Bookstore credit card application

We recently created an application to simulate a credit card transaction. This application tests the z/OS Cryptographic Services as well as DB2 and IMS Data Encryption. We accomplished this by enhancing our existing Bookstore application to use z/OS cryptographic services to process payment for books bought from the zPET bookstore. The goal of this application is to simulate the workflow, database structure, and response time requirements of a credit card application.

We used the following products in our implementation:

- WebSphere MQ for z/OS version V7.1
- WebSphere Application Server V7.0
- IMS V12
- DB2 V10
- z/OS Cryptographic Services ICSF
- z/OS Security Server RACF
- InfoSphere Guardium Data Encryption for DB2 and IMS v1.02.00
- Rational Application Developer

Application overview and flow. Figure 1 shows the basic application set up and flow.

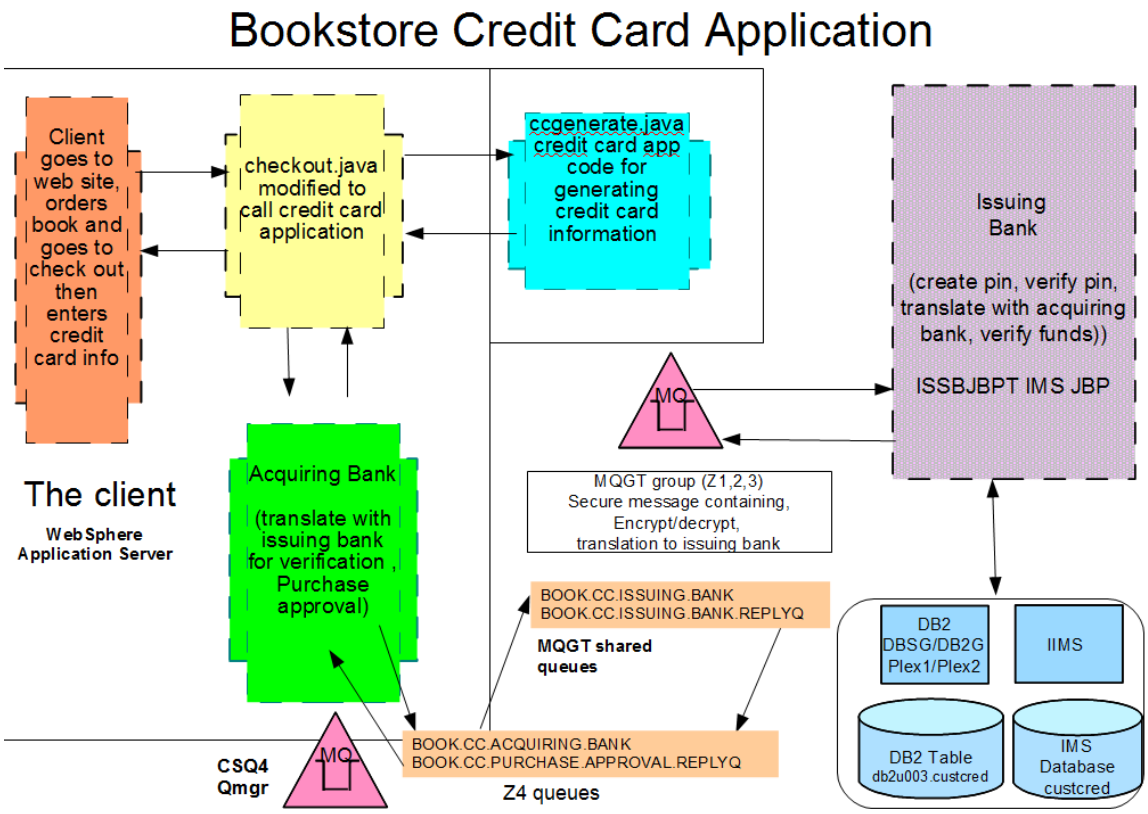

Figure 1. Bookstore credit card application

This enhancement to our bookstore application builds off of the existing Bookstore workload. When a customer buys a book the new credit card piece of the Bookstore application will require credit card information and need to verify that information. This is done by first going to the acquiring bank and then sending encrypted information to the issuing bank application to verify that this customer has valid credit card information and available credit to purchase a book. If credit is available, the customer will be able to purchase a book.

Bookstore purchase flow

When a customer purchases a book from our Bookstore the program `checkout.java` calls the `ccgenerate java` class for generating a credit card number, expiration date, service code and Card Verification Value (CVV) number.

When the credit card information is obtained the acquiring bank class is called along with the customer name, credit card number, expiration date, service code, CVV, and order total amount.

The `acquiringbank.java` and `issuingbank.java` programs use JNI calls for native ICSF services by Common Cryptographic Architecture (CCA) Support Program (that is similar to CCA JNI, which is not available for z/OS). See Secure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide, SC33-8294 and the IBM 4764 PCI-X Cryptographic Coprocessor web site. for additional information.

Checkout.java

- The following summarizes the process:Calls `ccgenerate` java class and acquiring bank java class
- Passes the customer id to `ccgenerate` java class for credit card information generation.
- Upon return from `ccgenerate` java `class`, calls acquiring bank java class for credit card verification
- Upon return from acquiring bank java class, verifies whether credit card was approved or denied and if approved, completes the buy. If not, produces an error.

CCGenerate.java

The following summarizes the process:

- Generates credit card number (equal to customer number), expiration date (1050), service code (112).
- Generates CVV using CSNBCSG callable service.

Acquiring Bank.java

The following summarizes the process:

- Receives credit card number, expiration date, service code and CVV number from the checkout java class., and also sends the order total.
- Creates a DATA and MAC key using CSNBKGN callable service
- Encrypts customer data using CSNBENC callable service
- Generates MAC using the CSNBMGN callable service
- Sends keys, encrypted and mac encrypted data to issuing bank through WebSphere MQ using SSL channels.
- Receives issuing banks response and decrypts it.
- Returns result of response to the checkout java class for return to client

Issuing Bank.java

The following summarizes the process:

- Receives DATA and MAC keys, MAC and Encrypted Message from acquiring bank java class via WebSphere MQ channel
- Receives DATA key, MAC key using CSNBKIM callable service.
- Verifies MAC using CSNBMVR callable services
- If MAC ok, decrypt customer data using CSNBDEC callable service
- Using credit card number, get information from DB2 or IMS (based on odd/even) and compare that information to what was sent (simulate comparing of credit card info to what is on file at the issuing bank).
- If there is credit card information in DB2 or IMS then verify CVV using CSNBCSV callable service.
- Encrypt conclusion of the card transaction.
- Returns encrypted response back to acquiring bank java class for return to client

Database Setup

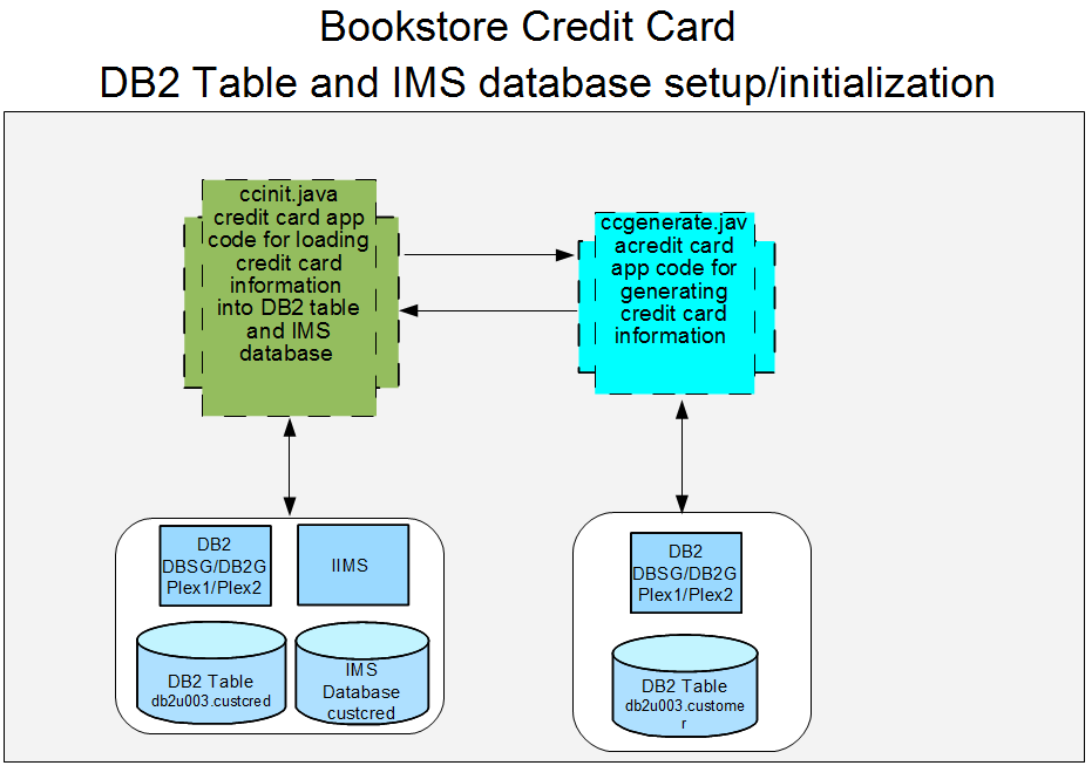

Figure 2. DB2 table and IMS database setup/initialization of the Bookstore credit card application

IBM InfoSphere Guardium Data Encryption for DB2 and IMS tool is used when storing customer credit information in the IMS/DB2 tables. Both IMS and DB2 are loaded with the same information (that is, both even and odd credit card numbers). A credit card ending in an even number uses IMS tables and one ending in an odd number uses DB2 tables.

Two programs were used to populate the credit card table `custcred`. The first `ccinit.java` loads the data into DB2 and IMS. The second `ccgenerate`.java creates the credit card information. `CCinit` calls `ccgenerate` to get that information and then loads it into the databases.

DB2

We added a new table to hold the customer credit card information. This table was encrypted by using an `editproc` as shown below.

```
PROFILE PREFIX(ISR)
 ISPSTART CMD(%DECENC02 DB2 CCRDEXIT -
 CREDITCARDDB2ENCRYPTKEY
```

```
The DDL used to define the table looks like this
CREATE TABLE CUSTCRED (
  "CUSTOMER_ID" INTEGER NOT NULL,
  "FIRST_NAME" VARCHAR(30) FOR SBCS DATA NOT NULL,
  "LAST_NAME" VARCHAR(30) FOR SBCS DATA NOT NULL,
  "CCNUMBER" INTEGER,
  "EXPDATA" INTEGER,
  "SERVCODE" INTEGER,
  "CREDLIM" INTEGER
 )
 IN BOOK.TSCUSCRE
 EDITPROC CCRDEXIT;
```

We ran DB2 BINDs for the IMS programs ISSBANK and LISTCRED as shown below. The plan names are the same as the IMS PSB names being used:

```
BIND PLAN(ISSBANK) PKLIST(*.NULLID.*,ISSBANK.*) -
   ACT(REP) ISOLATION(UR)  -
   EXPLAIN(NO) VALIDATE(BIND) REL(C) DEGREE(ANY)      -
   CURRENTDATA(YES) OWNER(DB2U003)

BIND PLAN(LISTCRED) PKLIST(*.NULLID.*,LISTCRED.*) -
   ACT(REP) ISOLATION(UR) -
   EXPLAIN(NO) VALIDATE(BIND) REL(C) DEGREE(ANY)      -
   CURRENTDATA(YES) OWNER(DB2U003)
```

IMS setup for the credit card application

We created a database to manage the IMS customer data and encrypted it using the InfoSphere Guardium Data Encryption v1.02.00 product. In order to perform the encryption, we used the exit JCL sample provided by the product. Before running this exit, we updated the JCL for our environment as follows:

```
//*************************************************************
//* LINKEDIT THE ICSF EDITPROC
//*************************************************************
//LINK     EXEC PGM=IEWL,PARM='LIST,XREF,RENT'
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SDECLMD0  DD DSN=MVSBUILD.DEI120.SDECLMD0,DISP=SHR
//SCSFMOD0  DD DSN=CSF.SCSFMOD0,DISP=SHR
//SYSUT1    DD UNIT=SYSDA,SPACE=(1024,(50,50))
//SYSLMOD   DD DSN=IMS1210T.PET.DYNALLOC,DISP=SHR
//SYSLIN    DD *
  ENTRY DECENB01
  INCLUDE SDECLMD0(DECENB01)
  INCLUDE SCSFMOD0(CSNBSYE)
  INCLUDE SCSFMOD0(CSNBSYD)
  NAME CLEARJON(R)
//*
//BATCHTSO EXEC PGM=IKJEFT01,DYNAMNBR=25,REGION=0M,COND=EVEN
```

```
//SYSLIB   DD DISP=SHR,DSN=IMS1210T.PET.DYNALLOC
//ISPLLIB  DD DISP=SHR,DSN=SYS1.MIGLIB
//ISPPLIB  DD DISP=SHR,DSN=MVSBUILD.DEI120.SDECPLIB
//         DD DISP=SHR,DSN=ISP.SISPPENU
//ISPSLIB  DD DISP=SHR,DSN=MVSBUILD.DEI120.SDECSLIB
//         DD DISP=SHR,DSN=ISP.SISPSLIB
//ISPMLIB  DD DISP=SHR,DSN=MVSBUILD.DEI120.SDECMLIB
//         DD DISP=SHR,DSN=ISP.SISPMENU
//ISPTLIB  DD DISP=SHR,DSN=IMSSYST.DEC11.ISPPROF
//         DD DISP=SHR,DSN=ISP.SISPTENU
//SYSEXEC  DD DISP=SHR,DSN=MVSBUILD.DEI120.SDECCEXE
//         DD DISP=SHR,DSN=ISP.SISPCLIB
//ISPPROF  DD DISP=SHR,DSN=IMSSYST.DEC11.ISPPROF.BATCH
//SYSTSPRT DD SYSOUT=*
//ISPLOG   DD SYSOUT=*,DCB=(BLKSIZE=800,LRECL=80,RECFM=FB)
//SYSTSIN  DD *
  PROFILE PREFIX(GGJ)
    ISPSTART CMD(%DECENC02 IMS CLEARJON -
    CREDITCARDIMSENCRYPTKEY
/*
```

We then implemented the encryption by updating the SEGM macro within the DBD source as follows:

```
***************************
*        SEGMENT NUMBER 1
***************************
    SEGM    NAME=CCARDSEG,
               PARENT=0,
               BYTES=094,
               RULES=(LLL,LAST),
               PTR=(TWINBWD,,,,),
               COMPRTN=(CLEARJON,DATA,INIT)
```

The DBD names and associated BMP names were then added to the IMS system gen and a JBP dependent region was added.

PSBs

We created two PSBs, one for the program that populates the IMS database (LISTCRED) and one for the issuing bank program (ISSBANK). These were then added to DFSJVMAP with references to the paths where the java programs reside.

LISTCRED=CCInit
ISSBANK=bookstore/creditcard/IssuingBank

DB2 bind jobs had to be run for these PSB's. See the section on DB2 for examples.

SSM Definitions

The IMS JBP program connects to DB2 using the DB2 location name and to WebSphere MQ using the queue manager name. To enable this type of connection, we created the following SSM definition:

```
SST=DB2,SSN=DB1Z,COORD=RRS
CSQ1,MQM1,CSQQESMT,,R,-
```

Note: For information on setting up connections from IMS to DB2 see Specifying DB2 for z/OS groups for IMS online regions in the IMS and DB2 Information Center. The IMS Java application uses a properties file to define the connection values for DB2 as shown below.

```
DB2_URL=jdbc:db2:USIBMTESTDB2              DB2 location name
DB2_Driver=com.ibm.db2.jcc.DB2Driver       DB2 driver being used
DB2_Name=DB2U003.CUSTCRED                  Name of the credit card table
```

IMS Java Setup

We have shared databases set up in IMS so any of our IMS regions are able to run the issuing bank program ISSBANK. We used the following IMS Java settings to access our IMS1 region.

```
IMS_Driver=com.ibm.ims.db.DLIDriver
```

The `acquiringbank.java` and `issuingbank.java` programs use JNI calls for native ICSF services by Common Cryptographic Architecture (CCA) Support Program. (This program is similar to CCA JNI, which is not available for z/OS.) See Linux for System zSecure Key Solution with the Common Cryptographic Architecture Application Programmer's Guide SC33-8294, and IBM 4764 PCI-X Cryptographic Coprocessor web site for additional information.

```
IMS_URL=jdbc:dli:IMSViewIssB/IMS1 – IMSViewIssB extends DLIDatabaseView
```

This reference is to a java program used to declare fields in the IMS database for the credit card data.

It contains all the information about PCB, PSB names, fields short IMS names and their long alias that we use in JDBC.

Several definitions were made for the IMS Java Batch processing (JBP) program.
DFSJVMMS – classpath definitions
DFSVMEV – libpath definitions
DFSJVMAP – maps locations of jar files and associates with PSB names.

WebSphere MQ

Two queue managers are used by this application. One, CSQ4, resides on the same system as the WebSphere Application Server running the Bookstore and the other, CSQ1, resides on a remote system running IMS and DB2. These two queue managers are not in the same queue sharing group so a remote queue is used to send messages to a shared queue. Shared queues are used for the back-end Issuing Bank for availability. Any member of the queue sharing group can process the request for credit card verification.

Remote queues are used to send messages from the front-end Bookstore application to the system running the IMS Issuing Bank program. The Issuing Bank application sends replies back to the Acquiring Bank using a remote queue. Communication between the `qmgrs` is done using MQ sender/receiver shared channels with SSL cipher spec SSL 3.0, Message Digest (version) 5 Hash, 128-bit RC4 encryption.

ICSF

For this application, we attempted to simulate a credit card transaction. We simulate two institutions, the acquiring bank, which is the bank that deals with the retailer, and the issuing bank, the bank that issued the card to the card holder. The acquiring bank and the issuing bank exchange data to determine if the customer supplied valid credit card information and if there is available credit at the account. The data that is passed back and forth between the banks is kept secure by using ICSF callable services and cryptographic hardware.

Key encrypting keys were created beforehand and exchanged between the two banks. These keys were then used to encrypt other keys which would be used to encrypt the customer data and ensure that the data passed between the banks was not tampered with in transit.

Following are the static keys we generated prior to starting the credit card transaction. We created these keys through the ICSF panels or through the ICSF callable services.

- InfoSphere Guardium Data Encryption for DB2 and IMS keys – We created these keys through the ICSF panels to be used in encrypting and decrypting customer data stored in the IMS and DB2 databases.
- CVV keys – We used ICSF callable services CSNBSKM (Multiple Secure Key Import), CSNBKRC (Key Record Create) and CSNBKRW (Key Record Write) to generate keys used in the creation of the CVV value which is the numerical value on the back of your credit card.
- Key encrypting keys for DATA and MAC keys – We used the ICSF panels to create keys for encrypting and decrypting the DATA and MAC keys

For each transaction, the encrypted customer data and the generated MAC (message authentication code) are sent to the issuing bank from the acquiring bank. The issuing bank decrypts the customer data and verifies, using the MAC that the data has not been

tampered with. The issuing bank, then searches for that customer in the IMS or DB2 database to verify valid credit card information and available credit. Note that the IMS and DB2 tables, when created were encrypted using InfoSphere Guardium Data Encryption for DB2 and IMS. If the customer credit card information is valid, we then check to see if there is available credit on the account. The issuing bank, similar to what the acquiring bank did, encrypts the customer data including the approval or disapproval and the generated MAC and sends this information to the acquiring bank via WebSphere MQ. The acquiring bank decrypts the data, verifies the MAC and notifies the retailer.

Following are the ICSF callable services we used in the credit card transaction:

- CSNBCSG (VISA CVV Service Generate) – We use the CVV keys and this callable service to generate a CVV, VISA Card Verification Value, which is the numerical value on the back of your credit card. We generated the CVV to simulate a customer entering this code when placing the order.
- CSNBKGN (Key Generate) – We use this service along with the key encrypting keys created above to generate the DATA key and the MAC key.
- CSNBENC (Encipher) – We then use the DATA key and this callable service to encipher the customer data.
- CSNBMGN (MAC Generate) – Using this callable service and the MAC key, we generate a MAC.
- CSNBKIM (Key Import) – This service is used to reencipher the DATA key and the MAC key from encryption under the key encrypting keys created above, to encryption under the master key.
- CSNBMVR (MAC Verify) – To verify that the date has not been tampered with in transit, we use this callable service and the reenciphered MAC key.
- CSNBDEC (Decipher) - Once we have determined the data has not been tampered with, we use the reenciphered DATA key and this service to decipher the customer data.
- CSNBCSV (VISA CVV Service Verify) – We use this service and the CVV key generated above to verify that indeed the CVV (simulated) entered by the customer is correct.

RACF

In our environment, we use RACF to control which applications can use the generated keys and ICSF callable services . This ensures that the keys and services are used only by authorized users and jobs. For this application, we had to give the appropriate userids access to the callable services and keys mentioned above.

WebSphere Application Server

We developed the Credit Card enhancement to the Bookstore application using Rational Application Developer (RAD). An EAR file was exported from within RAD and installed into a WebSphere Application Server using the WebSphere Application Server Administrative Console. Updates to the application deployed to WebSphere Application

Server were required in order for the Java application to call the C program which invokes the ICSF callable services.

The following updates were made in the WebSphere Application Server Admin Console:

Application servers > wst1s74 > Process definition > Servant > Environment Entries – a libpath entry was added set to the path where our java program files are located.

Environment -> Shared Libraries – Classpath and Native Library Path entries were added referring to the application jar files and shared library file (.so).

Enterprise Applications > zipSeriesStore > Shared library references - A shared library reference was added to the Bookstore Credit Card application.

Our current Bookstore application already had resources defined to access DB2 and WebSphere MQ so no new references were required for our Credit Card version.