CloudPOS Remote Key Injection

Demo System

V1.1

1 Summary	
2 Definitions	
3 Key Injection	4
3.1 Core Process	4
3.2 Connection Protection	
3.3 POS terminal Initialization	
4 Generate Public Keys	7
4.1 Create an XCA DB	7
4.2 Create Owner Key Pair	
4.3 Create Key Injector Host Key Pair	11
4.4 Prepare For POS terminal Initialization	
5 Initialization	
6 Demo System	15
6.1 Functions	
6.2 Running RemoteKeyInjectServer	
6.3 Running InjectKeyDemo	
6.4 Secure Communication	
6.4.1 Portecle Usage	
6.5 Agent in POS terminal (InjectKeyDemo)	
6.5.1 Manifest and Permissions	
6.5.2 Source Code Structure	
6.6 Host Application(RemoteKeyInjectServer)	
6.6.1 Main data structure	
6.6.2 Configuration file	
7 POS terminal Key Injection API Guide	
7.1 Key Injection AIDL Java API	
7.1.1 getAuthInfo	
7.1.2 importKeyInfo	
7.2 Permission	

目录

Version	Author	Date	Description
1.0	Hans	2018-03-02	

1 Summary

The cloudPOS remotely key injection mechanism support mutual authentication between POS terminal and Key Injector.

2 Definitions

Host	Key Injector Host
POS Root Key	Private key of root asymmetric key pair, stored in HSM of POS terminal.
POS Root Public Key	Public key of root asymmetric key pair, stored in host.
Host Root Key	Private key of host asymmetric key pair, stored in host.
Host Root Public Key	Public key of root asymmetric key pair, stored in terminal.

3 Key Injection

3.1 Core Process

Basically, the key injecting process includes 4 steps:



POS Send AuthInfo: The application in POS gets the AuthInfo via the HSM API, then sends it to host. AuthInfo contains:

PubKeyP: The public key of each POS. It is stored in a certificate file signed by POS Root Key. So it can be verified by the POS Root Public Key.

Rand: The 32 bytes random data generated by the HSM of POS. It's unique for each key injection transaction.

SN: The hardware serial number. It's unique for each POS terminal.

SignatureP: The signature of the data including SN and Rand. The algorithm is SHA256withRSA.

- Host Verify AuthInfo: Host verify the AuthInfo data, after the host receive them. Host use the POS Root Public Key to verify the PubKeyP, then use the PubKeyP to verify the SignatureP. So the Host will know if the AuthInfo is come from a trusted POS terminal.
- Host Send KeyInfo: The Host generate the KeyInfo data and send to POS terminal. The KeyInfo data contains:

PubKeyH: The public key of the Host. It is stored in a certificate file signed by Host Root Key and it can be already injected in the POS when initializing POS terminal.

Rand: The 32 bytes random data which is received from POS terminal in the AuthInfo data.

ENC(KeyData): The encrypted data of KeyInfo. It's encrypted by PubKeyP (the public key of the POS terminal). The algorithm is RSA/ECB/PKCS1Padding.

SignatureH: The signature of the data including Rand and ENC(KeyData). The algorithm is SHA256withRSA.

POS terminal Verify KeyInfo: The application in POS terminal will inject the KeyInfo, after it get it from Host. The HSM model of the POS terminal will verify the PubKeyH by the existing Host Root Public Key, and verify the SignatureH by the PubKeyH. If success, the HSM get the decryption data the KeyInfo by its own POS terminal PrivKey and store the data.

3.2 Connection Protection



The POS terminal and the Key Injector Host can connect each other in internet. So the application in POS terminal can use SSL/HTTPS connection with Host.

3.3 POS terminal Initialization

The Host Root Public Key should be stored in the POS terminal as the trusted key at the beginning. cloudPOS terminal is designed to use simple certificate file to protect and store the trusted Host Root Public Key.



Figure 1.

Vendor Root PubKey: Verndor Root Public Key. The public key of the vendor. It is been initialized in the ROM of the POS terminal. It is used to verify the first POS Owner PubKey (the future POS Owner PubKey are verified by the previous Owner PubKey).

Owner PubKey: Owner Public Key. The public key of the POS terminal owner who buy the POS terminal. It controls what the public key of which Key Injector Host can be loaded to POS terminal.

Host Root PubKey: Host Root Public Key. The public key stored in POS terminal to authenticate the Key Injector Host.

PriKey/PubKey: The private/public key pair is unique of each POS terminal. And the private key is only stored in one POS terminal.

POS terminal Root PubKey: POS Root Public Key. This public key is used to authenticate the POS terminal PubKey. It can be used by Key Injector Host.

The PriKey and PubKey of POS terminal are already initialized when terminal is in factory.

We as the POS vendor will help the POS terminal owner to initialize the Host Root Public Key to POS terminal.

4 Generate Public Keys

There are many tools to generate the public keys, including OPENSSL, XCA... In this document, we use XCA GUI tool to demonstrate how to generate the public keys.

4.1 Create an XCA DB

To create an XCA DB, follow these steps:

1. Launching the XCA.

2. Select **File > New Database**, input the db name, to create your owner public keys database. Please keep it safely and privately.

4.2 Create Owner Key Pair

To create the Owner Key Pair, follow these steps:

- 1. In Private Keys tab, click New Key to create the key pair with 2048 bit size.
- 2. In New Key window, input the name, select 2048 bit, click Create.

ternal nam	ne 💌 🛞 🗊 🛛 X Cer	tificate and Key mana	igement			Kau
	New key Please give a	name to the new key a	nd select the o	desired keysize		ort
	Name	es la companya de la comp				(PKCS#12)
	Karkara	MyOwner				etails
	Keytype	RSA		ete		
	Keysize	2048 bit			▼	
	Rememb	er as default		<u>C</u> ancel Cre	eate	
	_	_	-	0	HDX Y	

Database: /home/disk500g/temp/remotelyKeyInjectionDemo/RKIDemo.xdb

Figure 2. New Key

rivate Keys	Certifi	icate signi	ng requests	Certificates	Templates	Revocation lists	
Internal nar	ne 🔻	Туре	Size	Use	Password		Nove Kone
My0	wner	RSA	2048 bit	- 0 C	ommon		New Key
							Export
							Import
							mport PFX (PKCS#12)
							Show Details
							Delete
						6	HOX

Figure 3. Private Keys of Main window after key created

Then generate the CSR for Owner Key. This CSR file will be used to generate self-signed certificate in XCA, and it will be submit to us. We will issue the new owner certificate

To Create the Owner CSR and send to us, follow these steps:

- 1. In Certificate signing requests tab, click New Request.
- 2. In Source tab, set signature algorithm as SHA 256.

	Subject	Extensions	Key usage	Netscape	Advanced		
Signing	request						
unst	ructuredNa	ame					
chal	lengePassw	vord					
۲	Create a se	elf signed cert	ificate with th	ne serial 🚺			
	Use <u>t</u> his Ce	ertificate for s	igning				4
	Use this Ce	ertificate for s	igning		(FUA 256		
) Signatu	Use this Ce re algorithr	ertificate for s	igning		SHA 256		4
© Signatu Templa	Use <u>t</u> his Ce re algorithr te for the i	ertificate for s m new certifical	igning te		SHA 256		:
Gignatu Templa	Use this Ce re algorithr te for the r fault] CA	ertificate for s m new certifica	igning te		SHA 256		:
Signatu Fempla	Use this Ce re algorithr te for the r fault] CA	ertificate for s m new certifica	igning te		SHA 256	ions Apply subj	ect) Apply all

Figure 4. Source of Create Certificate signing request

3. In **Subject** tab, set Subject's distinguished name as your company information. Please set the internal name, organizationName, countryName, organizationUnitName, stateOrProvinceName, commonName, localityName, emailAddress to your real information. Select the private key created just now.

	Subject	Extension	ns Key usage	Netscape	Advanced		
isting	uished nar	ne					
Inter	Internal name		MyOwner		organizationName		
cour	ntryName				organizationalUnitName	e [
state	stateOrProvinceName				commonName	MyOwne	ſ
loca	lityName				emailAddress		
		Туре			Content		Add
							Delete
F							

Figure 5. Subject of Create Certificate signing request

4. In Extension tab, set type as certificate authority (actually your certificate needn't to be CA, it's just for XCA manage the certificate easily).

Source	Subject	Extensions	Key usage	Netscape	Advanced			
X509v3	Basic Con	straints					Key identifier	
Туре	C	ertification A	uthority		÷)	Subject K	ey Identifier
Path	length					Critical	Authority	/ Key Identifier
raudicy		Paratalana		Time	range			
	1.	Frank and a						
Not l	after	2018-03-0	7 01:39 GMT 7 01:39 GMT	▼ 1 ▼	Midnight	Local time	Years 🛟	Apply ned expiration
Not I Not a	oerore after Subject Al	2019-03-0	7 01:39 GMT	▼ [1 ▼	Midnight	Local time	Vears	Apply ned expiration Edit
Not I Not a K509v3 K509v3	serore after Subject Al Issuer Alte	2018-03-0 2019-03-0 ternative Name	7 01:39 GMT	▼ 1 ▼	Midnight	Local time	Vears :	Apply ned expiration
Not I Not a X509v3 X509v3 X509v3	serore after Subject Al Issuer Alte CRL Distri	2019-03-0 2019-03-0 ternative Name ernative Name bution Points	7 01:39 GMT	▼ 1 ▼	Midnight	Local time	Vears	Apply ned expiration

Figure 6. Extensions of Create Certificate signing request

5. In Key Usage tab, set the proper key usage flag for owner key according TerminalCertificateGuide_en.pdf.

ource	Subject	Extensions	Key usage	Netscape	Advanced	
(509v3	Key Usage	1			X509v3	Extended Key Usage
	Critical				c	Critical
Digi Non Key Dat Key Cert CRL Enci Dec	ital signatu Repudiatic Encipherm a Encipherm a Encipherm da Encipherm tificate Sign Sign ipher Only ipher Only	re on ent t			TLS V TLS V Code E-ma Time Micro Micro Micro Micro Micro IPSee IPSee IPSee IPSee EAP (EAP (KDC)	Veb Server Authentication veb Client Authentication veb Client Authentication signing soft Individual Code Signing soft Trust List Signing soft Forver Gated Crypto soft Ers File Recovery tend System Tunnel User curity end entity soft FS File Recovery Signing over PPP over Lan Authentication

Figure 7. Key usage of Create Certificate signing request

6. Click **OK** to create the owner CSR.

Certif	icate signing requests	Certificates	Templates	Revocation lists	
ne 💌	commonName	Signed			Novi Desurati
wner	MyOwner	Unhandled			New Request
					Export
					Import
					Show Details
					Delete
				Ĩ	Farmineeta Dingono 7 in
	Certif	Certificate signing requests	Certificate signing requests Certificates ne Certificates Ne CommonName Signed Wner Unhandled	Certificate signing requests Certificates Templates	Certificate signing requests Certificates Templates Revocation lists ne commonName Signed wner MyOwner Unhandled

Figure 8. Certificate signing requests of Main window after create owner CSR 7. In Certificate signing requests tab, click **Export**.

Certific	ate request export	a) reminant yes
ame Myowi	er	
Filename	C:\Users\Administrator\Myowner.pem	
PEM Text	format with headers	Export Format PEM (*. pem) 🔻

Figure 9. Certificate request export

8. Click **OK** to export the CSR. Then send it to <u>support@wizarpos.com</u>, and wait the reply.

- To Import the certificate replied, follow these steps:
- 1. In Certificates tab, click Import.

2. Select the replied certificate to import.



Figure 10. Certificates of Main window after import certificate

4.3 Create Key Injector Host Key Pair

Assume the Key Injector Host Key Pair will be used in a Tomcat server, so we generate the key in a JKS keystore.

To get and export the host certificate, follow these steps:

1. Use follow command to generate the key pair in myhost.jks file:

keytool -genkeypair -keystore **myhost.jks** -keyalg RSA -keysize 2048 -alias **myHost** -dname "CN=**MyHost**,EMAILADDRESS=**myname@abc.com**" -validity **7300**

In the command, bold part should be modified to real information. We only write CN and EMAIL in -dname, you can write other option dname information. -validity is to set valid

days, default is 90 days.

2. Use follow command to generate the CSR of host key:

keytool -certreq -keystore myhost.jks -alias myHost > myHost.csr

In the command, key alias name, myHost, should same with the key alias in step 1.

3. In **Certificate signing requests** tab, click **Import** to import the myHost.csr to RKIDemo.xdb created before.



Figure 11. Certificate signing requests of Main window after import host CSR

4. Sign the MyHost CSR by MyOwner. Set the proper valid time and the key usage flag according key loader root cert in TerminalCertificateGuide_en.pdf, as follows:



Database: /home/disk500g/temp/remotelyKeyInjectionDemo/RKIDemo.xdb

Figure 12. Sign of Right click menu

	Extensions	Key usage	Netscape	Advanced			
igning	request						
	Sign this Certi	ificate signing	request		MyHost		\$
	Copy extensio	ons from the r	equest		SI	now request	
	Modify subjec	t of the requ	est				
			3		(*)
ignatu	re algorithm				SHA 256		ļ
	ice for the nev	w certificate					
empla	faulti CA						

Figure 13. Source of Create X509 Certificate

	rtificate					
rce Extensio	ons Key usage	Netscape Ad	dvanced			
09v3 Basic Co	nstraints				Key identifier	
Туре	Not defined		*)	Subject Key	Identifier
Path length				Critical	Authority Ke	y Identifier
2000 - W	2010.02.00.07		20.5		Vears +	Apply
Not before	2018-03-09 07:	55 GMT -				Apply
Not before Not after	2018-03-09 07:	55 GMT 👻	☐ Midnight	Local time	No well-defined	d expiration
Not before Not after 29v3 Subject A	2018-03-09 07: 2038-03-09 07:	55 GMT V	Midnight	Local time	No well-defined	Apply J expiration Edit
Not before Not after 19v3 Subject A 19v3 Issuer Alt	2018-03-09 07: 2038-03-09 07: Iternative Name ernative Name	55 GMT V 55 GMT V	☐ Midnight	Local time	No well-defined	Edit Edit
Not before Not after 19v3 Subject A 19v3 Issuer Alt 19v3 CRL Distr	2018-03-09 07: 2038-03-09 07: Iternative Name ernative Name ibution Points	55 GM1 V 55 GM1 V	D Midnight	Local time	No well-defined	Edit Edit Edit

Figure 14. Extensions of Create X509 Certificate

ource	Extensions	Key usage	Netscape	Advanced	
(509v3	Key Usage				X509v3 Extended Key Usage
	Critical				Critical
Digi Not Key Dati Key Cat CRL Enci Dec	tal Signature Reputation Encipherment a Encipherment ificate Sign Sign pher Only ipher Only	t.	Key usa	ge	TLS Web Client Authentication TLS Web Client Authentication Code Signing E-mail Protection Time Stamping Microsoft Individual Code Signing Microsoft Trust List Signing Microsoft Server Gated Crypto Microsoft Ers File Recovery IPSec End System IPSec Tunnel IPSec User IP security end entity Microsoft Signing EAP over LPP EAP over LPP EAP over LAN KDC Authentication

Figure 15. Key usage of Create X509 Certificate

Click OK, the signing process has finished and the host certificate has been created. 5. Click Certificates tab, find MyHost certificate.

rivate Ke	ys Certi	ficate signing r	equests	Certificates	Templates	Revocation lists
Interna	Îname	commonName	CA	Serial	Ex	New Certificate
• A:	Myowner	dasg MyHost	VYes	2179D4A0F72B	3DBC 20	Export
J.C.	85					Import
						Show Details
						Delete
						Import PKCS#12
						Import PKCS#7
						Plain View
						Josephineta Divistano6 J Jim

Figure 16. Certificates of Main window after host certificate created 6. Select MyHost certificate, click Export

Certificate export	(a) Diministry The
ame MyHost	
Filename C:\Users\Administrator\MyHost.pem	
Concatenated text format of the complete certificate chain in one PEM file	Export Format PEM chain (*. pem)

Figure 17. Certificate export

7. Click OK, then get the host certificate.

4.4 Prepare For POS terminal

Initialization

There are two ways to do certificates initialization:

- 1. Configure the new owner cert file and host key loader cert file from wizarview, after the POS terminal restart and connect to network, it will get the new owner and host key loader cert file.
- 2. Send the owner cert file and host key loader cert to us, we will create an initialize APK. Run the APK to do initialization.

Normally, customer use wizarview to do the initialization.

5 Initialization

After initialize the POS terminal, it is ready to remotely key injection.

6 Demo System

Please prepare certificates for POS terminal and remote injector according as chapter 4

The server application: **RemoteKeyInjectServer (eclipse project)** The client application: **InjectKeyDemo (android-studio project)**

6.1 Functions

• POS terminal Agent(client) ask Host to inject the MK/SK or DUKPT keys to termnal.

- POS terminal Agent(client) ask Host to get check data to check the key injecting.
- MK/SK or DUKPT initial key component could be updated in Host's configure file,

6.2 Running RemoteKeyInjectServer

```
1. Import RemoteKeyInjectServer into eclipse
   2. Find class com.cloudpos.rki.Starter.java
   3. Edit keylist.txt file, and add your key information.Such as
## key type: 1 <u>dukpt</u> key 2 master key 3 transport key
##
## dukpt key
## 1.<u>sn</u> = key_index, usage, counter, <u>ksn</u>, initial key
##
## usage, O:PIN Key, 1:MAC Key, 2:Data Key
##
## master key
## 2.sn = key_index, hex_master_key(length: 32 or 48)
##
## transport key
## 3.<u>sn</u> = key_index, hex_transport_key(length: 32 or 48)
##
1.WP15461Q00002422
                                 2,
                                       2,
                                                    FFFF9876543210E0,
                                              О,
6AC292FAA1315B4D858AB3A3D7D5933A
```

4. Click the right mouse button --> Run As --> Java Application

6.3 Running InjectKeyDemo

- 1. Import InjectKeyDemo into Android-Studio
- 2. Edit SSLConnect.java file, and enter your own IP address as host value

```
// Change to your own host address
```

```
private String host = "121.199.23.212";
```

- // Change to your own host port
- private int port = 11060;
 - 3. Connect terminal to your computer
 - 4. Runing the project and install it into terminal.
 - 5. The terminal screenshot:



Figure 18.

Click "REQUEST INJECT MASTER KEY" button will trigger following activities:

- Read AuthInfo from HSM
- Send AuthInfo to remote server
- Read master key from remote server
- Import key information into HSM
- Request cipher data information which is encrypted by key from remote server

Click "REQUEST INJECT TRANSPORT KEY" button will trigger following activities:

- Read AuthInfo from HSM
- Send AuthInfo to remote server
- Read transport key from remote server
- Import key information into HSM
- Request cipher data information which is encrypted by key from remote server

Click "REQUEST INJECT DUKPT KEY" button will trigger following activities:

- Read AuthInfo from HSM
- Send AuthInfo to remote server
- Read dukpt key from remote server
- Import key information into HSM

6.4 Secure Communication

In order to ensure the security of communication, the demo application uses two-way SSL

links and TLSv1.2 protocol.

The server-side key store: ks-server.jks and ts-server.jks The ks-server.jks path: RemoteKeyInjectServer/ks-server.jks The ts-server.jks path: RemoteKeyInjectServer/ts-server.jks

The client-side key store: ks-client.bks and ts-client.bks

The ks-client.bks path: KeyInjectDemo/app/src/main/assets/ks-client.bks The ts-client.bks path: KeyInjectDemo/app/src/main/assets/ts-client.bks

Notice:

When the application running in production, please replace key store files to ensure security.

6.4.1 Portecle Usage

Key store tool recommendation: Portecle Download url: <u>https://sourceforge.net/projects/portecle/</u> Running tool: java -jar portecle.jar 1. Crate server-side key store: ks-server.jks

. Crate server-side key store: ks-server.jks

This process create keystore that server project used.

Fortecle	
<u>Eile Tools Examine H</u> elp	
Alias Name	New Keystore Type
	Select the type of the new Keystore: JKS PKCS #12 JCEKS JKS (case sensitive) BKS BKS-V1 UBER BCEKS GKR OK Cancel
No keystore loaded	

Figure 19.

🛐 [Untitled] - Portecle	a local		
Eile Iools Examine Hel	p * 19 11 11 11 11 11 11 11 11 11 11 11 11 		
Alias Name		Last Modified	
	Generate Key Pair Key Algorithm: ODSA (Key Size: 2048 OK Cance	® RSA	
Keystore type: JKS, provider:	SUN, size: 0 entries		

Figure 20.

[Untitled] - Portec	le		
<u>File Tools Exami</u>	ne <u>H</u> elp		
	Generate Certificate		
Alias Name	Signature Algorithm:	SHA256withRSA	
	Validity (days):	365	
	Common Name (CN):	ks-server	
	Organisation Unit (OU):	IT	
	Organisation Name (O):		
	Locality Name (L):		
	State Name (ST):		
	Country (C):	US	
	Email (E):		
	Subject Alternative DNS Name:		
	ОК	Cancel	
Keystore type: JKS, p	provider: SUN, size: 0 entries		

Figure 21.

Edit the certificate contents at above picture:

🛐 [Untitled] - Portecle	
Eile Iools Examine Help	
Alias Name	Last Modified
Key Pair Entry Password Enter New Password: Confirm New Password: OK Cancel	
Keystore type: JKS, provider: SUN, size: 0 entries	

Figure 22.

ks-server.jks has created.

I [Untitled] - Portecle		23
Eile Tools Examine Help		
I Alias Name	Last Modified	1
😹 ks-server	2019-11-11下午06时01分08秒	
Set Keystore Password Enter New Password: Confirm New Password: OK Cancel		
They store type. 510, provider. 5014, 5128. Telluy		

Figure 23.

[Untitled] - Portecle	X
File Tools Examine Help	
T AI Save Keystore As	
Save In: Save In:	
58d9ddad2146e4b73f	r
📑 360downloads 📑 eclipse-w	orkspace
iniLibs	
CoudpossDKinterraceimpi_001_en-siimiotenetso	t-0_0_3_3
download Skypewor	kspace
	Þ
File Name: ks-server.jks	
Files of Type: Java Keystore Files (*.jks;cacerts)	
	Save Cancel
Keystore type: JKS, provider: SUN, size: 1 entry	

Figure 24.

2. Create ts-client.bks

This process export certificate from server keystore, put it to trust store of client app. Right click ks-server, click export,

🛐 [Untitled] - Portecle		
<u>File Tools Examine H</u> elp		
r. = B 👫 🗏 🜿 🖻		
Alias Name	Export Keystore Entry 'ks-ser	ied
ks-server	Export Type Head Certificate Certificate Chain Private Key and Certificates Export Format DER Encoded PKCS #7 PkiPath PKCS #12 OK Cancel	1 下午06时01分08秒
Keystore type: JKS, provider: SUN	, size: 1 entry	

Figure 25.

🛐 [Untitled] - Portecle		83
<u>File Tools Examine Help</u>		
C, 5 B <u>\$</u> \$ \$ \$ B \$ \$ \$		
T AI Export certificate	8	
ks Look In: SLocal Disk (F:)		
t(signed by WizarPOS).crt is testcase_app_install.pem t(signed by WizarPOS).crt is testcase_owner.crt totapps_fangwei2.pem		
File Name: ks-server.cer		
Files of Type: X.509 Certificate Files (*.cer;*.crt;*.cert;*.pem)	-	
Export	ancel	
Keystore type: JKS, provider: SUN, size: 1 entry		

Figure 26.

🛐 F:\ks-server.jks - Portecle		
<u>File T</u> ools <u>E</u> xamine <u>H</u> elp		
Alias Name	New Keystore Type	odified
ks-server	Select the type of the new Keystore: JKS PKCS #12 JCEKS JKS (case sensitive) BKS-V1 UBER BCEKS GKR OK Cancel	1-11下午06时101分08秒
Keystore type: JKS, provider: SUN	V, size: 1 entry	

Figure 27.

🕼 [Untitled] - Portecle	
Eile Iools Examine Help Image: Constraint of the second	
Alias Name	Last Modified
Import Trusted Certificate	
Look In: 😂 Local Disk (F:)	
HB_App_Root_Cert_1.crt HB_Terminal_Owner_Root_ KevInjectDemo_MyOwner_1 Ks-server.cer MyHost.crt	Cert(signed by WizarPOS).crt 🔤 otaam .crt 👜 testca: totapp
MyOwner.crt	
File <u>N</u> ame: ks-server.cer	
Files of Type: X.509 Certificate Files (*.cer;*.cr	t,*.cert,*.pem)
	Import Cancel
Keystore type. BKS, provider. BC, size. o entries	

Figure 28.



Figure 29.

	Certificate 1 of 1 >>
Version:	3
Subject:	CN=ks-server, OU=IT, O=testa, L=testb, ST=testc, C=US, EMAILADDRES
Issuer:	CN=ks-server, OU=IT, O=testa, L=testb, ST=testc, C=US, EMAILADDRES
Serial Number:	5DC9 3126
Valid From:	2019-11-11下午06时00分06秒
Valid Until:	2020-11-10下午06时00分06秒
Public Key:	RSA (2,048 bits)
Signature Algorithm:	SHA256withRSA
SHA-1 Fingerprint:	03:44:4F:6D:96:AE:19:BB:84:01:19:07:5A:13:82:7B:6D:82:37:4E
MD5 Fingerprint:	B5:A5:93:94:6E:28:7C:8D:7A:1F:B7:E9:F5:6D:82:54
	Extensions PEM Encoding

Figure 30.

I [Untitled] - Portecle		_ 0	83
Eile Tools Examine Help			
Trusted Certificate Entry Alias Enter Alias: ts-client OK Cancel	S3		
Keystore type: BKS, provider: BC, size: 0 entries			

Figure 31.

🛐 [Untitled] - Portecle	
Eile Iools Examine Help	
Alias Name	Last Modified
🔛 ts-client	2019-11-13 上午10时30分22秒
Set Keystore Password Enter New Password: Confirm New Password: OK Cance	el
Keystore type: BKS, provider: BC, size: 1 entry	

Figure 32.

[Untitled] - Portecle		n) 23
Eile Iools Examine		0	57	
AI Save Keyston	e As	Latitud		
Save In:	Local Disk (F:)	▼ III		
58d9ddad 560downlu cert cloudposS DiffFingery DTLFolder	2146e4b73f bads SDKInterfaceImpI_001_en-slim print(1) 	eclipse-workspace jniLibs lotenetsoft-0_6_3_ networkdownload skypeworkspace	3	
Files of <u>Type</u> :	BKS Keystore Files (*.bks)	Save	e Cancel	

Figure 33.

3. Create client-side key store: ks-client.bks

This process create keystore that client app used.

F:\ts-client.bks - Portecle		
<u>File Tools Examine Help</u>		
🖪 🗉 🐰 🖾 😫		
Alias Name	New Keystore Type	odified
ts-client	Select the type of the new Keystore: JKS PKCS #12 JCEKS JKS (case sensitive) BKS-V1 UBER BCEKS GKR OK Cancel	1-13 上午10时30分22秒
Keystore type: BKS, provider: BC	, size: 1 entry	

Figure 34.

Select BKS at this window, then click OK, the other process is like create ks-server.jks. 4. Create ts-server.jks

This process export the client certificate, put it to trust store of server project.

F:\ks-client.bks - Portecle		
<u>File Tools Examine Help</u>		
		D
Alias Name	New Keystore Type	odified
ks-client	Select the type of the new Keystore:	1-13 上午10时37分36秒
Keystore type: BKS, provider: BC,	size: 1 entry	

Figure 35.

Select JKS at this window, the other process is like Create ts-client.bks.



(InjectKeyDemo)

6.5.1 Manifest and Permissions

The demo application needs following permissions:

```
<!-- Access inject key service permission -->
<uses-permission
android:name="android.permission.CLOUDPOS_REMOTE_KEY_INJECTION" />
```

6.5.2 Source Code Structure

The main class diagram



powered by Astah

MainActivity.java

It provides a demo UI which can operate by user.

Click "REQUEST INJECT MASTER KEY" will call doInjectMasterKey method. Click "REQUEST INJECT TRANSPORT KEY" will call doInjectTransportKey method. Click "REQUEST INJECT DUKPT KEY" will call doInjectDukptKey method.

SSLConnect.java

It provides a secure connection to remote inject server. If you want to running application and connect to your own remote inject server, you must modify the host and port information in SSLConnect.java. Such as:

```
// Change to your own host address
private String host = "121.199.23.212";
// Change to your own host port
private int port = 11060;
```

The aidl interface, IKeyLoaderService provides an enter to operate PIN Pad. Example code:

```
boolean isSuccess = startConnectService(this, comp, this);
return isSuccess;
}
protected synchronized boolean startConnectService(Context context, ComponentName comp,
ServiceConnection connection) {
    Intent intent = new Intent();
    intent.setPackage(comp.getPackageName());
    intent.setComponent(comp);

    boolean isSuccess = context.bindService(intent, connection,
Context. BIND_AUTO_CREATE);
    Logger. debug("(%s)bind service (%s, %s)", isSuccess, comp.getPackageName(),
comp.getClassName());
    return isSuccess;
}
```

6.6 Host

Application(RemoteKeyInjectServer)

6.6.1 Main data structure



Figure 36.

MessagePack.java

It's the enter class that communication with POS terminal.

AuthInfo.java

There is an same name AuthInfo file in host with Agent in POS terminal. It's used to parse

auth info from POS terminal. The class file full name: **com.cloudpos.rki.pinpad.AuthInfo**. Example:

```
byte[] data = {};
AuthInfo authInfo = new AuthInfo(data);
```

CKeyInfo.java

Create cipher key info such as master key that will be injected into POS terminal can use **CKeyInfo**. The class file full name: **com.cloudpos.rki.pinpad.CKeyInfo**

Example about master key:

```
CKeyInfo cKeyInfo = new CKeyInfo(authInfo);
byte[] data = cKeyInfo.setMasterKey(keyIndex, key).build();
```

Example about dukpt key:

```
CKeyInfo cKeyInfo = new CKeyInfo(authInfo);
byte[] data = cKeyInfo.setDukptKey(keyIndex, ksn, int counter, key).build();
```

6.6.2 Configuration file

Path: RemoteKeyInjectServer/config.properties

```
Content
# Inject server port
localPort=11060
# The ssl key store
keystore.path=ks-server.jks
keystore.pass=wizarpos
# The ssl trust key store
truststore.path=ts-server.jks
truststore.pass=wizarpos
# PINPad key store
pinpad.keystore=MyHostSelf.p12
pinpad. keystore. passwd=myhost
pinpad.key.alias=MyHostSelf
pinpad.key.passwd=myhost
# key length
key. 1en. q2=32
key. 1en. q1v2=32
key. 1en. k2=32
```

7 POS terminal Key Injection API Guide

7.1 Key Injection AIDL Java API

7.1.1 getAuthInfo

byte[] getAuthInfo()

This API let the application to get the authentication information buffer from HSM module. The AuthInfo buffer format is described below.

Field	PubKeyP Length	PubKeyP	Random	SN length	SN	Signature
Length (byte)	4	4096	32	1	31	256

PubKeyP Length: 4 bytes little-endian. It's the real length of the contents in next PubKey field.

PubKeyP: Fixed 4096 bytes buffer to store the PubKey in simple certificate in PEM format.

Random: Fixed 32 bytes random number.

SN Length: The real length of contents in next SN field.

SN: Fixed 31 bytes buffer to store the SN.

Signature: Fixed 256 bytes buffer to store the signature.

7.1.2 importKeyInfo

int importKeyInfo(in byte[] KeyInfo)

This API let the application to import the KeyInfo which is transferred from Host. The KeyInfo buffer format is described below.

Field	PubKeyH Length	PubKeyH	Random	Cipher KeyInfo	Signature
Length (byte)	4	4096	32	256	256

PubKeyH Length: 4 bytes little-endian. It's the real length of the contents in next PubKey field.

PubKeyH: Fixed 4096 bytes buffer to store the Host public key in simple

certificate in PEM format.

Random: Fixed 32 bytes random number.

Cipher KeyInfo: The fixed 256 bytes buffer. It store the cipher text of the KeyInfo data, which is encrypted by POS terminal public key. The plain text of the KeyInfo buffer has two format. The format depends on the first byte(KeyType) of the KeyInfo:

	-					
Field	КеуТуре	KeyIndex	Reserved	KSN	Counter	Initial Key
Length (byte)	1	1	2	8	4	16
/	•					

KeyType=1. DUKPT schema:

KeyType: 1 means DUKPT key.

KeyIndex: The index number of this DUKPT key. We support 3 suit of DUKPT.

KSN: The Initially Loaded Key Serial Number.

Counter: The initially counter.

Initial Key: The initially loaded PIN entry device key.

Reserved: Two bytes, byte[1] is for key usage, 0:PIN Key, 1:MAC Key, 2:Data Key, byte[0] is reserved. From our demo server, we set 0xFF, that means, the dukpt key has not designated use, so you can use it to calculate pinblock, or mac....

KeyType=2, 3. Master Key/Session schema:

Field	КеуТуре	KeyIndex	Key Length	Reserved	Key
Length (byte)	1	1	1	1	24/32

KeyType: 2 means master key, 3 means transport key.

KeyIndex: The index of this master key. We support 10 suit of master key/session key.

Key Length: The real length of the Key field. It can be 16 or 24.

Key: The fixed 24 or 32 bytes buffer to store master key or transport key. For Q1-4G and Q2/K2, the length is 32, for Q1, the length is 24.

Reserved: Not used, please set 0.

Signature: Fixed 256 bytes buffer to store the signature of the Random + Cipher KeyInfo.

7.2 Permission

To access the key injector API, the application should declare the proper permission in its AndroidManifest file.

<!-- Access inject key service permission -->

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