DSPREAD

QPOS BOOK

This book is only providing general information, not instead of banking, financial, payment gateway and other professional suggestions and service. The author makes no representation or warranty, either express or implied, with respect to the book, its quality, accuracy, or fitness for a specific application.

Therefore, the author shall have no liability to you or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by the book. This include, but is not limited to, interruption of service, loss of data, loss of consulting or anticipatory profits or consequential damages from the use of the book

BJ-16-09-09

Summary

QPOS is a mobile payment card reader device with pinpad that works with mobile devices such as smart phone. It provides merchants and consumers a safe and convenient way to make mobile payments. QPOS can communicate with the mobile device through many methods, such as: audio jack, Bluetooth and USB cable.

QPOS BT Reader is another mobile payment card reader device without pinpad. QPOS Reader can communicate with the mobile device through audio jack or USB cable.

Features:

· Ensure secure transactions: integrated keyboard and multiple encryption algorithms to ensure secure transactions.

· Accept all types of bank card: supports magnetic stripe card, contact EMV IC card and contactless EMV IC card.

· Adapts to more smart devices through audio jack: Supports over 2,000 smart devices through audio jack.

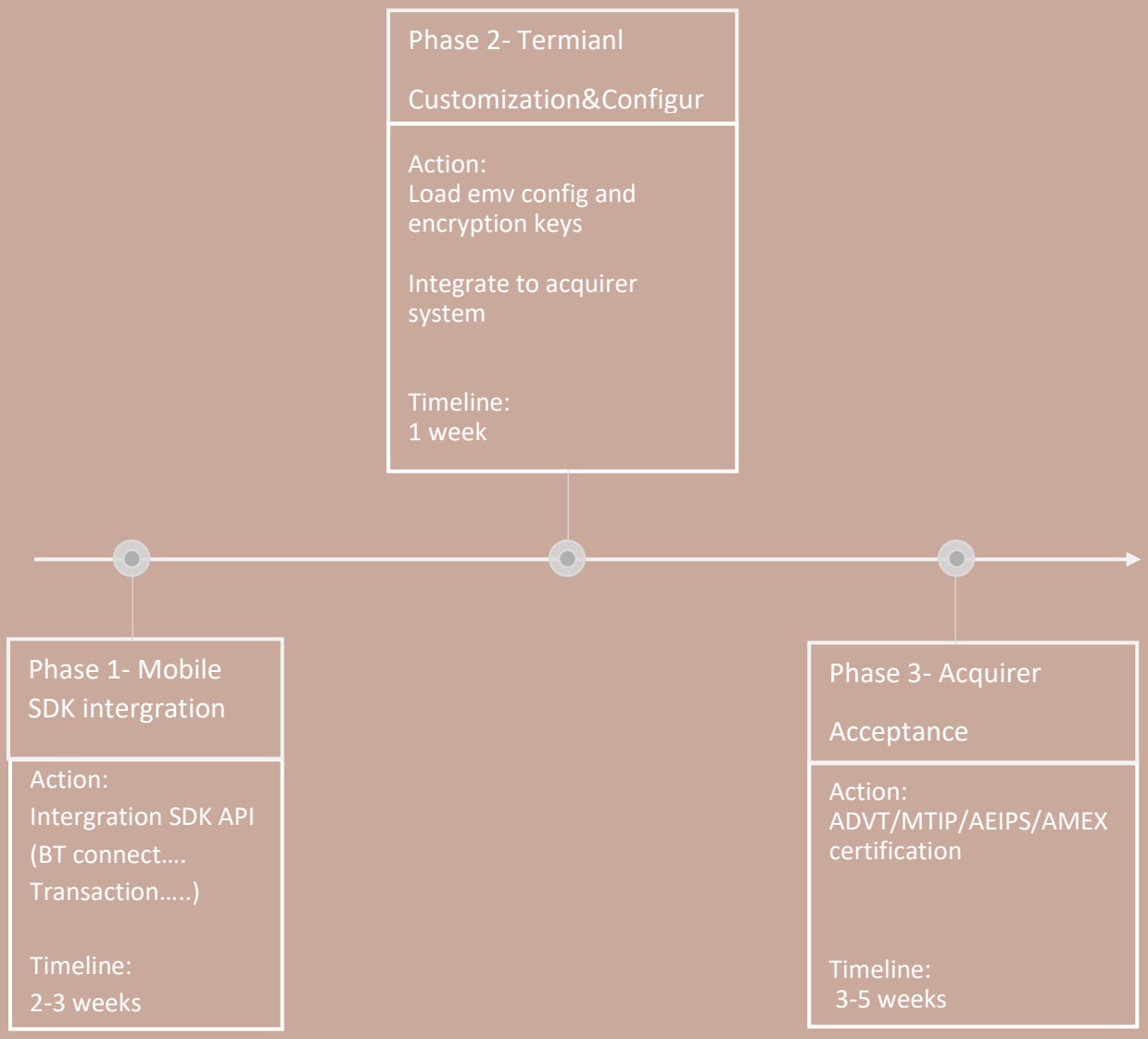
· Fulfill global standards: EMV L1&L2, PCI PTS and more.

· Supports all types of mobile systems: Such as iOS, Android, Windows phone and PC OS.

This document is to help readers to integrate QPOS android SDK into their mobile payment APPs.

Overview

POS Integration Development Phases



CONTENT

1. Request Sample Devices

1. Fill application form
2. Receive samples and SDK

2. Integration DSPREAD SDK

1. Work on QPOS demo project
   - Import demo project
   - Simulate real transaction
   - Point to Point Encryption (P2PE)
2. Implement QPOS SDK
   - Start transaction
   - EMV transaction for IC
   - Online processing
   - Completion

3. Key Management (DUKPT)

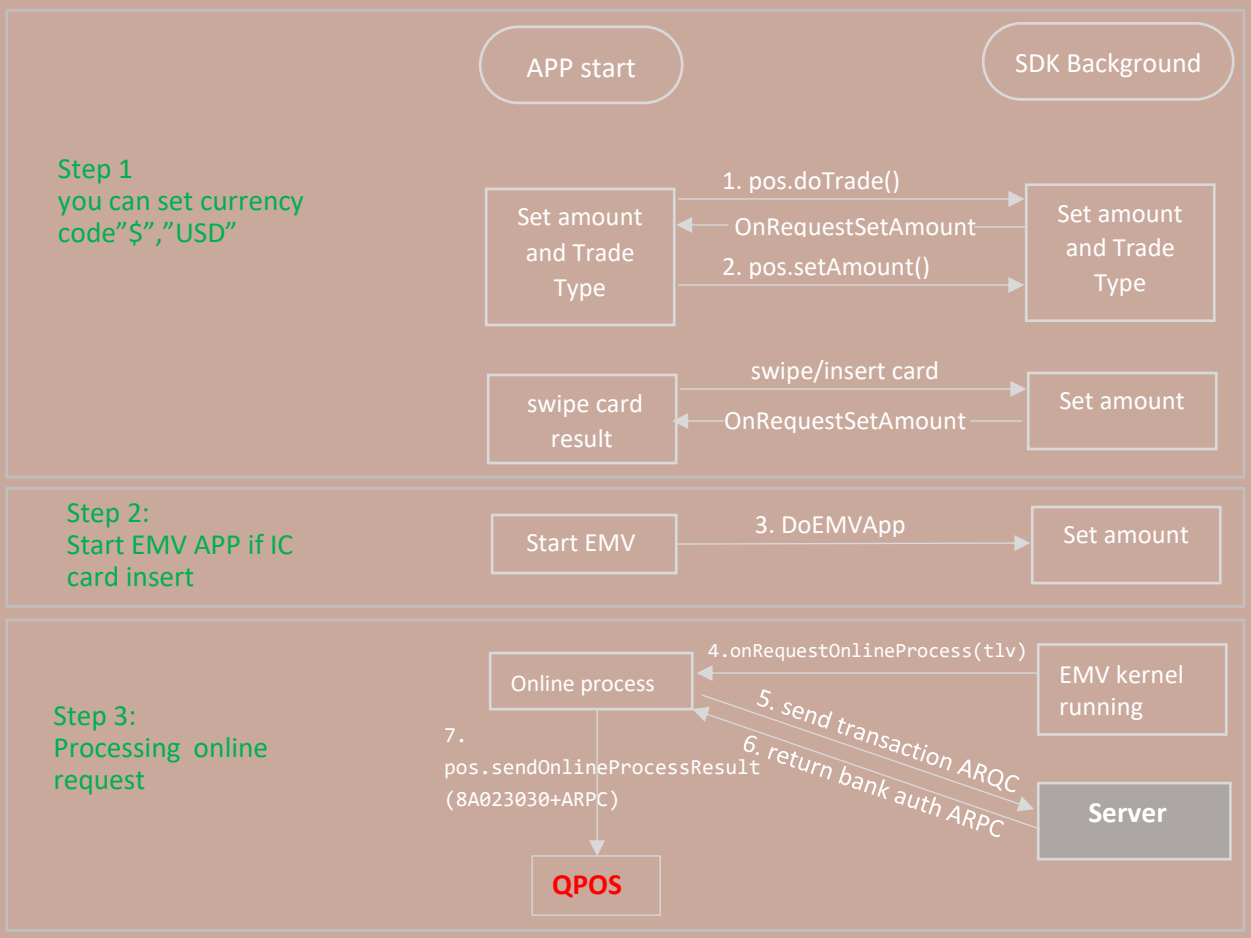
1. Key generation
2. Key distribution
3. Key updating

4. EMV AID and CAPK List Download

Annex-Production Preparation

Phase 1

Mobile SDK Intergration



This chapter we will focus on bellow three part

- Transaction Processing API
- Key Injection And EMV AID/CAPK Loading API
- Firmware Upgrade OTA (Over The Air) API

1 Transaction Processing API

Transaction Execution Flow

1.pos.doTrade(HashTable)/pos.doTrade(timeout)

setAmount()

2. onDoTradeResult()

stratEmvApp()

SetTransactionTime

Input Pin

Start EMV app-multiple app

MSR Completed

3. onRequestOnlineProcess(String tlv)

sendOnlineResult()

4. onRequestTransactionResult

ICC Completed

Get Transaction Result

The application will be notified by the SDK regarding the transaction result by:

```
    @Override
    public void onRequestTransactionResult(
       TransactionResult transactionResult) {
       if (transactionResult == TransactionResult.APPROVED) {
       } else if (transactionResult == TransactionResult.TERMINATED) {
       } else if (transactionResult == TransactionResult.DECLINED) {
       } else if (transactionResult == TransactionResult.CANCEL) {
       } else if (transactionResult == TransactionResult.CAPK_FAIL) {
       } else if (transactionResult == TransactionResult.NOT_ICC) {
       } else if (transactionResult == TransactionResult.SELECT_APP_FAIL) {
       } else if (transactionResult == TransactionResult.DEVICE_ERROR) {
       } else if (transactionResult == TransactionResult.CARD_NOT_SUPPORTED) {
       } else if (transactionResult == TransactionResult.MISSING_MANDATORY_DATA) {
       } else if (transactionResult == TransactionResult.CARD_BLOCKED_OR_NO_EMV_APPS) {
       } else if (transactionResult == TransactionResult.INVALID_ICC_DATA) {
       }
    }
  }
```

Batch Data Handling

When the transaction is finished. The batch data will be returned to the application by below callback.

```
  @Override
     public void onRequestBatchData(String tlv) {
     }
```

Note, if there is issuer's script result inside the tlv, the mobile app need to feedback it to the bank. Decoding the tlv inside **onRequestBatchData** is similar to decoding **onRequestOnlineProcess**.

Reversal Handling

If the EMV chip card refuse the transaction, but the transaction was approved by the issuer. A reversal procedure should be initiated by the mobile app. The requred data for doing reversal can be got by below call back:

```
    @Override
    public void onReturnReversalData(String tlv) {
       ...
    }
```

2 Key Injection And EMV AID/CAPK Loading API

DUKPT Scheme Key Injection

a) Digital envelope

MainActivity.java - pos.udpateWorkKey(String envelope)

```
String str="680100005b2253fa80.............................fffffffffff";
pos.udpateWorkKey(str);
```

a) Transmission Key

MainActivity.java - packInjectKey(trackksn, trackipek, emvksn, emvipek, pinksn,

```
String trackipek = "76A368F34F31363B8EE561F52BDC1D12";
String trackksn = "00000332100300E00001";

String emvipek = "76A368F34F31363B8EE561F52BDC1D12";
String emvksn = "00000332100300E00001";

String pinipek = "76A368F34F31363B8EE561F52BDC1D12";
String pinksn = "00000332100300E00001";

String transmissionkey = "11111111111111111111111111111111";
String protectKey = "8FFF16FC564609EA11E91A6783FD2744";

String result = packInjectKey(trackksn, trackipek, emvksn, emvipek, pinksn,
pinipek, tmk, protectKey);
```

MK/SK Scheme Key Injection

MainActivity.java - udpateWorkKey(pik, pikCheck, trk, trkCheck, mak,makCheck,keyIndex)

```
//step 1.
//Update Master key.New master key must be encrypted by original master key
//check value: new master key encrypt 8 byte 00
pos.setMasterKey(encryptionResult, checkValue, keyIndex);

//step 2.
//Update work key.work key must be encrypted by new master key
//checkValue: specifc work key encrypt 8 byte 00
pos.udpateWorkKey(pik, pikCheck, trk, trkCheck, mak,makCheck,keyIndex);
```

Key Words: [#tag]

Digital Envelope MK/SK DUKPT Transmission Key

EMV AID/CAPK Loading

MainActivity.java - updateEmvConfig(emvAppCfg,emvCapkCfg);

```
//emvcfg_app.bin : EMV cards AID list, terminal capabilities,currency code...
//emvcfg_capk.bin: EMV cards CAPK list.
//There 2 bin files need included into project “assets” folder
//for further details , please refer to Appendix

String emvAppCfg = QPOSUtil.byteArray2Hex(readLine("emvcfg_app.bin"));
String emvCapkCfg = QPOSUtil.byteArray2Hex(readLine("emvcfg_capk.bin"));
pos.updateEmvConfig(emvAppCfg,emvCapkCfg);
```

Key Words[#tag]

Appendix

3 Firmware Upgrade OTA (Over The Air) API

MainActivity.java - updatePosFirmware(data, blueTootchAddress)

```
// Frist, you need request specific firmare from DSPREAD to update QPOS
// then, copy the firmware .asc file into “assets” folder
// In case that QPOS will be destroyed by incorrect firmware
// Besides OTG upgrade, we also support PC tool upgrade
// Please refer to the “alternative solution”

byte[] data = readLine("firmware.asc");
pos.updatePosFirmware(data, blueTootchAddress);
```

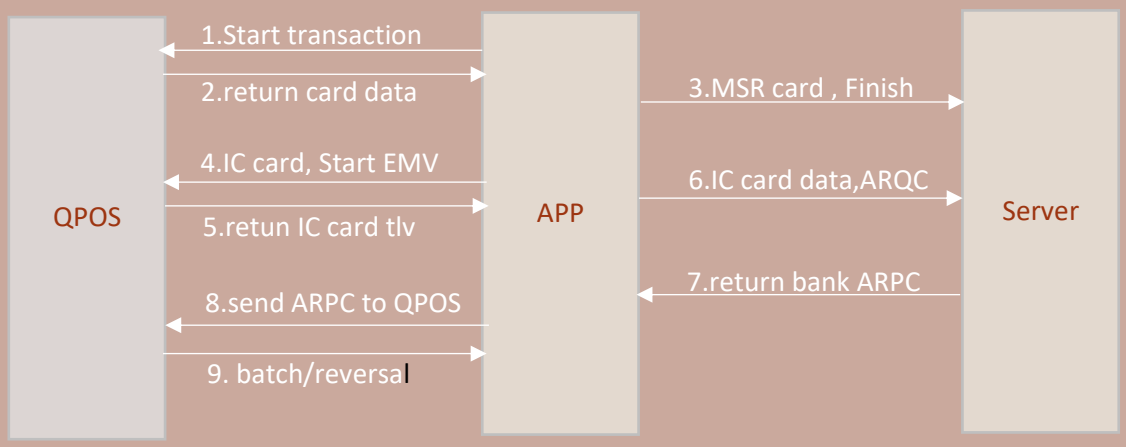
[#tag]

PC tool (windows)

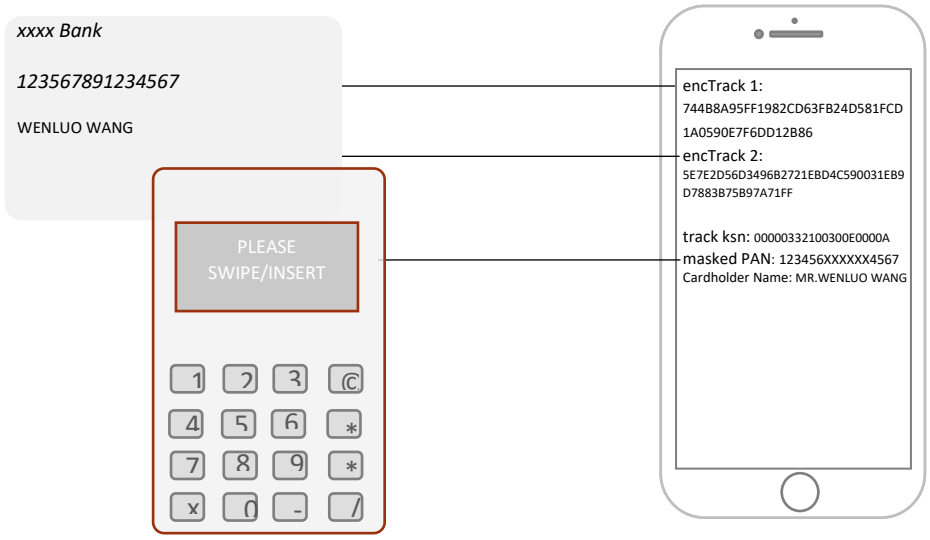
- Please download dspread driver : https://mega.nz/#driver
- Then request firmware loader tool

Phase 2 *DATA PROCESSING*

P2PE-Encryption and Decryption



1 Transaction Result Data : MSR



track ksn 00000332100300E0000A can be used to decode the track data:

enc Track 1 data: 744B8A95FF1982CD63FB24D581FCD1A0590E7F6DD12B86ED1B1D26E687EA853A128598C16B

E14964A34607452511C4B6CBDCACD72BEB566E32094937C18C2424

Track 2 data: 5E7E2D56D3496B2721EBD4C590031EB9D7883B75B97A71FF

Below python script demostrate how to decode track data encrypted with DataKey Variant in 3DES CBC mode:

```
def decrypt_card_info(ksn, data):
    BDK = unhexlify("0123456789ABCDEFFEDCBA9876543210")
    ksn = unhexlify(ksn)
    data = unhexlify(data)
    IPEK = GenerateIPEK(ksn, BDK)
    DATA_KEY_VAR = GetDataKeyVariant(ksn, IPEK)
    print hexlify(DATA_KEY_VAR)
    res = TDES_Dec(data, DATA_KEY_VAR)
    return hexlify(res)
```

Using data key variant to decrypt enc track 1 and enc track2, you will get plaintext track Data: track 1 and track 2. For saving memory space, track 1 and track 2 data are compressed in certain format. So in order to get raw track data, first you need decompress these data to restore raw track 1 and raw track 2 data.

Track 1 data format:

Using data key variant to decrypt track 1, will get:
16259249 54964104 16598554 553FADC8 EEA8BF50 23A25BA7 02886F00 00000000 0003E450 45145059
15D44964 10653590 41041041 F0000000 00000000 00000000

Each character in Track 1 is 6 bits in length, 4 characters are packed into 3 bytes.
Each character is mapped from 0x20 to 0x5F. So to get the real ASCII value of each character, you need to add 0x20 to each decoded 6 bits.

For example, the leading 4 bytes of above track 1 data is 16,25,92,49,54,96

Which in binary is: 00010110 00100101 10010010
Unpacked them to 4 bytes: 000101 100010 010110 010010
Which in hex is:0x05,0x22,0x16,0x12
Add 0x20 to each byte:25423632
Which is in ASCII :%B62

Example – The leading 6 chars in track 1 bit mapping

	Character 1 0x16	Character 2 0x25	Character 3 0x92	Character 4 0x49	Character 5 0x54	Character 6 0x96
	00 01 01 10	00 10 01 01	10 01 00 10	01 00 10 01	01 01 01 00	10 01 01 10
	00 0101	10 0010	01 0110	01 0010	01 0101	01 0010	01 0110
	0000 0101 0x05	0010 0010 0x22	0001 0110 0x16	0001 0010 0x12	0001 0010 0x12	0001 0101 0x15	0001 0010 0x12	0001 0110 0x16
Add 0x20	0x25 (%)	0x42 (B)	0x36 (6)	0x32 (2)	0x32 (2)	0x35 (5)	0x32 (2)	0x35 (6)
Result : %B622526……								

Track 2&Track 3 data format:

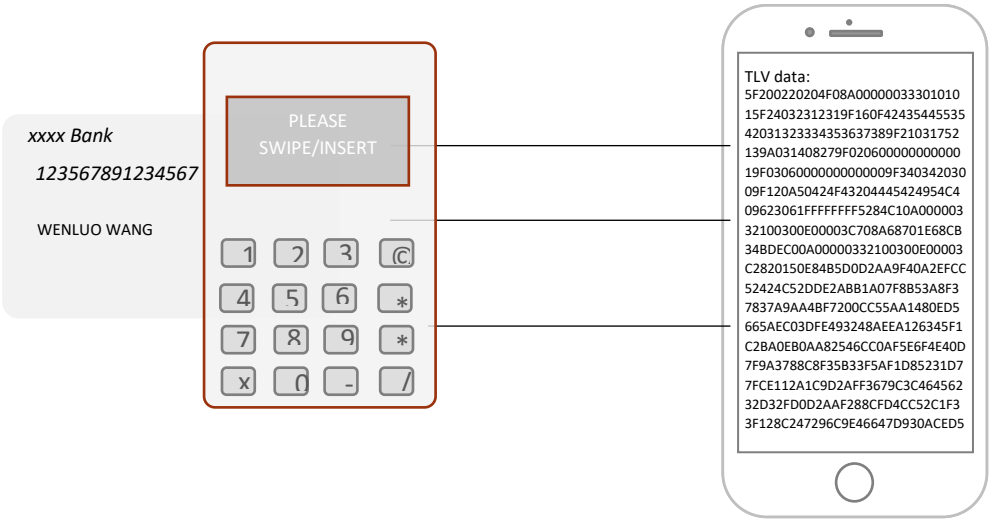
Using data key variant to decrypt track 2, will get:
62252600 06685453 D1011106 17426936 FFFFFFFF FFFFFFFF

Each character in Track 2 & Track 3 is 4 bits in length. 2 characters are packed into 1 byte and padded with zero before encryption

Note:

"D" is stand for **"="**
So PAN=62252600 06685453 "D" = Separator Expiry data=1011 Service Code=106

2 Transaction Result Data :ICC



Below is an exmple of tlv data received by onRequestOnlineProcess:
2014-08-27 17:52:21.221 qpos-ios-demo[391:60b] onRequestOnlineProcess tlv =

5F200220204F08A00000033301010 15F24032312319F160F42435445535420313233343536373 89F21031752139
A031408279F020600000000000019F030600000000000009F34034203009F120A50424F43204445424954C4096230
61FFFFFFFF5284C10A00000332100300E00003C708A68701E68CB34BDEC00A00000332100300E00003C2820150E
84B5D0D2AA9F40A2EFCC52424C52DDE2ABB1A07F8B53A8F37837A9AA4BF7200CC55AA1480ED5665AEC03DFE4
93248AEEA126345F1C2BA0EB0AA82546CC0AF5E6F4E40D7F9A3788C8F35B33F5AF1D85231D77FCE112A1C9D2A
FF3679C3C46456232D32FD0D2AAF288CFD4CC52C1F33F128C247296C9E46647D930ACED5B34CFD0C2A823B3F9
1BEC60E8280005CB96C3EFCCC352F0A30F77A2A033361B5C2C720D8B6E85BFA3C589ADBD6FAF15D3C520085A5
276B736860441BB15DBF8FA537708654EE90E32C194D1487362498F59346706FD797DFC8DD28FCF31E7D49886B
A62779EC42411A54F03FE22B9431969B780E8280005CB96C3EEF460C1F76C0F2217EAC9B999E3E03128A93A11A
4FC6885E4106A4EA4D815D10900AC6AC95E3325D585CB8678AE17A4DEE4C45E2E44209B9493B5FD94F3F46CCF
730CD8FED9430B7574CE670018A94907B2AA4B475A93ABF

The tlv data can be decoded using the online EMVlab tool: http://www.emvlab.org/tlvutils/

As we can see from the decoded table:

Tag	Tag Name	Value
5F20	Cardholder Name	
4F	AID	A000000333010101
5F24	App Expiration Date	231231
9F16	Merchant ID	B C T E S T 1 2 3 4 5 6 7 8
9F21	Transaction Time	175213
...	...	...
C4	Masked PAN	623061FFFFFFFF5284
C1	KSN(PIN)	00000332100300E00003
C7	PINBLOCK	A68701E68CB34BDE
C0	KSN Online Msg	00000332100300E00003
C2	Online Message	E84B5D0D2AA9F40A2EFC....

Inside the table, there are:
1. Some EMV TAGs (5F20,4F,5F24 ...) with plain text value.
2. Some Proprietary tags starting with 0xC, in our case C4,C1,C7,C0 and C2.

The defination of proprietary tags can be found below:

Tag	Name	Length(Bytes)
C0	KSN of Online Msg	10
C1	KSN of PIN	10
C2	Online Message(E)	var
C3	KSN of Batch/Reversal Data	10
C4	Masked PAN	0~10
C5	Batch Data	var
C6	Reversal Data	var
C7	PINBLOCK	8

It's the responsibility of the app to handle the online message string, sending them to the bank(the cardd issuer), and check the bank processing result.

C0 : KSN of online Msg
C2 : the encrypted Online Message, usually the app need to send it to the back end system, along with the tag C0 value. The backend system can derive the 3DES key from C0 value, and decrypt the C2 value and get the real online data in plain text format.
The example above is just a demostration. "**8A023030**" is a fake result from back end system

C2 C5 and **C7** length are variable. Please refer to bellow **Coding of Length field**

Byte	Length	Coding
1	0-127	0xxx xxxx
2	128-255	1000 0001 xxxx xxxx
3	256-65535	1000 0010 xxxx xxxx xxxx xxxx

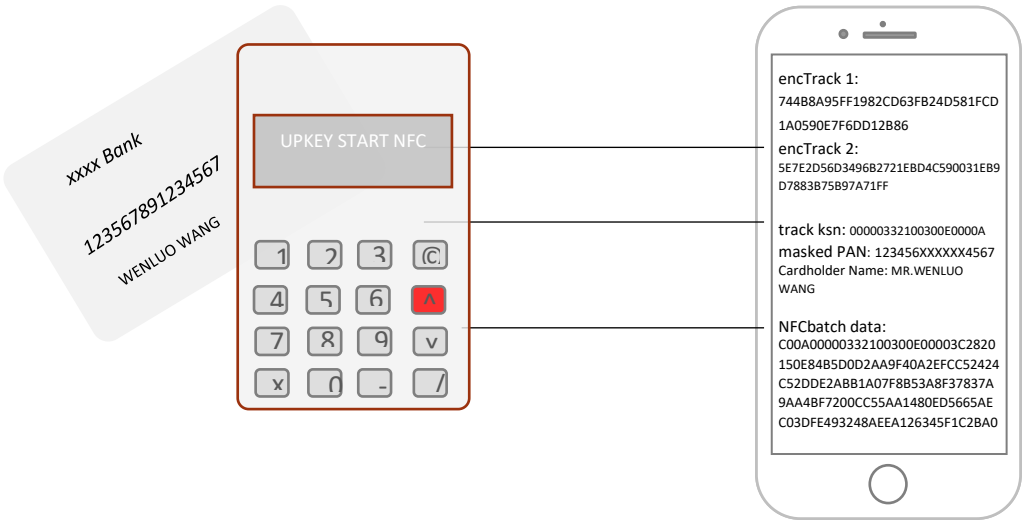
The length field (L) consists of one or more consecutive bytes. It indicates the length of the following field. The length field of the data objects described in this specification which are transmitted over the card-terminal interface is coded on one, two or three bytes
As an exmple of decoding the online message, please find below some demo scripts:

```
BDK = unhexlify("0123456789ABCDEFFEDCBA9876543210")
ksn  = unhexlify("00000332100300E00003")
data = unhexlify(
"E84B5D0D2AA9F40A2EFCC52424C52DDE2ABB1A07F8B53A8F37837A9AA4BF7200CC55AA1480ED5665
AEC03DFE493248AEEA126345F1C2BA0EB0AA82546CC0AF5E6F4E40D7F9A3788C8F35B33F5AF1D85231
D77FCE112A1C9D2AFF3679C3C46456232D32FD0D2AAF288CFD4CC52C1F33F128C247296C9E46647D930
ACED5B34CFD0C2A823B3F91BEC60E8280005CB96C3EFCCC352F0A30F77A2A033361B5C2C720D8B6E85
BFA3C589ADBD6FAF15D3C520085A5276B736860441BB15DBF8FA537708654EE90E32C194D1487362498F
59346706FD797DFC8DD28FCF31E7D49886BA62779EC42411A54F03FE22B9431969B780E8280005CB96C3E
EF460C1F76C0F2217EAC9B999E3E03128A93A11A4FC6885E4106A4EA4D815D10900AC6AC95E3325D585
CB8678AE17A4DEE4C45E2E44209B9493B5FD94F3F46CCF730CD8FED9430B7574CE670018A94907B2AA4
B475A93ABF")
 IPEK = GenerateIPEK(ksn, BDK)
 DATA_KEY = GetDataKey(ksn, IPEK)
 res = TDES_Dec(data, DATA_KEY)
```

The decoded Result:

708201479f02060000000000015a0962306157101017528457136230615710101752 84d231222086038214069f9f1
01307010103a02000010a01000000000013f6c0429f160f4243544553542031323334353637389f4e0f61626364000
0000000000000000000082027c008e0e0000000000000000042031e031f005f24032312315f25031307304f08a000000
3330101019f0702ff009f0d05d8609ca8009f0e0500100000009f0f05d8689cf8009f2608059aae950d0b7a679f27018
09f3602008d9c01009f3303e0f8c89f34034203009f3704c1cdd24a9f3901059f4005f000f0a0019505088004e0009b0
2e8008408a0000003330101019a031408275f2a0201565f3401019f03060000000000009f0902008c9f1a0206439f1
e083833323031494343 9f3501229f4104000000015f200220205f300202205f28020156500a50424f4320444542495
40000000000

3 Transaction Result Data :NFC



To start NFC transaction, You need **press "^" button** to trigger, then tap the bank card that support contactless transaction. Then you will get track data in **onDoTradeResult().** Regarding the NFC batch data, you need call pos.getNFCBatchData()

Track data is same as swipe card and Batch data is same as IC card. So you can refer to swipe and IC card data example for NFC data decryption

PIN format (important):

In case **encrypted PIN** is needed by the transaction, the app can also send pin to back end system by decrypt tag **C7,C1**.
As X9.8-1 requirement. For online interchange transactions, PINs are only encrypted using ISO 9564–1 PIN block formats 0, 1 or 3. By default, QPOS reader is using format 0 (same as ANSI X9.8)

ISO-0 (Format 1, same as ANSI X9.8)

By default, QPOS is using the format 0 under pos.dotrade(). Alternatively we also provide format 1 pinblock for specific purpose
Example:

```
PIN blocks: PIN block encrypt operation finished
 ****************************************
 PAN:            43219876543210987
 PIN:            1234
 PAD:            N/A
 Format:         Format 0 (ISO-0)
 ----------------------------------------
 Clear PIN block:0412AC89ABCDEF67

PIN blocks: PIN block decode operation finished
 ****************************************
 PIN block:      0412AC89ABCDEF67
 PAN:            43219876543210987
 PAD:            N/A
 Format:         Format 0 (ISO-0)
 ----------------------------------------
 Decoded PIN:    1234
```

ISO-0 (Format 0)

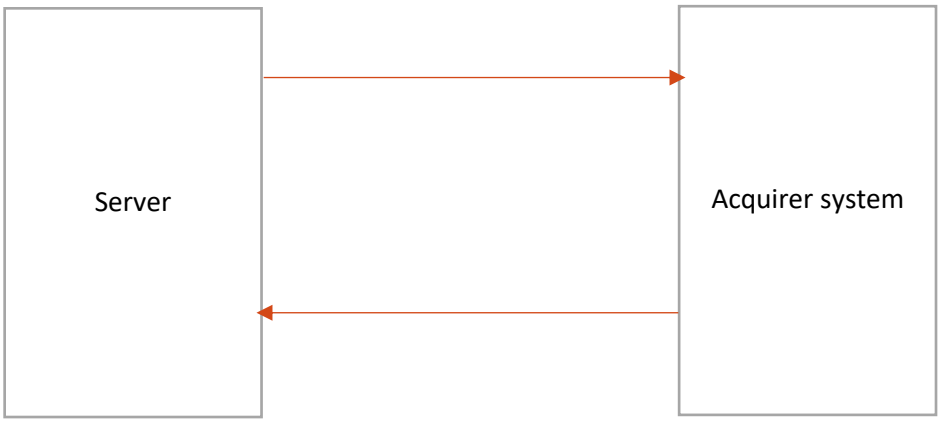
This pinblock format can be achieved by call pos.docheckcard() instead of pos.doTrade(). Then call pos.getPin().
Example:

```
PIN blocks: PIN block encrypt operation finished
 ****************************************
 PAN:            43219876543210987
 PIN:            1234
 PAD:            N/A
 Format:         Format 1 (ISO-1)
 ----------------------------------------
 Clear PIN block:141234CE8C767872

PIN blocks: PIN block decode operation finished
 ****************************************
 PIN block:      141234CE8C767872
 PAN:            N/A
 PAD:            N/A
 Format:         Format 1 (ISO-1)
 ----------------------------------------
 Decoded PIN:    1234
```

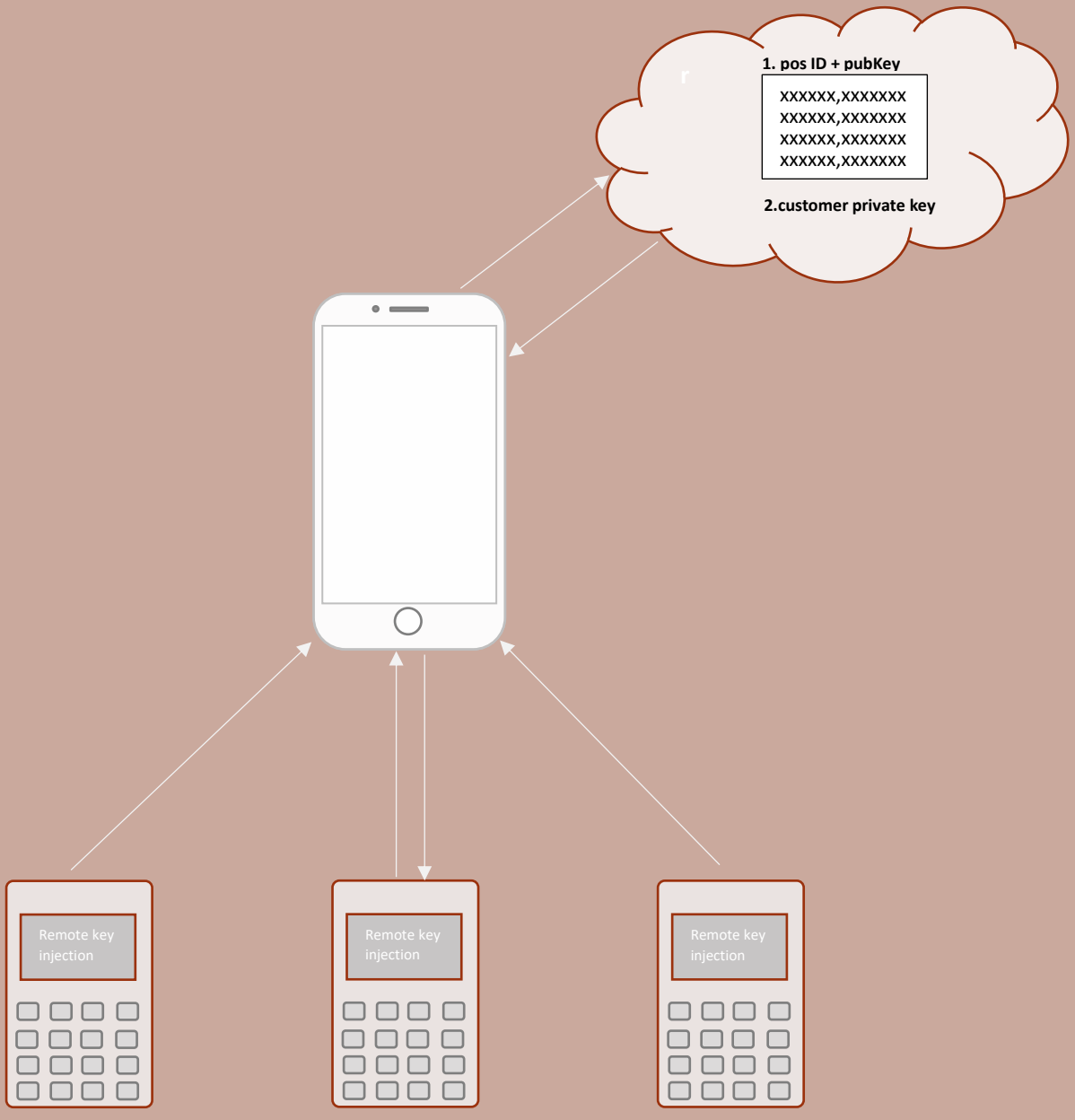
3 Server-Acquirer

Send to Server System



Phase 3 *TRM SYSTEM*

Terminal Management System



1 TRM system-terminal management system

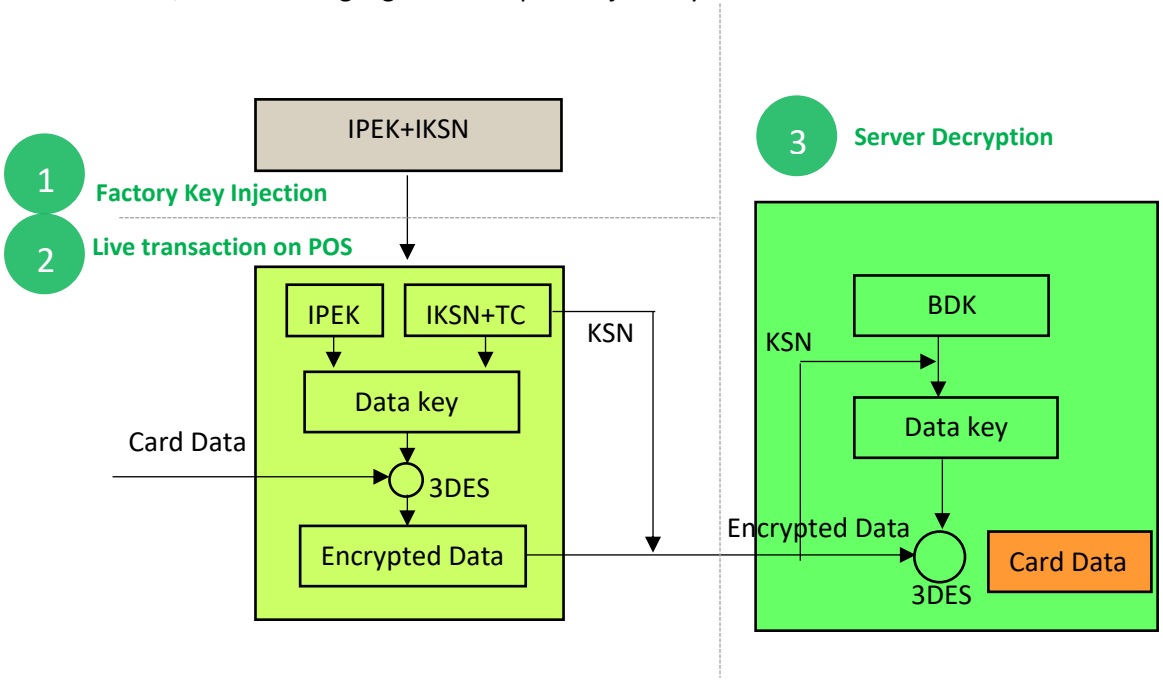
QPOS life cycle is mainly including two steps: Production in factory and TRM on merchant field. This chapter is mainly focus on QPOS key management scheme. Currently QPOS is supporting DUKPT and MK/SK key management scheme. As hardware vendor, DSPREAD will play a role of key injection facility. All the key related part is implemented according to ANSI X9.24 and VISA pin security specification, which is also compliant with PCI pin security requirements.

Since the best industrial practice for key management that most adopted internationally is DUKPT(Derived Unique Key Per Transaction). Hence, this document will only concentrate on illustrate DUKPT (https://en.wikipedia.org/wiki/Derived_unique_key_per_transaction)

Overview on DUKPT

A single BDK (base derived key) able to facilitate 500,000 devices by deriving same amount IPEK keys. Therefore, by having several BDKs you may able to support millions of devices which able to use a unique key for every instance they are going to encrypt data.

You can store your BDKs in an industrial used HSM (Hardware Secure Module). Then send IPEK (Initial Key) file to vendor for devices key injection, also you can inject and update keys on your own app with DSPREAD SDK API, which is using digital envelope to inject keys into terminal



1- First factory will inject the encryption key from Client/Bank, then inject these keys into QPOS

2- After above step, QPOS is capable of protecting card and transaction data by encrypting them. Then send the encrypted card to acquirer.

3- Finally, acquirer will receive and decrypt card and transaction data, then send card data to issuer bank for confirm.

Bellow will illustrate expected key files and format for production by factory

1 Production – key exchange Stage

Before massive production, Customer will be required to send bellow keys to DSPREAD for device initial keys.

Let's say production 1000 device on 2017/02/27. so POSID: xxxxxxxxx17022700001~1000

a) Customers send customer public key (PEM format) and IPEK key files to vendor

IPEK.TXT			
NO	Field	Length	Description
1	KSN	14	This is first 14 digits of KSN that related to last 14 digits of POSID
2	Field separator	1	Comma "', " as field separator
3	IPEK	32	Encrypted IPEK under transmission key
4	Field separator	1	Comma "', " as field separator
5	KCV	16	IPEK key check value

IPEK.txt template

xxx17022700001,AF8167638F879DF76E576E69AACFCE1E,5AA690, 76E576E69AACF EE9
xxx17022700002,F25451C5B3BD929B00BEE907B3F9EBC3,388D18, B3BD929B00BEE9AC
xxx17022700003,403E5A3E174DA57C571F4C5FCACBC7D9,C2919B,74DA57C571F4C5FC
xxx17022700004,1121978EFBEA242DC003FABBD52097E0,377B90, BEA242DC003FABBD

First Column: KSN 1(4 hexadecimal), third column: key check value (6 hexadecimal)
Second Column: encrypted IPEK under transmission key (32 hexadecimal)
Third Column: IPEK key check value

Vendor will load the customer public key and IPEK into devices while manufacture. The public key will be used to verify customer authentication when update customer own keys.

b) Vendor export device(vendor) public key file(csv format) to customer for RKI purpose

PubKey.CSV			
NO	Field	Length	Description
1	POS ID	20	POS ID corresponding to each device public key
2	Field separator	1	Comma "', " as field separator
3	PSAM ID	ID	Device PCB ID
4	Field separator	1	Comma "'," as field separator
5	Public key	16	Each device public key

pubkey.csv template

xxxxxxxxx17022700001, BB8ECD444E8771F91DA1680433F303BFD5571A048E4DBDE73C95A527C353B.....
xxxxxxxxx17022700002, 9615E170F3221DBC7D2C34210323067CFE5616BBFF0CE83C4798B76855FC9C.....
xxxxxxxxx17022700003, BC654B87AD4D9B1A9889E6679B5DC8F261F39D2100405159392E 6DAB2E87.....
xxxxxxxxx17022700004, 6EE02F924178198AA6D0DC6DA6055CD554F2FBF92D88393606A3C091F487D.....

First Column: pos id(20 difits) second column: public key

Device public key is used to encrypt the keys when customer update their own keys so that device can decrypt with its inside device private key to verify the terminal authentication.

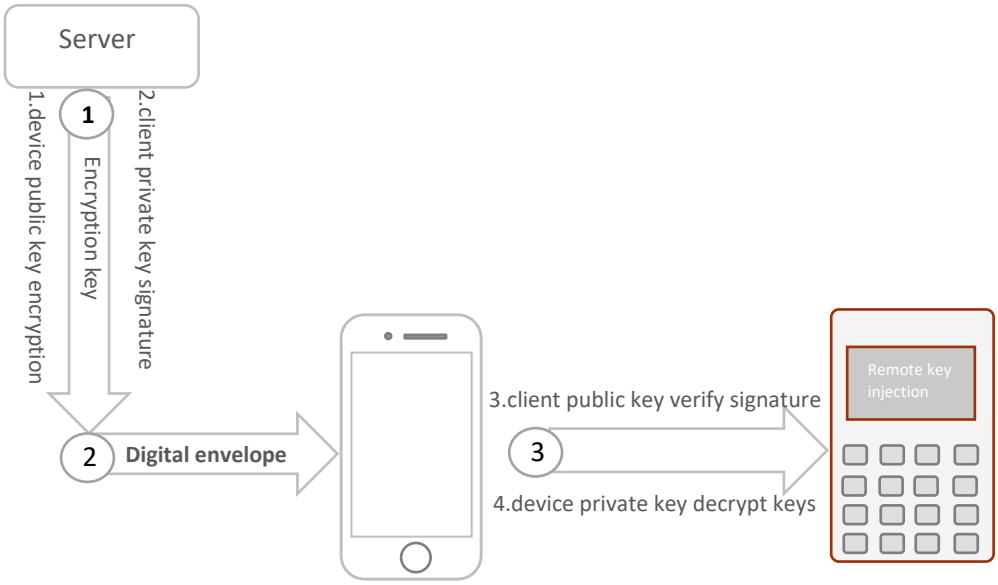
Additional:

so Customer need send the specific production keys to us before production. For TRM management, we will provide each device public rsa key and api so that customers can manage their device remotely with digital envelope.

Digital envelope is achieved under the condition that both **terminal vendor** and **terminal customer** possess one pair of RSA key: **private key** and **public key**. Private key must be kept secret on HSM and never be sharable. And public key will be exchanged between vendor and customer for **digital envelope dual authentication** while customer **RKI(remote key injection)** management , Please refer to bellow specific implementation steps:

2 Key update-RKI (Remote Key Injection)

While at production stage, customer and vendor will exchange each other public key. Because RSA key (private/public key) should be one pair and matched. So it will be used to authenticate each both party identify. After exchange, Device and customer will possess the other party public key, which make secure RKI feasible, the theory is illustrated as bellow:



1 APP request encryption key, server will use device public key to encrypt keys, then use client private key sign on it and return digital envelope to app

2 APP call pos.updateWorkKey(envelope) to inject it into device

3 Terminal use client public key to verify the signature and then decrypt the keys , if both successfully, then save keys into memory. RKI completed !

Digital envelope implementation source code

Firstly, please download the source code by this link:

https://mega.nz/#!1UlXiSxA!bH95hFaBvR9w50rVx67rc0ZdmU91jNFWh6jp-mnmLAE

Then import this project into eclipse, there are only 2 main file to interact-
DukptKeys.java: this file only contains dukpt keys and device public key
Envelope.java: this file is used to implement digital envelope generation

DukptKeys.java

```
public static String dukptGroup="00";        // indicate update specific index group keys

// IPEK and KSN keys for dukpt
public static String trackipek    ="93D67D62337B36F0D739663CABEEFD3C";
public static String emvipek     ="93D67D62337B36F0D739663CABEEFD3C";
 public static String pinipek      ="93D67D62337B36F0D739663CABEEFD3C";

public static String trackksn     = "FFFF9876543210E00000";
public static String emvksn      = "FFFF9876543210E00000";
public static String pinksn       = "FFFF9876543210E00000";
//IPEK should be encrypted under tmk for security, so above IPEK is not in plaintext
public static String tmk          ="5F8B2B8818966C5CD4CC393AF9FC7722";
//RSA_public_key is used to generate digital envelope
public static String RSA_public_key="84F5F1C1CF03D7A9FE7BBA5E8C276B……"
```

Add your own server private key into “keys” folder in the project, then reference it in Envelope.java

Envelope.java

```
//reference your private key in Envelope.java file
senderRsa.loadPrivateKey(new FileInputStream("keys/rsa_private_pkcs8.pem"));
```

Finally, run envelope.java as java application, then you will get digital envelope , call android sdk pos.updateWorkKey(envelope) to inject keys into device

Key Map Inside QPOS

```
public static String dukptGroup="00";        // indicate update specific index group keys
```

Currently QPOS support 4 group dukpt keys, each group contains 3 pair IPEK+KSN
Track IPEK/KSN: used to encrypt track data
Pin IPEK/KSN: used to encrypt pinblock
EMV IPEK/KSN: used to encrypt ICC data

keydukptGroup(0-3):
key index that indicating which group key that you want to update ,

4. Tags included

The reader now included the below EMV standards tags in onRequestOnlineProcess,.

The existence of tags which sourced from ICC depends on ICC card.

Tag	Name	Description	Source
4F	Application Identifier (AID) – card	Identifies the application as described in ISO/IEC 7816-5	ICC
50	Application Labe	Mnemonic associated with the AID according to ISO/IEC 7816-5	ICC
57	Track 2 Equivalent Data	Contains the data elements of track 2 according to ISO/IEC 7813, excluding start sentinel, end sentinel, and Longitudinal Redundancy Check (LRC), as follows: Primary Account Number (n, var. up to 19) Field Separator (Hex 'D') (b) Expiration Date (YYMM) (n 4) Service Code (n 3) Discretionary Data (defined by individual payment systems) (n, var.) Pad with one Hex 'F' if needed to ensure whole bytes (b)	ICC
5A	Application Primary Account Number (PAN)	Valid cardholder account number	ICC
5F20	Cardholder Name	Indicates cardholder name according to ISO 7813	ICC
5F24	Application Expiration Date	Date after which application expires	ICC
5F25	Application Effective Date	Date from which the pplication may be used	ICC
5F30	Service Code	Service code as defined in ISO/IEC 7813 for track 1 and track 2	ICC
5F34	Application Primary Account Number (PAN) Sequence Number	Identifies and differentiates cards with the same PAN	ICC

5F2A	Transaction Currency Code	Indicates the currency code of the transaction according to ISO 4217	ICC
82	Application Interchange Profile	Indicates the capabilities of the card to support specific functions in the application	ICC
84	Dedicated File (DF) Name	Identifies the name of the DF as described in ISO/IEC 7816-4	ICC
89	Authorisation Code	Value generated by the authorisation authority for an approved transaction	Issuer
8A	Authorisation Response Code	Code that defines the disposition of a message	Issuer/Terminal
8E	Cardholder Verification Method (CVM) List	Identifies a method of verification of the cardholder supported by the application	ICC
95	Terminal Verification Results	Status of the different functions as seen from the terminal	Terminal
99	Transaction Personal Identification Number (PIN) Data	Data entered by the ardholder for the purpose of the PIN verification	Terminal
9A	Transaction Date	Local date that the transaction was authorised	Terminal
9B	Transaction Status Information	Indicates the functions performed in a transaction	Terminal
9C	Transaction Type	Indicates the type of financial transaction, represented by the first two digits of ISO 8583:1987 Processing Code	Terminal
9F 02	Amount, Authorised (Numeric)	Authorised amount of the transaction (excluding adjustments)	Terminal
9F 03	Amount, Other (Numeric)	Secondary amount associated with the transaction	Terminal

		representing a cashback amount	
9F 06	Application Identifier (AID) – terminal	Identifies the application as described in ISO/IEC 7816-5	Terminal
9F 07	Application Usage Control	Indicates issuer's specified restrictions on the geographic usage and services allowed for the application	ICC
9F 09	Application Version Number	Version number assigned by the payment system for the application	Terminal
9F0D	Issuer Action Code – Default	Specifies the issuer's conditions that cause a transaction to be rejected if it might have been approved online, but the terminal is unable to process the transaction online	ICC
9F 0E	Issuer Action Code – Denial	Specifies the issuer's conditions that cause the denial of a transaction without attempt to go online	ICC
9F 0F	Issuer Action Code – Online	Specifies the issuer's conditions that cause a transaction to be transmitted online	ICC
9F 10	Issuer Application Data	Contains proprietary Application data for transmission to the issuer in an online transaction	ICC
9F 12	Application Preferred Name	Preferred mnemonic associated with the AID	ICC
9F 16	Merchant Identifier	When concatenated with the Acquirer Identifier, uniquely	Terminal

		identifies a given merchant	
9F1A	Terminal Country Code	When concatenated with the Acquirer Identifier, uniquely identifies a given merchant	Terminal
9F 1E	Interface Device (IFD) Serial Number	Unique and permanent serial number assigned to the IFD by the manufacturer	Terminal
9F 26	Application Cryptogram	Cryptogram returned by the ICC in response of the GENERATE AC command	ICC
9F 27	Cryptogram Information Data	Indicates the type of cryptogram and the actions to be performed by the terminal	ICC
9F 33	Terminal Capabilities	Indicates the card data input, CVM, and security capabilities of the terminal	Terminal
9F 34	Cardholder Verification Method (CVM) Results	Indicates the results of the last CVM performed	Terminal
9F 35	Terminal Type	Indicates the environment of the terminal, its communications capability, and its operational control	Terminal
9F 36	Application Transaction Counter (ATC)	Counter maintained by the application in the ICC(incrementing the ATC is managed by the ICC)	ICC
9F 37	Unpredictable Number	Value to provide variability and uniqueness to the generation of a cryptogram	Terminal
9F 39	Point-of-Service (POS) Entry Mode	Indicates the method by which the PAN was entered, according to the first two digits of the ISO 8583:1987 POS Entry Mode	Terminal

9F 40	Additional Terminal Capabilities	Indicates the data input and output capabilities of the terminal	Terminal
9F 41	Transaction Sequence Counter	Counter maintained by the terminal that is incremented by one for each transaction	Terminal
9F 4E	Merchant Name and Location	Indicates the name and location of the merchant	Terminal
9F 53	Transaction Category Code	Transaction Category Code	Terminal