Dspread PIN CVM APP

Software-based PIN Entry on COTS

Version 1.0.9

July 2023

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 ExpressPay, Discovey 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.

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

CONTENTS

SPoC 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.

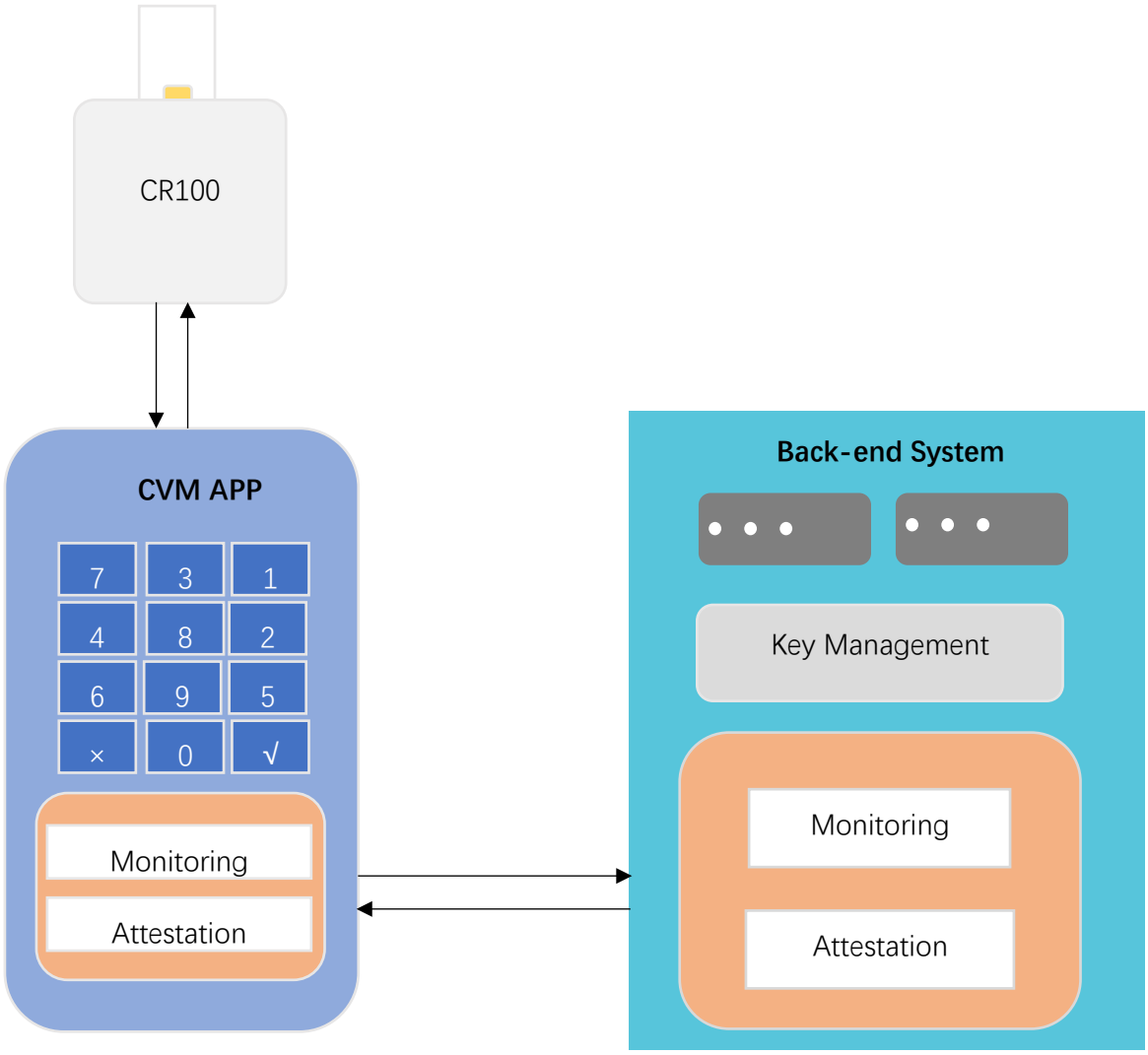


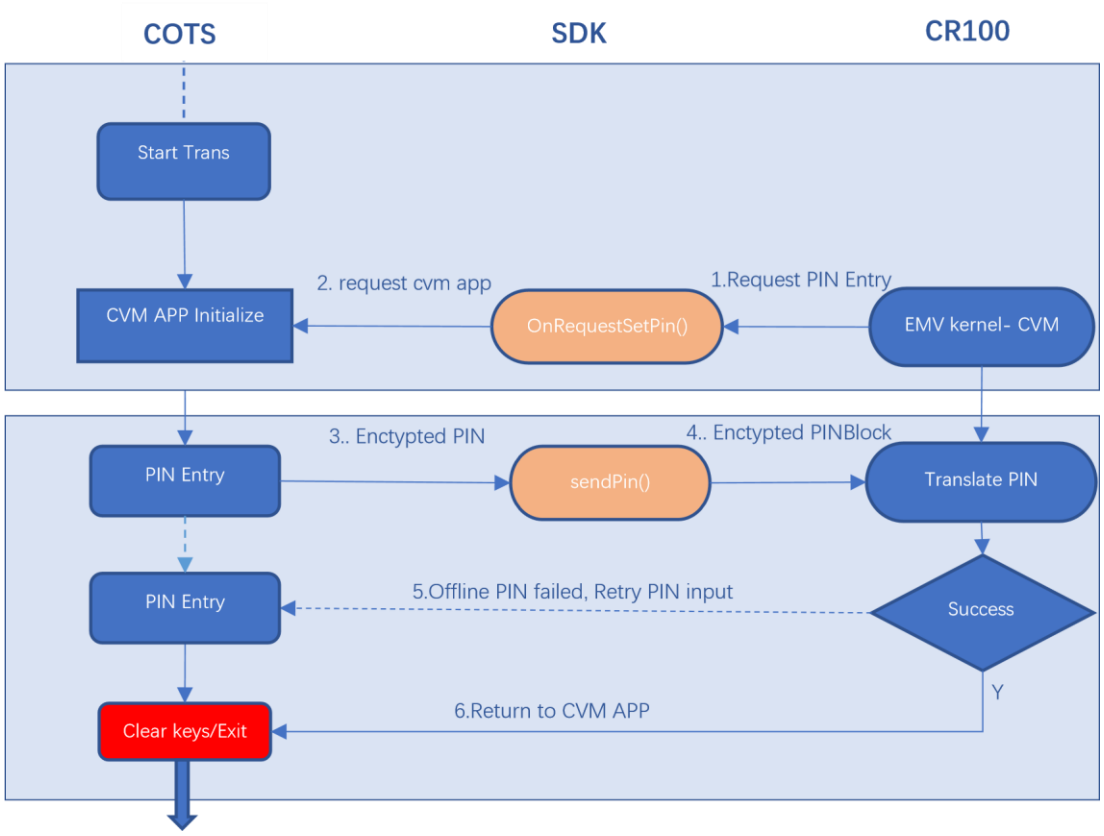
Figure 1 Software-based PIN Entry Solution

DSPREAD PIN CVM APP API

DSPREAD PIN CVM APP API enforce the security between COTS and CR100. Bellow 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.



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.

SCRP and APP Mutual Authentication

The Protection Key is used to protect Session key, and Session key is used to protect communication data,

- Protection key:
  1. Get random seed from the SCRP
  2. Generate random number based on random seeds in SDK
  3. Use random number, primitive root and public prime to generate Key A (based on Diffie Hellman)
  4. Send key A to SCRP, and obtain key B and Signed hash value from SCRP, this value uses private key to signature.
  5. SDK uses corresponding pubic key to verify signature, if failed, APP is not allowed to communicate with SCRP.
  6. In SDK, use key B, random number and public prime to generate protection key.
     In SCRP, use key A, its own random number and public prime to generate protection key.
     Based on Diffie Hellman, the protection keys are the same.

- Session key:
  1. Get the ciphertext session key comes from the SCRP, the session key is randomly generated by SCRP and stored in the security chip of SCRP.
  2. Use protection key to decrypt ciphertext session key, and the decryption is using AES CBC algorithm.

SCRP Enablement Token API

Token example

0F20000000020000000000112A4537790020123

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 loaded 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:

000200812F0B2F83BF59583A2B93FC30236AF7E190D72740D2F0D36DCB34A430F9DE66D7E658BE1C
9AE1BA2D1A666EE346A7ED778ED0F1870D29D1CFF042521E1103A957A717D6D917DF3DDEE8AC68
17EF69C7B4EABEF49D580F2A3F7B18256378F89D4E3F72C99E42EC3DBBE7B5880B8EC749CE200DBA
BFC07692A1D6BA140D10959A9072E75CDE634395BEE8D6D1C1AD767DED447849DCB42975522C46
51867CF3A31E72ED8EDA108E7F7C2DD36F6D4161915E208C9D181471811B2AAA2C64F7ECAC63878
0D61295BCDCE27C89DBE6494685CED20C20D7BEC47D621A897B55ED0C71096B56F564B9E82A6E93
D01F8BBE6C0F5D19E69E457C7C742004D2C01BCD5D4D0226EF886878B2B453CDEDAC9B119EDB601
11038A26786494C027A1B0B90391B0CD33B283CEE70493F5ACB473C4D09C4DED7BB5CDCA8148F0E
853605C9067293D82102381DDEA12FF95E340443130B21FA98D13D99A75CC675D3AB660570854FD
2E6E47E579FF784CA9F3E6B5EA0636EBD90606DCA51CD980457864C5F40B67D88161F0AD27163492
C8FC0FEE7576CDCEB15A196032206CDDBC4F932B0569E95DF2B5209B4FD3D58FA50BC50D5081D4E
42FC4292BD91E820B2BFAABC699F6094D4F21545D464B055274F9637AA9564363912BF9185DC91ED
DAA1D43235864955B48548333C5601C592044B5CD03A80C11A947B243A45EB4550C6BCB646752B8
313FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Enablement Token API

pos.updateWorkKey(String envelope)

APPENDIX

Build ISO-format4 PIN (Android)

Source Code

Note: This part of source code can be the reference when passing SPOC certification, they have been included in SDK for mutual authentication process, there is no need to call in APP processing.

```
private String buildCvmPinBlock(Hashtable<String, String> value, byte[] pin) {
    String randomData = value.get("RandomData") == null ? "" : value.get("RandomData");
    String pan = value.get("PAN") == null ? "" : value.get("PAN");
    String AESKey = value.get("AESKey") == null ? "" : value.get("AESKey");
    String isOnline = value.get("isOnlinePin") == null ? "" : value.get("isOnlinePin");
    String pinTryLimit = value.get("pinTryLimit") == null ? "" : value.get("pinTryLimit");
    //iso-format4 pinblock
    int pinLen = pin.length;
    String newPin = "";
    for (byte p: pin
        ) {
       newPin += Integer.toHexString(p).toUpperCase();
    }
    newPin = "4" + Integer.toHexString(pinLen) +newPin;
    for (int i = 0; i < 14 - pinLen; i++) {
       newPin = newPin + "A";
```

```
}
newPin += randomData.substring(0, 16);
String panBlock = "";
int panLen = pan.length();
int m;
if (panLen < 12) {
   panBlock = "0";
   for (int i = 0; i < 12 - panLen; i++) {
      panBlock += "0";
   }
   panBlock = panBlock + pan + "0000000000000000000";
} else {
   m = pan.length() - 12;
   panBlock = m + pan;
   for (int i = 0; i < 31 - panLen; i++) {
      panBlock += "0";
   }
}
String pinBlock1 = AESUtil.encrypt(AESKey, newPin);
newPin = Util.xor16(Util.HexStringToByteArray(pinBlock1), Util.HexStringToByteArray(panBlock));
String pinBlock2 = AESUtil.encrypt(AESKey, newPin);
return pinBlock2; }
```

Build ISO-format4 PIN (iOS)

Source Code

Note: This part of source code can be the reference when passing SPOC certification, they have been included in SDK for mutual authentication process, there is no need to call in APP processing.

```
//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;
}
```

Key Exchange for Mutual Authentication (Android)

Source Code

Note: This part of source code can be the reference when passing SPOC certification, they have been included in SDK for mutual authentication process, there is no need to call in APP processing.

```
CommandDownlink dc = new CommandDownlink(0x41, 0x1E, 10,Util.HexStringToByteArray("03"));
        pos.sendCommand(dc);
        CommandUplink uc = pos.receiveCommandWaitResult(timeout);
        Tip.i("random = "+Util.byteArray2Hex(uc.getAllBytes()));
        int len = uc.getByte(0);
        String randomData = Util.byteArray2Hex(uc.getBytes(1,len));
        SecureRandom secureRandom = new SecureRandom();
        int hex = Integer.parseInt(randomData,16);
        int secureRandomD = secureRandom.nextInt(hex);
        random = secureRandom.nextInt(hex);
        if(random == 0){
           random ++;
        }
```

```
        Tip.e("random 2= "+random+" secureRandomD = "+hex+" "+secureRandomD);
        A = (long) AESUtil.largeMOD(n,random,p);
        if (!isPosExistFlag()) {
            return false;
        }
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                flag = doSafetyMode();
            }
        });
        thread.start();
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

private boolean doSafetyMode(){
    if (!connect(1)) {
        return false;
    }
    CommandDownlink dc = null;
    CommandUplink uc = null;
    String aValue = Util.convertAscIIStringToHex(A+"");
```

```
String paras = "02"+Util.intToHexStr(aValue.length()/2)+aValue;
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;
}
Tip.d("safety mode: " + Util.byteArray2Hex(uc.getBytes(0, uc.length())));
if(uc.result() == 0){
   String result = Util.byteArray2Hex(uc.getBytes(0, uc.length()));
   int index = 0;
   int keyBLen = uc.getByte(index++);
   String keyB = Util.byteArray2Hex(uc.getBytes(index,keyBLen));
   index+=keyBLen;
   int keyBShaLen = uc.getByte(index++);
   String keyBSha = Util.byteArray2Hex(uc.getBytes(index,keyBShaLen));
   index+=keyBShaLen;
   int sessionKeyLen = uc.getByte(index++);
   String sessionKey = Util.byteArray2Hex(uc.getBytes(index,sessionKeyLen));
   boolean signResult =Util.check(keyB,keyBSha);
   if(!signResult) return false;
   long B = Long.parseLong(Util.convertHexToString(keyB));
   long keyResult = (long)AESUtil.largeMOD(B,random,p);
   productKey = keyResult+posId;
```

```
            if(productKey.length() < 32){
                int len = productKey.length();
                for(int i = 0 ; i < 32 - len; i++){
                    productKey+="0";
                }
            }
            productKey = Util.convertAscIIStringToHex(productKey);
            usbSessionKey = AESUtil.decryptCBC(productKey,sessionKey);
            disConnect("doSafetyMode");
            return true;
        }
        disConnect("doSafetyMode");
        return false;
    }
```

Key Exchange for Mutual Authentication (iOS)

Source Code

Note: This part of source code can be the reference when passing SPOC certification, they have been included in SDK for mutual authentication process, there is no need to call in APP processing.

```
-(BOOL)buildSafetyMode{
    NSDictionary *result = [self sycnDoGetPosId:self.pos delay:10];
    __block BOOL flag = false;
    if(result != nil && result.count > 0){
        NSString *posId = [result objectForKey:@"posId"];
        CommandDownlink *dc = [[CommandDownlink alloc] init:0x41 aSubCode:0x1E aDelay:10
aParas:[QPOSUtil HexStringToByteArray:@"03"]];
```

```
        CommandUplink *uc = nil;
        [self.pos sendCommand:dc];
        uc = [self.pos receiveCommandWaitResult:5];
        NSInteger index = 0;
        NSInteger len = [uc getByte:index++];
        NSInteger random = [QPOSUtil byteArrayToInt:[uc getBytes:index Length:len]];
        random = arc4random_uniform((uint32_t)random)+1;
        NSInteger keyA = [QPOSUtil largeMOD:n y:random z:p];
        dispatch_async(self.self_queue, ^{
            flag = [self doSafetyMode:keyA random:random p:p posId:posId];
        });
    }
    return flag;
}


-(BOOL)doSafetyMode:(NSInteger)keyA random:(NSInteger)random p:(NSInteger)p posId:(NSString *)posId{
    NSString *aValue = [QPOSUtil byteArray2Hex:[QPOSUtil stringFormatTAscii:@(keyA).stringValue]];
    NSString *paras = [NSString stringWithFormat:@"02%@%@",[QPOSUtil byteArray2Hex:[QPOSUtil
IntToHexOne:aValue.length/2]], aValue];
    CommandDownlink *dc = [[CommandDownlink alloc] init:0x41 aSubCode:0x1E aDelay:10 aParas:[QPOSUtil
HexStringToByteArray:paras]];
    CommandUplink *uc = nil;
    [self.pos sendCommand:dc];
    uc = [self.pos receiveCommandWaitResult:5];
    BOOL f = [self checkCmdId:uc];
    if (!f) {return false;}
```

```
if(uc.result == 0){
    NSInteger index = 0;
    NSInteger keyBLen = [uc getByte:index++];
    NSString *keyB = [QPOSUtil asciiFormatString:[uc getBytes:index Length:keyBLen]];
    index += keyBLen;

    NSInteger keyBShaLen = [uc getByte:index++];
    NSString *keyBSha = [QPOSUtil byteArray2Hex:[uc getBytes:index Length:keyBShaLen]];
    index += keyBShaLen;

    NSInteger sessionKeyLen = [uc getByte:index++];
    NSString *enSessionKey = [QPOSUtil byteArray2Hex:[uc getBytes:index Length:sessionKeyLen]];
    index += keyBShaLen;

    BOOL signResult = [QPOSUtil check:keyB shaContent:keyBSha];
    if (!signResult) {return false;}

    NSInteger B = keyB.integerValue;
    NSInteger keyResult = [QPOSUtil largeMOD:B y:random z:p];
    NSString *protectKey = [NSString stringWithFormat:@"%ld%@",(long)keyResult,posId];
    if (protectKey.length < 32) {
        NSInteger len = protectKey.length;
        for (NSInteger i = 0; i < 32 - len; i++) {
            protectKey = [protectKey stringByAppendingString:@"0"];
        }
    }
```

```
        protectKey = [QPOSUtil byteArray2Hex:[QPOSUtil stringFormatTAscii:protectKey]];

        NSData *tempKey = [QPOSUtil AES256CBCDecrypt:[QPOSUtil HexStringToByteArray:enSessionKey]
key:protectKey];

        self.sessionKey = [QPOSUtil byteArray2Hex:tempKey];

        [self disConnect];

        return true;

    }

    [self disConnect];

    return false;

  }
```

FAQ

1. How is the secure channel provided between the Android application and the CR100?
   The android application and the CR100 use the secure bluetooth communication as the secure channel which transfer the encrypt datas to each other.

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 secure channel between the Android application and the CR100 provide mutual authentication of the two components?

You can check the page 8 - **SCRP and APP Mutual Authentication**

4. What Bluetooth security modes and security levels are permitted?
   The mode 4 level 3 are permitted.

5. What cryptographic algorithms are used between the Android application and the CR100 at the software SDK layer?
   We use the **Diffie–Hellman** cryptographic algorithm as the mutual authentication between the Android application and the CR100 at the software SDK layer. And pls check the below **Diffie–Hellman** cryptographic algorithm generation steps

6. How dose the DH key generate? how many bits does the DH key have?
   1) Get random seed from the SCRP
   2) Generate random number based on random seeds in SDK
   3) Use random number, primitive root and public prime to generate Key A (based on Diffie Hellman)
   4) Send key A to SCRP, and obtain key B and Signed hash value from SCRP, this value uses private key to signature.
   5) SDK uses corresponding pubic key to verify signature, if failed, APP is not allowed to communicate with SCRP.
   6) In SDK, use key B, random number and public prime to generate protection key.

In SCRP, use key A, its own random number and public prime to generate protection key.Based on Diffie Hellman, the protection keys are the same.

And pls check the below **signature verification** source code.

```
if(uc.result() == 0){
    String result = Util.byteArray2Hex(uc.getBytes(0, uc.length()));
    int index = 0;
    int keyBLen = uc.getByte(index++);
    String keyB = Util.byteArray2Hex(uc.getBytes(index,keyBLen));
    index+=keyBLen;
    int keyBShaLen = uc.getByte(index++);
    String keyBSha = Util.byteArray2Hex(uc.getBytes(index,keyBShaLen));
    index+=keyBShaLen;
    int sessionKeyLen = uc.getByte(index++);
    String sessionKey = Util.byteArray2Hex(uc.getBytes(index,sessionKeyLen));
    boolean signResult =Util.check(keyB,keyBSha);

    if(!signResult) return false;
    long B = Long.parseLong(Util.convertHexToString(keyB));
    long keyResult = (long)AESUtil.largeMOD(B,random,p);
    Tip.i("keyres = "+keyResult);
    productKey = keyResult+posId;
    if(productKey.length() < 32){
        int len = productKey.length();
        for(int i = 0 ; i < 32 - len; i++){
            productKey+="0";
        }
    }
    productKey = Util.convertAscIIStringToHex(productKey);
    usbSessionKey = AESUtil.decryptCBC(productKey,sessionKey);
    disConnect("doSafetyMode");
    return true;
}
```

We use the 2048 bits p and 224 bits q to generate the private-public key which is the random DH key, so the DH key bits is random.

7. 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.