requirements
- How to integrate data integrity with encrypted data
- Database design considerations
- Performance data for encrypting data using secure keys

z/TPF Secure Key Management Components



Description of 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)

Defining the Keystore

Defining Keystore Restores

- 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

Increasing the Size of the Keystore

- Master Keystore
  - Increase the number of #IKEYS fixed file records in the FCTB
  - Load the new FCTB
    - Can be done without an outage using alternate FCTB loader
- Memory Keystore
  - Two options:
    - Generate and load a new CTKC
      - Update KEYSENT parameter on SKEYS macro in SIP
      - Load the updated CTKC
    - Update the size online
      - Use ZKEYS KEYSENT operator command
      - Must IPL each active processor to pick up the new value
        - Can be done one at a time to avoid complex-wide outage
- Warning messages are issued when the master or a memory keystore becomes 90% full, 91% full, 92% full, 93% full, and so on

How Key Names Are Used to Encrypt and Decrypt Data

Keystore Entry Contents - Displayable Fields

- Encryption key name
  - Name applications use to encrypt data using this key
  - Input to the *tpf_encrypt_data* API
  - Multiple entries can have the same encryption key name
- Decryption key name
  - Name applications use to decrypt data that was encrypted by this key
  - Output of the *tpf_encrypt_data* API
  - Input to the *tpf_decrypt_data* API
  - Name is unique for all keystore entries
- Encryption key status
  - Active or not active
- Cipher
  - DES, TDES, AES128, DES-CBC, TDES-CBC, AES128-CBC
- Date when key was activated
- Statistics on key usage

Keystore Entry Contents - Other Fields

- Key
  - The (encrypted) value of the key used to encrypt/decrypt data
- Validity values
  - Used to make sure the contents of an entry have not been corrupted
- Hash chain pointers
  - Exist in memory keystore only
  - Used to enable quick look up of a key

How Key Names Work

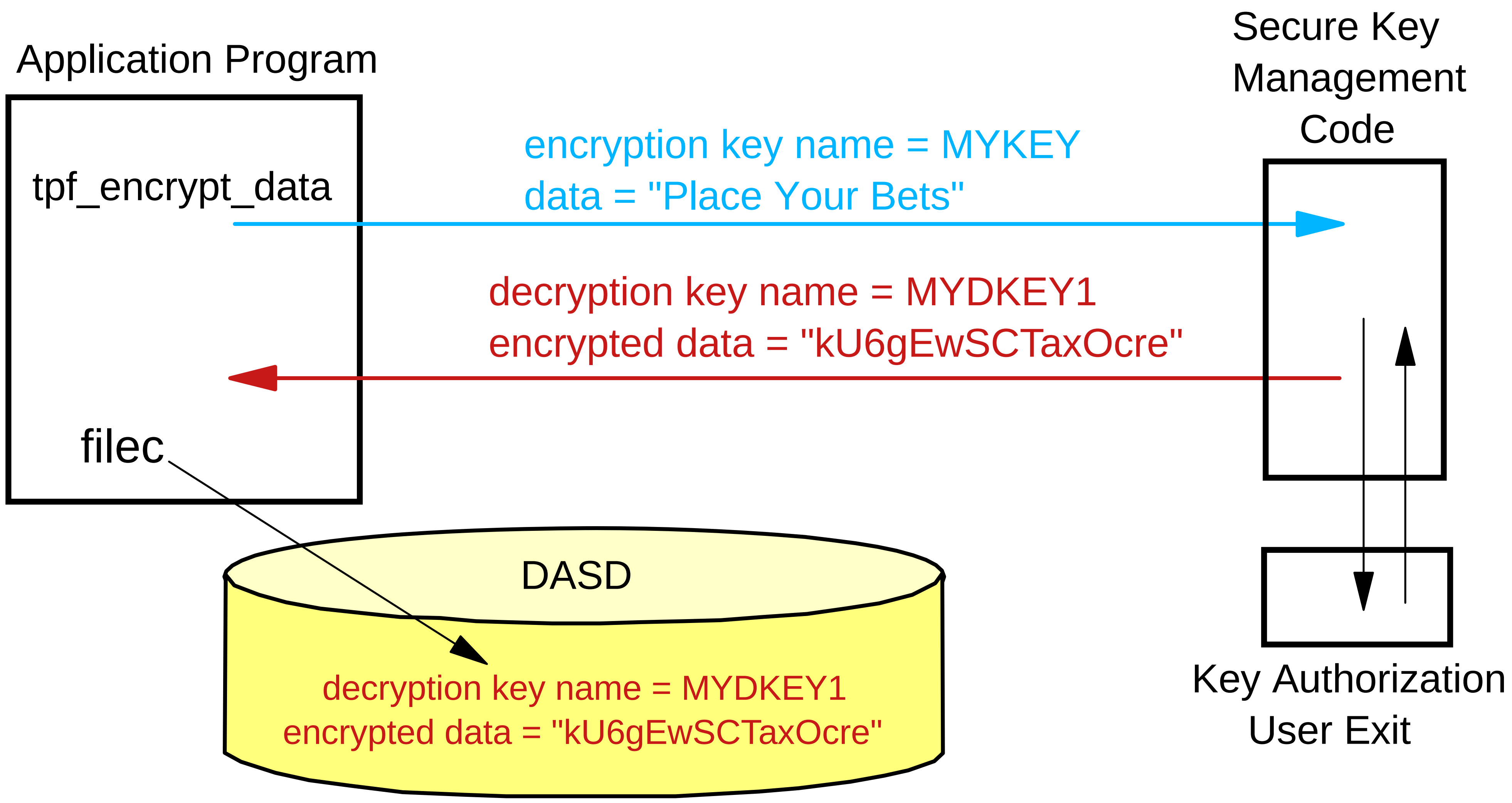
- Each key has two names:
  - ► Encryption key name
  - ► Decryption key name
- Having two names allows you change key values without having to modify application programs
- Encryption key name can be hard coded into your application program
  - ► Name does not change, even when key value changes
- Decryption key name should be saved in the same record as the encrypted data (or transmitted across the network along with the encrypted data)
  - ► Need to know which key was used to encrypt the data when you want to decrypt the data later on

In the Following Example...

- Keystore entry exists with the following information:
  - Encryption key name is MYKEY
  - Decryption key name is MYDKEY1
  - The encryption key is marked as active
  - Cipher is TDES
  - The value of the key is "KEY1"

```
Encryption Key Name  Decryption Key Name  Active  Cipher   Key
-------------------  -------------------  ------  ------  ------
       MYKEY               MYDKEY1          YES    TDES   "KEY1"
```

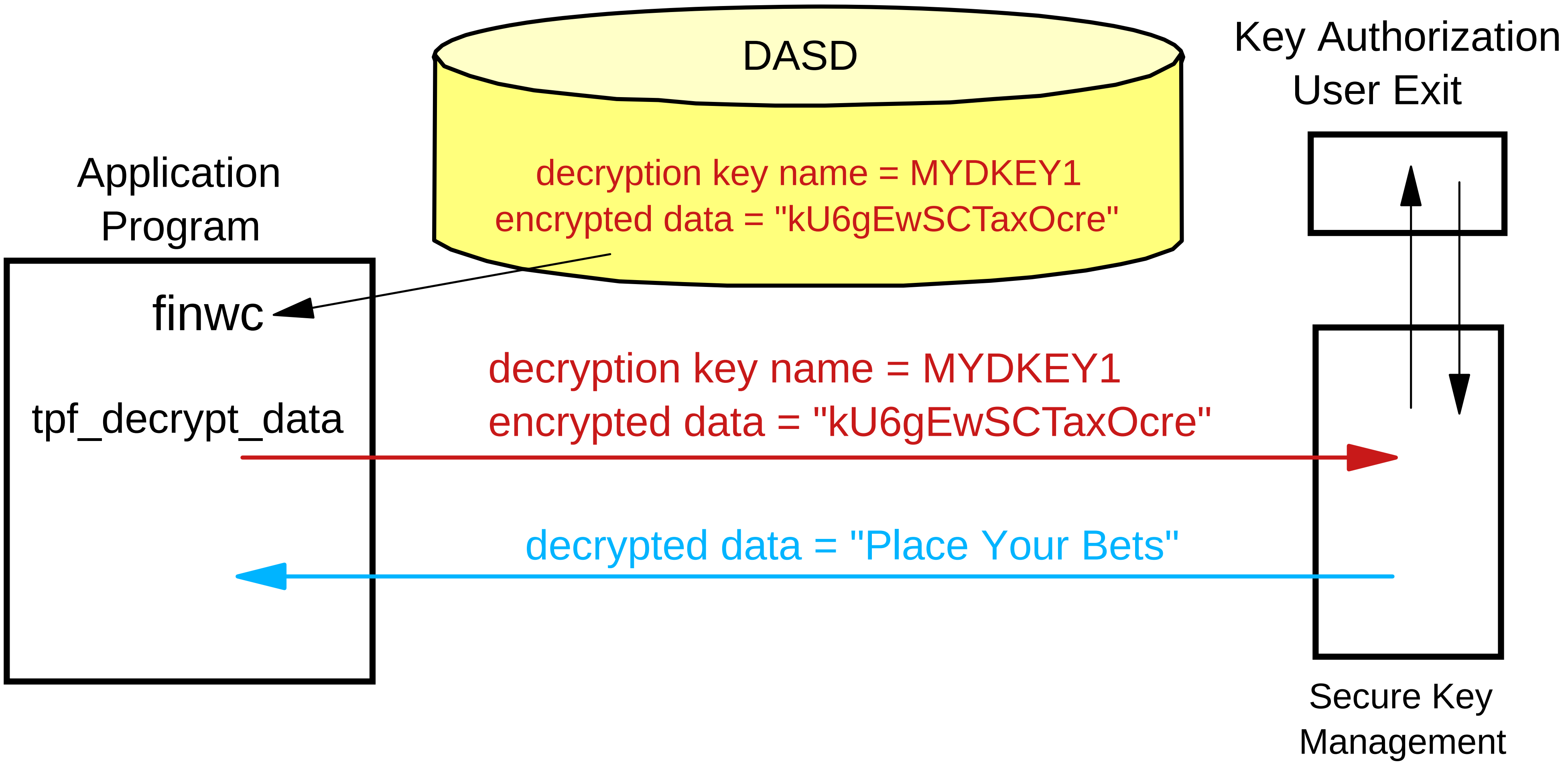
Data Encryption Example


Application Program
tpf_encrypt_data
filec
Secure Key Management Code
encryption key name = MYKEY
data = "Place Your Bets"
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
DASD
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
Key Authorization User Exit

Data Encryption Example - Steps

1. Application issues the tpf_encrypt_data API to encrypt data that is to be written out to file
   - Encryption key name (MYKEY) hardcoded into the application program is passed as input to the API
2. Secure Key Authorization User Exit is called to verify that this application program is allowed to use key MYKEY
3. Secure key management code searches the memory keystore to find (and validate) the active entry with encryption key name MYKEY
4. Secure key management code invokes CPACF to encrypt the data using the cipher (TDES) and key ("KEY1") in the memory keystore entry
5. Control is returned to the application program
   - Decryption key name (MYDKEY1) from the memory keystore entry is passed back to the application program
6. Application program files out a record containing the encrypted data and decryption key name (MYDKEY1) to use to decrypt this data

Data Decryption Example


DASD
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
Key Authorization User Exit
Application Program
finwc
tpf_decrypt_data
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
decrypted data = "Place Your Bets"
Secure Key Management

Data Decryption Example - Steps

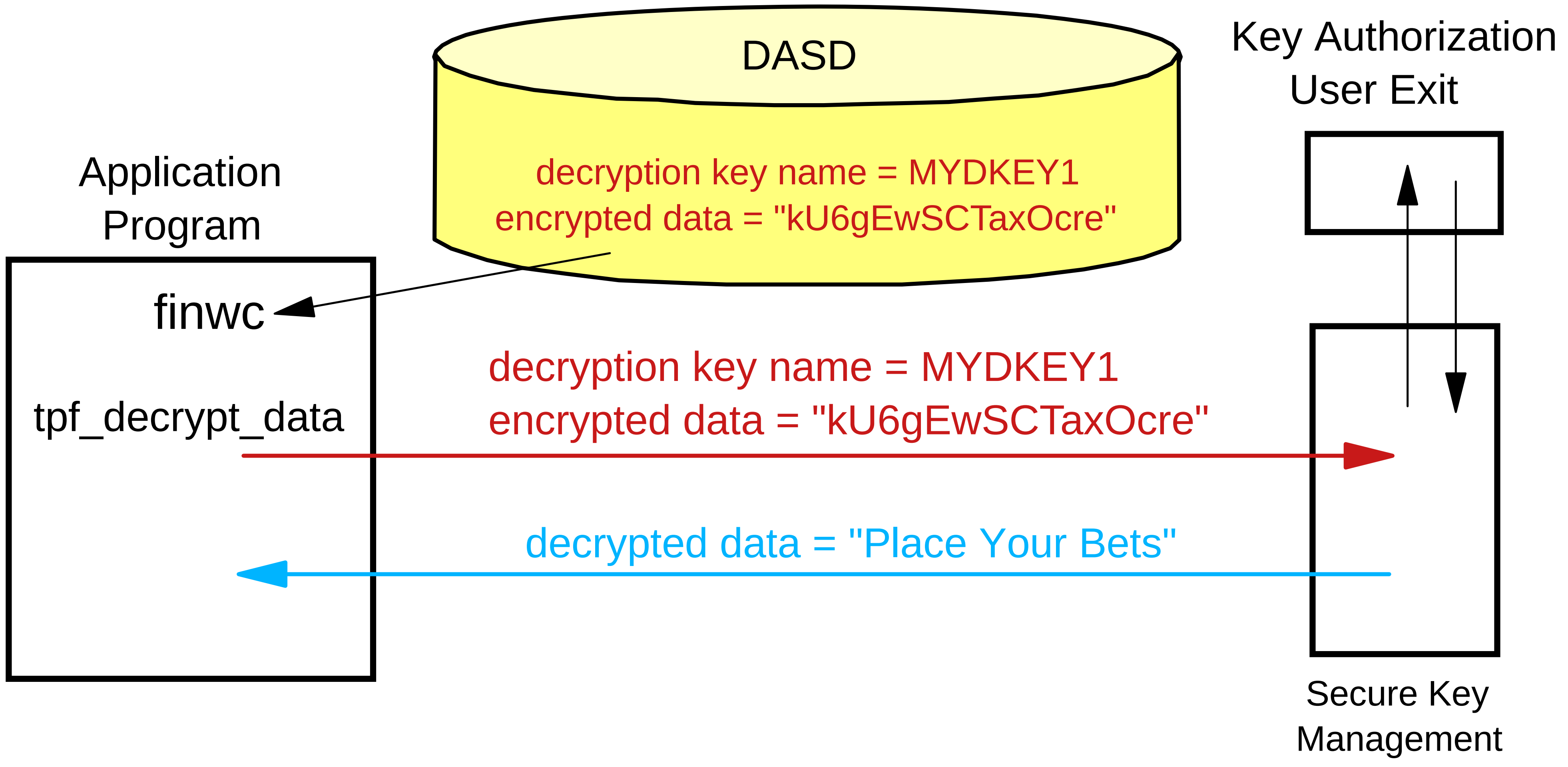
1. Application program reads a record containing encrypted data and decryption key name (MYDKEY1) to use to decrypt this data
2. Application issues the tpf_decrypt_data API to decrypt data
   - Decryption key name (MYDKEY1) that was saved in record is passed as input to the API
3. Secure Key Authorization User Exit is called to verify that this application program is allowed to use key MYDKEY1
4. Secure key management code searches the memory keystore to find (and validate) the entry with decryption key name MYDKEY1
5. Secure key management code invokes CPACF to decrypt the data using the cipher (TDES) and key ("KEY1") in the memory keystore entry
6. Control is returned to the application program

Changing Keys

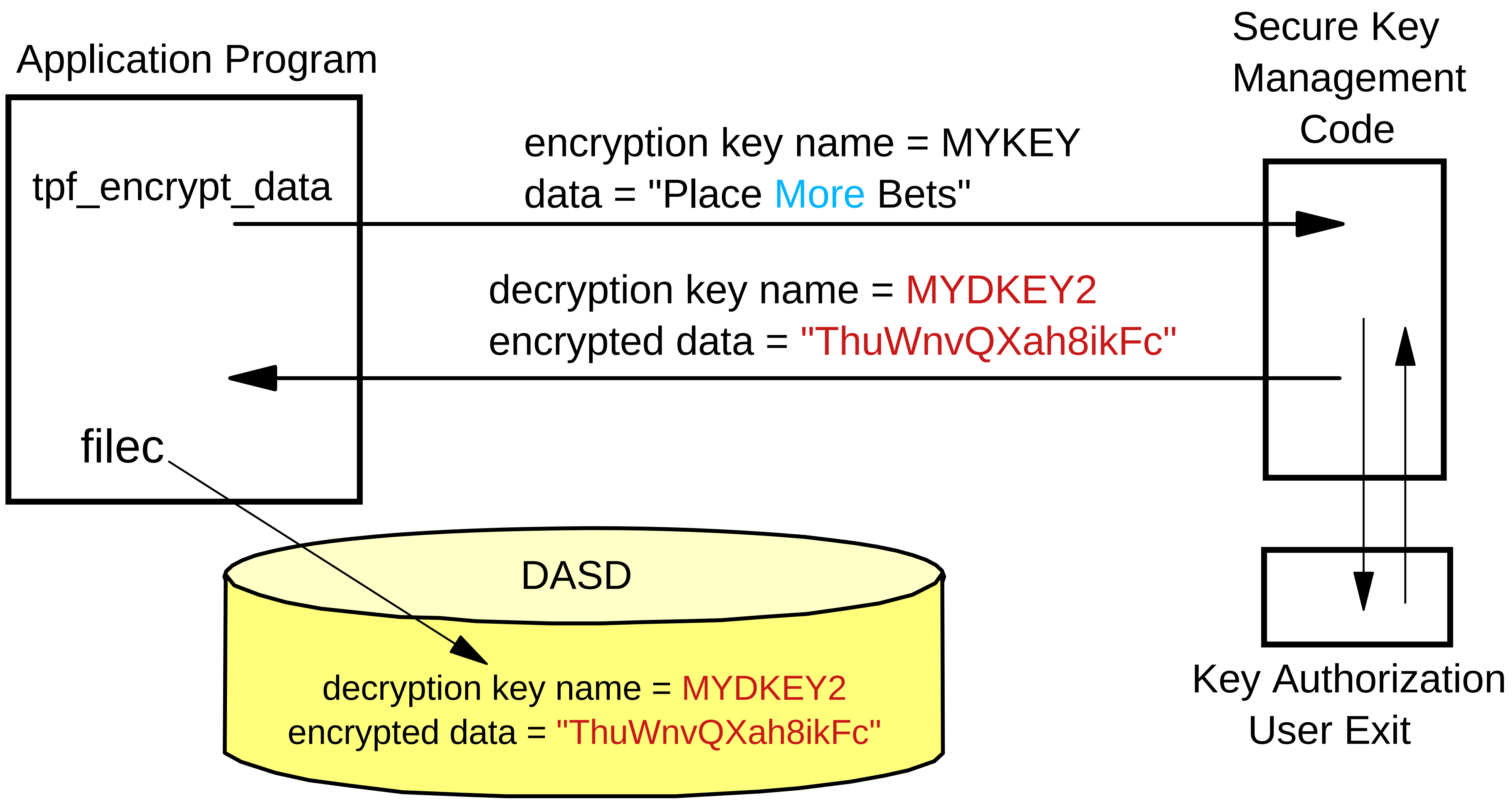
- Data was encrypted using "KEY1" in the previous example
- A new key is created and activated that changes the key value used to encrypt data with encryption key name MYKEY from "KEY1" to "KEY2"
- The following example decrypts the data, changes the data, then re-encrypts the data using the new key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

```
Encryption Key Name  Decryption Key Name  Active  Cipher    Key
-------------------  -------------------  ------  ------  ------
       MYKEY                MYDKEY1         NO            "KEY1"
       MYKEY                MYDKEY2         YES    TDES   "KEY2"
                                               TDES
```

Changing Keys Example - Part 1


DASD
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
Key Authorization User Exit
Application Program
finwc
tpf_decrypt_data
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
decrypted data = "Place Your Bets"
Secure Key Management

Changing Keys Example - Part 2

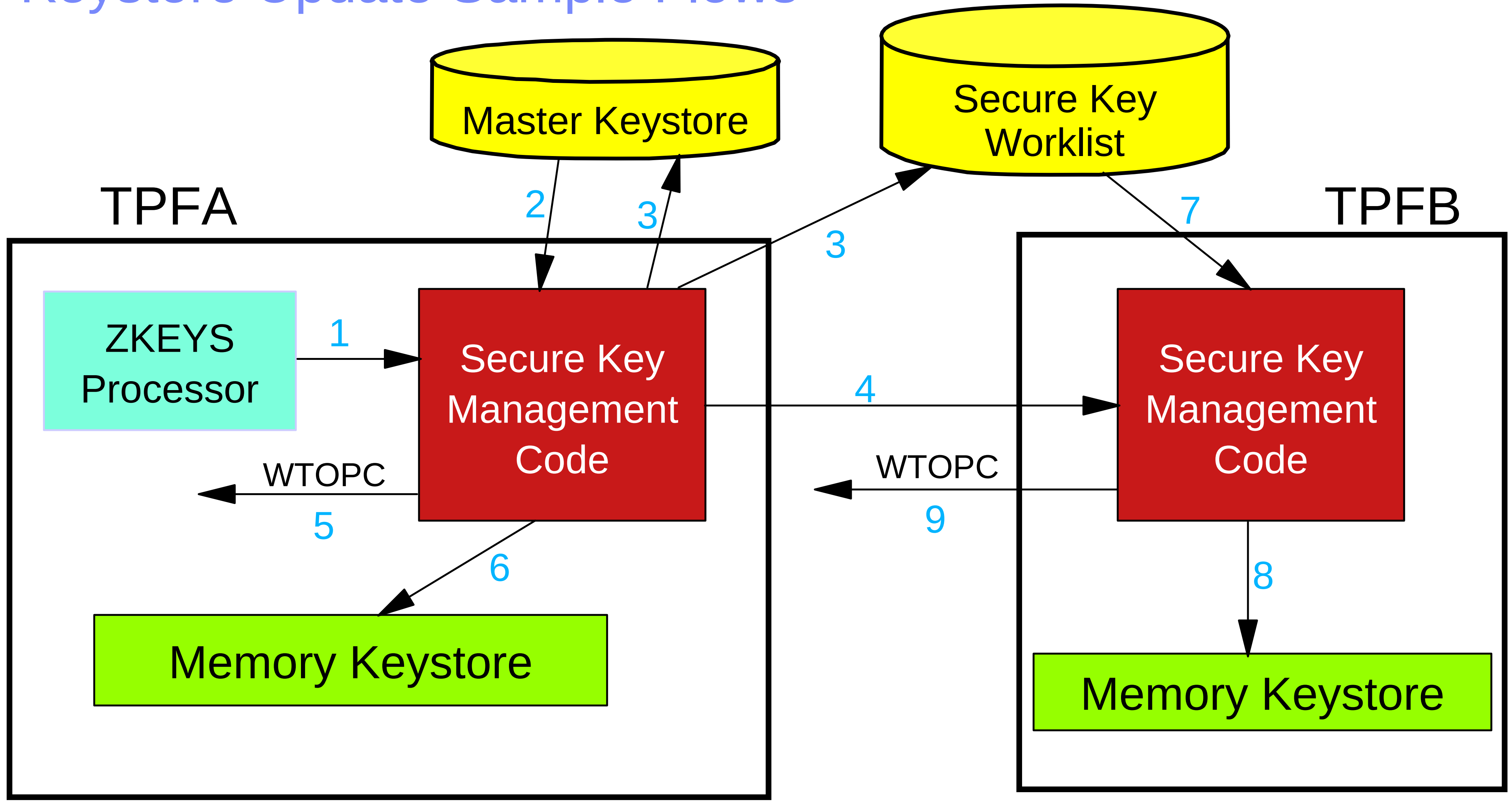

Application Program
tpf_encrypt_data
filec
encryption key name = MYKEY
data = "Place More Bets"
decryption key name = MYDKEY2
encrypted data = "ThuWnvQXah8ikFc"
Secure Key Management Code
DASD
decryption key name = MYDKEY2
encrypted data = "ThuWnvQXah8ikFc"
Key Authorization User Exit

How Keystore Updates are Made

Generating, Activating, Deactivating, and Deleting Keys

- Use the ZKEYS commands
- Any change in key status is done in two phases:
  - The master keystore is updated first, then the memory keystore on each active processor is updated
- Secure Key Worklist
  - File structure containing updates that need to be made to the memory keystore on one or more active processors
  - Entry added to the worklist when the master keystore is updated
  - Worklist is processed on each processor:
    - When notified that the master keystore has been updated (and therefore the worklist has been updated)
    - Periodically - for example, to process entries for a processors that was just deactivated

Keystore Update Sample Flows



Keystore Update Sample Flow Details

1. Operator on TPFA issues ZKEYS ACTIVATE command to activate a key
2. Secure key code on TPFA searches the master keystore to make sure that the specified key is defined and is not active
3. Secure key code on TPFA marks the key as active in the master keystore and updates the secure key worklist to indicate the key was just activated. Both file updates are done at the same time using a commit scope.
4. Secure key code on TPFA issues SIPCC to inform all active processors that the master keystore has been updated
5. Secure key code on TPFA issues WTOPC to send a message back to the operator indicating the key is active in the master keystore
6. Secure key code on TPFA marks the key as active in its memory keystore
7. Secure key code on TPFB having received the SIPCC message reads the secure key worklist to find out what keystore updates have been made
8. Secure key code on TPFB marks the key as active in its memory keystore
9. Secure key code on TPFB sends issues WTOPC to send a message to operator indicating the key has been activated in all memory keystores

Operator Procedures

for Managing Individual Keys

Creating a New Key

- ZKEYS GENERATE command
- Creates a key based on the specified cipher and adds it to the keystore
- Key is not marked as active
- The ENC parameter is the encryption key name
- The DEC parameter is the decryption key name

```
ZKEYS GENERATE ENC-MYKEY DEC-MYDKEY1 CIPHER-TDES 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
```

Activating a Key

- ZKEYS ACTIVATE command
- Activates the specified key making it available for use to encrypt data by applications
- If another key exists with the same encryption key which is active, that other key is deactivated
  - Can only be 1 active key per encryption key name
- You must backup the keystore (ZKEYS BACKUP command) after a key is defined and before you activate it
  - Must have backup copy before you start using a key

```
ZKEYS ACTIVATE ENC-MYKEY DEC-MYDKEY1

KEYS0006I 08:16:11 KEY ENC-MYKEY DEC-MYDKEY1 IS NOW ACTIVE IN MASTER KEYSTORE

KEYS0007I 08:16:11 KEY ENC-MYKEY DEC-MYDKEY1 IS NOW ACTIVE IN MEMORY KEYSTORE
                   ON ALL PROCESSORS
```

Changing Keys Example

```
KEYS0022I 17.17.11 ZKEYS DISPLAY PROCESSING STARTED
ENC NAME DEC NAME ACT  ACT DATE    CIPHER
-------- -------- --- ---------- ---------
MYKEY    MYDKEY1   Y  17MAR2007 TDES
MYKEY     MYDKEY2   N            TDES
END OF DISPLAY

ZKEYS ACTIVATE ENC-MYKEY DEC-MYDKEY2
KEYS0006I 17:17:18 KEY ENC-MYKEY DEC-MYDKEY2 IN NOW ACTIVE IN MASTER KEYSTORE
KEYS0007I 17:17:18 KEY ENC-MYKEY DEC-MYDKEY2 IN NOW ACTIVE IN MEMORY KEYSTORE


KEYS0022I 17.17.22 ZKEYS DISPLAY PROCESSING STARTED
ENC NAME DEC NAME ACT  ACT DATE    CIPHER
-------- -------- --- ---------- ---------
MYKEY    MYDKEY1   N  17MAR2007 TDES
MYKEY     MYDKEY2   Y 17APR2007 TDES
                   ON ALL PROCESSORS
END OF DISPLAY
```

Considerations for Creating and Activating a Key

- Your applications/utilities can also add keys to the keystore using the *tpf_keystore_add_key* API
  - Use to migrate your existing crypto keys to z/TPF secure key management support
  - Can also be used to add keys you imported from a remote key manager to the z/TPF keystore
  - Add Secure Key Authorization user exit controls what programs are allowed to add keys to the keystore
- Adding a key to the keystore and activating the key are separate steps
  - You must make a backup copy of keystore before you use the key
  - You want to make sure the key is added to all memory keystores before you use the key
    - For example, if TPFA encrypted data in a file record using a newly created key and TPFB read that file record before the key was added to the memory keystore on TPFB, the application on TPFB would not be able to decrypt the data

Deactivating a Key

- ZKEYS DEACTIVATE command
- Deactivates the specified key such that it cannot be used to encrypt data anymore
- You can still use the key to decrypt data that was encrypted previously with this key

```
ZKEYS DEACTIVATE ENC-MYKEY DEC-MYDKEY1

KEYS0012I 08:19:16 KEY ENC-MYKEY DEC-MYDKEY1 IS NO LONGER ACTIVE IN MASTER KEYSTORE

KEYS0013I 08:19:16 KEY ENC-MYKEY DEC-MYDKEY1 IS NO LONGER ACTIVE IN MEMORY KEYSTORE
                        ON ALL PROCESSORS
```

Deleting a Key

- ZKEYS DELETE command
- Deletes a key from the keystore
- You can only delete a key if the key has never been activated
  - Keys that were active at any point in time may have been used to encrypt data that will need to decrypted at a later point in time

```
ZKEYS DELETE ENC-MYKEY DEC-MYDKEY3

KEYS0014I 08:11:44 KEY ENC-MYKEY DEC-MYDKEY3 DELETED FROM MASTER KEYSTORE

KEYS0015I 08:11:44 KEY ENC-MYKEY DEC-MYDKEY3 DELETED FROM MEMORY KEYSTORE
                                   ON ALL PROCESSORS
```

Operator Procedures

for Backing up and Restoring the Keystore

Backing Up Keystore Data

- ZKEYS BACKUP command
- Copies the master keystore information to the specified file in the z/TPF file system
  - Information in the master keystore is encrypted; therefore, information in the file is also encrypted
  - Keystore information is validated as well
- You can FTP a copy of the backup file to another system for disaster recovery purposes
- Optional PASSWORD parameter on ZKEYS BACKUP command allows you to control access to the backup file

```
ZKEYS BACKUP PATH-/keys/keysback.fil PASSWORD-MYSECRET

KEYS0004I 09:41:01 MASTER KEYSTORE BACKUP STARTED
KEYS0005I 09:41:02 MASTER KEYSTORE BACKUP COMPLETED
```

Validating Backup Copy of Keystore Data

- ZKEYS VALIDATE command
- Used to verify that the specified keystore backup file is still usable and current
  - Makes sure the backup file has not been corrupted
  - Checks to see if any changes have been made to the master keystore since the backup file was created
- PASSWORD parameter is optional
  - If specified, it must be the same password that was entered on the ZKEYS BACKUP command that created the backup file

```
ZKEYS VALIDATE PATH-/keys/keysback.fil

KEYS0008I 15:41:01 ZKEYS VALIDATE PROCESSING STARTED
KEYS0009I 15:41:02 ZKEYS VALIDATE PROCESSING COMPLETED
```

Restoring the Master Keystore

- ZKEYS RESTORE command
- Rebuilds the master keystore using the information in the specified keystore backup file
- If a password was specified on ZKEYS BACKUP command that created the backup file, you must specify the same password on the ZKEYS RESTORE command
- This command does ***not*** affect the contents of the memory keystore on any processor

```
ZKEYS RESTORE PATH-/keys/keysback.fil PASSWORD-MYSECRET

KEYS0010I 17:32:02 ZKEYS RESTORE PROCESSING STARTED
KEYS0011I 17:32:03 ZKEYS RESTORE PROCESSING COMPLETED
```

Keystore Corruption

Restart and Memory Keystore Corruption

- During restart after an IPL, the master keystore is read from DASD and used to build the memory keystore
  - The master keystore information is validated as part of this process
  - If the master keystore is found to be corrupted:
    - You will need to restore the master keystore (using ZKEYS RESTORE command)
    - The system will continue restart processing and be allowed to reach 1052 state so that you can FTP a keystore backup file into z/TPF if necessary
- Each entry in the memory keystore is validated before it is used
  - If the memory keystore is found to be corrupted
    - Catastrophic system error is taken causing the memory keystore to be rebuilt from the master keystore after the IPL

Master Keystore Corruption After System Restart

- If secure key restart processing completed successfully, that means the memory keystore is usable
- The master keystore is validated:
  - During the processing of most ZKEYS commands
  - Periodically by the Keystore Monitor
    - How frequently the monitor runs is defined by the KEYSVAL parameter on the SKEYS macro in SIP
    - Use the ZKEYS KEYSVAL command to dynamically change how frequently the keystore monitor runs
- If the master keystore is found to be corrupted
  - The master keystore is rebuilt using the memory keystore information
  - Applications can continue to encrypt/decrypt data while the master keystore is being rebuilt

Keystore Backup and Restore Considerations

- Keystore backup recommended procedures:
  - Backup the master keystore (ZKEYS BACKUP) whenever keys are added, activated, or deactivated
    - You are required to do a backup before activating new keys
  - Keep at least one current backup copy in the z/TPF file system
  - Keep a backup copy on at least one remote system as well
    - Use FTP to send a copy of the backup file to the remote system
- Options for recovering a corrupted master keystore
  - The system will automatically rebuild the master keystore using the memory keystore if the memory keystore is built (secure key restart completed successfully) and the memory keystore is not corrupted
  - Issue ZKEYS RESTORE to restore the master keystore using a backup copy in the z/TPF file system
  - FTP a backup copy saved on a remote system to the z/TPF file system, then issue ZKEYS RESTORE

Displaying Keystore Information

ZKEYS DISPLAY Commands

- ZKEYS DISPLAY SUMMARY
  - Shows information about keystore resources
- ZKEYS DISPLAY STATS
  - Shows statistical information about secure key operations at a system wide level
- Other ZKEYS DISPLAY commands options
  - Displays information about one or more keys in the keystore
    - For example, display all keys, a specific key, active keys, all keys with the same encryption key name
  - Display includes statistical information about key usage on this processor

ZKEYS DISPLAY SUMMARY Example

```
ZKEYS DISPLAY SUMMARY
KEYS0024I 17.34.09 ZKEYS DISPLAY SUMMARY

KEYSTORE ENTRIES IN USE : 39
MAXIMUM ENTRIES IN MASTER KEYSTORE : 975
HIGHEST MASTER KEYSTORE RECORD ORDINAL IN USE : 2
MAXIMUM ENTRIES IN MEMORY KEYSTORE : 100
MAXIMUM ENTRIES IN MEMORY KEYSTORE ON NEXT IPL : 100
LAST UPDATE TO MASTER KEYSTORE : 17.16.37 - MAR 19, 2007
LAST UPDATE IN KEYSTORE BACKUP : 17.16.37 - MAR 19, 2007
MASTER KEYSTORE VALIDATION INTERVAL : 1

END OF DISPLAY
```

ZKEYS DISPLAY SUMMARY Example Description

```
KEYSTORE ENTRIES IN USE : 39
MAXIMUM ENTRIES IN MASTER KEYSTORE : 975
MAXIMUM ENTRIES IN MEMORY KEYSTORE : 100
MAXIMUM ENTRIES IN MEMORY KEYSTORE ON NEXT IPL : 100
```

39 = number of keys defined in the keystore

975 = maximum number of keys that will fit in the master keystore based on the number of #KEYS fixed file records defined

100 = value of KEYSENT in CTKC when this processor IPL'd

100 = current value of KEYSENT in CTKC

ZKEYS DISPLAY SUMMARY Example Description

```
LAST UPDATE TO MASTER KEYSTORE : 17.16.37 - MAR 19, 2007
LAST UPDATE IN KEYSTORE BACKUP : 17.16.37 - MAR 19, 2007
```

- If the two timestamps above are equal
  - The backup keystore file is current, meaning no changes have been made to master keystore since the last keystore

  backup (ZKEYS BACKUP) was done

```
MASTER KEYSTORE VALIDATION INTERVAL : 1
```

1 = value of KEYSVAL in CTKC

ZKEYS DISPLAY Keys Example

```
ZKEYS DISPLAY ENC-AP1EKEY
KEYS0022I 17.17.11 ZKEYS DISPLAY PROCESSING STARTED

ENC NAME DEC NAME ACT  ACT DATE  CIPHER   ENC/SEC DEC/SEC MAX ENC MAX DEC
-------- -------- --- --------- --------- ------- ------- ------- -------
AP1EKEY  AP1DKY06  Y  13APR2007 TDES                19265   40567   28213
AP1EKEY  AP1DKY05  N  13MAR2007 TDES                    2       0      18
                                         25781
AP1EKEY  AP1DKY04  N  13FEB2007 TDES                    0       0       2
AP1EKEY  AP1DKY03  N  13JAN2007 DES                     0       0       0
AP1EKEY  AP1DKY02  N  13DEC2006 DES                     0       0       0
                                        0
AP1EKEY  AP1DKY01  N  13NOV2006 DES                     0       0       0
                                        0
END OF DISPLAY
                                        0
                                        0
                                        0
```

Installation Procedures

Initialize Master Keystore

- ZKEYS INITIALIZE command
- Initializes the master keystore
- You need to do this once when you first install the code and should be the only time you use this command in a production system
  - If the master keystore becomes corrupted, restore it (using the ZKEYS RESTORE command) rather than reinitializing the keystore
- Operator will be prompted to confirm the initialization of the master keystore

Installation Procedures

1. Install the APAR PJ31450 code
2. Determine how many secure keys you need
3. Define master and memory keystore resources
4. Initialize master keystore
5. Create secure keys and add your existing keys to the keystore
6. Make backup copy of the keystore
7. Activate the keys
8. Make another backup copy of the keystore
9. Assign keys to applications
10. Update secure key authorization user exit
11. Update applications to use secure key APIs

Installation Requirements

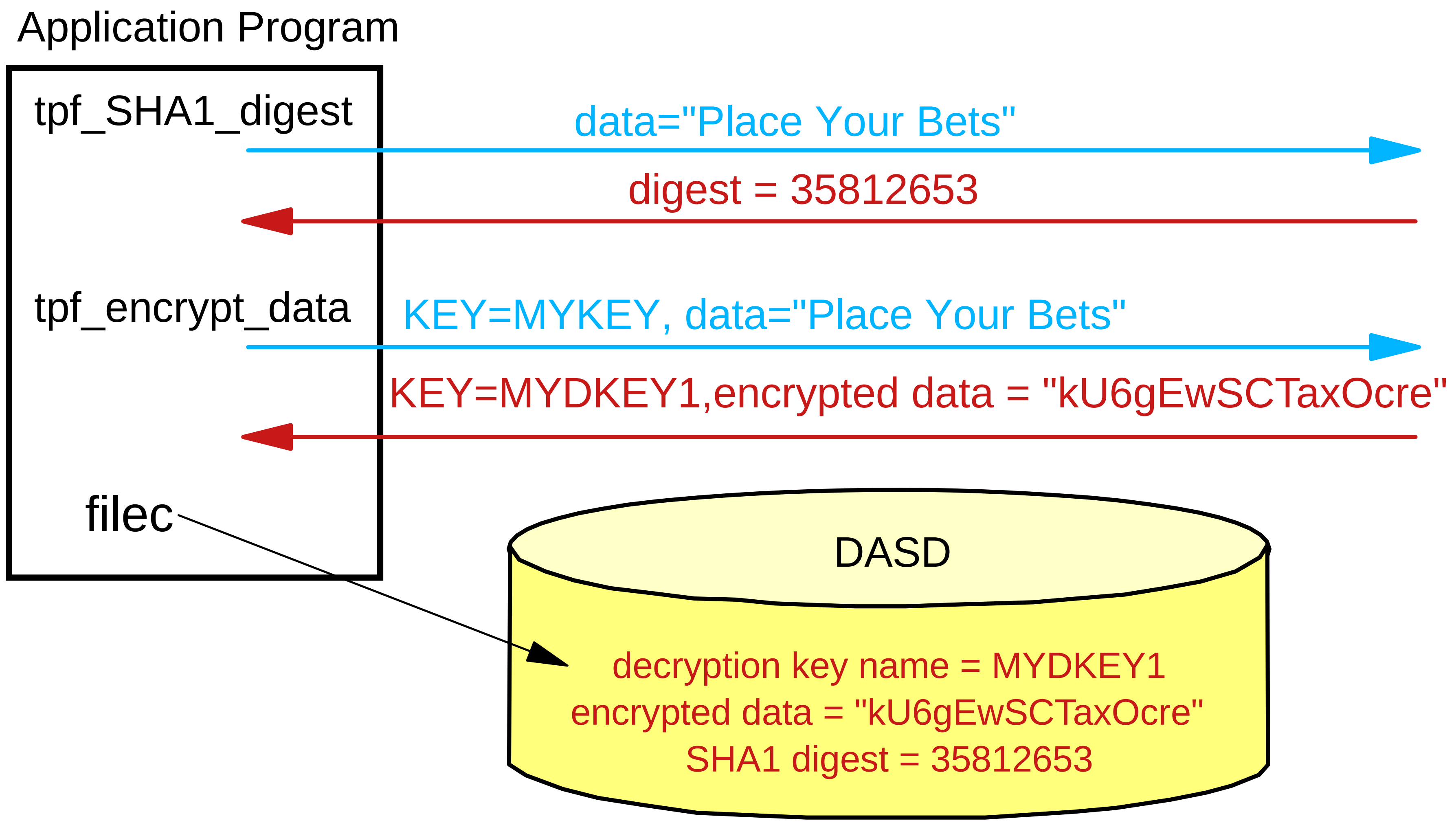
- Enabling secure key support:
  - APAR PJ31450 must be installed on all processors
  - All processors must be z990 or higher
    - Secure key support requires CPACF
  - DES/TDES feature on CPACF must be enabled
- To use AES128 or AES128-CBC ciphers
  - Must be running z9 or higher

Data Integrity

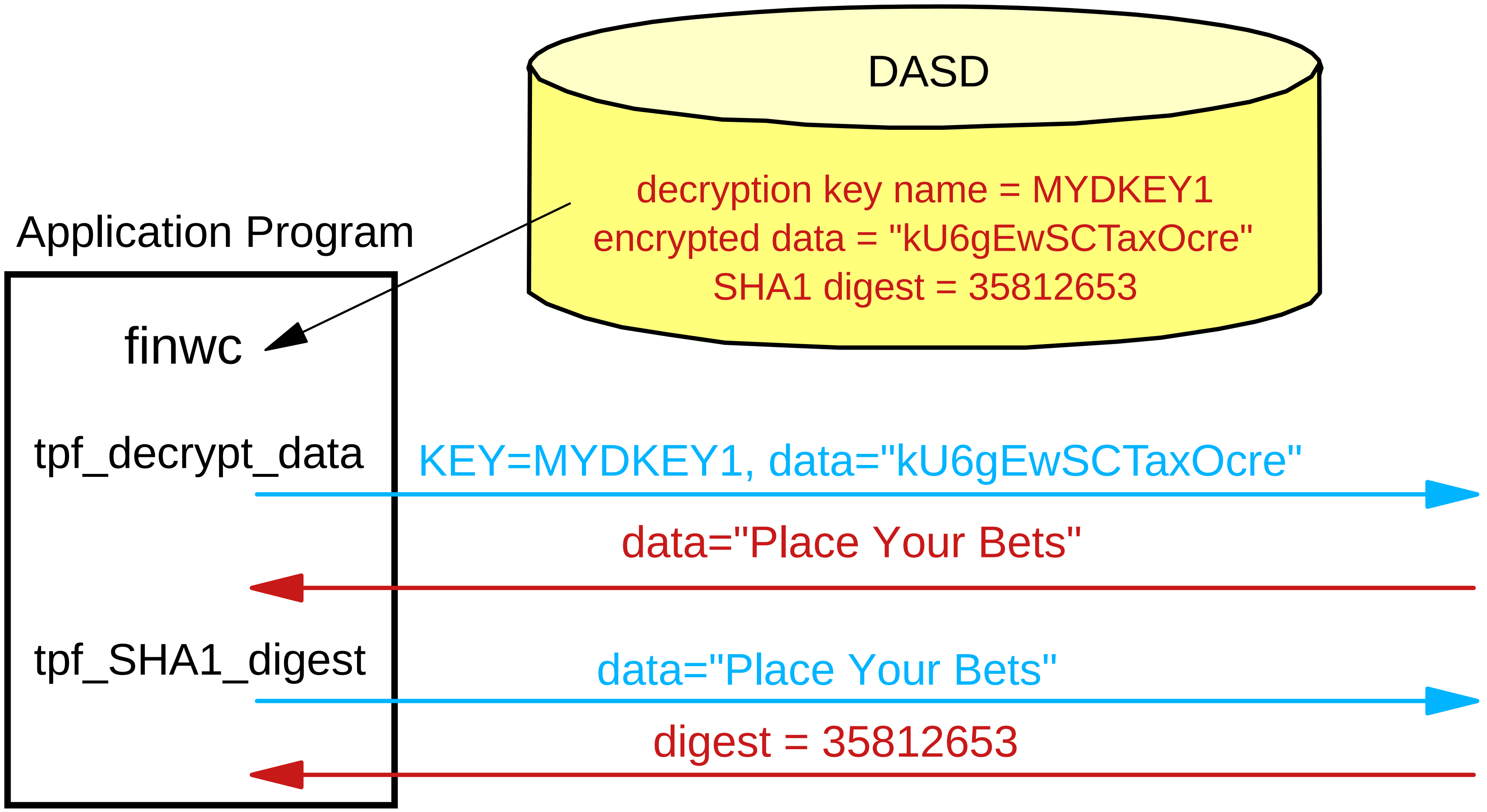
Data Integrity

- Recommended usage:
  1. Create a digest of the (unencrypted) data
  2. Save the digest along with the encrypted data
     - Also save what digest algorithm was used in case you change algorithms in the future
  3. Recalculate the digest after data is decrypted and compare to the original digest to verify that the data has not been altered
- Use SHA-1 APIs to calculate digests
  - ► Provided by z/TPF APAR PJ31336
- Statement of direction to provide SHA-256 APIs

Data Encryption Example with Data Integrity


Application Program
tpf_SHA1_digest
data="Place Your Bets"
digest = 35812653
tpf_encrypt_data
KEY=MYKEY, data="Place Your Bets"
KEY=MYDKEY1,encrypted data = "kU6gEwSCTaxOcre"
filec
DASD
decryption key name = MYDKEY1
encrypted data = "kU6gEwSCTaxOcre"
SHA1 digest = 35812653

Data Decryption Example with Data Integrity

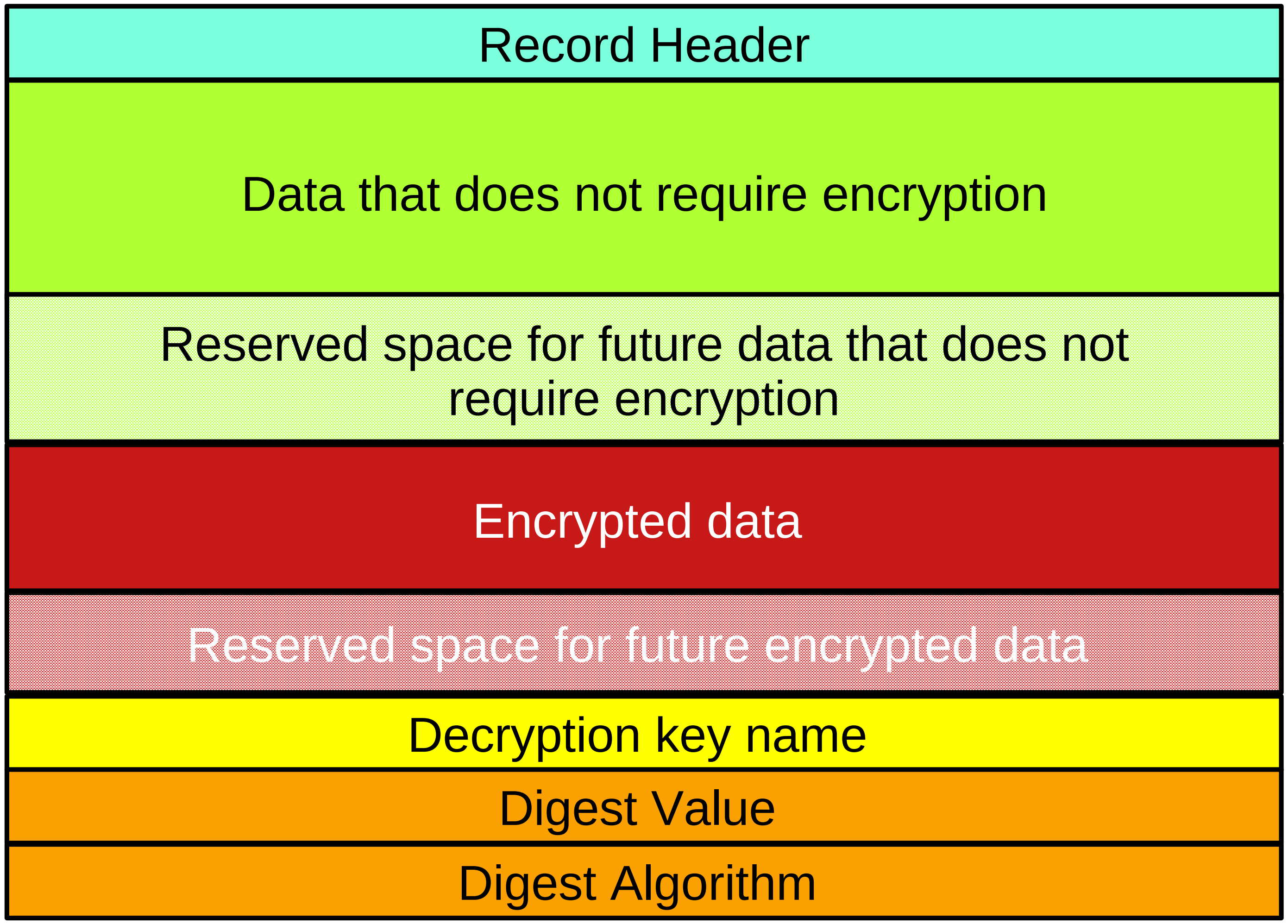


Design Considerations Data Record

Data Record Design Considerations

- For records containing data encrypted at the application level that are stored on DASD, stored on tape, or sent across the network
- Try to group together all data that needs to be encrypted
  - If data to be encrypted is contiguous
    - One tpf_encrypt_data API can encrypt all that data
  - If data to be encrypted is not contiguous
    - Multiple *tpf_encrypt_data* APIs are needed
    - Make sure the application issues all the *tpf_encrypt_data* APIs for a given record without giving up control to prevent the encryption key value from changing in the middle of processing a record
- Reserve space for both regular data (that does not require encryption) and encrypted data
- Reserve 32 bytes for digest to accommodate SHA-256
  - 20 bytes needed for SHA-1 digest

Suggested Record Layout


Record Header
Data that does not require encryption
Reserved space for future data that does not require encryption
Encrypted data
Reserved space for future encrypted data
Decryption key name
Digest Value
Digest Algorithm

Performance

Data

Secure Key Performance Implications

- Number of keys defined does not impact performance
  - For example, the system can process the same number of secure key APIs per second with 2 keys defined in the keystore as when 2,000 keys are defined
- Number of secure key APIs that can be processed per second grows with the number of I-streams
  - Each I-stream has its own CPACF
  - Roughly linear scaling
- Secure Key versus Clear Key
  - Ran tests with different data sizes first using clear key APIs (*tpf_cryptc*), then with secure key APIs (*tpf_encrypt_data* and *tpf_decrypt_data*)
    - The number of crypto operations (APIs) per second is 2-10% less using secure key compared to clear key

Secure Key Data Operations/Second - Single I-stream, Different Ciphers

Data Size	DES	TDES	TDES-CBC	AES128
16	449,292	440,516	434,490	430,800
32	442,148	428,586		422,658
64	433,236	400,434		410,058
256	386,958	310,818	304,200	322,955
1024	274,970	158,508		188,974
4096	124,352	53,780	53,246	69,222
32,768	20,622	7,480		9,785
65,536	10,461	3,753	3,734	4,968
1,048,576	669	236		313

Tests performed on z9 processor - your results may vary

Secure Key Data Operations/Second - TDES, Multiple I-streams

Data Size	1 I-Stream	2 I-streams	Scaling Factor
	400,434	740,449	1.85
4096	53,780	105,305	1.96
65,536	3,753	7,466	1.99
1,048,576 64	236	469	1.99

	5 I-Streams	Scaling	9 I-streams	Scaling
	1,847,119		3,339,130	
	265,512	4.61	480,407	8.34
65,536	18,762	4.94	33,770	8.8399
1,048,576 4096 64	1,176	4.99	2,122	
		4.99		8.99

Tests performed on z9 processor - your results may vary

Summary

z/TPF Secure Key Management Support Highlights

- Ability to change encryption key values (and in some cases upgrade the cipher) without requiring any application program changes
- Can scale to hundreds of thousands of crypto operations per second
- Ability to control and log key usage by applications
- Ability to control who can create keys (operators and applications)
- Ability to backup and restore keys
- Ability to migrate your existing keys to this support
- Safeguards to prevent a corrupted key (accidental or intentional) from ever being used
- APIs to encrypt and decrypt data using secure keys
  - Use in conjunction with message digest APIs enables your applications to ensure data integrity (detect data corruption)
- Performance and archive advantages over external crypto box solutions
- To summarize, a solution that enables you to protect vital data, and does so with traditional TPF scalability and performance characteristics

Closing Message - The Choice is Yours ...

with secure key management

without secure key management

Trademarks