Dspread PIN CVM APP

Software-based PIN Entry on COTS

Version 1.1.2

December 2024

Document Changes

Date	Version	Author	Description
10th June 2020	1.0.0	Qianmeng Chen	Add CVM API Description
4th July 2020	1.0.1	Wenluo Wang	Add CVM APP transaction Flow Chart
4th Jan 2021	1.0.2	Wenluo Wang	Add enablement token API
30thAug 2021	1.0.3	Mengqiu Ma	Add buid ISO-format4 pinblock code
16th Nov 2021	1.0.4	Zizhou Xu	Add SCRP and APP Mutual Authentication
6th Dec 2021	1.0.5	Zizhou Xu	Add mutual authentication source code in Appendix
17th Jan 2022	1.0.6	Qianmeng Chen	Add the FAQ
22nd Feb 2022	1.0.7	Zizhou Xu	Add Note for Mutual Authentication usage clarification in Appendix; Add sample code for Mutual Authentication on iOS
22nd June 2022	1.0.8	Qianmeng Chen	Add Diffie Hellman details and implement code
26th July 2023	1.0.9	Qianmeng Chen	Update the RSA key size length to 2048
26th Sep 2023	1.1.0	Zhengwei Fang	Add build ISO-4 pinblock iOS code
18nd Dec 2023	1.1.1	Mengqiu Ma	Add SCRP token verification content
2th Dec 2024	1.1.2	Shasha Lv	Update the algorithm to ECDH

Content

1. Background Introduction

Dspread has a long history of being the leader in mPOS solutions. We also have a Universal SDK that can be used for all DSPREAD products making integration easy. CR100 is a superior product with exceptional performance, flexibility and reliability for payment businesses. With CR100, you can accept payments wherever you go.

CR100 is secure card reader with contact and contactless EMV capability, which includes all certified payment kernel (EMV contact L1 & L2, VISA payWave, MasterCard PayPass, AMEX Express Pay, Discover ZIP, China Union qPBOC, Mifare etc.). It is certified with PCI PTS v5.X PIN reader (SCRP) hardware that enforce security PIN entry on any COTS device. And CR100 is using our Universal SDK that provide one set generic PIN CMV APP API that ensure security of PIN convey between COTS and CR100.Based on CR100 PIN CVM APP API, client can easily develop their own PIN CVM APP and integrate to their own Monitoring/Attestation system.

2. MPoC Solution Overview

The following diagram illustrates the flow of a PIN transaction in a software-based PIN entry solution. Steps 1-7 are detailed on the following page.

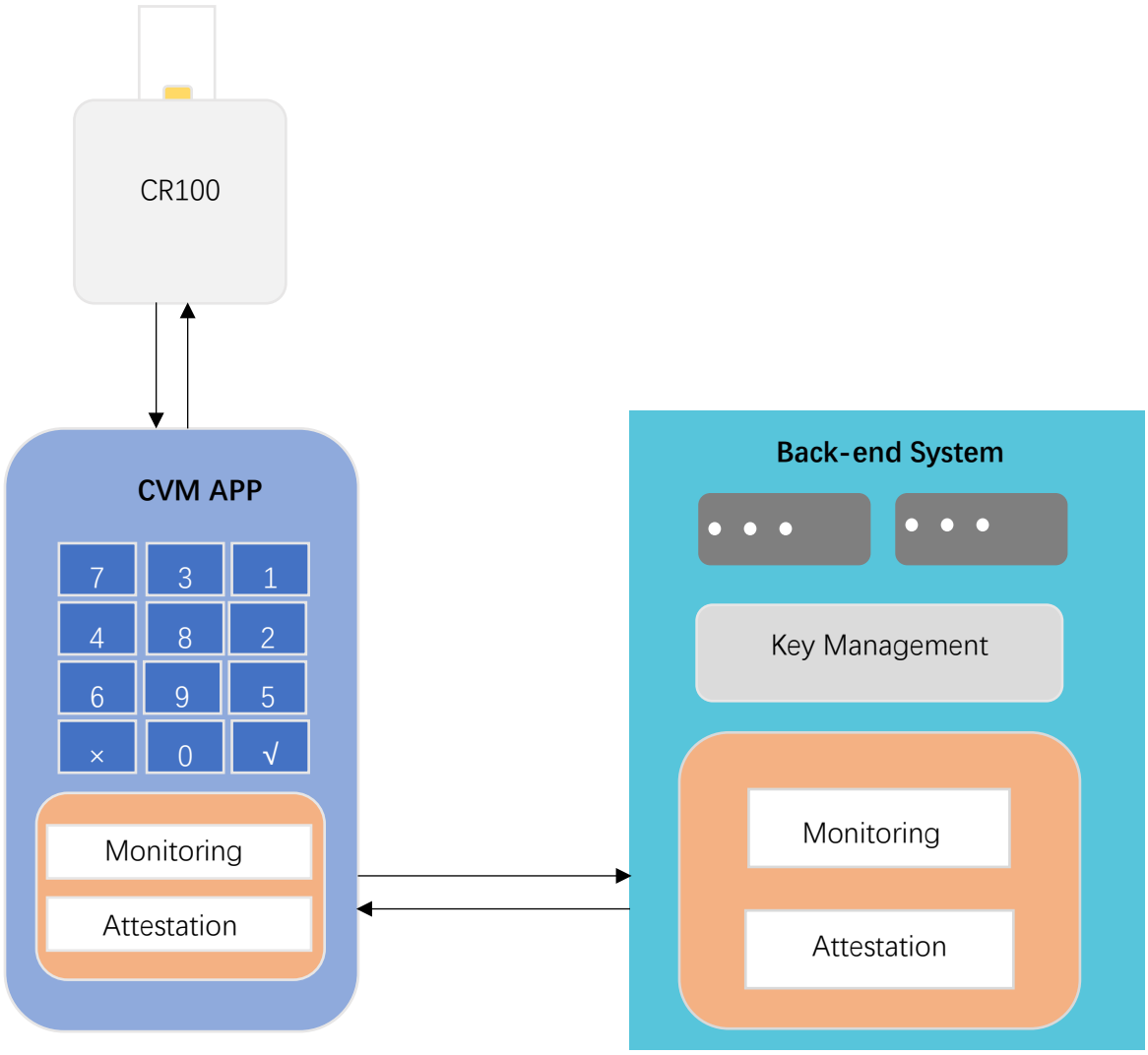


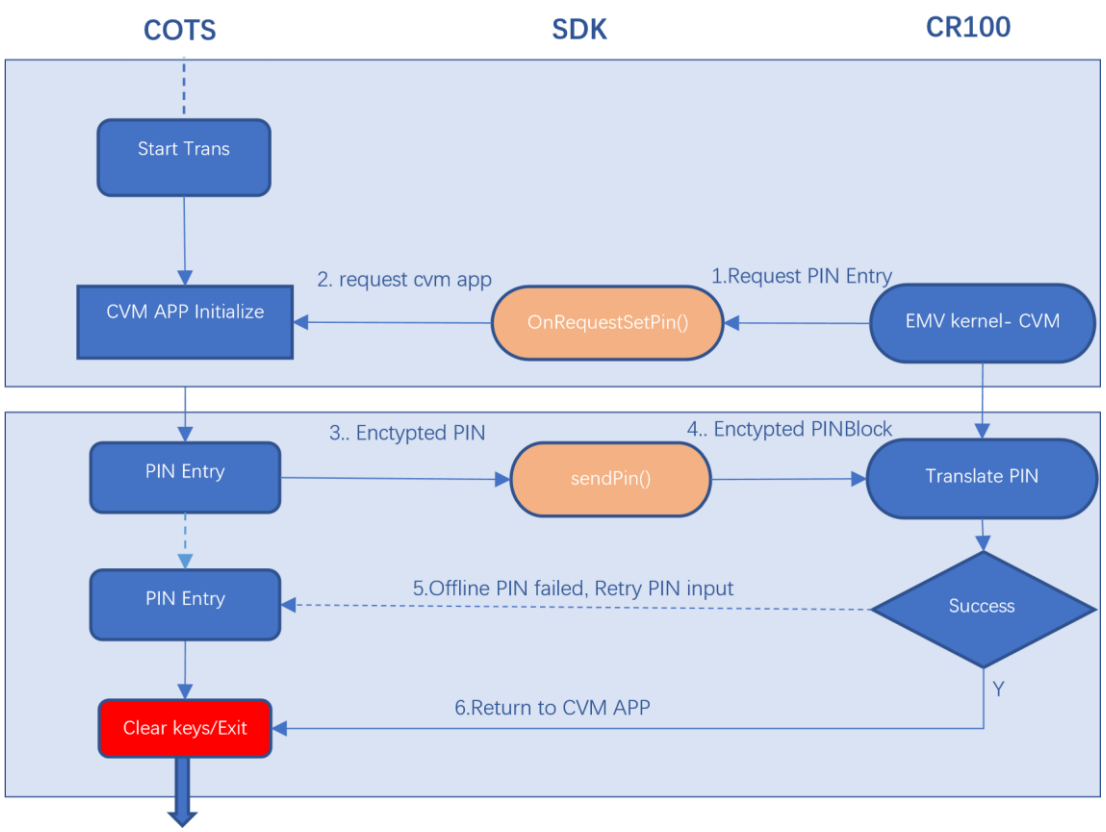
Figure 1 Software-based PIN Entry Solution

2.1 DSPREAD PIN CVM APP API

DSPREAD PIN CVM APP API enforce the security between COTS and CR100. Below is the comparison implementation PIN entry for online pin and offline pin scenario. Both will call below API

- onRequestSetPin()
- sendPin(pinBlock)

After cardholder present the card to CR100, the EMV kernel will determine if PIN entry needed. Then CR100 will initial PIN entry request to CVM APP by calling SDK delegate: onRequestSetPin(). Mobile CVM APP need draw PIN entry layout in this callback. Random Scattered digits layout keyboard with "****" mask can enforce PIN input security.



2.2 Pin On Mobile Steps

1. PIN CVM Application and SCRP are initialized with their financial keys (this may be asynchronous with the transaction).

2. A secure communication channel between the PIN CVM Application and the back-end monitoring system is established.

3. The back-end monitoring system determines the security status of the mobile payment-acceptance platform (SCRP, COTS platform and PIN CVM Application) using the attestation component.

4. An EMV card, contact or contactless, or an NFC-enabled mobile EMV payment device is presented to the CR100.

5. CR100 EMV kernel determine if PIN required for current transaction by check card CVM list. And send PIN Entry request to CVM APP

6. The PIN CVM APP PIN entry component renders a PIN entry screen on the COTS platform and the cardholder enters their PIN using the rendered PIN pad from the PIN CVM Application. The resulting information is enciphered and sent to the SCRP by the PIN CVM Application.

7. The SRED component of the SCRP enciphers the account data using preloaded data-encryption keys according to either **Case 1** (online PIN verification) or **Case 2** (offline PIN verification) above. Offline will validated by ICC card. If PIN incorrect, CVM app will request to input PIN again. You can also retrieve remaining pin try limit by calling getCVMPinTryLimit().

8. The payment transaction is processed.

3. MPoC Solution Details

3.1 CR100 related keys

This section details the keys used in the solution, as well as the purpose, algorithm, length, generation, exchange, storage, and destruction of the keys. Each key complies with the MPoC security specification. For details, please see the table "CR100 related keys" as below.

3.2 Secure Channels

MPoC SDK and CR100 use Bluetooth for communication. Before the Bluetooth connection is successful, RSA 3072 will be used for mutual authentication and ECDH 256 will be used to exchange and generate AES 128 session key and session mac key. After the Bluetooth connection is successful, the session key and session mac key will be used to establish a secure channel to protect the communication between MPoC SDK and CR100. For details, please see the flowchart "Secure Channel Flow Chart" as below.

3.3 Secure PIN entry

The entire transaction, including the input of PIN, is conducted in a secure channel. Before the PIN is input, a random key is exchanged with the protection key. Then CR100 uses the random key to derive a 16-byte aes encryption key and a 16-byte data encryption key. Then the aes encryption key is used to encrypt the AES key, and the data encryption key is used to encrypt the dummy pan, random number and other data. Then the ciphertext is sent to the SDK. The SDK decrypts and sends the decrypted AES key, dummy pan and random number to the app. The app requests the user to enter the PIN, and uses the AES key, dummy pan and random number to get the pinblock through the ISO-4 algorithm and sends it to CR100. CR100 uses the AES key, dummy pan and random number to decrypt the plaintext PIN, and then uses ISO-0 to encrypt the plaintext PIN to get the pinblock, and then returns the pinblock and transaction data to the app. Please see the flowchart "PIN Input Flow Chart" as below.

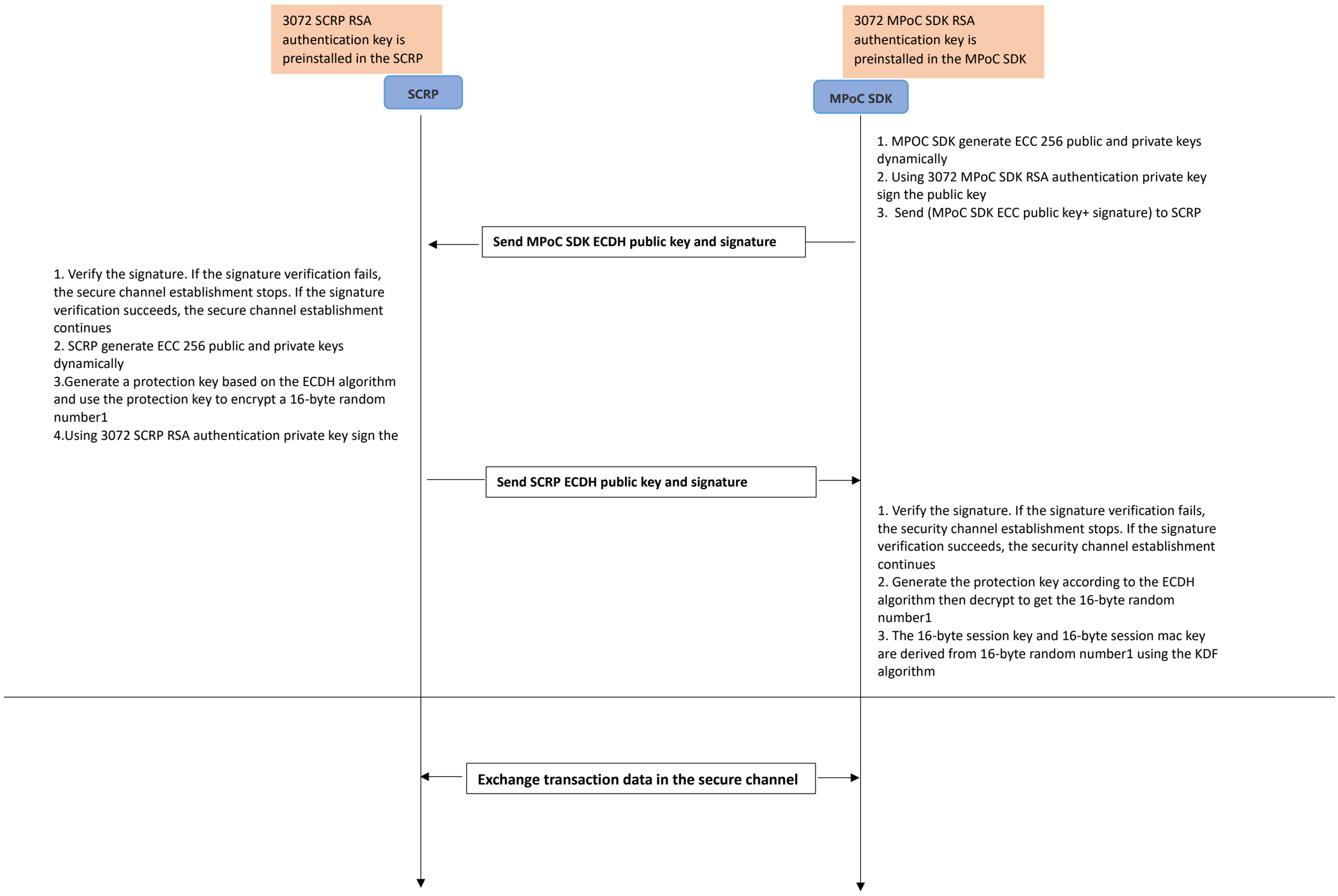
CR100 related keys

Key Name	Purpose	Algorithm / Size	Unique-per-	Component	Generation / Loading & Protection	Lifetime	Usage Location	Protection in Storage	Erasure
MPoC SDK ECC key	Derive Protection Key	ECDH 256	Operation	SDK	1.MPoC SDK dynamically generates ECC public and private keys 2.Use the MPoC SDK key authentication key to sign the MPoC SDK ECC public key 3.Send MPoC SDK ECC public key and signature to SCRP	Operation	SCRP	Not stored	End of operation
SCRP ECC key	Deriver Protection Key	ECDH 256	Operation	SCRP	1.SCRP dynamically generates ECC public and private keys 2.Use the SCRP key authentication key to sign the SCRP ECC public key 3.Send SCRP ECC public key and signature to MPoC SDK	Operation	SDK	Not stored	End of operation
Protection Key	Protect 16 bytes random number 1 and random 2	AES 128	Transaction	SDK	Derived from MPoC SDK ECC key and SCRP ECC key	Transaction	SCRP MPoC SDK	Ephemeral key managed by the MPoC SDK and SCRP	End of transaction
				SCRP	Derived from MPoC SDK ECC key and SCRP ECC key		See PTS POI report	See PTS POI report	See PTS POI report
AES Key	Encrypt PIN to send to the CR100	AES 128	Transaction	SDK	16-byte random number 2 derives 16-byte aes encryption key and 16-byte data encryption key, AES Key is encrypted by derived aes Key	Transaction	SDK process memory	Not stored	End of operation
				SCRP	See PTS POI report		See PTS POI report	See PTS POI report	See PTS POI report
IPEK	DUKPT IPEK to protect the plain PIN sent to the backend	TDES 128	SCRP device	SCRP	See PTS POI report	See PTS POI report	See PTS POI report	See PTS POI report	See PTS POI report
Session key	Protect messages on the SCRP&SDK secure channel	AES 128	Bluetooth Connect	SDK	1.Random number 1 derives session key and session mac key by X9.63KDF algorithm 2.Protected by Protection Key	Bluetooth Connect	SCRP MPoC SDK	Ephemeral key managed by the MPoC SDK and SCRP	Destruction when app and SCRP disconnect
				SCRP	See PTS POI report		See PTS POI report	See PTS POI report	See PTS POI report

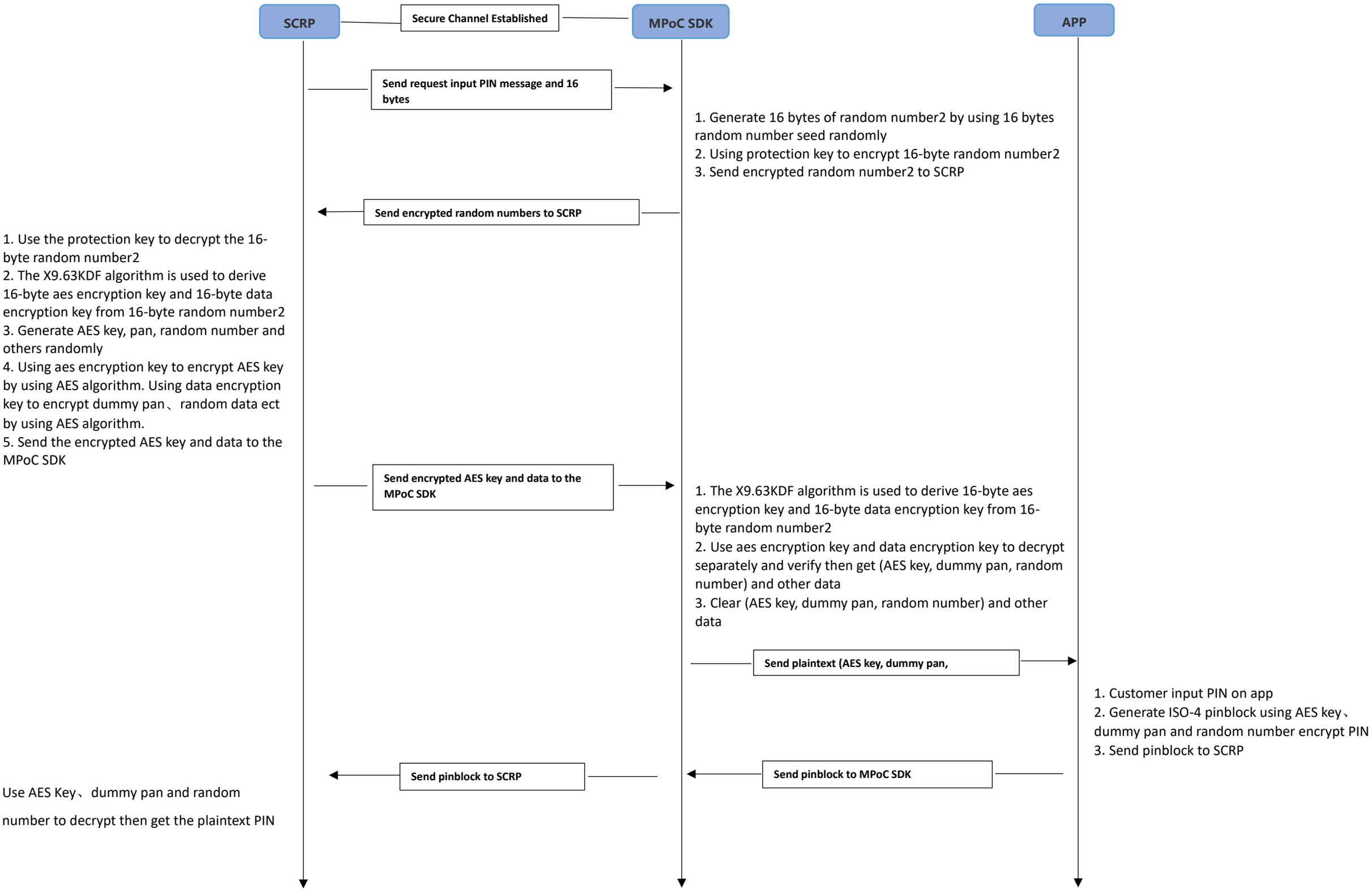
Session mac key	Verify messages on the SCRP&SDK secure channel	CMAC AES 128	Bluetooth Connect	SDK	1.Random number 1 derives 16 bytes session key and 16 bytes session mac key by X9.63KDF algorithm 2.Protected by Protection Key	Bluetooth Connect	SCRP MPoC SDK	Ephemeral key managed by the MPoC SDK and SCRP	Destruction when app and SCRP disconnect
				SCRP	See PTS POI report		See PTS POI report	See PTS POI report	See PTS POI report
SCRP key authentication key	Authentication of SCRP DH key	RSA 3072 SHA 256	Solution	SCRP	Hardcoded in the MPoC SDK（public key）	Valid until update it	MPoC SDK	Hardcoded in the MPoC SDK	Valid until update it
					Loaded during manufacture（private key）		SCRP	SCRP	
MPoC SDK key authentication key	Authentication of MPoC SDK DH key	RSA 3072 SHA 256	Solution	MPoC SDK	Hardcoded in the MPoC SDK（private key）	Valid until update it	MPoC SDK	Hardcoded in the MPoC SDK	Valid until update it
					Loaded during manufacture（public key）		SCRP	SCRP	
Device Key	encrypt enablement token to send and active the device	RSA2048	SCRP device	SCRP	Loaded during manufacture	Valid until updating it	SCRP	Stored in SCRP (Public key)	Valid until update it
				Server	Loaded by backend person		Server	Stored in Server (Private key)	
recieverRSA	sign enablement token to send and active the device	RSA2048	Solution	SCRP	Loaded during manufacture	Valid until updating it	SCRP	Stored in SCRP (Public key)	Valid until update it
				Server	Loaded by backend person		Server	Stored in Server (Private key)	
aes encryption key	Protect the AES Key transmitted by SCRP to the SDK	AES 128	Transaction	SDK	Derived from random number 2 using the X9.63KDF algorithm	Transaction	MPoC SDK	Ephemeral key managed by the MPoC SDK	Destruction when app get “AES Key, dummy Pan,

									random number" etc
data encryption key	Protect the "dummy Pan, random number" etc transmitted by SCRP to the SDK	AES 128	Transaction	SDK	Derived from random number 2 using the X9.63KDF algorithm	Transaction	MPoC SDK	Ephemeral key managed by the MPoC SDK	Destruction when app get "AES Key, dummy Pan, random number" etc

Secure Channel Flow Chart



PIN Input Flow Chart



3.4 SCRP Enablement Token API

Token example

0F2000000002000000000112A4537790020123

0F	20000000020	0000000001	12A4537790	02	0123
Length of Token Type: Fixed	Last 10 digits of SN	Token counter	Random	Length of token interval	Interval of token seconds

Generate digital envelope from token

The digital envelope will require mutual authentication between terminal and server host, need exchange server public key and terminal public key during terminal manufacture. For development terminal, it will load with test server public key, and the test server private key can be found in demo project keys folder

Len(4) Bytes	Public RSA Encryption(256B)	3DES Encrypt	Private RSA Signature(256B)
1	2	3	4

Digital Envelope Demo Source Code

```
RSA senderRsa = new RSA();
RSA receiverRsa = new RSA();
senderRsa.loadPrivateKey(new FileInputStream("/usr/local/envelope/rsa_private_pkcs8_2048.pem"));
 String devicePosPublicKey=
"BEB7AFB87AF92A20ED69071C144DAD99EAAFAE183E6A01A6693B300C805F7E6BA4196EFF18C8FCB87CF65280B49E128D189AC27A44BC6D39261E124393C6DADD212FA6
5CA53D8A3AABD936F5C27BC053257D2497080D8D17139C6170662DF617BC01B671571CD0B60E249EDA6B4201A01CDA5AE7EAEBEF14F9B78DE2E1E210DC3F90222778EB6
611A269041C4B5F880B561AEE4F5B81B11A2448869CD17853E7B009E1508B5D898330A8701F650E00C618998E30CCB1FAE9203051F4C34B82997AAE3A47DE99471AE928
95DEF86AC63B6FD323362D122B239A89BD0473A1823103081D682B56F3B5C62E58F0E0E1C0F627108E35BCAA2FC8E103205D0F25159";
 String e = "010001";
 receiverRsa.loadPublicKey(devicePosPublicKey, e);
 byte[] de = envelope.envelope(message2, senderRsa, receiverRsa, 2048);
```

Device Public Key: device public key exported from terminal in factory and save in client server database

receiverRSA: client server private key generated by client. Client need send its public key to vender and injected into terminal while manufacture

Digital Envelope Process:

1.RSA encrypt the 3DES KEY, then we get section **(2) [Public RSA Encryption]** of the digital envelope

2.use the 3DES key encrypt the plain text, then get section **(3) [Encrypted Message]** of the digital envelope

3.using private key signature (2) and (3). then use sending private RAS key encrypt the signature result. then get **(4) [Private RSA Key encryption]** of the digital envelope

4.finally, calculate the total length of digital envelope, (1) the resulting digital envelope: len(4B) Public RAS Encryption(256B) 3DES Encrypt Private RAS Signature(256B)

the resulting digital envelope:

000200812F0B2F83BF59583A2B93FC30236AF7E190D72740D2F0D36DCB34A430F9DE66D7E658BE1C9AE1BA2D1A666EE346A7ED778ED0F1870D29D1CFF042521E1103A957A717D6D917DF3DDEE8AC6817EF69C7B4EABEF49D580F2A3F7B18256378F89D4E3F72C99E42EC3DBBE7B5880B8EC749CE200DBABFC07692A1D6BA140D10959A9072E75CDE634395BEE8D6D1C1AD767DED447849DCB42975522C4651867CF3A31E72ED8EDA108E7F7C2DD36F6D4161915E208C9D181471811B2AAA2C64F7

**ECAC638780D61295BCDCE27C89DBE6494685CED20C20D7BEC47D621A897B55ED0C71096B56F5
64B9E82A6E93D01F8BBE6C0F5D19E69E457C7C742004D2C01BCD5D4D0226EF886878B2B453CDE
DAC9B119EDB60111038A26786494C027A1B0B90391B0CD33B283CEE70493F5ACB473C4D09C4DE
D7BB5CDCA8148F0E853605C9067293D82102381DDEA12FF95E340443130B21FA98D13D99A75CC6
75D3AB660570854FD2E6E47E579FF784CA9F3E6B5EA0636EBD90606DCA51CD980457864C5F40B6
7D88161F0AD27163492C8FC0FEE7576CDCEB15A196032206CDDBC4F932B0569E95DF2B5209B4F
D3D58FA50BC50D5081D4E42FC4292BD91E820B2BFAABC699F6094D4F21545D464B055274F9637A
A9564363912BF9185DC91EDDAA1D43235864955B48548333C5601C592044B5CD03A80C11A947B243
A45EB4550C6BCB646752B8313FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFF**

Enablement Token API

pos.updateWorkKey(String envelope)

SCRP verification

1. Verify serial number | counter | random data

SN：The SCRP will verify whether the serial number and the serial number of the device are the same. If they are the same, the serial number will be considered valid.

Counter：The SCRP will verify whether the counter is larger than the last counter received. If it is larger, the counter number will be considered valid.

Random data：The random number will be verified to see if it is different from the last time. If it is different, the random number will be considered valid.

2. Interval of token

The token is valid within this time interval. The device can do the transaction. After this time, the token will become invalid and the terminal cannot be used.

4. MPoC Android SDK Source Code

4.1 Key Exchange for Mutual Authentication

```
private boolean buildECDHSafetyMode() throws NoSuchAlgorithmException, InvalidAlgorithmParameterExcepti
on, InvalidKeySpecException, InvalidKeyException, InvalidParameterSpecException {
    String ECGenParameter = "secp521r1";
    String reservedStr = "00000000";
    String ECDHtype = "03";
    if(ECDHtype.equals("01")){
       ECGenParameter = "secp256r1";
    } else if (ECDHtype.equals("03")) {
       ECGenParameter = "secp521r1";
    }
    ECGenParameterSpec parameterSpec = new ECGenParameterSpec(ECGenParameter);
    KeyPairGenerator keyGen = KeyPairGenerator.getInstance("EC");
    keyGen.initialize(parameterSpec);
    KeyPair sdkKeyPair = keyGen.generateKeyPair();
    PublicKey sdkPubKey =  sdkKeyPair.getPublic();
    PrivateKey sdkPriKey = sdkKeyPair.getPrivate();
    byte[] sdkPubKeyBytes = sdkPubKey.getEncoded();
    Tip.d("sdkPubKey: " + Util.byteArray2Hex(sdkPubKeyBytes));
    parseASN1(Util.byteArray2Hex(sdkPubKeyBytes));
    String sdkPubKeyBitchar = updateServerECCPublicKeyParam.substring(2);
    String sdkPubKeyBitcharLength = Util.intToHexStr(sdkPubKeyBitchar.length()/2);
    while (sdkPubKeyBitcharLength.length()<4){
       sdkPubKeyBitcharLength = "0"+sdkPubKeyBitcharLength;
    }
    RSAUtil rsaUtil = new RSAUtil();
    Tip.d("sign origin:\n"+reservedStr+ECDHtype+sdkPubKeyBitcharLength+sdkPubKeyBitchar);
    byte[] signedSDKPubKeyBitchar = rsaUtil.sign(Util.HexStringToByteArray(reservedStr+ECDHtype+sdkPub
KeyBitcharLength+sdkPubKeyBitchar));
    String signedSDKPubKeyBitcharLength = Util.intToHexStr(signedSDKPubKeyBitchar.length);
    while (signedSDKPubKeyBitcharLength.length()<4){
```

```
            signedSDKPubKeyBitcharLength = "0"+signedSDKPubKeyBitcharLength;
        }
        Tip.d("char len: sign len"+sdkPubKeyBitcharLength+" "+signedSDKPubKeyBitcharLength);
        CommandDownlink dc = null;
        CommandUplink uc = null;
        String paras = "04"+reservedStr+ECDHtype+sdkPubKeyBitcharLength+sdkPubKeyBitchar+signedSDKPubKeyBitcharLength+Util.byteArray2Hex(signedSDKPubKeyBitchar);
        Tip.i("paras "+paras);
        dc = new CommandDownlink(0x41, 0x1E, 10,Util.HexStringToByteArray(paras));
        pos.sendCommand(dc);
        uc = pos.receiveCommandWaitResult(timeout);
        boolean f = checkCmdId(uc);
        if (!f) {
            return false;
        }
        if(uc.result() == 0x0C) return false;
        String fixedECDHKeyPrefix = "30819b301006072a8648ce3d020106052b810400230381860004";
        if(ECDHtype.equals("01")){
            fixedECDHKeyPrefix = "3059301306072A8648CE3D020106082A8648CE3D03010703420004";
        } else if(ECDHtype.equals("03")) {
            fixedECDHKeyPrefix = "30819b301006072a8648ce3d020106052b810400230381860004";
        }
        if(uc.result() == 0){
            String result = Util.byteArray2Hex(uc.getBytes(0,uc.length()));
            int index = 0;
            int deviceECDHPubKeyLen = Util.byteArrayToInt(uc.getBytes(index,2));
            index += 2;
            String deviceECDHPubKey = Util.byteArray2Hex(uc.getBytes(index,deviceECDHPubKeyLen));
            Tip.d("pubkey from device:"+deviceECDHPubKeyLen+" "+deviceECDHPubKey);
            index += deviceECDHPubKeyLen;
            int encryptedRandomNumber1Len = uc.getByte(index);
            index++;
```

```
        String encryptedRandomNumber1= Util.byteArray2Hex(uc.getBytes(index,encryptedRandomNumber1Len)
);

        Tip.d("encryptedRandomNumber1Len: "+encryptedRandomNumber1Len +"encryptedRandomNumber1: "
+encryptedRandomNumber1);

        index += encryptedRandomNumber1Len;

        int deviceECDHPubKeySignatureLen = Util.byteArrayToInt(uc.getBytes(index,2));

        index += 2;

        String deviceECDHPubKeySignature = Util.byteArray2Hex(uc.getBytes(index,deviceECDHPubKeySignat
ureLen));

        Tip.d("pubkeysignature from device:"+deviceECDHPubKeySignatureLen+" "+deviceECDHPubKeySignat
ure);

        index += deviceECDHPubKeySignatureLen;

        int signatureConetentLen = 4+deviceECDHPubKeyLen*2+2+encryptedRandomNumber1Len*2;

        Tip.d("signatureConetent:"+result.substring(0,signatureConetentLen));

        boolean signatureCheck = rsaUtil.check(Util.HexStringToByteArray(result.substring(0,signatureConetentLe
n)),Util.HexStringToByteArray(deviceECDHPubKeySignature));

        if(!signatureCheck) return false;

        //get ECDH key

        Security.addProvider(new BouncyCastleProvider());

        String deviceECDHKey = fixedECDHKeyPrefix+deviceECDHPubKey;

        Tip.d("deviceECDHKey:"+deviceECDHKey);

        KeyFactory deviceKeyFactory = KeyFactory.getInstance("EC");

        X509EncodedKeySpec deviceKeySpec = new X509EncodedKeySpec(Util.HexStringToByteArray(deviceE
CDHKey));

        PublicKey devicePublicKey = deviceKeyFactory.generatePublic(deviceKeySpec);

        //exchange

        KeyAgreement keyAgreement = KeyAgreement.getInstance("ECDH");

        keyAgreement.init(sdkKeyPair.getPrivate());

        keyAgreement.doPhase(devicePublicKey,true);

        protectKey = keyAgreement.generateSecret();

        Tip.d("sharedkey:"+Util.byteArray2Hex(protectKey));

        byte[] plaintextRandomNumber1Bytes = AESUtil.decryptCBC(protectKey,Util.hexStringToByteArray(encr
yptedRandomNumber1));

        Tip.d("plaintextRandomNumber1Bytes: "+Util.byteArray2Hex(plaintextRandomNumber1Bytes));
```



```
            byte[] derivedRandomKey = X963KDFUtil.deriveKey(plaintextRandomNumber1Bytes,32);
            usbSessionKey = Arrays.copyOfRange(derivedRandomKey,0,16);
            Tip.d("usbSessionKey:"+Util.byteArray2Hex(usbSessionKey));
            usbSessionMACKey = Arrays.copyOfRange(derivedRandomKey,16,32);
            disConnect("doSafetyMode");
            Arrays.fill(sessionKeyBytes, (byte)0);
            Arrays.fill(derivedSessionKey , (byte)0);
            return true;
        }
        return false;
    }
```

4.2 Clearing of AES Key

```
if (uc.result() == Constants.RESP_REQUIRE_APP_ENTER_ONLINE_PIN) {//CR100 cvm online PIN
    esc.setCvmPin(true);
    esc.setOnlinePin(true);
    int index = 0;
    int randomSeedLen = Util.byteArrayToInt(uc.getBytes(index++, 1));
    byte[] randomSeed = uc.getBytes(index, randomSeedLen);
    index += randomSeedLen;
    byte[] randomR2 = SP800SecureRandomUtil.getSecureRandomByte(randomSeed, 16);
    String encryptResult = AESUtil.encrypt(esc.protectKey,randomR2);
    byte[] derivedKey = X963KDFUtil.deriveKey(randomR2,32);
    byte[] aesEncryptionKey = Arrays.copyOfRange(derivedKey,0,16);
    byte[] dataEncryptionKey = Arrays.copyOfRange(derivedKey,16,32);
    Tip.i("aesEncryptionKey == " + Util.byteArray2Hex(aesEncryptionKey));
    Tip.i("dataEncryptionKey == " + Util.byteArray2Hex(dataEncryptionKey));
    uc = doCvmPin.EMVChangePin(pos,encryptResult);
    if(uc != null){
    if(uc.result() == 0){
    String result = Util.byteArray2Hex(uc.getBytes(0,uc.length()));
    Tip.i("resut  == "+result);
```

```
    Hashtable<String,StringBuffer> encryptData = new Hashtable<>();
    int ciphterTxtIndex = 0;
    int aesKeyCiphterLen = Integer.valueOf(result.substring(ciphterTxtIndex, ciphterTxtIndex + 2), 16) * 2;
    ciphterTxtIndex += 2;
    StringBuffer aesKeyCiphterBuffer = new StringBuffer(result.substring(ciphterTxtIndex, ciphterTxtIndex + aesKeyLen));
    ciphterTxtIndex += aesKeyLen;
    StringBuffer aesKeyBuffer = new StringBuffer(Util.byteArray2Hex(AESUtil.decrypt(aesEncryptionKey, new String(aesKeyCiphterBuffer))));//decrypt
    int dataCiphterLen = Integer.valueOf(result.substring(ciphterTxtIndex, 2), 16) * 2;
    Tip.e("dataCiphterLen"+ dataCiphterLen );
    StringBuffer dataCiphterBuffer = new StringBuffer(result.substring(2, 2 + dataCiphterLen));
    Tip.e("dataCiphterBuffer:"+ dataCiphterBuffer);
    ciphterTxtIndex = dataCiphterLen + 2;
    String plainData = Util.byteArray2Hex(AESUtil.decrypt(dataEncryptionKey, new String(dataCiphterBuffer)));//decrypt
    ciphterTxtIndex = 0;
    Tip.e("randomDataLen start");
    int randomDataLen = Integer.valueOf(plainData.substring(ciphterTxtIndex, 2), 16) * 2;
    Tip.e("randomDataLen "+randomDataLen);
    StringBuffer randomDataBuffer = new StringBuffer(plainData.substring(2, 2 + randomDataLen));
    Tip.e("randomdata:"+randomDataBuffer);
    ciphterTxtIndex = randomDataLen + 2;
    int panLen = Integer.valueOf(plainData.substring(ciphterTxtIndex,ciphterTxtIndex + 2),16) *2;
    ciphterTxtIndex +=2;
    StringBuffer panBuffer = new StringBuffer(Util.convertHexToString(result.substring(ciphterTxtIndex, ciphterTxtIndex + panLen)));
    ciphterTxtIndex += panLen;
    // 随机表生成加密 pin
    if(ciphterTxtIndex < result.length()) {
        int keyListLen = Integer.valueOf(result.substring(ciphterTxtIndex, ciphterTxtIndex + 2), 16) * 2;
        ciphterTxtIndex += 2;
        String keyList = plainData.substring(ciphterTxtIndex, ciphterTxtIndex + keyListLen);
```

```
        esc.setCvmKeyList(keyList);
    }
    Tip.i("data ="+randomDataBuffer);
    encryptData.put("RandomData", randomDataBuffer);
    encryptData.put("AESKey", aesKeyBuffer);
    encryptData.put("PAN", panBuffer);
    encryptData.put("isOnlinePin", new StringBuffer(String.valueOf(true)));
    encryptData.put("ResetTimes", new StringBuffer(""));
    encryptData.put("pinTryLimit", new StringBuffer(String.valueOf(pinTyrLimit)));
    doCvmPin.setChangePin(false);
    esc.setEncryptData(encryptData);
    esc.onRequestSetPin();
    aesKeyCiphterBuffer.delete(0, aesKeyCiphterBuffer.length())
    aesKeyBuffer.delete(0,aesKeyBuffer.length());
    dataCiphterBuffer.delete(0, dataCiphterBuffer.length());
    randomDataBuffer.delete(0,randomDataBuffer.length());
    panBuffer.delete(0,panBuffer.length());
    Arrays.fill(randomR1, (byte)0);
    Arrays.fill(randomR2, (byte)0);
    Arrays.fill(protectKey, (byte)0);
    Arrays.fill(derivedKey, (byte)0);
    Arrays.fill(aesEncryptionKey, (byte)0);
    Arrays.fill(dataEncryptionKey, (byte)0);
}
```

Clean up the Hashtable type global variables that store AES Key

```
public Hashtable<String, String> getEncryptData() {
    if (encryptData != null) {
      Hashtable<String, String> tempEncryptData = encryptData;
      encryptData.clear();
      return tempEncryptData;
         }
    return null;
}
```


4.3 Usage of Session Key

Using session key to decrypt the data from SCRP.

```
byte[] b = read();
Tip.d(" isEncryptSecureData "+isEncryptSecureData);
try{
   if(isEncryptSecureData){
      String result = Util.byteArray2Hex(b);
      int len = Integer.valueOf(result.substring(2,6),16)*2;
      String data = result.substring(8,8+len-2);
      data = AESUtil.decryptCBC(Util.byteArray2Hex(QPOSService.getUsbSessionKey()),data);
      int dataLen = Integer.valueOf(data.substring(0,4),16)*2;
      String plainData = data.substring(44);
      data = "4D"+data.substring(0,4)+plainData;
      Tip.w("result = "+data);
      byte crcbyte = Util.HexStringToByteArray(data)[0];
      for (int i = 1; i < Util.HexStringToByteArray(data).length ; i++) {
         crcbyte = (byte) (crcbyte ^ Util.HexStringToByteArray(data)[i]);
      }
      data = data + Util.byteArray2Hex(new byte[]{crcbyte});
      b = Util.HexStringToByteArray(data);
   }
} catch (Exception e) {
   b = new byte[0];
      e.printStackTrace();
}
```

4.4 Usage of receiverRSA

```
public static byte[] byteEvelope(byte[] message, RSA deviceRsa, RSA receiverRsa, int RSA_len) throws Exception
{
      TRACE.d("message:"  + QPOSUtil.byteArray2Hex(packageMessage(message)));
      byte[] encrypedMessage = deviceRsa.encrypt(packageMessage(message));
      byte[] toSha1Message = new byte[encrypedMessage.length];
      System.arraycopy(encrypedMessage, 0, toSha1Message, 0, encrypedMessage.length);
```



```
        byte[] signedMessage = receiverRsa.sign(toSha1Message);
        byte[] results = new byte[4 +  encrypedMessage.length + signedMessage.length];
        int len = encrypedMessage.length + signedMessage.length;
        byte[] lenBytes = Utils.int2Byte(len);
        if (RSA_len == 2048) {
            lenBytes[3] = (byte) 0x81;
        }
        TRACE.d( "encrypedMessage:" + encrypedMessage.length + "\n" + "signedMessage:" + signedMessage.length);
        System.arraycopy(lenBytes, 0, results, 0, lenBytes.length);
        System.arraycopy(encrypedMessage, 0, results, lenBytes.length , encrypedMessage.length);
        System.arraycopy(signedMessage, 0, results, lenBytes.length  + encrypedMessage.length, signedMessage.length);
        return results;
    }
```

4.5 Build ISO-4 Format Pinblock

```
StringBuffer randomData = value.get("RandomData") == null ? new StringBuffer() : new StringBuffer(value.get("RandomData"));
StringBuffer pan = value.get("PAN") == null ? new StringBuffer() : new StringBuffer(value.get("PAN"));
StringBuffer AESKey = value.get("AESKey") == null ? new StringBuffer(""): new StringBuffer(value.get("AESKey"));
StringBuffer isOnline = value.get("isOnlinePin") == null ? new StringBuffer() : new StringBuffer(value.get("isOnlinePin"));
StringBuffer pinTryLimit = value.get("pinTryLimit") == null ? new StringBuffer() : new StringBuffer(value.get("pinTryLimit"));
Tip.d("randombuffer:"+randomData.toString()+"\n 123"+value.get("RandomData"));
//iso-format4 pinblock
int pinLen = pin.length;
StringBuffer newPin = new StringBuffer();
for (byte p : pin) {
   newPin.append(Integer.toHexString(p).toUpperCase());
}
newPin.insert(0,"4" + Integer.toHexString(pinLen));
Tip.d("newPin 2 ==" + newPin);
```

```
for (int i = 0; i < 14 - pinLen; i++) {
    newPin.append("A");
}
newPin.append(randomData.substring(0, 16));
StringBuffer panBlock;
int panLen = pan.length();
int m = 0;
if (panLen < 12) {
    panBlock = new StringBuffer("0");
    for (int i = 0; i < 12 - panLen; i++) {
        panBlock.append("0");
    }
    panBlock.append(pan).append("0000000000000000000");
} else {
    m = pan.length() - 12;
    panBlock = new StringBuffer("" + m + pan);
    for (int i = 0; i < 31 - panLen; i++) {
        panBlock.append("0");
    }
}
byte[] derivedKey = X963KDFUtil.deriveKey(AESKey,32);
byte[] aesKeySource = Arrays.copyOfRange(derivedKey,0,16);
byte[] macKeySource = Arrays.copyOfRange(derivedKey,16,32);
int keyLen = AESKey.length()/2;
Tip.d("pinblock:aesKeySource:"+Util.byteArray2Hex(aesKeySource)+" "+ Util.byteArray2Hex(aesKeySource,keyLen));
Tip.d("pinblock:macKeySource:"+Util.byteArray2Hex(macKeySource)+" "+ Util.byteArray2Hex(macKeySource,keyLen));
this.pinblockMacKey = Arrays.copyOf(macKeySource, keyLen);
byte[] pinBlock2 = new byte[0];
try {
    byte[] pinBlock1 = AESUtil.encrypt(Arrays.copyOfRange(aesKeySource,0,keyLen), newPin);
```

```
    newPin = new StringBuffer(Util.xor16(pinBlock1, panBlock));
    pinBlock2 = AESUtil.encrypt(Arrays.copyOfRange(aesKeySource,0,keyLen), newPin);
} catch (Exception e){
    e.printStackTrace();
}
randomData.delete(0,randomData.length());
pan.delete(0,pan.length());
AESKey.delete(0,AESKey.length());
value.clear();
Arrays.fill(derivedKey, (byte)0);
Arrays.fill(aesKeySource, (byte)0);
Arrays.fill(macKeySource, (byte)0);
panBlock.delete(0,panBlock.length());
newPin.delete(0,newPin.length());
return pinBlock2;
```


5. MPoC iOS SDK Source Code

5.1 Build ISO-format4 PIN

```
//callback of input pin on phone
-(void) onRequestPinEntry{
    UIAlertController *alertController = [UIAlertController alertControllerWithTitle:NSLocalizedString(@"Please set pin", nil) message:@"" preferredStyle:UIAlertControllerStyleAlert];
    [alertController addAction:[UIAlertAction actionWithTitle:NSLocalizedString(@"Cancel", nil) style:UIAlertActionStyleCancel handler:^(UIAlertAction * _Nonnull action) {
        [pos cancelPinEntry];
    }]];
    [alertController addAction:[UIAlertAction actionWithTitle:NSLocalizedString(@"Confirm", nil) style:UIAlertActionStyleDefault handler:^(UIAlertAction * _Nonnull action) {
        UITextField *titleTextField = alertController.textFields.firstObject;
        NSString *pinStr = titleTextField.text;
        NSString *newPin = [self getNewPinByCvmKeyList:[pos getCvmKeyList] pin:pinStr];
        NSString *pinBlock = [self buildCvmPinBlock:newPin dict:[pos getEncryptDataDict]];
        [pos sendCvmPin:pinBlock isEncrypted:YES];
    }]];
    [alertController addTextFieldWithConfigurationHandler:nil];
    [self presentViewController:alertController animated:YES completion:nil];
}
- (NSString *)getNewPinByCvmKeyList:(NSString *)cvmKeyList pin:(NSString *)pin{
    NSString *newPin = @"";
    if(cvmKeyList != nil && cvmKeyList.length > 0 && pin != nil && pin.length > 0){
        cvmKeyList = [QPOSUtil asciiFormatString:[QPOSUtil HexStringToByteArray:cvmKeyList]];
        for (NSInteger i = 0; i < pin.length; i++) {
            NSRange range = [cvmKeyList rangeOfString:[pin substringWithRange:NSMakeRange(i, 1)]];
            newPin = [newPin stringByAppendingString:[QPOSUtil getHexByDecimal:range.location]];
        }
    }
    return newPin;
}
// use iso-4 format to encrypt pin
- (NSString *)buildCvmPinBlock:(NSString *)pin dict:(NSDictionary *)dict{
```

```
    NSString *random = [dict objectForKey:@"RandomData"];
    NSString *aesKey = [dict objectForKey:@"AESKey"];
    NSString *pan = [dict objectForKey:@"PAN"];
    NSString *pinStr = [NSString stringWithFormat:@"4%lu%@",pin.length,pin];
    NSInteger pinStrLen = 16 - pinStr.length;
    for (int i = 0; i < pinStrLen; i++) {
        pinStr = [pinStr stringByAppendingString:@"A"];
    }
    NSString *newRandom = [random substringToIndex:16];
    pinStr = [pinStr stringByAppendingString:newRandom];
    NSString *panStr = @"";
    if(pan.length < 12){
        panStr = @"0";
        for (int i = 0; i < 12 - pan.length; i++) {
            [panStr stringByAppendingString:@"0"];
        }
        panStr = [[panStr stringByAppendingString:pan] stringByAppendingString:@"0000000000000000000"];
    }else{
        panStr = [NSString stringWithFormat:@"%lu%@",pan.length - 12,pan];
        for (int i = 0; i < 32-pan.length-1; i++) {
            panStr = [panStr stringByAppendingString:@"0"];
        }
    }
    NSString *blockA = [self encryptOperation:kCCEncrypt value:pinStr key:aesKey];
    NSString *blockB = [self pinxCreator:panStr withPinv:blockA];
    NSString *pinblock = [self encryptOperation:kCCEncrypt value:blockB key:aesKey];
    return pinblock;
}
```

6. FAQs

6.1 FAQs of solution

1. How is the secure channel provided between the Android application and the CR100?

   Before the Bluetooth connection, CR100 and Android application authenticate each other through RSA 3072, generate a shared key through ECDH algorithm, exchange a key seed through the shared key and use the key seed to derive a 16-byte session key and 16-byte session mac key for encryption and verification of the secure channel.

2. How are data confidentiality, integrity and authenticity maintained between the Android application and the CR100?

   First, the android app and CR100 use secure Bluetooth as a secure channel. The transmission channel in the air is encrypted, and the transmitted data is also mutually encrypted by the app and CR100. After transmission, both parties perform data encryption, decryption, and signature verification through two-way authentication. This ensures the confidentiality, integrity and authenticity of data communications.

3. How does the DSPREAD SDK ensure that the SCRP is a real CR100 and not an emulator or tampered device?

   The sdk will active the SCRP devices every 5 minutes to check if the SCRP is the emulator or tampered, and you can check the page 9 - **SCRP Enablement Token API** to know about the active process.

4. Could you please ask Dspread what tool they use to perform the obfuscation?

   We use Proguard and StringFog tools to do the obfuscation.

5. I note the CR100 I'm testing with has hardware version CR100D and firmware version 589.87.239.13, which don't line up with those listed in the SP (HW 1.1.0, FW 1.1.3). Could you please explain?

   589.87.239.13 is the version number used for our internal debugging not the PCI version. Our PCI version is SP (HW 1.1.0, FW 1.1.3)

6. could you please provide evidence that only encrypted account data is ever sent from the SCRP to the MPoC SDK, and that the key used for decrypting the account data is never sent to the MPoC SDK?

   Our account data is always be encrypted and the key used for decrypting the account data is never sent to the MPoC SDK. You can refer this link of CR100 Security Policy (pcisecuritystandards.org).

   For encrypted account data is ever sent from the SCRP to the MPoC SDK , please refer below screenshot.

5.3 Account Data Protection

For the SRED module, account data can be encrypted by AES with 128/192 bits MK/SK, AES with 128/192/256 bits DUKPT, TDES 128 bits DUKPT and TDES 192bits MK/SK encryption or masked. The firmware of device doesn't support white listing for the pass-through of clear-text account data; also, it always provides SRED functionality, and does not support the disablement (turning off) of SRED functionality. For more details please refer to < Secure Software Development Guide >.

For the key used for decrypting the account data is never sent to the MPoC SDK, our device has passed PCI certification so it should conform to PCI security Policy.

7. The MPoC SDK key authentication key private key is hardcoded in the Dspread library, could you please explain how this key is protected?

   The private key file is putted in the res directory. When packaged, the private key file is hardcoded into the APK, which needs to be obfuscated and hardened to protect it.

8. Could you please provide the details (including DRBG algorithm, NIST SP800-90A adherence, what seeds/entropy are used, the length of each seed, and how seeds are combined) of the RNG used to generate this key? (Key A & B)

   Key A RNG:

   1. Obtain 32 bytes of hard random number from SCRP security chip as seed
   2. MPoC SDK uses random seeds, Java SecureRandom and BouncyCastle's SP800SecureRandom to generate secure random numbers through specific algorithms. For details, please see the following code screenshot.

```java
public static byte[] getSecureRandomByte(byte[] secureRandomSeed,int len) {  ouyangwanli *
    SP800SecureRandom sp800SecureRandom = null;
    SP800SecureRandomBuilder sp800SecureRandomBuilder = new SP800SecureRandomBuilder(k -> new EntropySource() {
        @Override  no usages  ouyangwanli
        public boolean isPredictionResistant() { return true; }

        @Override  no usages  ouyangwanli *
        public byte[] getEntropy() {
            SecureRandom secureRandom = new SecureRandom();
            byte[] entropy = new byte[(k + 7) / 8];
            byte[] seedFromChallenge = secureRandomSeed;
            byte[] seedFromDevice = secureRandom.generateSeed(entropy.length);
            byte[] userSeed = new byte[Math.max(seedFromDevice.length, seedFromChallenge.length)];
            // XOR the key parts to get our seed, repeating them if they lengths
            for (int i = 0; i < userSeed.length; i++) {
                userSeed[i] = (byte) (seedFromChallenge[i] ^ seedFromDevice[i]);
            }
            return userSeed;
        }
        @Override  no usages  ouyangwanli
        public int entropySize() { return k; }
    });
    sp800SecureRandom = sp800SecureRandomBuilder.buildHash(new SHA512Digest(),  nonce: null,  predictionResistant: false);
    return sp800SecureRandom.generateSeed(len);
}
```

Key B RNG:

There is a hard random data of two bytes is generated by SCRP secure chip directly. Details as below:

```
u32 lark_rand(pu8 pRand,u32 randLen )
{
    pDrv_Param_t pDrvParam;
    pAlg_Rand_t AlgRand;
    u32 len;

    pDrvParam = lark_alloc_mem(sizeof(Drv_Param_t));
    AlgRand = lark_alloc_mem(sizeof(Alg_Rand_t));
    pDrvParam->DrvID = DEV_ALG_RAND;
    AlgRand->Buf = pRand;
    AlgRand->Len = randLen;
    pDrvParam->pValue = AlgRand;
    LARK_API_(pDrvParam);
    len = AlgRand->Len;
    lark_free_mem(pDrvParam);
    lark_free_mem(AlgRand);
    return len;
}
```

6.2 FAQs of source code

1. Could you please give some more information on the X963KDFUtil.deriveKey() function? Which algorithms and hash functions are used?

   We used X9.63 KDF algorithm and SHA-256 hash algorithm.

2. Section "clearing of AES Key" calls getUsbSessionMACKey() and section "usage of Session Key"

calls getUsbSessionKey(), do these refer to Session Key and MAC Key or other keys that need to be added to the keytable?

getUsbSessionMACKey() is get session mac key in "loading and verification of Key B PK, generation of Protection Key, all usages of Protection Key, clearing of Protection Key" this part.

getUsbSessionKey() is get session key in "loading and verification of Key B PK, generation of Protection Key, all usages of Protection Key, clearing of Protection Key" this part.

3. Source snippets showing that the expiry of the enablement token is less than 24 hours, and SCRP will stop processing transactions once the expiry is reached

1. SCRP will judge valid time interval if be three minutes to eleven minutes in this range when app send enablement token to SCRP, if so, the device will be activated otherwise will be failed.

```
Length = pValue[cnt++];
if(Length >= 0x00 && Length <= CUSTOM_TOKEN_CHECK_TIME_LEN_MAX)
{
  Memset_(pTempBuffer, 0x00,CUSTOM_TOKEN_CHECK_TIME_LEN_MAX+1);
  Memcpy_(&pTempBuffer[CUSTOM_TOKEN_CHECK_TIME_LEN_MAX-Length], &pValue[cnt], Length);
  time = BE_PtrToShort(pTempBuffer);
  if(time >CHECK_TOKEN_MIN_TIME && time <CHECK_TOKEN_MAX_TIME )
  {
    rw_param(RW_WRITE, TOKEN_CHECK_TIME, pTempBuffer);
  }
  else
  {
      res = RES_CODE_DATA_FORMAT_ERROR;
      goto ↓jump_4010_envelope_end;
  }
}
else
{
  res = RES_CODE_DATA_FORMAT_ERROR;
  goto ↓jump_4010_envelope_end;
}
```

```
#if defined(CFG_CHECK_TOKEN)
#define CHECK_TOKEN_MIN_TIME 3*60
#define CHECK_TOKEN_MAX_TIME 11*60
#endif
```

2. When device activated, device will count based on token validity period and every valid second will decrease one until the value decrease to zero the token will be expired then SCRP will forbit to do transaction until to active again will be on.

```
#if defined(CFG_CHECK_TOKEN)
    if(app_process_check_token() == false)
    {
       command_make_and_send_noparam_packet(pmsg->wparam,CMD_APP_TOKEN_INVALID);
       return;
    }
#endif
```

```
bool app_process_check_token(void )
{
    pu8 pBuffer=NULL;
    bool ret_code = true;
    pBuffer = lark_alloc_mem(32);
    Memset_(pBuffer, 0x00, CUSTOM_TOKEN_LEN_MAX+1);
    rw_param(RW_READ, TOKEN_ID, pBuffer);
    if(Memcmp_(&pBuffer[CUSTOM_TOKEN_LEN_MAX-2], "\x00\x00", 2) == 0)
        ret_code = false;
    else
        ret_code = true;

_app_process_check_token_end:
#if defined(COMMAND_DBG)
        TRACE(DBG_TRACE_LVL,"ret_code = %d\r\n", ret_code);
#endif

    lark_free_mem(pBuffer);
    return ret_code;
} « end app_process_check_token »
```

