Overview

This document is applicable to the Android platform SDK of POS(Z90, Z91), MPOS(Z70) and other products

Get start

Android Studio is described here as the default IDE

- Copy SmartPos_xxx.jar to `app\libs`, then right click on it —>add as library
- Copy jniLibs to `src/main`
- Base libs: libSmartPosJni.so，libSmartPos.so， libEmvCoreJni.so，SmartPos_xxx.jar
- QR scanning lib: libjava_camera.so，libsyno_getparam.so，libsyno_jni.so，libsynochip_qrcode.so(Business needs to be contacted for authentication password)
- Declare permissions

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Noted

- Most functions return int type values, detailed type refers to `com.zcs.sdk.SdkResult`
- **It is recommended to obtain the single-threaded thread pool of the SDK through `DriverManager.getInstance().getSingleThreadExecutor` to ensure sequential execution.**

Common class

Instruction

- DriverManager: Used to generate instances of each module operation class
- Sys Used to obtain various device hardware information and system package interfaces
- CardReaderManager: Find cards and get various types of card operations
  - ICCard
  - MagCard
  - RfCard: RF、m1、mifare plus、felica
  - SLE4428Card
  - SLE4442Card
  - IDCard: Only for China
- EmvHandler: Run emv
- PinPadManager: About key and encryption
- Printer: Use to print
- FingerprintManager: Use to get fingerprint
- Led: Use to swich Led light
- Beeper: Use to opearte beeper
- BluetoothHandler: For the Z70 Bluetooth card reader

- ExternalCardManager: For ZCS160 card reader device

Class Instantiation

Get the various module operation classes through the various `getXXX()` functions of `DriverManager`

```
DriverManager mDriverManager= DriverManager.getInstance();
Sys mSys = mDriverManager.getBaseSysDevice();

CardReaderManager mCardReadManager = mDriverManager.getCardReadManager();
ICCard mICCard = mCardReadManager.getICCard();
MagCard mMagCard = mCardReadManager.getMAGCard();
RfCard mRfCard = mCardReadManager.getRFCard();
SLE4428Card mSLE4428Card = mCardReadManager.getSLE4428Card();
SLE4442Card mSLE4442Card = mCardReadManager.getSLE4442Card();
IDCard idCard = mCardReadManager.getIDCard();

EmvHandler mEmvHandler = EmvHandler.getInstance();

PinPadManager mPadManager = mDriverManager.getPadManager();

Printer mPrinter = mDriverManager.getPrinter();

FingerprintManager mFingerprintManager = mDriverManager.getFingerprintManager();

Beeper mBeeper = mDriverManager.getBeeper();
Led mLed = mDriverManager.getLedDriver();

BluetoothHandler mBluetoothHandler = mDriverManager.getBluetoothHandler();

ExternalCardManager mExternalCardManager =
mDriverManager.getExternalCardManager();
```

Initialization

SPI(Default， such as Z90)

```
int statue = mSys.getFirmwareVer(new String[1]);
if (statue != SdkResult.SDK_OK) {
    int sysPowerOn = mSys.sysPowerOn();
    Log.i(TAG, "sysPowerOn: " + sysPowerOn);
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
int i = mSys.sdkInit();
if (i == SdkResult.SDK_OK) {
}
```

Bluetooth(For Z70)

```
private void initSdk() {
```

```
    // Config the SDK base info
    mSys = mDriverManager.getBaseSysDevice();
    mSys.showLog(true);
    mBluetoothManager = BluetoothManager.getInstance()
            .setContext(mActivity)
            .setBluetoothListener(new BluetoothListener() {
                @Override
                public boolean isReader(BluetoothDevice bluetoothDevice) {
                    // Get device searched by bluetooth
                    mAdapter.addDevice(bluetoothDevice);
                    mAdapter.notifyDataSetChanged();
                    return false;
                }

                @Override
                public void startedConnect(BluetoothDevice device) {
                    Log.e(TAG, "startedConnect: ");
                }

                @Override
                public void connected(BluetoothDevice device) {
                    Log.e(TAG, "connected: ");
                    mHandler.obtainMessage(MSG_TOAST,
"Connected").sendToTarget();
                    int sdkInit = mSys.sdkInit(ConnectTypeEnum.BLUETOOTH);
                    String initRes = (sdkInit == SdkResult.SDK_OK) ?
getString(R.string.init_success) : SDK_Result.obtainMsg(mActivity, sdkInit);

                    // mBluetoothManager.connect called in sub thread, u need to
switch to main thread when u need to change ui
                    mHandler.obtainMessage(MSG_TOAST, initRes).sendToTarget();
                }

                @Override
                public void disConnect() {
                    Log.e(TAG, "disConnect: ");
                    mHandler.obtainMessage(MSG_TOAST,
"Disconnect").sendToTarget();
                }

                @Override
                public void startedDiscovery() {
                    Log.e(TAG, "startedDiscovery: ");
                }

                @Override
                public void finishedDiscovery() {
                    Log.e(TAG, "finishedDiscovery: ");
                }
            })
            .init();
}
```

USB

```
int openUsb() {
    if (mUsbHandler != null) {
```

```
            mUsbHandler.close();
        }
        mUsbHandler = UsbHandler.getInstance().setContext(this).init();
        int nRet = mUsbHandler.connect();
        showLog("openUsb: " + nRet);
        if (nRet == USBConstants.USB_NO_PERMISSION) {
            mUsbHandler.checkPermission();
            nRet = mUsbHandler.connect();
        }
        if (nRet == 0) {
            nRet = mSys.sdkInit(ConnectTypeEnum.USB);
            showLog("sdkInit:" + nRet);
        }
        return nRet;
    }
```

COM (For Z91)

```
int openSerialPort() {
    int statue = mSys.getFirmwareVer(new String[1]);
    if (statue != SdkResult.SDK_OK) {
        int sysPowerOn = mSys.sysPowerOn();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    return mSys.sdkInit(ConnectTypeEnum.COM);
}
```

Bluetooth card reader Z70 own API

The following only provides unique functions to Z70, ** for more specific calls, please refer to PbocActivity and EmvActivity** in the demo

PBOC(For PBOC card)

** This function reads the card limited, only supports the PBOC card in China, the EMV card please refer to later EMV**

Search card

```
int timeout = 10;
CardReaderTypeEnum cardType = CardReaderTypeEnum.MAG_IC_RF_CARD;
int ret = mBluetoothHandler.searchCard(cardType, timeout);
ret = mBluetoothHandler.cancelSearchCard();
```

Add listener

Search card, keyboard keys, pboc kernel execution result, timeout and other will trigger various callbacks

1. `onKeyEnter`, `onKeyCancel`: trigger when the ok key and the cancel key are clicked
2. `onCardDetect`: `BluetoothHandler.searchCard` will trigger this callback

3. `onEnterPasswordTimeout`: `BluetoothHandler.startInputPin` will trigger this callback
4. `onEmvTimeout`, `onEmvStatus`: trigger by function `BluetoothHandler.emv`

Add listener

```
BluetoothManager mBluetoothManager = BluetoothManager.getInstance();
BluetoothHandler mBluetoothHandler = mDriverManager.getBluetoothHandler();

mBluetoothHandler.addEmvListener(new OnBluetoothEmvListener() {
    @Override
    public void onKeyEnter() {
        Log.e(TAG, "onKeyEnter: ");
    }

    @Override
    public void onKeyCancel() {
        Log.e(TAG, "onKeyCancel: ");
    }

    @Override
    public void onCardDetect(CardDetectedEnum cardDetectedEnum) {
        Log.e(TAG, "onCardDetect: " + cardDetectedEnum.name());
        switch (cardDetectedEnum) {
            case INSERTED:
                showLog("IC card insert");
                emvRet = mBluetoothHandler.emv(CardReaderTypeEnum.IC_CARD, 100,
"20180811121212", 10);
                showLog("Start emv ret:  " + emvRet);
                break;
            case SWIPED:
                getMagData();
                break;
            case CONTACTLESS_FR:
                showLog("Rf card");
                emvRet = mBluetoothHandler.emv(CardReaderTypeEnum.RF_CARD, 100,
10);
                showLog("Start emv ret:  " + emvRet);
                break;
            case REMOVED:
                showLog("IC card remove");
                break;
        }
    }

    @Override
    public void onEmvTimeout() {
        Log.e(TAG, "onEmvTimeout: ");
        showLog("onEmvTimeout");
    }

    @Override
    public void onEnterPasswordTimeout() {
        Log.e(TAG, "onEnterPasswordTimeout: ");
    }

    @Override
    public void onEmvStatus(EmvStatusEnum emvStatusEnum) {
```

```
        Log.e(TAG, "onEmvStatus: " + emvStatusEnum.name());
        showLog("onEmvStatus: " + emvStatusEnum.name());
        if (emvStatusEnum == EmvStatusEnum.PBOC_OK || emvStatusEnum ==
EmvStatusEnum.QPBOC_OK) {
            getEmvData();
        }
    }
});
```

Read PBOC card

```
    private void getEmvData() {
        String filed = mBluetoothHandler.get55Field();
        String track = mBluetoothHandler.getTrack();
        String cardNo = mBluetoothHandler.getCardNo();
        String cardHolder = mBluetoothHandler.getCardHolder();
        String expDate = mBluetoothHandler.getExpDate(track);
        String icSeq = mBluetoothHandler.getIcSeq();
        NFCCardType cardType = mBluetoothHandler.getNFCCardType();
        String encryptTrack = mBluetoothHandler.getEncrypTrackData(0);
        //if (mEmvDialog != null && mEmvDialog.isShowing()) {
        //    mEmvDialog.dismiss();
        //}
        showLog("cardNo:  " + cardNo);
        showLog("field55:  " + filed);
        showLog("track:  " + track);
        showLog("encryptTrack:  " + encryptTrack);
        showLog("cardHolder:  " + cardHolder);
        showLog("expDate:  " + expDate);
        showLog("icSeq:  " + icSeq);
        showLog("cardType:  " + cardType.name());

    }
```

Read magnetic stripe card

```
    private void getMagData() {
        showLog("Mag card swipe");
        CardInfoEntity magReadData = mBluetoothHandler.getMagData();
        MyApp.cardInfoEntity = magReadData;
        if (magReadData.getResultcode() == SdkResult.SDK_OK) {
            String tk1 = magReadData.getTk1();
            String tk2 = magReadData.getTk2();
            String tk3 = magReadData.getTk3();
            String expiredDate = magReadData.getExpiredDate();
            String cardNo = magReadData.getCardNo();
            showLog("tk1:  " + tk1);
            showLog("tk2:  " + tk2);
            showLog("tk3:  " + tk3);
            showLog("expiredDate:  " + expiredDate);
            showLog("cardNo:  " + cardNo);
            showLog("isICCard:  " + mBluetoothHandler.isICChip());
            //searchCard(CardReaderTypeEnum.MAG_CARD);
        } else {
            showLog("Mag card read error:  " + magReadData.getResultcode());
        }
```

```
    }
```

Pinblock

```
// get pin block in OnBluetoothEmvListener.onKeyEnter()
// or stop input in OnBluetoothEmvListener.onKeyCancel()

boolean isEncrypted = true;
// the last param is true means encrypted pinblock, or is plaintext
 mBluetoothHandler.startInputPin((byte) 4, (byte) 12, 0, isEncrypted);

 mBluetoothHandler.closeInputPin();
 mBluetoothHandler.getPinBlock((byte) 0, 0, pan[0]);
```

EMV card

Please refer to the following: EMV

Screen

```
private void setLcdMain() {
    int ret = mBluetoothHandler.LCDMainScreen();
    showLog("Set mpos Lcd main:  " + ret);
}

private void setLcdAmount() {
    int ret = mBluetoothHandler.LCDAmount(50 * 100);
    showLog("Set mpos Lcd amount ￥50:  " + ret);
}

private void showLcdQR() {
    int ret = mBluetoothHandler.LCDQRCodeShow(100, "www.google.com");
    showLog("Set mpos Lcd qr:  " + ret);
}
```

Z91

Read card

Z91 used the native nfc api in android. Reader SDK interface is currently not supported. Please refer to the link for details: Android NFC API

Print

Print using a universal print interface, see below: Print

Card

Search card

Through the `CardInfoEntity` entity class, you can get information about the card return, such as card type, non-card reset information, card ID, etc.

```
mCardReadManager.cancelSearchCard();
mCardReadManager.searchCard(CardReaderTypeEnum.MAG_IC_RF_CARD, 0, mListener);

boolean isM1 = false;
boolean isMfPlus = false;
OnSearchCardListener mListener = new OnSearchCardListener() {
    @Override
    public void onCardInfo(CardInfoEntity cardInfoEntity) {
        CardReaderTypeEnum cardType = cardInfoEntity.getCardExistslot();
        switch (cardType) {
            case RF_CARD:
                // only can get SdkData.RF_TYPE_A / SdkData.RF_TYPE_B /
SdkData.RF_TYPE_FELICA / SdkData.RF_TYPE_MEMORY_A / SdkData.RF_TYPE_MEMORY_B
                byte rfCardType = cardInfoEntity.getRfCardType();
                Log.e(TAG, "rfCardType: " + rfCardType);
                if (isM1) {
                    readM1Card();
                } else if (isMfPlus) {
                    readMFPlusCard();
                } else {
                    if (rfCardType == SdkData.RF_TYPE_FELICA) { // felica card
                        readFelica();
                    } else if (rfCardType == SdkData.RF_TYPE_A || rfCardType ==
SdkData.RF_TYPE_B) {
                        readCpuCard();
                    }
                }
                break;
            case MAG_CARD:
                readMagCard();
                break;
            case IC_CARD:
                readICCard(CardSlotNoEnum.SDK_ICC_USERCARD);
                break;
            case PSIM1:
                readICCard(CardSlotNoEnum.SDK_ICC_SAM1);
                break;
            case PSIM2:
                readICCard(CardSlotNoEnum.SDK_ICC_SAM2);
                break;
        }
    }

    @Override
    public void onError(int i) {
        isM1 = false;
        isMfPlus = false;
        mHandler.sendEmptyMessage(i);
    }

    @Override
    public void onNoCard(CardReaderTypeEnum cardReaderTypeEnum, boolean b) {

    }
};
```

Magnetic stripe cards

`getMagTrackData()` return the magnetic stripe card track data
`getMagReadData()` The track data will be parsed according to the standard bank card format and returned

```
private void readMagCard() {
    // use `getMagReadData` to get mag track data and parse data
    // use `getMagTrackData` to get origin track data
    //CardInfoEntity cardInfo = mMagCard.getMagReadData();
    CardInfoEntity cardInfo = mMagCard.getMagTrackData();
    Log.d(TAG, "cardInfo.getResultcode():" + cardInfo.getResultcode());
    if (cardInfo.getResultcode() == SdkResult.SDK_OK) {
        //String exp = cardInfo.getExpiredDate();
        //String cardNo = cardInfo.getCardNo();
        String tk1 = cardInfo.getTk1();
        String tk2 = cardInfo.getTk2();
        String tk3 = cardInfo.getTk3();
    }
    mMagCard.magCardClose();
}
```

Contact CPU card

```
public static final byte[] APDU_SEND_IC = {0x00, (byte) 0xA4, 0x04, 0x00, 0x0E,
0x31, 0x50, 0x41, 0x59, 0x2E, 0x53, 0x59, 0x53, 0x2E, 0x44, 0x44, 0x46, 0x30,
0x31, 0x00};
public static final byte[] APDU_SEND_RF = {0x00, (byte) 0xA4, 0x04, 0x00, 0x0E,
0x32, 0x50, 0x41, 0x59, 0x2E, 0x53, 0x59, 0x53, 0x2E, 0x44, 0x44, 0x46, 0x30,
0x31, 0x00};
public static final byte[] APDU_SEND_RANDOM = {0x00, (byte) 0x84, 0x00, 0x00,
0x08};
public static final byte[] APDU_SEND_FELICA = {0x10, 0x06, 0x01, 0x2E, 0x45,
0x76, (byte) 0xBA, (byte) 0xC5, 0x45, 0x2B, 0x01, 0x09, 0x00, 0x01, (byte) 0x80,
0x00};


private void readICCard(CardSlotNoEnum slotNo) {
    int icCardReset = mICCard.icCardReset(slotNo);

    int[] recvLen = new int[1];
    byte[] recvData = new byte[300];

    if (icCardReset == SdkResult.SDK_OK) {
        int icRes;
        byte[] apdu;
        if (slotNo != CardSlotNoEnum.SDK_ICC_USERCARD) {
            apdu = APDU_SEND_RANDOM;
        } else {
            apdu = APDU_SEND_IC;
        }
        icRes = mICCard.icExchangeAPDU(slotNo, apdu, recvData, recvLen);
        if (icRes == SdkResult.SDK_OK) {
            String apduRecv =
StringUtils.convertBytesToHex(recvData).substring(0, recvLen[0] * 2);
        }
    }
```

```
    int icCardPowerDown =
mICCard.icCardPowerDown(CardSlotNoEnum.SDK_ICC_USERCARD);
}
```

Contactless CPU card

```
private void readRfCard(byte realRfType) {
    int rfReset = mRfCard.rfReset();
    if (rfReset == SdkResult.SDK_OK) {
        byte[] apduSend;
        if (realRfType == SdkData.RF_TYPE_FELICA) { // felica card
            apduSend = APDU_SEND_FELICA;
        } else {
            apduSend = APDU_SEND_RF;
        }
        byte[] recvData = new byte[300];
        int[] recvLen = new int[1];
        int rfRes = mRfCard.rfExchangeAPDU(apduSend, recvData, recvLen);
        int powerDownRes = mRfCard.rfCardPowerDown();
        if (rfRes == SdkResult.SDK_OK) {
            mHandler.sendEmptyMessage(rfRes);
            String recv = StringUtils.convertBytesToHex(recvData).substring(0,
recvLen[0] * 2);
        }
    }
}
```

Mifare Classic 1k(S50)

- 1k

```
String keyM1 = "FFFFFFFFFFFF";
private void readM1Card() {
    StringBuilder m1_message = new StringBuilder();
    byte[] key = StringUtils.convertHexToBytes(keyM1);
    int status;
    do {
        // sector 10 = 4 * 10
        status = mRfCard.m1VerifyKey((byte) (4 * 10), keyType, key);
        if (status != SdkResult.SDK_OK) {
            break;
        }
        m1_message.append("Read sector 10:");
        for (int i = 0; i < 4; i++) {
            byte[] out = new byte[16];
            status = mRfCard.m1ReadBlock((byte) (4 * 10 + i), out);
            if (status == SdkResult.SDK_OK) {
                m1_message.append("\nBlock").append(i).append(":")
                        .append(StringUtils.convertBytesToHex(out));
            } else {
                break;
            }
        }
    } while (false);
}
```

```
void writeM1() {
    // 1. verify the sector key
    // 2. write it
    byte[] key = StringUtils.convertHexToBytes(keyM1);
    int status = mRfCard.m1VerifyKey((byte) (4 * 10), keyType, key);
    if (status == SdkResult.SDK_OK) {
        for (int i = 0; i < 3; i++) {
            byte[] input =
StringUtils.convertHexToBytes("0123456789ABCDEF0123456789ABCDEF");
                mRfCard.m1WirteBlock((byte) (4 * 10 + i), input);
            }
        }
    }
```

- 4k

```
/**
 * 1K : 16 sector * 4 blocks * 16 bytes = 1024
 * 4K : 32 sector * 4 blocks * 16 bytes(big sector: 0 ~ 127)
 * + 8 sector * 16 blocks * 16 bytes(little sector: 127 ~ 255) = 4096
 */
String keyM1 = "FFFFFFFFFFFF";

void readM14K() {
    byte[] key = StringUtils.convertHexToBytes(keyM1);
    int ret = -1;
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < 32; i++) {
        ret = mRFCard.rfSearchCard(SdkData.RF_TYPE_A, new byte[1], new
byte[300]);
        Log.e(TAG, "readM14K: " + i + "  " + ret);
        ret = mRFCard.m1VerifyKey((byte) (i * 4), (byte) 0, key);
        if (ret == SdkResult.SDK_OK) {
            for (int j = 0; j < 4; j++) {
                byte[] out = new byte[16];
                ret = mRFCard.m1ReadBlock((byte) (i * 4 + j), out);
                if (ret == SdkResult.SDK_OK) {
                    sb.append("\nsector").append(i)
                            .append(" block").append(j).append(": ")
                            .append(StringUtils.convertBytesToHex(out));
                } else {
                    sb.append("\nsector").append(i)
                            .append(" block").append(j).append(": ")
                            .append("read error");
                }
            }
        } else {
            sb.append("\nsector").append(i)
                    .append(" key is error");
        }
    }
    Log.e(TAG, "readM14K: " + sb);
    sb.setLength(0);

    for (int i = 0; i < 8; i++) {
        ret = mRFCard.rfSearchCard(SdkData.RF_TYPE_A, new byte[1], new
byte[300]);
```

```
        Log.e(TAG, "readM14K: " + (32 + i) + "  " + ret);
        ret = mRFCard.m1VerifyKey((byte) (128 + i * 16), (byte) 0, key);
        if (ret == SdkResult.SDK_OK) {
            for (int j = 0; j < 16; j++) {
                byte[] out = new byte[16];
                ret = mRFCard.m1ReadBlock((byte) (128 + i * 16 + j), out);
                if (ret == SdkResult.SDK_OK) {
                    sb.append("\nsector").append(i + 32)
                            .append(" block").append(j).append(": ")
                            .append(StringUtils.convertBytesToHex(out));
                } else {
                    sb.append("\nsector").append(i + 32)
                            .append(" block").append(j).append(": ")
                            .append("read error");
                }
            }
        } else {
            sb.append("\nsector").append(32 + i)
                    .append(" key is error");
        }
    }
    Log.e(TAG, "readM14K: " + sb);
}
```

Mifare Plus

```
String keyMfPlus = "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF";
byte[] addressMfPlus = {0x40, 0x00};
private void readMFPlusCard() {
    StringBuilder m1_mf_puls = new StringBuilder();
    byte[] key = StringUtils.convertHexToBytes(keyMfPlus);
    int status = mRfCard.mFPlusFirstAuthen(addressMfPlus, key);
    if (status == SdkResult.SDK_OK) {
        m1_mf_puls.append("Read sector 0:");
        byte[] outdata = new byte[64];
        if (mRfCard.mFPlusL3Read(StringUtils.convertHexToBytes("0000"), (byte)
4, outdata) == SdkResult.SDK_OK) {
            m1_mf_puls.append(StringUtils.convertBytesToHex(outdata));
        }
    }
}
```

SLE4442

```
private String key = "FFFF";
private int startAddr = 0;
private int len = 127;
private byte[] data;
private byte[] protectedData;

SLE4442Card mSLE4442Card =
DriverManager.getInstance().getCardReadManager().getSLE4442Card();
private int init() {
    int ret = mSLE4442Card.init();
    Log.e(TAG, "init: " + ret);
    showLog("init: " + ret);
```

```
    if (ret != SdkResult.SDK_OK) {
        showLog(getString(R.string.init_failed), Color.RED);
    }
    return ret;
}

private int verifyKey() {
    int ret = mSLE4442Card.verifyKey(StringUtils.convertHexToBytes(key));
    Log.e(TAG, "verify: " + key + "\t" + ret);
    showLog("verify: " + key + "  \t" + ret);
    if (ret != SdkResult.SDK_OK) {
        showLog(getString(R.string.verify_failed), Color.RED);
    }
    return ret;
}

private void read() {
    data = new byte[len];
    int ret = mSLE4442Card.readData(startAddr, len, data);
    Log.e(TAG, "readData: " + ret + "\t" + StringUtils.convertBytesToHex(data));
    String prefix = "readData: " + ret + "\t";
    showLog(prefix + StringUtils.convertBytesToHex(data), Color.RED,
prefix.length() + 32 * 2, 2);
}

private void readProtected() {
    protectedData = new byte[32];
    int ret = mSLE4442Card.readProtectedData(protectedData);
    Log.e(TAG, "readProtectedData: " + ret + "  \t" +
StringUtils.convertBytesToHex(protectedData));
    showLog("readProtectedData: " + ret + "\t" +
StringUtils.convertBytesToHex(protectedData));
}


private void changeKey() {
    int ret = mSLE4442Card.changeKey(StringUtils.convertHexToBytes(key));
    Log.e(TAG, "changeKey: " + ret);
    showLog("changeKey: " + ret);
}

private void write() {
    if (data == null || data.length == 0) {
        showLog(getString(R.string.read_first), Color.RED);
        return;
    }
    int ret = mSLE4442Card.writeData(startAddr, len, data);
    Log.e(TAG, "writeData: " + StringUtils.convertBytesToHex(data) + "\t" +
ret);
    showLog("writeData: " + StringUtils.convertBytesToHex(data) + "\t" + ret);
}

private void writeProtected() {
    if (protectedData == null || protectedData.length == 0) {
        showLog(getString(R.string.read_first), Color.RED);
        return;
    }
```

```
    int ret = mSLE4442Card.writeProtectedData((byte) 0, (byte) 32,
protectedData);
    Log.e(TAG, "writeProtected: " + StringUtils.convertBytesToHex(protectedData)
+ "\t" + ret);
    showLog("writeProtected: " + StringUtils.convertBytesToHex(protectedData) +
"\t" + ret);
}
```

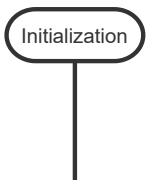
SLE4428

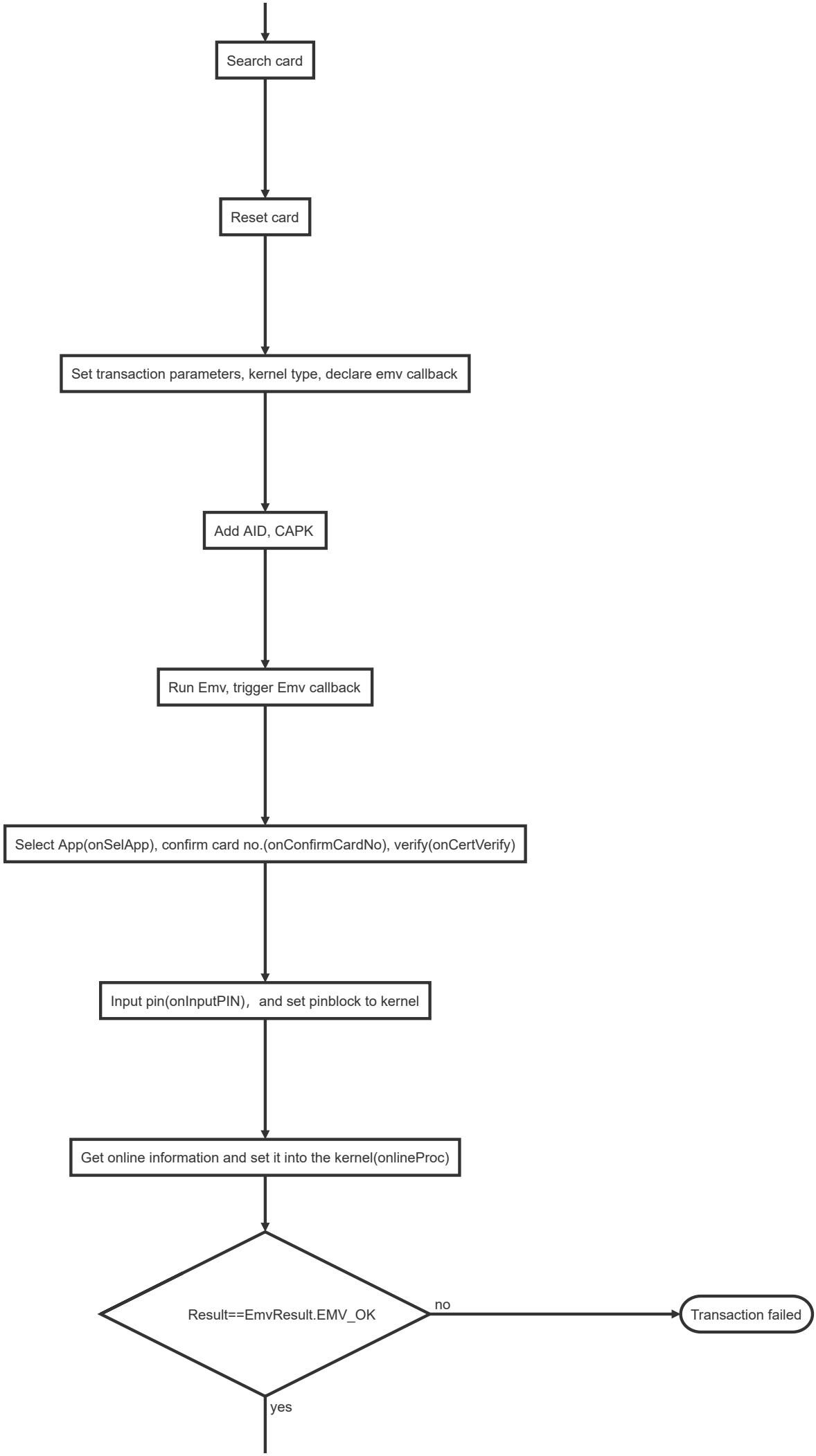
```
private String key = "FFFF";
private int startAddr = 0;
private int len = 127;
private byte[] data;
private byte[] protectedData;

SLE4428Card mSLE4428Card =
DriverManager.getInstance().getCardReadManager().getSLE4428Card();

private int init() {
    int ret = mSLE4428Card.init();
    Log.e(TAG, "init: " + ret);
    showLog("init: " + ret);
    if (ret != SdkResult.SDK_OK) {
        showLog(getString(R.string.init_failed), Color.RED);
    }
    return ret;
}

private int verifyKey() {
    int ret = mSLE4428Card.verifyKey(StringUtils.convertHexToBytes(key));
    Log.e(TAG, "verify: " + key + "\t" + ret);
    showLog("verify: " + key + "  \t" + ret);
    if (ret != SdkResult.SDK_OK) {
        showLog(getString(R.string.verify_failed), Color.RED);
    }
    return ret;
}

private void read() {
    data = new byte[len];
    int ret = mSLE4428Card.readData(startAddr, len, data);
    Log.e(TAG, "readData: " + ret + "\t" + StringUtils.convertBytesToHex(data));
    String prefix = "readData: " + ret + "\t";
    showLog(prefix + StringUtils.convertBytesToHex(data), Color.RED,
prefix.length() + 32 * 2, 2);
}

private void readProtected() {
    protectedData = new byte[32];
    int ret = mSLE4428Card.readProtectedData(startAddr, len, protectedData);
    Log.e(TAG, "readProtectedData: " + ret + "  \t" +
StringUtils.convertBytesToHex(protectedData));
    showLog("readProtectedData: " + ret + "\t" +
StringUtils.convertBytesToHex(protectedData));
}
```

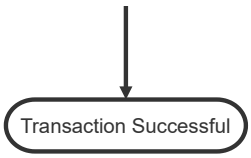
```
private void changeKey() {
    int ret = mSLE4428Card.changeKey(StringUtils.convertHexToBytes(key));
    Log.e(TAG, "changeKey: " + ret);
    showLog("changeKey: " + ret);
}

private void write() {
    if (data == null || data.length == 0) {
        showLog(getString(R.string.read_first), Color.RED);
        return;
    }
    int ret = mSLE4428Card.writeData(startAddr, len, data);
    Log.e(TAG, "writeData: " + StringUtils.convertBytesToHex(data) + "\t" +
ret);
    showLog("writeData: " + StringUtils.convertBytesToHex(data) + "\t" + ret);
}

private void writeProtected() {
    if (protectedData == null || protectedData.length == 0) {
        showLog(getString(R.string.read_first), Color.RED);
        return;
    }
    int ret = mSLE4428Card.writeProtectedData((byte) 0, (byte) 32,
protectedData);
    Log.e(TAG, "writeProtected: " + StringUtils.convertBytesToHex(protectedData)
+ "\t" + ret);
    showLog("writeProtected: " + StringUtils.convertBytesToHex(protectedData) +
"\t" + ret);
}
```

EMV

For specific interfaces and parameter meanings, please refer to EMV_API.doc




Search card
Reset card
Set transaction parameters, kernel type, declare emv callback
Add AID, CAPK
Run Emv, trigger Emv callback
Select App(onSelApp), confirm card no.(onConfirmCardNo), verify(onCertVerify)
Input pin(onInputPIN)，and set pinblock to kernel
Get online information and set it into the kernel(onlineProc)
Result==EmvResult.EMV_OK
no
Transaction failed
yes



Initialization

```
emvHandler = EmvHandler.getInstance();
mPinPadManager = mDriverManager.getPadManager();
 byte[] pucIsEcTrans = new byte[1];
 byte[] pucBalance = new byte[6];
 byte[] pucTransResult = new byte[1];
```

Callback

```
OnEmvListener onEmvListener = new OnEmvListener() {
    @Override
    public int onSelApp(String[] appLabelList) {
        Log.d("Debug", "onSelApp");
        return 0;
    }

    @Override
    public int onConfirmCardNo(String cardNo) {
        Log.d("Debug", "onConfirmCardNo");
        String[] track2 = new String[1];
        final String[] pan = new String[1];
        emvHandler.getTrack2AndPAN(track2, pan);
        int index = 0;
        if (track2[0].contains("D")) {
            index = track2[0].indexOf("D") + 1;
        } else if (track2[0].contains("=")) {
            index = track2[0].indexOf("=") + 1;
        }
        final String exp = track2[0].substring(index, index + 4);
        showLog("cardNum:" + pan[0]);
        showLog("exp:" + exp);
        return 0;
    }

    @Override
    public int onInputPIN(byte pinType) {
        // 1. open the secret pin pad to get pin block
```

```
        // 2. send the pinBlock to emv kernel
        if (emvTransParam.getTransKernalType() ==
EmvData.KERNAL_CONTACTLESS_ENTRY_POINT) {
            String[] track2 = new String[1];
            final String[] pan = new String[1];
            emvHandler.getTrack2AndPAN(track2, pan);
            int index = 0;
            if (track2[0].contains("D")) {
                index = track2[0].indexOf("D") + 1;
            } else if (track2[0].contains("=")) {
                index = track2[0].indexOf("=") + 1;
            }
            final String exp = track2[0].substring(index, index + 4);
            showLog("card:" + pan[0]);
            showLog("exp:" + exp);
        }
        Log.d("Debug", "onInputPIN");
        int iRet = 0;
        iRet = inputPIN(pinType);
        Log.d("Debug", "iRet=" + iRet);
        if (iRet == EmvResult.EMV_OK) {
            emvHandler.setPinBlock(mPinBlock);
        }
        return iRet;
    }

    @Override
    public int onCertVerify(int certType, String certNo) {
        Log.d("Debug", "onCertVerify");
        return 0;
    }

    @Override
    public byte[] onExchangeApdu(byte[] send) {
        Log.d("Debug", "onExchangeApdu");
        if (realCardType == CardReaderTypeEnum.IC_CARD) {
            return mICCard.icExchangeAPDU(CardSlotNoEnum.SDK_ICC_USERCARD,
send);
        } else if (realCardType == CardReaderTypeEnum.RF_CARD) {
            return mRFCard.rfExchangeAPDU(send);
        }
        return null;
    }

    @Override
    public int onlineProc() {
        // 1. assemble the authorisation request data and send to bank by using
get 'emvHandler.getTlvData()'
        // 2. separateOnlineResp to emv kernel
        // 3. return the callback ret
        Log.d("Debug", "onOnlineProc");
        byte[] authRespCode = new byte[3];
        byte[] issuerResp = new byte[512];
        int[] issuerRespLen = new int[1];
        int iSendRet = emvHandler.separateOnlineResp(authRespCode, issuerResp,
issuerRespLen[0]);
        Log.d("Debug", "separateOnlineResp iSendRet=" + iSendRet);
        return 0;
```

```
    }
};
```

Set transaction params

```
        // 2. set params, use `EmvTransParam` to set transaction date, amount,
transaction number etc.
        // setKernelType
        final EmvTransParam emvTransParam = new EmvTransParam();
        if (cardType == CardReaderTypeEnum.IC_CARD) {
            emvTransParam.setTransKernalType(EmvData.KERNAL_EMV_PBOC);
        } else if (cardType == CardReaderTypeEnum.RF_CARD) {

emvTransParam.setTransKernalType(EmvData.KERNAL_CONTACTLESS_ENTRY_POINT);
        }
        emvHandler.transParamInit(emvTransParam);
        final EmvTermParam emvTermParam = new EmvTermParam();
        emvHandler.kernelInit(emvTermParam);

        // 3. add aid or capk
        // add app and capk is persistently stored in the sdcard, if you need to
add dynamically, please be sure to clear the previous
        //emvHandler.delAllApp();
        //emvHandler.delAllCapk();
        //loadVisaAIDs(emvHandler);
        //loadMasterCardCapks(emvHandler);
```

Add AID, CAPK

Note that adding AID and CAPK needs to be executed after `emvHandler.kernelInit(emvTermParam)`, and AID and CAPK will be stored in the file system persistently, so avoid adding it repeatedly. Available via `emvHandler.delAllApp()`, `emvHandler .delAllCapk()` to clear before added

```
private void loadVisaAIDs(EmvHandler emvHandle) {
    // Visa Credit/Debit
    EmvApp ea = new EmvApp();

    ea.setAid("A0000000031010");
    ea.setSelFlag((byte) 0);
    ea.setTargetPer((byte) 0x00);
    ea.setMaxTargetPer((byte) 0);
    ea.setFloorLimit(1000);
    ea.setOnLinePINFlag((byte) 1);
    ea.setThreshold(0);
    ea.setTacDefault("0000000000");
    ea.setTacDenial("0000000000");
    ea.setTacOnline("0000000000");
    ea.settDOL("0F9F02065F2A029A039C0195059F3704");
    ea.setdDOL("039F3704");
    ea.setVersion("008C");
    ea.setClTransLimit("000000015000");
    ea.setClOfflineLimit("000000008000");
    ea.setClCVMLimit("000000005000");
```

```
    ea.setEcTTLVal("000000100000");

    emvHandle.addApp(ea);
}

private void loadMasterCardCapks(EmvHandler emvHandle) {
    EmvCapk capk = new EmvCapk();
    capk.setKeyID((byte) 0x05);
    capk.setRID("A000000004");
    capk.setModul("B8048ABC30C90D976336543E3FD7091C8FE4800"
            + "DF820ED55E7E94813ED00555B573FECA3D84AF6"
            + "131A651D66CFF4284FB13B635EDD0EE40176D8B"
            + "F04B7FD1C7BACF9AC7327DFAA8AA72D10DB3B"
            + "8E70B2DDD811CB4196525EA386ACC33C0D9D45"
            + "75916469C4E4F53E8E1C912CC618CB22DDE7C3"
            + "568E90022E6BBA770202E4522A2DD623D180E21"
            + "5BD1D1507FE3DC90CA310D27B3EFCCD8F83DE"
            + "3052CAD1E48938C68D095AAC91B5F37E28BB49EC7ED597");
    capk.setCheckSum("EBFA0D5D06D8CE702DA3EAE890701D45E274C845");
    capk.setExpDate("20211231"); // YYYYMMDD

    emvHandle.addCapk(capk);
    }
```

Run

```
int ret = emvHandler.emvTrans(emvTransParam, onEmvListener, pucIsEcTrans,
pucBalance, pucTransResult);
```

Result

```
- The return value of the function is a constant value defined by the
`EmvResult` class, and `ret==0` represents success.
- `pucTransResult` is a parameter. There are three return values
`EmvData.APPROVE_M` Approve, `EmvData.DECLINE_M` Decline, `EmvData.ONLINE_M`
Need online, only when `pucTransResult[0] == EmvData.APPROVE_M` On behalf of
online transaction success
```

PinPad

Download main key

```
String main_key = "31313131313131313232323232323232";
String pin_key = "BF1CA957FE63B286E2134E08A8F3DDA903E0686F";
String mac_key = "8670685795c8d2ea0000000000000000d2db51f1";
String tdk_key = "00A0ABA733F2CBB1E61535EDCFDC34A93AA3EA2D";
// index: means Key index, 0x00~0x0F
// key: The key length is a multiple of 8.
void upMainKey() {
    int ret = mPadManager.pinPadUpMastKey(0,
            StringUtils.convertHexToBytes(main_key),
            (byte) (main_key.length() / 2));
    Log.e(TAG, "upMainKey: " + ret);
}
```

Download work key

```
// Encrypted text
// Format (20 bytes) = 16 bytes of work key encrypted by the master key + Work key encryption 8 bytes 0 after taking the first 4 bytes
// lenght = 0: means dont download this key
void upWorkKey() {
    int ret = mPadManager.pinPadUpWorkKey(0,
            StringUtils.convertHexToBytes(pin_key), (byte) ((byte) pin_key.length() / 2),
            StringUtils.convertHexToBytes(mac_key), (byte) (mac_key.length() / 2),
            StringUtils.convertHexToBytes(tdk_key), (byte) (tdk_key.length() / 2));
    Log.e(TAG, "upWorkKey: " + ret);
}

// Plain text
// Sent null means dont download this key
status = pinPadManager.pinPadUpPlainWorkKey(index, mainKey, pin, mac, tdk)
```

pinblock (Secure random pin keyboard)

Need to declare the specified Activity in Manifest, and define its style

```
<activity
    android:name="com.zcs.sdk.pin.pinpad.PinPadPasswordActivity"
    android:theme="@style/Theme.WindowActivity">
</activity>

<style name="Theme.WindowActivity" parent="android:style/Theme.Dialog">
        <item name="android:windowNoTitle">true</item>
        <item name="android:windowIsTranslucent">true</item>
        <item name="android:windowBackground">@android:color/transparent</item>
        <item name="android:windowContentOverlay">@null</item>
        <item name="android:windowIsFloating">true</item>
        <item name="android:backgroundDimEnabled">true</item>
        <item name="android:windowAnimationStyle">@android:style/Animation.Dialog</item>
</style>
```

```
pinPadManager.inputOnlinePin(getActivity(), (byte) 6, (byte) 12, 60, true, "5187108106590784", (byte) 0, PinAlgorithmMode.ANSI_X_9_8, new PinPadManager.OnPinPadInputListener() {
    @Override
        public void onError(final int code) {

    }

    @Override
    public void onSuccess(final byte[] bytes) {
        Log.e(TAG, "PinBlock: " + StringUtils.convertBytesToHex(bytes));
    }
});
```

MAC

```
void mac() {
    String mac_data =
"0200302004C030C0981100000000000000000100000802100012325187108106590769 9B0E751AD
D38E0680104995187108106590784D1561561999999993001999000000034343413031 0DD0684236
0105980000521929 8D060D745153979CC00313233343536373831323334353637383930313233343
5313536117A7E3A0DFD41792610000000000000001422000335000601";
    byte[] mac = new byte[8];
    int ret = mPadManager.pinPadMac(0, PinMacTypeEnum.ECB,
            StringUtils.convertHexToBytes(mac_data), mac_data.length() / 2,
            mac);
    Log.e(TAG, "mac: " + ret + "  " + StringUtils.convertBytesToHex(mac));
}
```

Encrypted track data

```
void encryptTrack() {
    String track = "6258091644092434=20102010000089500000";
    // track is ascii string, one letter is one byte
    // hex string: two letter is one byte
    byte[] encryptedTrack = new byte[track.length()];
    int ret = mPadManager.pinPadEncryptTrackData(0,
MagEncryptTypeEnum.UNION_ENCRYPT,
            track.getBytes(), (byte) (track.length()),
            encryptedTrack);
    Log.e(TAG, "encryptTrack: " + ret + "  " + new String(encryptedTrack));
}
```

Encrypt data

```
void encryptData() {
    String dataForDes = "11111111111111111111111111111111";
    byte[] res = new byte[dataForDes.length() / 2];
    int ret = mPadManager.pinPadEncryptData(0, PinWorkKeyTypeEnum.MAC_KEY,
            StringUtils.convertHexToBytes(dataForDes), dataForDes.length() / 2,
res);
    Log.e(TAG, "encryptData: " + ret + "  " +
StringUtils.convertBytesToHex(res));
}
```

About dukpt key

- Download dukpt key

```
private void setDukptKey() {
    String key = "6AC292FAA1315B4D858AB3A3D7D5933A";
    String ksn = "FFFF9876543210E00000";
    int upDukpt = mPadManager.pinPadUpDukpt(0,
StringUtils.convertHexToBytes(key), (byte) (key.length() / 2),
StringUtils.convertHexToBytes(ksn));
}
```

- Get pinblock

```
    private void getPinBlockByDukpt() {
    final byte[] ksn = new byte[10];
        mPadManager.inputOnlinePinByDukpt(getActivity(), (byte) 6, (byte) 12,
60, true, "5187108106590784",
                (byte) 0, PinAlgorithmMode.ANSI_X_9_8, new
PinPadManager.OnPinPadInputListener() {
                    @Override
                    public void onError(final int code) {

                    }

                    @Override
                    public void onSuccess(final byte[] bytes) {
                        Log.e(TAG, "PinBlock: " +
StringUtils.convertBytesToHex(bytes));
                        Log.e(TAG, "ksn: " +
StringUtils.convertBytesToHex(ksn));
                        });
                    }
                }, ksn);
    }
```

- MAC

```
    private void getMacByDukpt() {
        String input =
"0200302004C030C0981100000000000000000010000080210001232518710810659076 99B0E751AD
D38E0680104995187108106590784D1561561999999993001999000000034343413031 0DD0684236
0105980000521929 8D060D745153979CC00313233343536373831323334353637383930313233343
5313536117A7E3A0DFD41792610000000000000000142200033500060 1";
        byte[] outData = new byte[8];
        byte[] ksn = new byte[10];
        int ret = mPadManager.pinPadMacByDukpt(0, PinMacTypeEnum.ECB,
StringUtils.convertHexToBytes(input), input.length() / 2, outData, ksn);
        Log.d(TAG, "pinPadMacByDukpt:" + ret);
        if (ret == SdkResult.SDK_OK) {
            Log.d(TAG, "outData:" + StringUtils.convertBytesToHex(outData));
            Log.d(TAG, "ksn:" + StringUtils.convertBytesToHex(ksn));
        }
    }
```

- Encrypt

```
    private void encryptByDukpt() {
        String input =
"0200302004C030C0981100000000000000000100000802100012325187108106590769 9B0E751AD
D38E0680104995187108106590784D1561561999999993001999000000034343413031 0DD0684236
0105980000521929 8D060D745153979CC00313233343536373831323334353637383930313233343
5313536117A7E3A0DFD417926100000000000000014220003";
        byte[] outData = new byte[input.length() / 2];
        byte[] ksn = new byte[10];
        int ret = mPadManager.pinPadEncryptDataByDukpt(0,
PinWorkKeyTypeEnum.PIN_KEY, StringUtils.convertHexToBytes(input), input.length()
/ 2, outData, ksn);
        Log.d(TAG, "sdkPadEncryptDataByDukpt:" + ret);
        if (ret == SdkResult.SDK_OK) {
            Log.d(TAG, "outData:" + StringUtils.convertBytesToHex(outData));
            Log.d(TAG, "ksn:" + StringUtils.convertBytesToHex(ksn));
        }
    }
```

Print

Built-in SDK printing

If you call the print barcode or QR code interface, please add ZXing-core as the project dependency, otherwise the barcode QR code will not be generated normally.

Print text

```
int printStatus = mPrinter.getPrinterStatus();
if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

} else {
    PrnStrFormat format = new PrnStrFormat();
    format.setTextSize(30);
    format.setAli(Layout.Alignment.ALIGN_CENTER);
    format.setStyle(PrnTextStyle.BOLD);
    format.setFont(PrnTextFont.CUSTOM);
    format.setPath(Environment.getExternalStorageDirectory() +
"/fonts/simsun.ttf");
    mPrinter.setPrintAppendString("POS SALES SLIP"), format);
    format.setTextSize(25);
    format.setStyle(PrnTextStyle.NORMAL);
    format.setAli(Layout.Alignment.ALIGN_NORMAL);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString("MERCHANGT NAME:" + " Test ", format);
    mPrinter.setPrintAppendString("MERCHANT NO:" + " 123456789012345 ", format);
    mPrinter.setPrintAppendString("TERMINAL NAME:" + " 12345678 ", format);
    mPrinter.setPrintAppendString("OPERATOR NO:" + " 01 ", format);
    mPrinter.setPrintAppendString("CARD NO: ", format);
    format.setAli(Layout.Alignment.ALIGN_CENTER);
    format.setTextSize(30);
    format.setStyle(PrnTextStyle.BOLD);
    mPrinter.setPrintAppendString("6214 44** **** **** 7816", format);
    format.setAli(Layout.Alignment.ALIGN_NORMAL);
    format.setStyle(PrnTextStyle.NORMAL);
    format.setTextSize(25);
```

```
    mPrinter.setPrintAppendString(" ----------------------------", format);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    printStatus = mPrinter.setPrintStart();
}
```

Print QR code (This need add ZXing library as dependency)

```
String QR_TEXT = "https://www.baidu.com";

int printStatus = mPrinter.getPrinterStatus();
if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

} else {
    PrnStrFormat format = new PrnStrFormat();
    mPrinter.setPrintAppendQRCode(QR_TEXT, 200, 200,
Layout.Alignment.ALIGN_NORMAL);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendQRCode(QR_TEXT, 200, 200,
Layout.Alignment.ALIGN_OPPOSITE);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendQRCode(QR_TEXT, 200, 200,
Layout.Alignment.ALIGN_CENTER);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    printStatus = mPrinter.setPrintStart();
    if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

    }
}
```

Print Bar code (This need add ZXing library as dependency)

```
String BAR_TEXT = "6922711079066";

int printStatus = mPrinter.getPrinterStatus();
if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

} else {
    PrnStrFormat format = new PrnStrFormat();
    mPrinter.setPrintAppendBarCode(getActivity(), BAR_TEXT, 360, 100, true,
Layout.Alignment.ALIGN_NORMAL, BarcodeFormat.CODE_128);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    printStatus = mPrinter.setPrintStart();
    if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

    }
}
```

Print bitmap

```
private Bitmap mBitmapDef;

int printStatus = mPrinter.getPrinterStatus();
if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

} else {
    if (mBitmapDef == null) {
        try {
            InputStream inputStream =
getActivity().getAssets().open("print_demo.bmp");
            Drawable drawable = Drawable.createFromStream(inputStream, null);
            mBitmapDef = ((BitmapDrawable) drawable).getBitmap();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    PrnStrFormat format = new PrnStrFormat();
    mPrinter.setPrintAppendBitmap(mBitmapDef, Layout.Alignment.ALIGN_CENTER);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    mPrinter.setPrintAppendString(" ", format);
    printStatus = mPrinter.setPrintStart();
    if (printStatus == SdkResult.SDK_PRN_STATUS_PAPEROUT) {

    }

}
```

Print labels (custom device support only)

1. `Printer.labelPrintLocationFeed()` Label repositioning
2. `Printer.labelPrintBackFeed()` Label back
3. `Printer.labelPrintForwardFeed()` Label forward

Every time to reposition to print labels

```
int paperWidth = 360;
int paperHeight = 240;
void singleLabel() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            int ret = 0;
            for (int i = 0; i < 1; i++) {
                ret = mPrinter.labelPrintLocationFeed();
                if (ret == SdkResult.SDK_OK) {
                    ret = mPrinter.labelPrintBackFeed();
                    if (ret == SdkResult.SDK_OK) {
                        mPrinter.setPrintAppendQRCode(QR_TEXT, 220, 220,
Layout.Alignment.ALIGN_CENTER, paperWidth);
                        int printStatus =
mPrinter.setLabelPrintStart(paperWidth, 240);
                    }
                } else
                    break;
```

```
            }
            ret = mPrinter.labelPrintLocationFeed();
        }
    }).start();
}
```

By adjusting the width and height of the paper, calculate the number of lines that should be forward and backward for each print

```
int paperWidth = 360;
int paperHeight = 240;

void loopLabel(final int total, final int interval) {
    isLoop = true;
    new Thread(new Runnable() {
        @Override
        public void run() {
            int count = 0;
            int ret = mPrinter.labelPrintLocationFeed();
            ret = mPrinter.labelPrintBackFeed();
            do {
                mPrinter.setPrintAppendBarCode(getActivity(), BAR_TEXT, 320,
220, true, Layout.Alignment.ALIGN_CENTER, BarcodeFormat.CODE_128, paperWidth);
                int printStatus = mPrinter.setLabelPrintStart(paperWidth,
paperHeight);
                mPrinter.labelPrintForwardFeed();

                if (interval != 0) {
                    SystemClock.sleep(interval * 1000);
                }
            } while (isLoop && (total == 0 || ++count < total));

            ret = mPrinter.labelPrintLocationFeed();
            isLoop = false;
        }
    }).start();
}
```

Virtual Bluetooth printing

In the list of Bluetooth devices you can see a Bluetooth device "ZCSPrint" that has been paired and always exists, which is virtualized by the operating system.
The printer device that came out does not actually exist.
The virtual Bluetooth printing method is to complete the printing by connecting the `ZCSPrint` Bluetooth device and sending ESC&POS commands.
Bluetooth scanning, connecting, sending can refer to the official Google documentation, or refer to the Demohttps://github.com/AnilMH/Android-studio-bluetooth-printer

Fingerprint

Initialization

```
mFingerprintManager = mDriverManager.getFingerprintManager();
mFingerprintManager.admFingerprintManager.init();dFignerprintListener(this);
mFingerprintManager.init();
```

Get fingerprint image

```
mFingerprintManager.capture();

@Override
public void onGetImageComplete(int result, byte[] imgBuff) {
    if (result == 0) {
        try {
            save2File("/sdcard/raw.data", imgBuff);
            SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmss");
            String name = sdf.format(new Date()) + ".bmp";
            // convert raw format image to bmp
            final Bitmap bitmap = mFingerprintManager.generateBmp(imgBuff, files
+ name);
            mHandler.post(new Runnable() {
                @Override
                public void run() {
                    mIvResult.setVisibility(View.VISIBLE);
                    mIvResult.setImageBitmap(bitmap);
                }
            });
        } catch (Exception e) {
            e.printStackTrace();
        }
    } else {
        showLog(result);
    }
}

mFingerprintManager.captureAndGetFeature();

@Override
public void onGetImageFeature(int result, byte[] feature) {
    showLog("onGetImageFeature: ret = " + result + (result == SdkResult.SDK_OK ?
"\tfeature = " + StringUtils.convertBytesToHex(feature) : null));
}

mFingerprintManager.captureAndGetISOFeature();

@Override
public void onGetImageISOFeature(int result, byte[] feature) {
    showLog("onGetImageISOFeature: ret =  " + result + (result ==
SdkResult.SDK_OK ? "\tISO feature = " + StringUtils.convertBytesToHex(feature) :
null));
}
```

Enroll

- Enroll a fingerprint is to store the fingerprint in the fingerprint chip. The size of the fingerprint library is 64 templates, that is, 64 users can be registered.
- The value corresponding to the fingerprint ID is 0~63
- The default registration of the fingerprint module requires 3 fingerprint images, so the callback will be triggered 3 times.
- Any error in the registration process means that the registration failed, that is to say, the three callbacks need to be returned successfully, otherwise the user needs to re-execute all the registration operations and try to register the fingerprint again.

```
mFingerprintManager.enrollment(0);

@Override
    public void onEnrollmentProgress(int fingerId, int remaining, int reason) {
        if (reason == 0 && remaining == 0) {
            showLog("Fingerprint ID:" + fingerId + "  Enrollment success!");
        } else {
            showLog("Fingerprint ID:" + fingerId);
            showLog("remaining times:" + remaining);
            showLog("reason:" + reason);
        }
    }
```

Verify

- verify the specified fingerprint id
- search in full fingerprint storages
- download fingerprint feature to verify with the finger on the sensor
- download fingerprint iso feature to verify with the finger on the sensor
- download fingerprint feature to search finger id in fingerprint module storage
- download fingerprint iso feature to search finger id in fingerprint module storage

```
// verify the specified fingerprint id
mFingerprintManager.authenticate(0, timeout);
// search in full fingerprint storage
mFingerprintManager.authenticate(timeout);
// download fingerprint feature to verify with the finger on the sensor
byte[] feature;
byte[] isoFeature;
mFingerprintManager.verifyWithFeature(feature)
// download fingerprint iso feature to verify with the finger on the sensor
mFingerprintManager.verifyWithISOFeature(isoFeature);
// download fingerprint feature to search finger id in fingerprint module
storage
mFingerprintManager.identifyWithFeature(feature);
// download fingerprint iso feature to search finger id in fingerprint module
storage
mFingerprintManager.identifyWithISOFeature(isoFeature);

@Override
public void onAuthenticationSucceeded(int fingerId, Object obj) {
    showLog("Fingerprint auth successfully:  fingerId = " + fingerId + "  score
= " + obj);
}
```

Beep、LED

```
int ret = mLed.setLed(LedLightModeEnum.RED, true);
int ret = mBeeper.beep(4000, 600);
```

Scan code

- Scan code uses the google open source scan code library ZXing, and has been partially optimized. For specific use, please refer to the java file in com.szzcs.smartpos.scan in demo.
- The company also provides a scan code library with stronger scanning performance. Please contact us for business details.

```
// active the scan lib
private void activateBarcode(String id, final Dialog loading, String sn) {
    SDKUtils.getInstance(MainActivity.this, id)
            .setDeviceId(sn)
            .activeBarcode(new OnActivateListener() {
                @Override
                public void onActivateResult(int code, String error) {
                    Log.e(TAG, "code = " + code + "   error = " + error);
                }

                @Override
                public void onActivateProcess(String msg) {
                    Log.e(TAG, msg);
                }

                @Override
                public void onActivateState(final boolean isActivated) {
                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            loading.dismiss();
                            if (isActivated) {
                                startActivity(new Intent(MainActivity.this,
ScanActivity.class));
                            } else {
                                Toast.makeText(MainActivity.this, "Activated
failed", Toast.LENGTH_SHORT).show();
                            }
                        }
                    });
                }
            });
}
```

About system

Silent installation uninstall

```
// Care about installation results
//SdkResult.SDK_INSTALL_SUCCESS
//SdkResult.SDK_INSTALL_ERROR
//SdkResult.SDK_INSTALL_NONE
//SdkResult.SDK_INSTALLING
//SdkResult.SDK_NO_SIGNATURE
//SdkResult.SDK_SIGNATURE_ERROR
int ret = mSys.installApp(context, "/sdcard/xxx.apk");
int ret = mSys.uninstallApp(context, "com.xxx.xxxx");

// No installation result
mSys.installApp2(context, "/sdcard/xxx.apk");
```

Others

Proguards

```
-keep class com.zcs.base.SmartPosJni{*;}
-keep class com.zcs.sdk.DriverManager{*;}
-keep class com.zcs.sdk.emv.**{*;}
```

Q & A