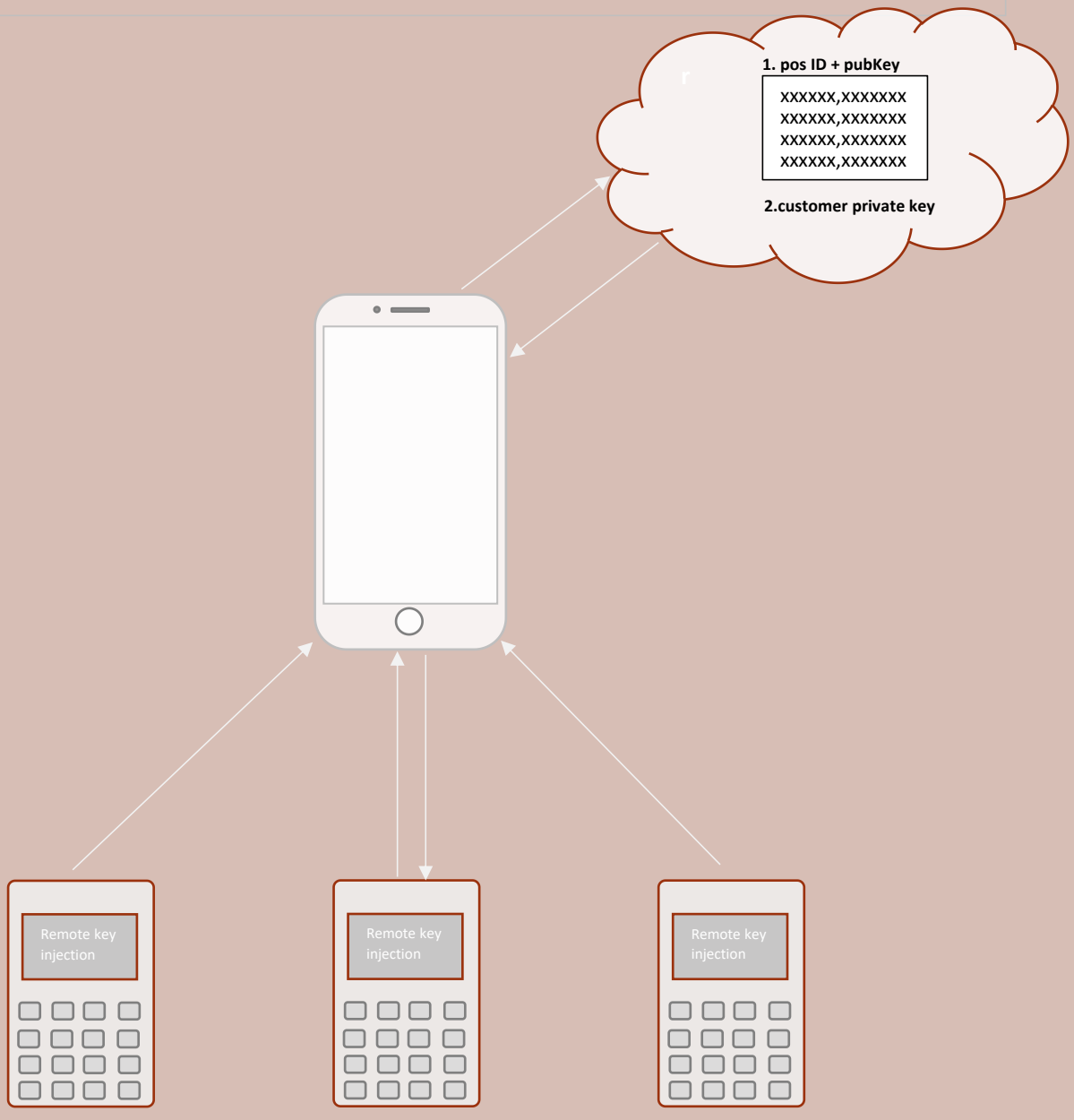
Phase 3 *TRM SYSTEM*

Terminal Management System



1 TRM system-terminal management system

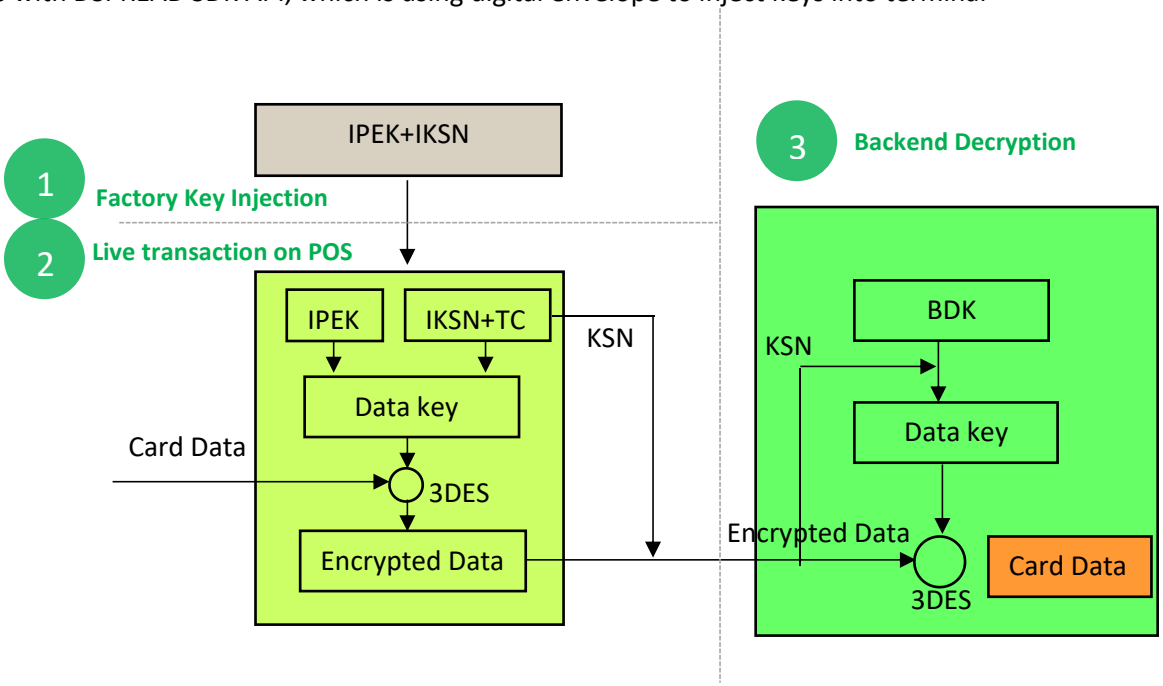
QPOS life cycle is mainly including two steps: Production in factory and TRM on merchant field. This chapter is mainly focus on QPOS key management scheme. Currently QPOS is supporting DUKPT and MK/SK key management scheme. As hardware vendor, DSPREAD will play a role of key injection facility. All the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

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

Overview on DUKPT

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

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



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

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

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

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

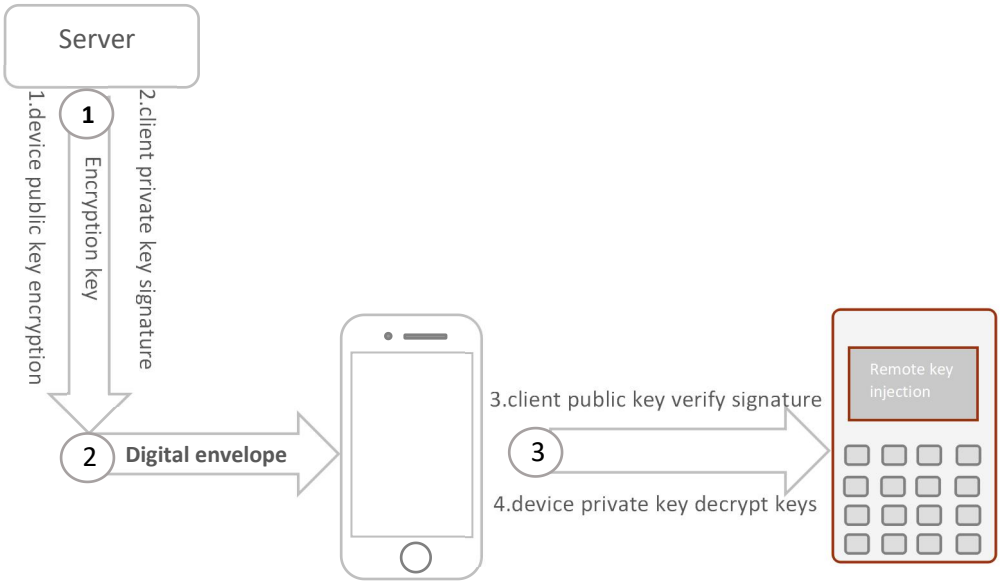
Additional:

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

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

2 Key update-RKI (Remote Key Injection)

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



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

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

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

Digital envelope implementation source code

Firstly, please download the source code by this link:

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

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

DukptKeys.java

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

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

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

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

Envelope.java

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

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

Key Map Inside QPOS

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

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

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