How Samsung Secures Your Wallet & How To Break It

HC Ma
Tencent's Xuanwu Lab
http://xlab.tencent.com @XuanwuLab

Who am I ?

- Security Researcher @ 

- hyperchemma#tencent.com
  - Embedded Device Security
  - Firmware Reverse-Engineering
  - Fan of IoT
  - Big Fan of 

Who am I ?

- Security Researcher @ 腾讯玄武实验室 TENCENT'S XUANWU LAB
- hyperchemma#tencent.com
  - Embedded Device Security
  - Firmware Reverse-Engineering
  - Fan of IoT
  - Big Fan of

Agenda

- What’s SamsungPay
- SamsungPay Architecture
- Steal Money from SamsungPay?!

What’s SamsungPay?

=

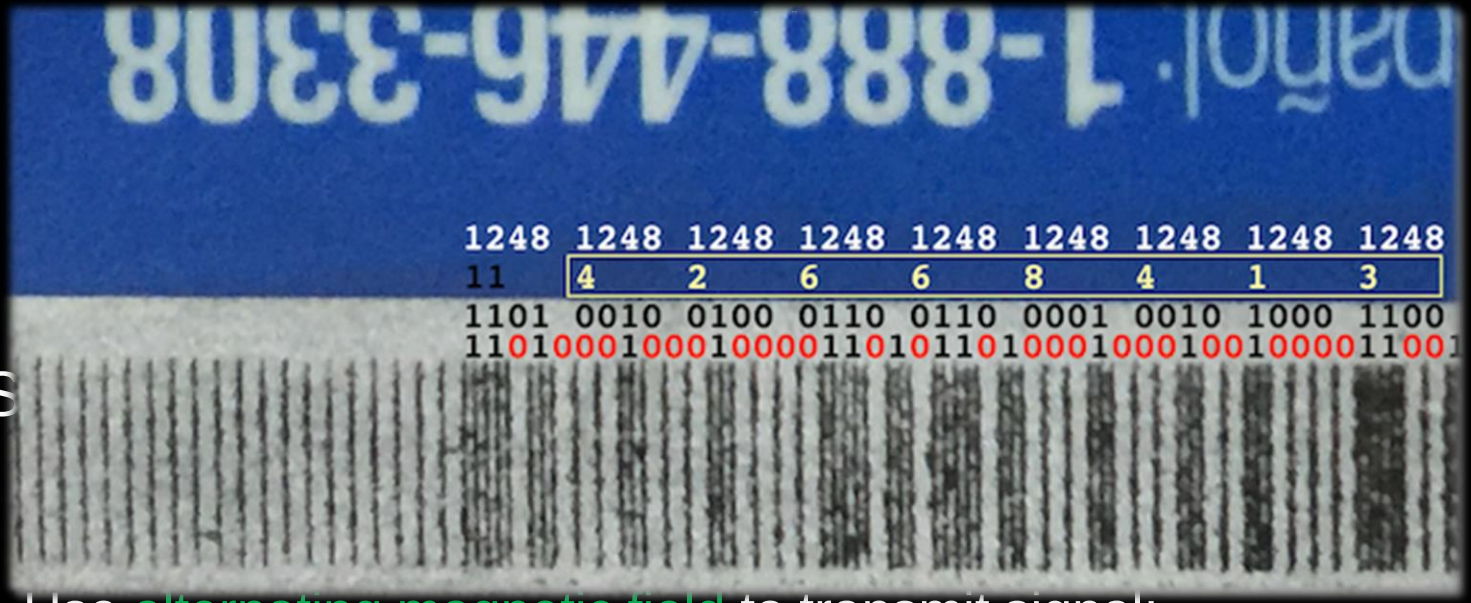
What's SamsungPay?

SAMSUNG pay =

Magnetic Card & MST

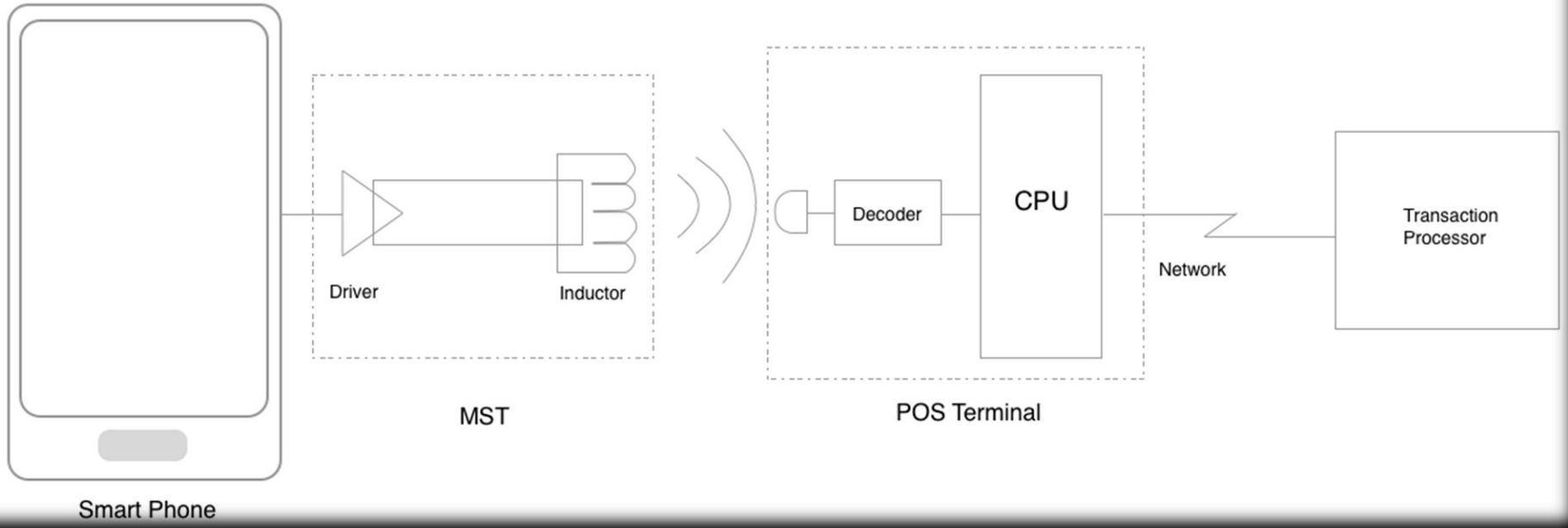
- Magnetic Card:
  - Store data using magnetic particles;
  - Physically 3 tracks on card;
  - Track2 is the only one needed for payment;
  - 6230744888888888888=2102777777777777;
  - Card Skimmer;
- MST:
  - Magnetic Secure Transmission;
  - Technology for simulating magnetic card;
  - Use alternating magnetic field to transmit signal;
  - Invented by LoopPay, bought by Samsung;
  - Now ported to Samsungpay;

Magnetic Card & MST

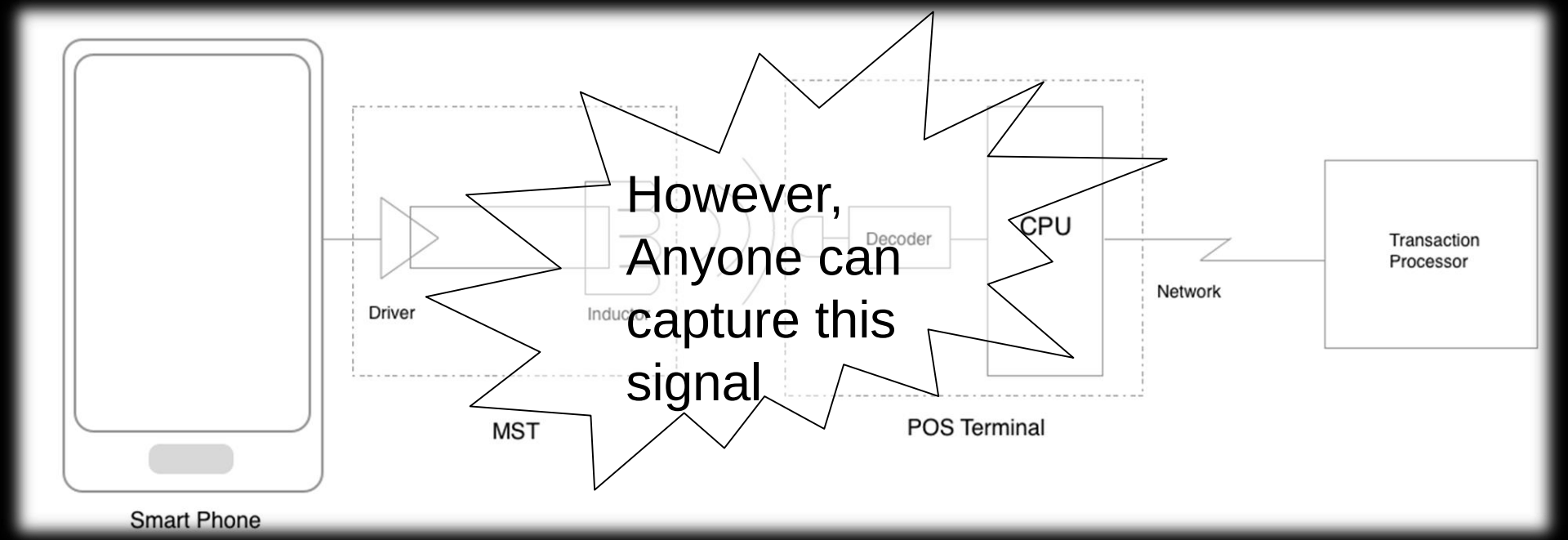
- Magnetic Card:
  - 

- MS
  - Use alternating magnetic field to transmit signal;
  - Invented by LoopPay, bought by Samsung;
  - Now ported to Samsungpay;

MST mechanism

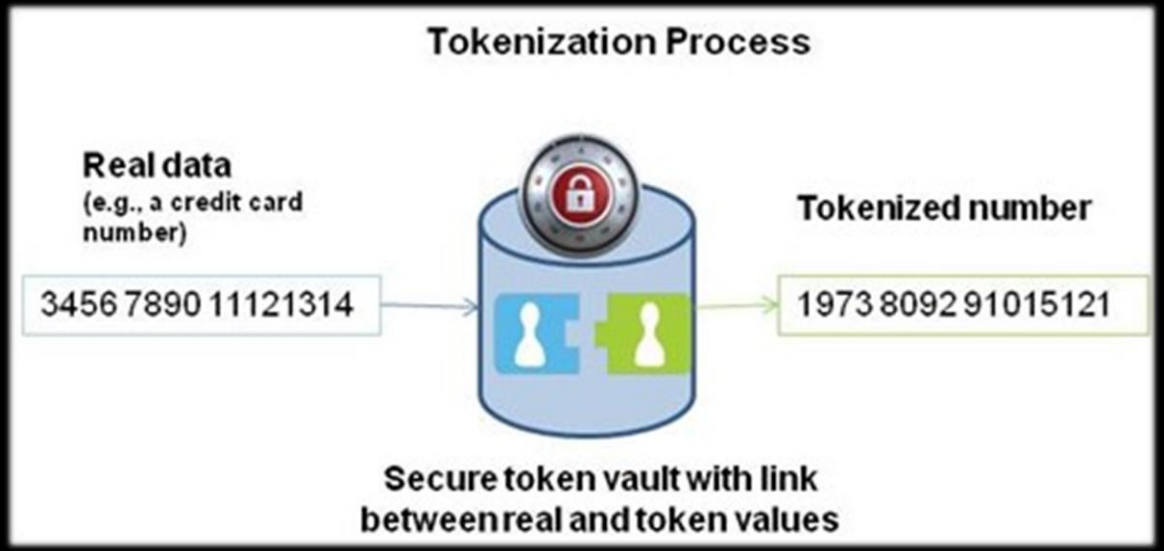


MST mechanism



Tokenization

- Reliable solution for processing sensitive information;
- Mathematically inreversible;
- NO Sensitive data leaked;
- But Where to store?



Secure Element

- Secure Element(SE) is a secure chip for securely hosting applications and their confidential and cryptographic data;
- SE has very high security level, and is the most essential part of mobile payment;
- Three types: UICC, MicroSD and Embedded SE;

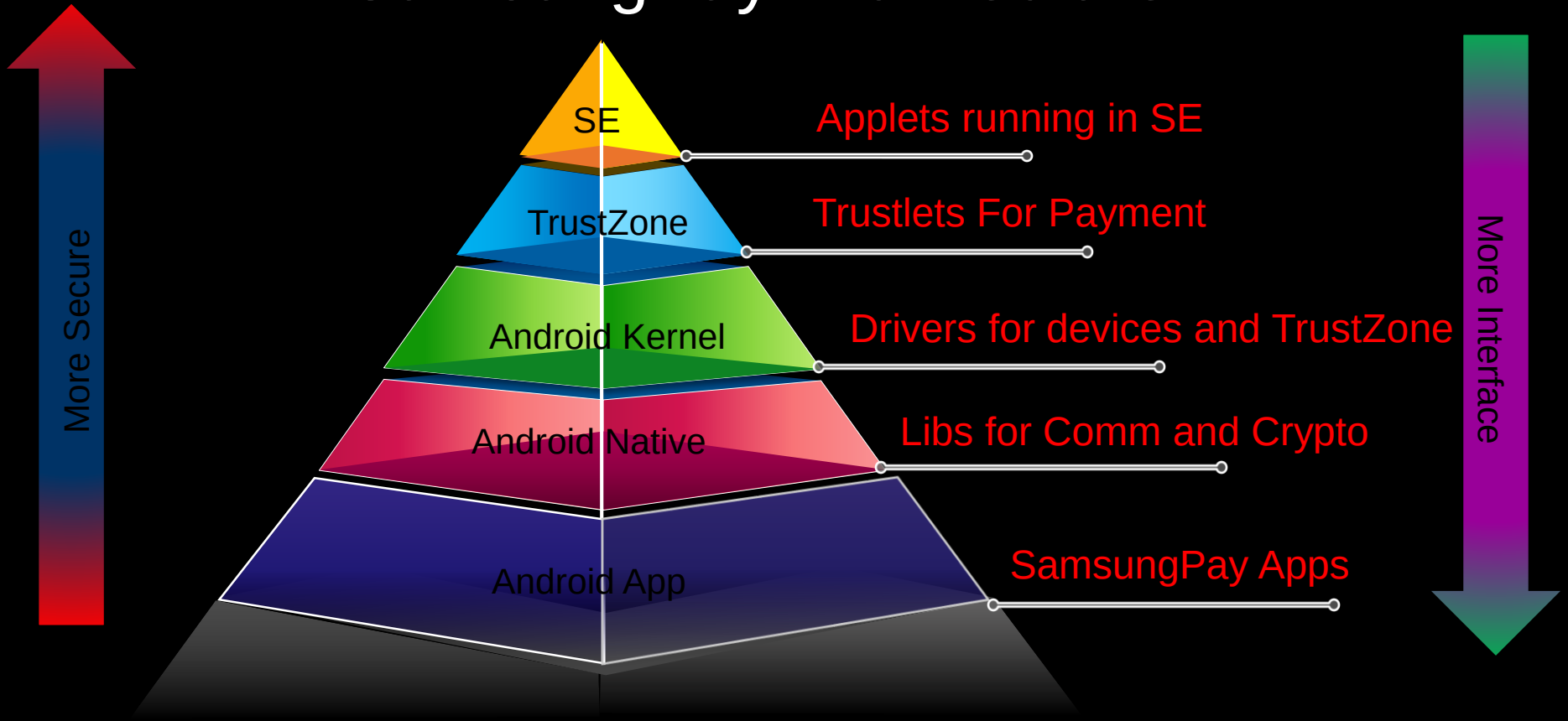
Secure Element

- Secure Element(SE) is a secure chip for securely hosting applications and their confidential and cryptographic data;
- SE has very high security level, and is the most essential part of mobile payment;
- Three types: UICC, MicroSD and Embedded SE;

Applet

- An OS resides in SE;
- Applet is an application running upon the OS, developed by Java;
- Compatible with JavaCard;
- Two methods required: install and process;
- Communicate with APDU;
- In CAP files forms;
- Confidential and cryptographic data for generating token also reside in SE;

SamsungPay Architecture



SamsungPayStub

- Pre-installed in official firmware released after 2016.03,located in /system/priv-app/SamsungPayStub;
- SamsungPay works fine without this;
- No payment function,just a stub;
- Download and install necessary App:
  - SamsungPay Main App;
  - SamsungPay Framework;
  - TSM Serivce App;

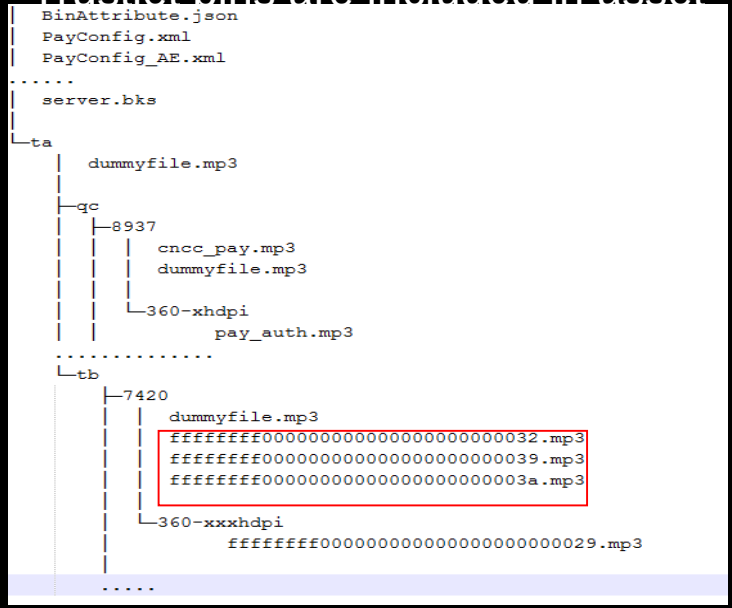
Main App & Framework

- Main App:
  - Update package for SamsungPayStub,shared the same package name;
  - Payment function,UI code and Card Management code included;
  - Save configuration in shared preferences:common_preferences.xml and prov_preferences.xml;
  - Save data in 8 SQLITE databases;
  - Most data encrypted by private algorithm (localefont);
- Framework:
  - Provide service for communicating with TrustZone;
  - Trustlet bins are included in asset directory;

Main App & Framework

- Main App:
  - Update package for SamsungPayStub,shared the same package name;
  - Payment function,UI code and Card Management code included;
  - Save configuration in shared preferences:common_preferences.xml and prov_preferences.xml;
  - Save data in 8 SQLITE databases;
  - Most data encrypted by private algorithm (localefont);

- Framework:
  - Provide service for communicating with TrustZone;
  - Trustlet bins are included in asset



TSM Service

- A bridge between Bank and SamsungPay;
- Different for different region, in China, Provided and signed by China UnionPay;
- Provide remote card management:
  - Enrollment
  - Download
  - Update
  - Revoke
  - Delete
- Main App call service exported by TSM to achieve card management;
- Communicate with Service Provider web server.

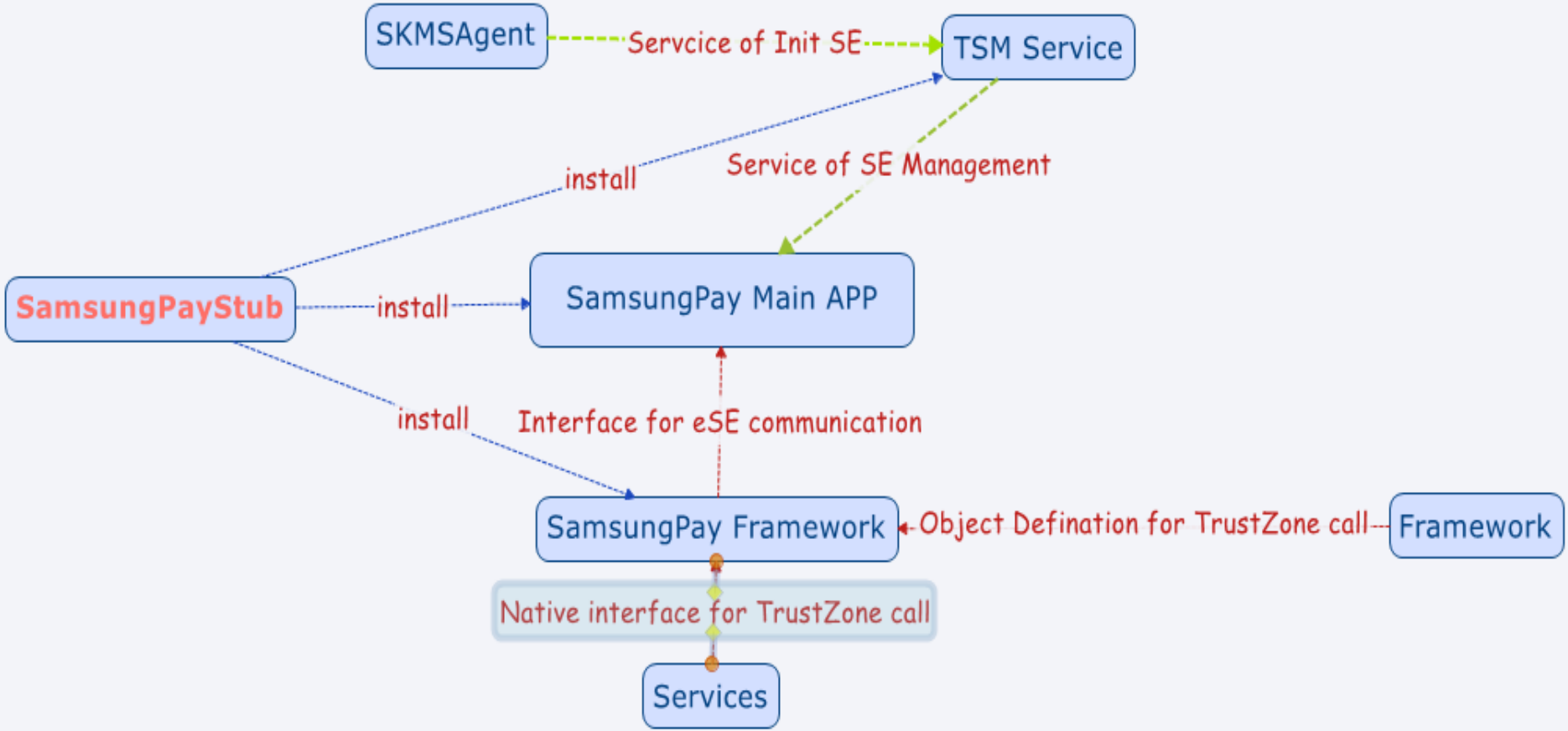
SKMS Agent

- Samsung Key Management Service Agent;
- Communicate with Samsung web server;
- Three versions:
  1. Pre-installed odex in /system/priv-app/SKMSAgent,obfuscated;
  2. dalvik-cache odex in /dalvik-cache/, clear code;
  3. Full apk Package bundled in some TSM install Package,obfuscated;
- Do SE initialization at very beginning phase;
- Collect SE information for every payment and registration;

Interface2Native

- Four methods for SamsungPay:
  - *nativeCreateTLCommunicationContext*
  - *nativeDestroyTLCommunicationContext*
  - *nativeProcessTACommand*
  - *nativeGenerateDeviceCertificates*

Android App


SKMSAgent
Servcice of Init SE
TSM Service
install
Service of SE Management
SamsungPayStub
install
SamsungPay Main APP
install
Interface for eSE communication
SamsungPay Framework
Object Defination for TrustZone call
Framework
Native interface for TrustZone call
Services

Android Native

- Few libs are involved in SamsungPay:
  - libandroid_servers.so -> wrapper for all native service;
  - libtlc_spay.so -> trustlet communication lib for samsungpay;
  - libtlc_direct_comm.so-> lower communication lib;
  - libMcClient.so -> MobiCore Client Lib;
- Daemon for communication:
  - mcDriverDaemon -> daemon for talking to driver, by read,write and ioctl;
- Device interfaces:
  - /dev/mobicore
  - /dev/mobicore-user

  } MobiCore Driver
  - /dev/mst_ctrl

  } mst_drv

Android Kernel

- Drivers related to SamsungPay:
  - MobiCore Driver -> Interface for Userland;
  - MobiCore Kernel Driver -> Talk to TrustZone;
  - mst_drv Driver -> Control MST Device;
- Source Code Available;

Android Kernel

- Drivers related to SamsungPay:
  - MobiCore Driver -> Interface for Userland;
  - MobiCore Kernel Driver -> Talk to TrustZone;
  - mst_drv Driver -> Control MST Device;
- Source Code Available;

Function	CmdID	Comments
turnonMST	1	Used
turnoffMST	0	
sendTrack1	2	Unused
sendTrack2	3	
sendTrack3	6	
sendTest	4	Used In Test APP
Escape	5	

TrustZone

- OS is closed-source, MobiCore, developed by Giesecke & Devrient;
- Trustlets run in it, with MCLF format;
- Signed but NOT encrypted;
- Different payment use different trustlets:
  - VISA,MASTER,UnionPay;
- Trustlet entry accepts two arguments: tci and its length;
  - tci points to WSM(World Shared Memory)
- After loaded, Trustlet does some initialization, then call tlApiWaitNotification api wait notification from normal world;
- Accept commands from normal world:*nativeProcessTACommand*

SE

- *Hardware:*
  - SmartMX2-P61 family;
  - Model: P61N1M3(maybe);
  - Integrated into NFC controller chip;
  - SmartMX2 CPU, 90nm CMOS;
  - ISA: Super Set of 80C51;
  - Fame2 crypto coprocessor for RSA/ECC;
  - SBC crypto coprocessor for DES/AES;

- *Hardware(cont.):*
  - 128KB E^2PROM,1.2MB Flash, 34KB RAM;
  - Five modes:
    - ~~Boot Mode;~~
    - ~~Test Mode;~~
    - ~~Firmware Mode;~~
    - System Mode;
    - User Mode;
  - SPI interface for connecting directly to SE;
  - EAL6+;

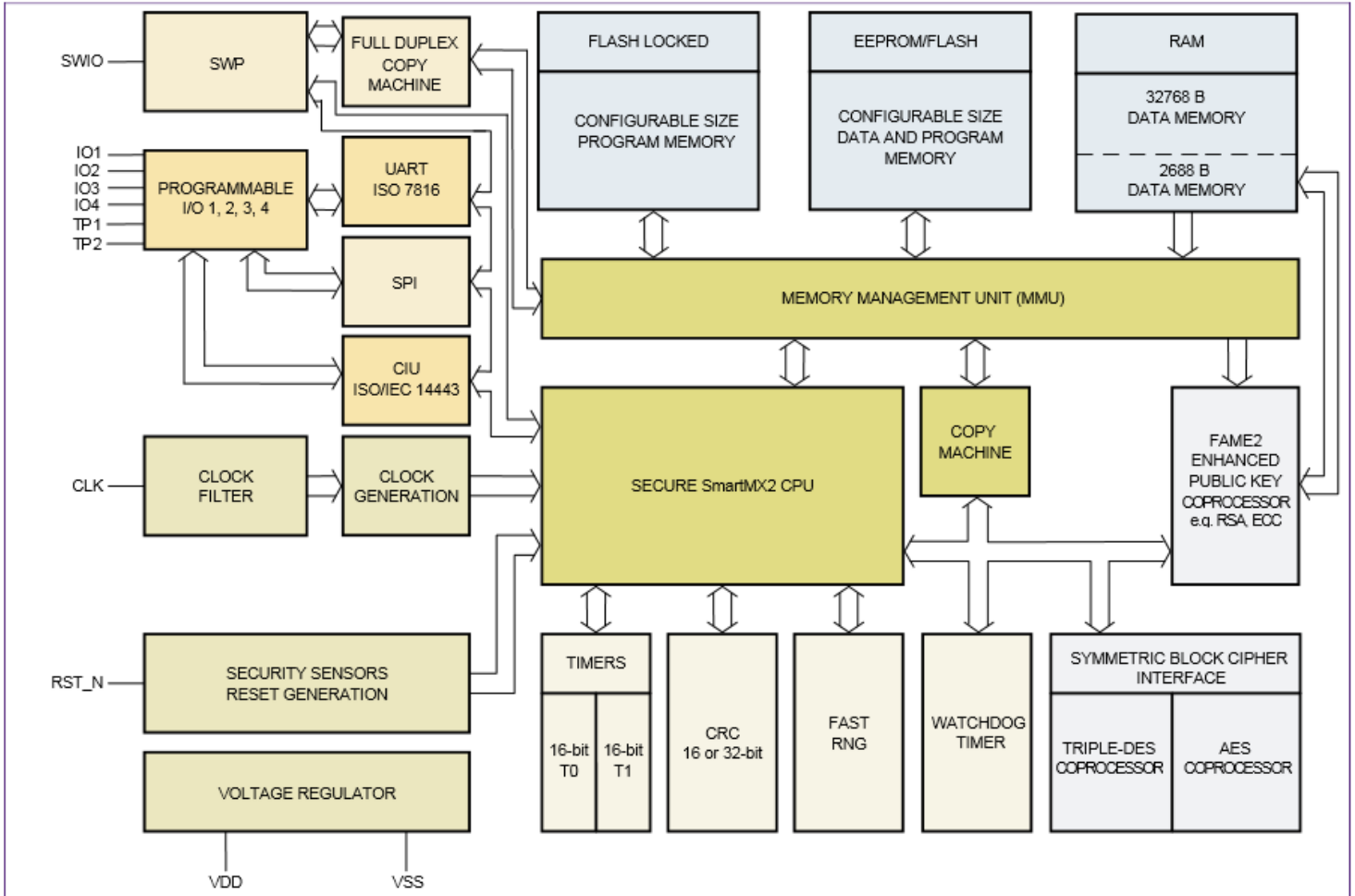


Fig 1. Block Diagram of P61N1M3PVD/VE

SE

- *Software:*
  - A Card OS inside, Regulated by
  - Java Card runtime;
  - Cryptographic and Hashing;
  - Security Domain;

- Global Platform API;
- Card Life Cycle Models;
- Secure Channel;

SE

- *Software:*
  - A Card OS inside, Regulated by
  - Java Card runtime;
  - Cryptographic and Hashing;
  - Security Domain;

- Isolated Environment for Running Applets and Storing Data(keys ,config data), like sandbox;
- Issuer Security Domain(ISD) own the top privilege(Samsung);
- Supplementary Security Domains(SSD) for Users, lower privilege;
- Cross Domains access is prohibited;

GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

- Global Platform API;
- Card Life Cycle Models;
- Secure Channel;

SE

- *Software:*
  - A Card OS inside, Regulated by
  - Java Card runtime;
  - Cryptographic and Hashing;
  - Security Domain;

- Isolated Environment for Running Applets and Storing Data(keys ,config data), like sandbox;
- Issuer Security Domain(ISD) own the top privilege(Samsung);
- Supplementary Security Domains(SSD) for Users, lower privilege;
- Cross Domains access is prohibited;

GLOBALPLATFORM®
THE STANDARD FOR SECURE DIGITAL SERVICES AND DEVICES

- Global Platform API;
- Card Life Cycle Models;
- Secure Channel;

- Built upon APDU;
- Negotiation and Authentication before doing any operation;
- Session Keys are negotiated for every connection;
- Traffic packets are encrypted by Session Keys;

In a word

- Many components in multi levels;
- Roughly 3 layers:
  - Android;
  - MobiCore(TrustZone);
  - Applets and OS in SE;
- We focus mostly on the latter two;

Steal Money from SamsungPay?!

Remote

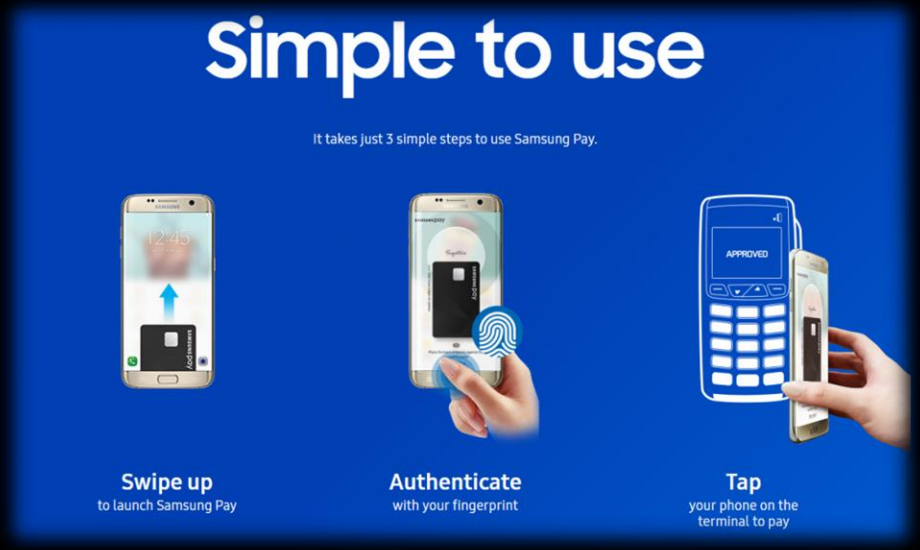
Local

Payment

Registration

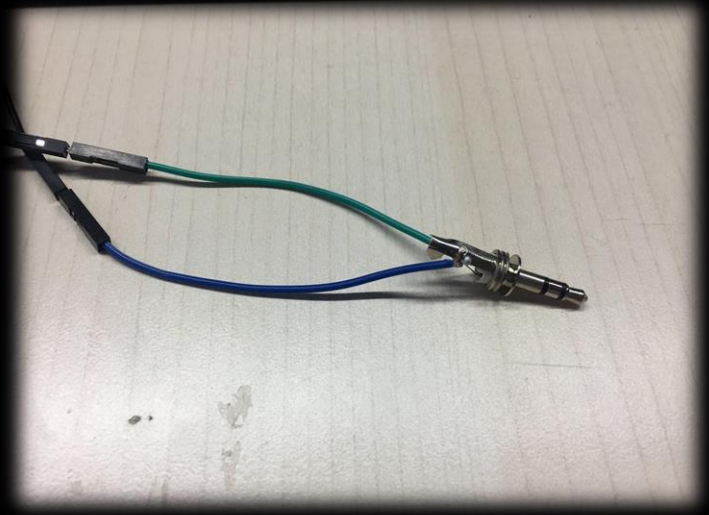
Payment-Basic

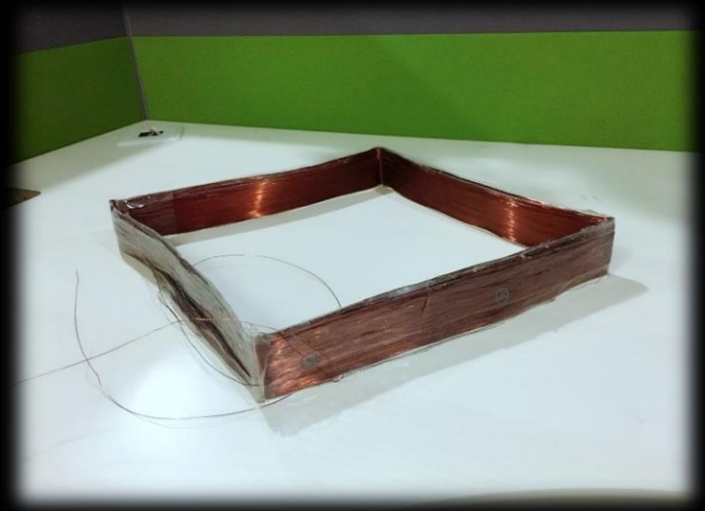
- Payment is the most frequently used feature;
- Step for using SamsungPay:
  - Select Card -> select one of virtual card you registered in SamsungPay
  - Authenticate -> password/fingerprint/iris
  - Tap on POS -> stay phone close to POS terminal;
- SamsungPay transmits NFC and MST signal at the same time;
- We focus on both hardware and software implementation of MST transaction;



Payment-Token Capture

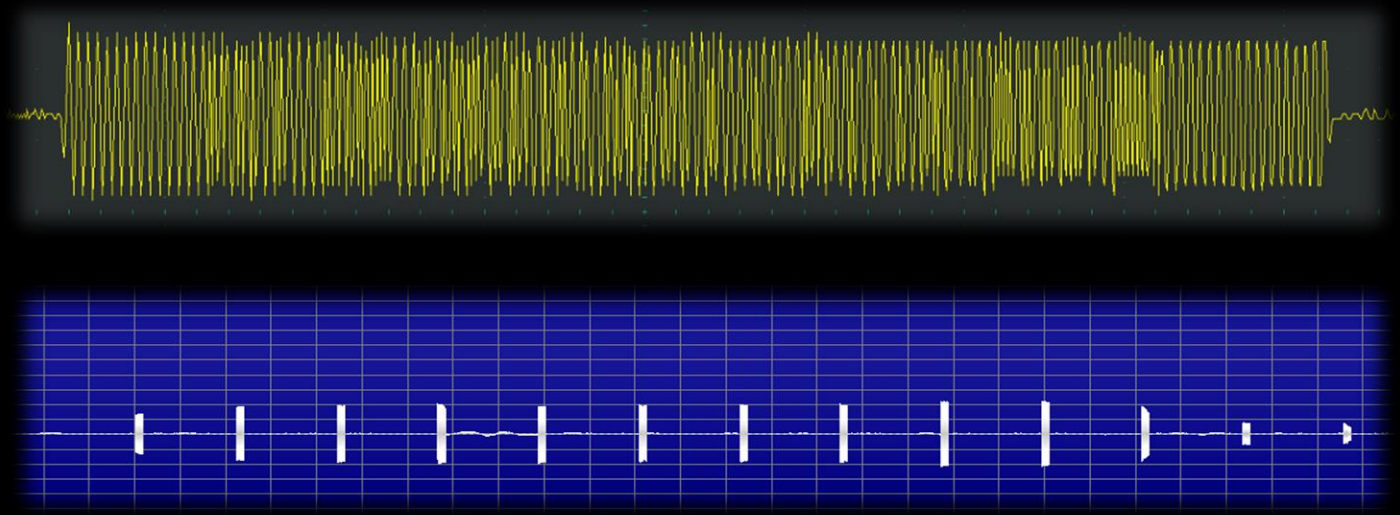
- MST signal can be captured by coil;
- The energy of this signal is high enough to be captured from a distance;
- Reported by 3 groups on BlackHat and USENIX;





Payment-Token Capture

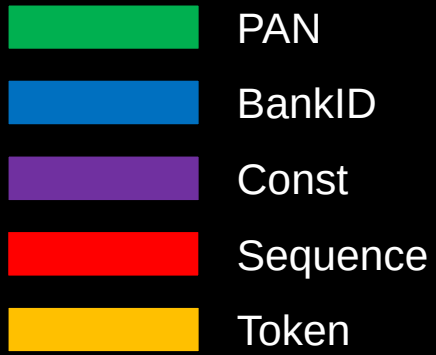
- Transmit Track2 Info Only;
- 30 times in 30s for each payment;



Payment-Token Analysis

- Different version was found in China;
- 6 digits token instead of 3(documented in BH USA 2015);
- No internet or cellular required while generating tokens;
- Synchronized by sequence number;

6230745372011888888=21021010051295089
6230745372011888888=21021010061045672
6230745372011888888=21021010071577380
6230745372011888888=21021010081608599
6230745372011888888=21021010091744699

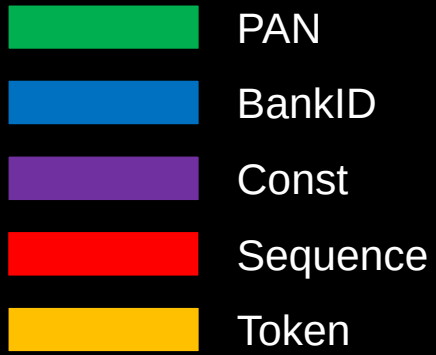


Payment-Token Analysis

- Different version was found in China;
- 6 digits token instead of 3(documented in BH USA 2015);
- No internet or cellular required while generating tokens;
- Synchronized by sequence number;

6230745372011888888=21021010051295089
6230745372011888888=21021010061045672
6230745372011888888=21021010071577380
6230745372011888888=21021010081608599
6230745372011888888=21021010091744699

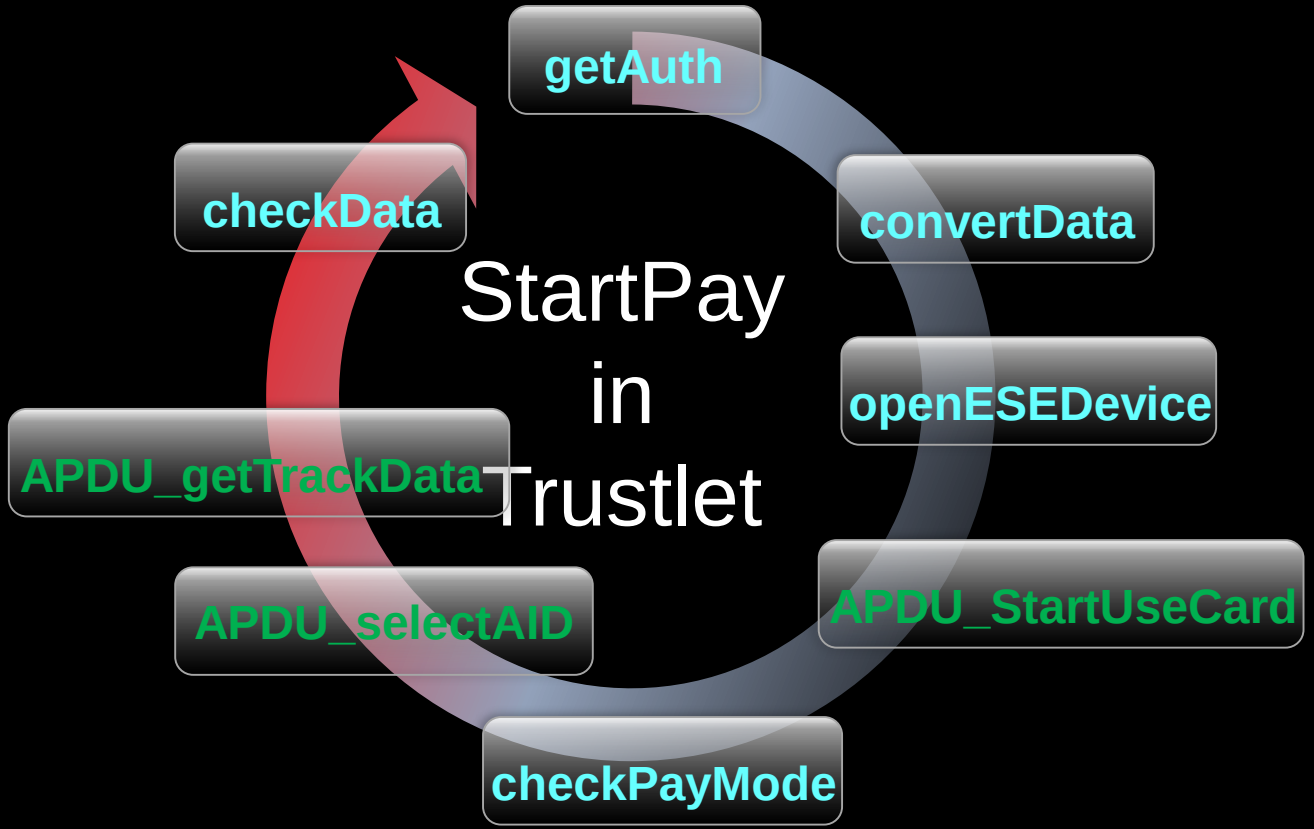
PRG + Seed ?



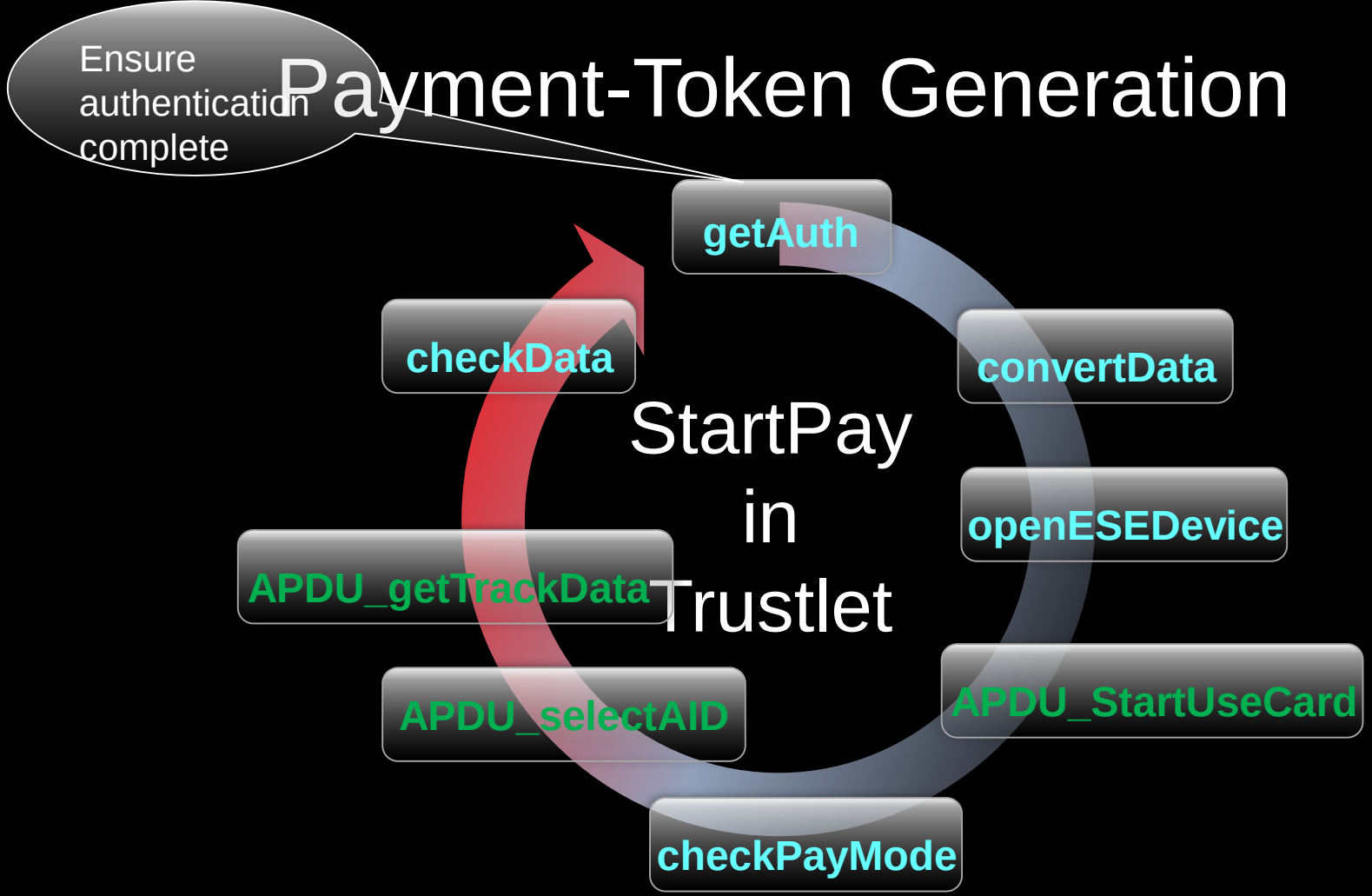
Payment-Token Generation

- Generating token securely is vital to mobile payment;
- Samsung uses layering model to minimize attacking surface;
- Most work are done in TrustZone and SE;
- Two procedures involved, and each accepts one argument from userland:
  - StartPay(AID)
  - transmitMSTData(ConfigData)

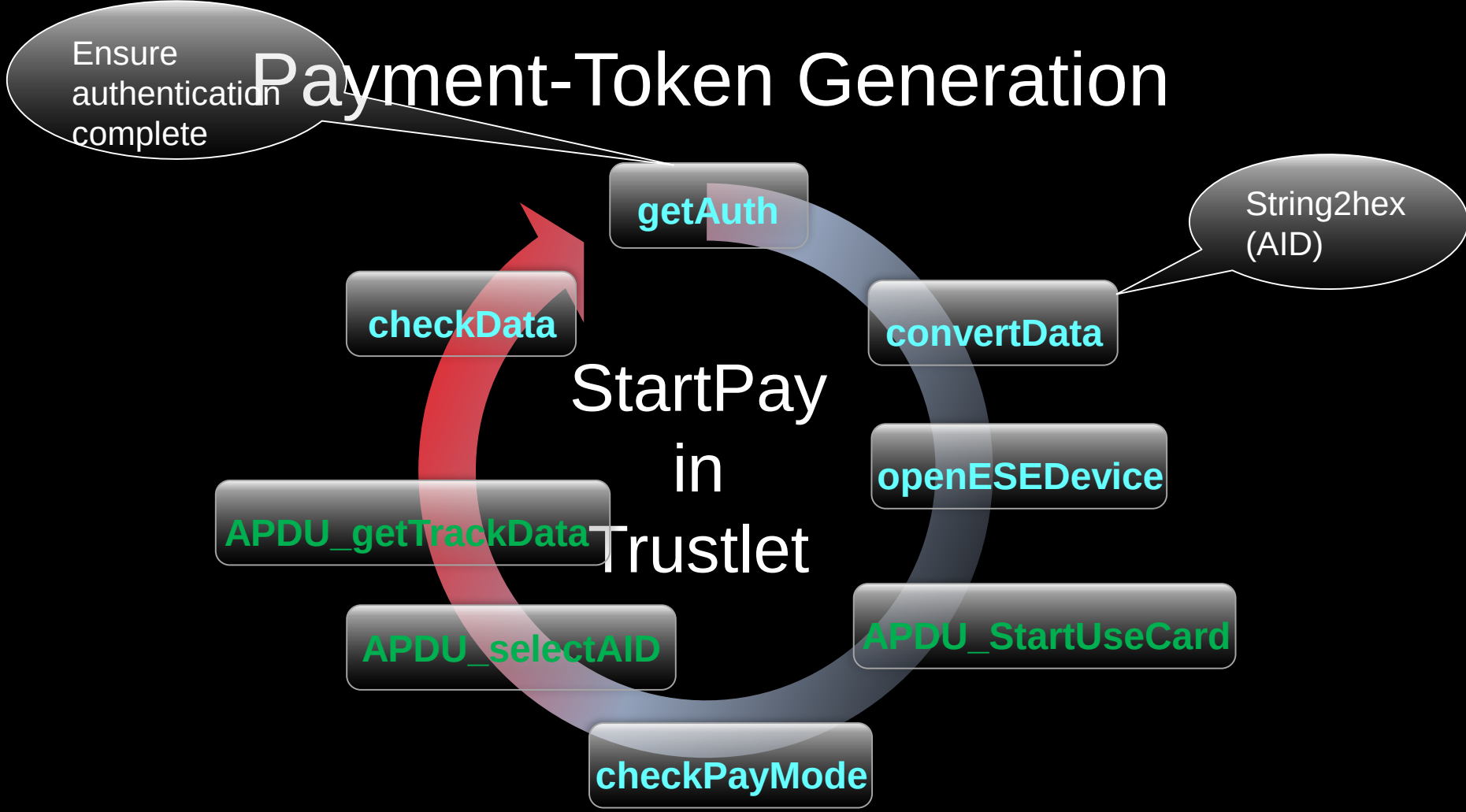
Payment-Token Generation



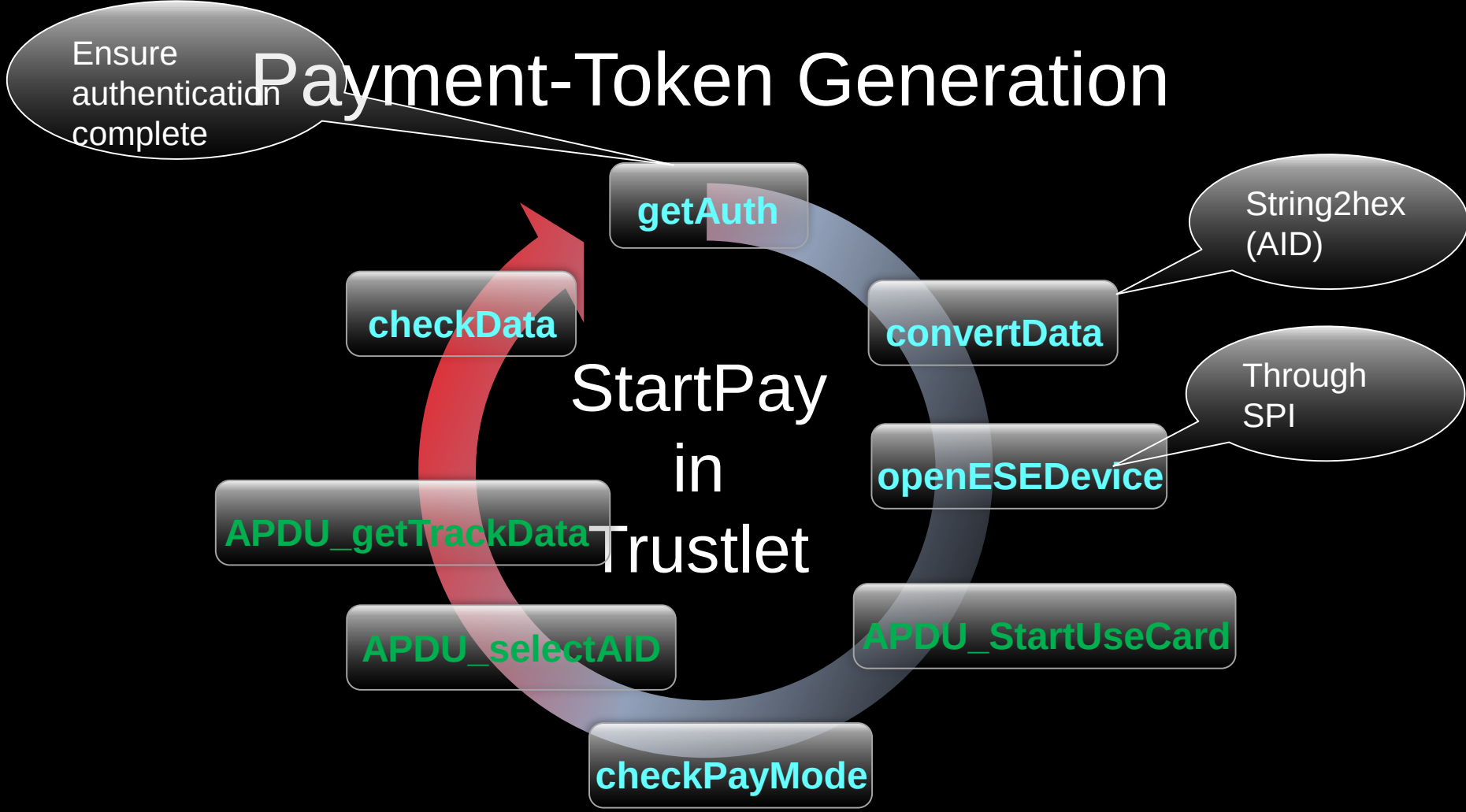
Payment-Token Generation



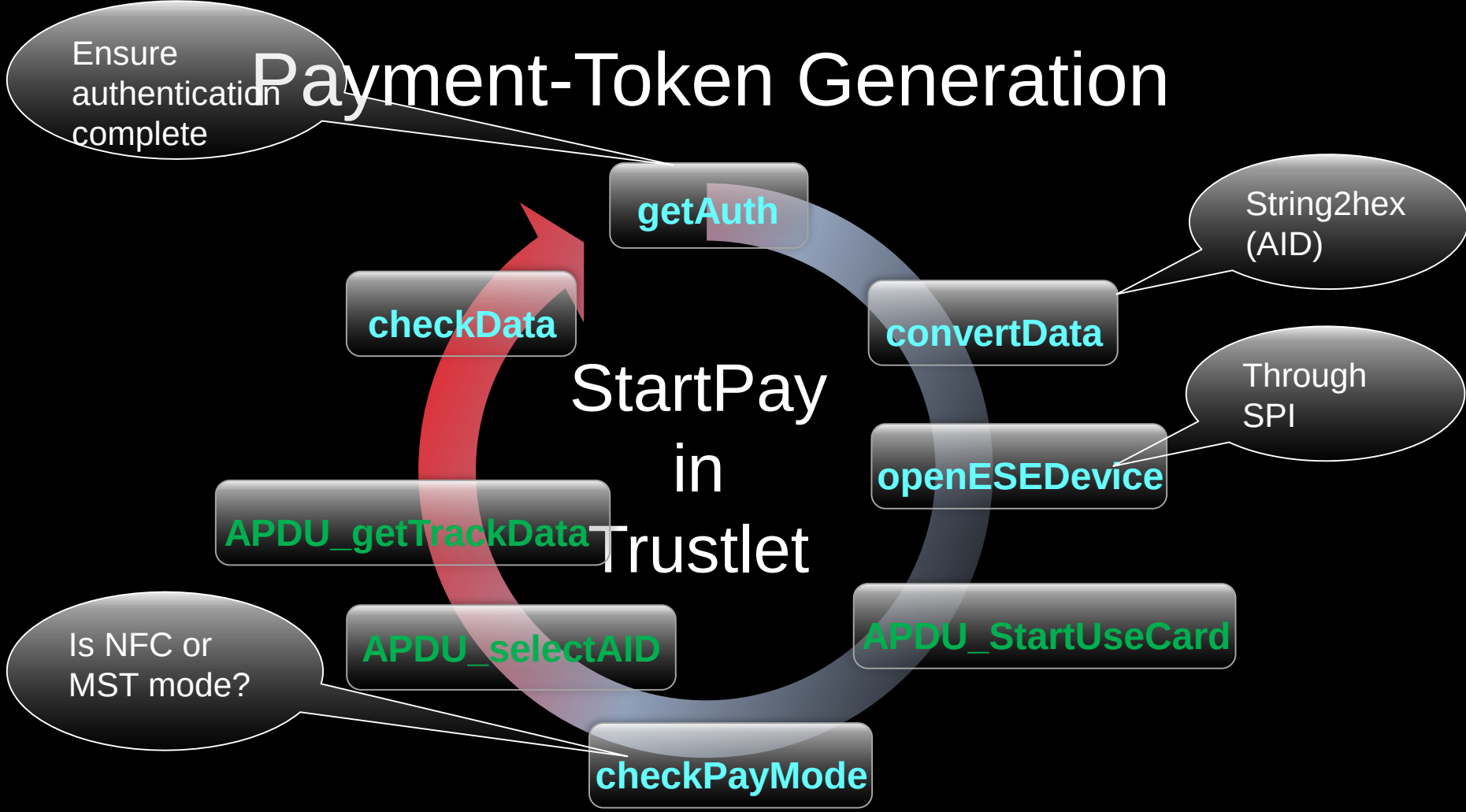
Payment-Token Generation



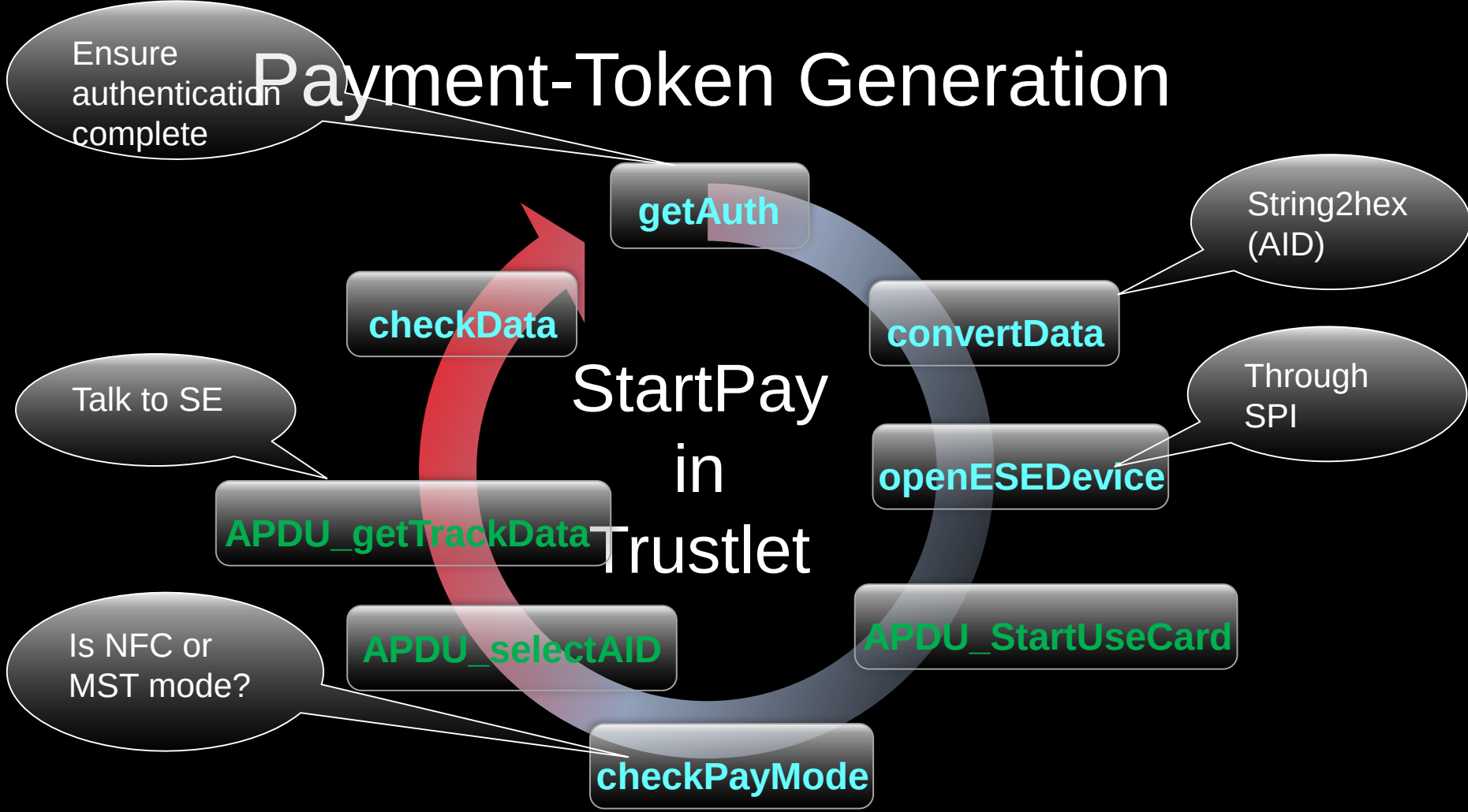
Payment-Token Generation



Payment-Token Generation

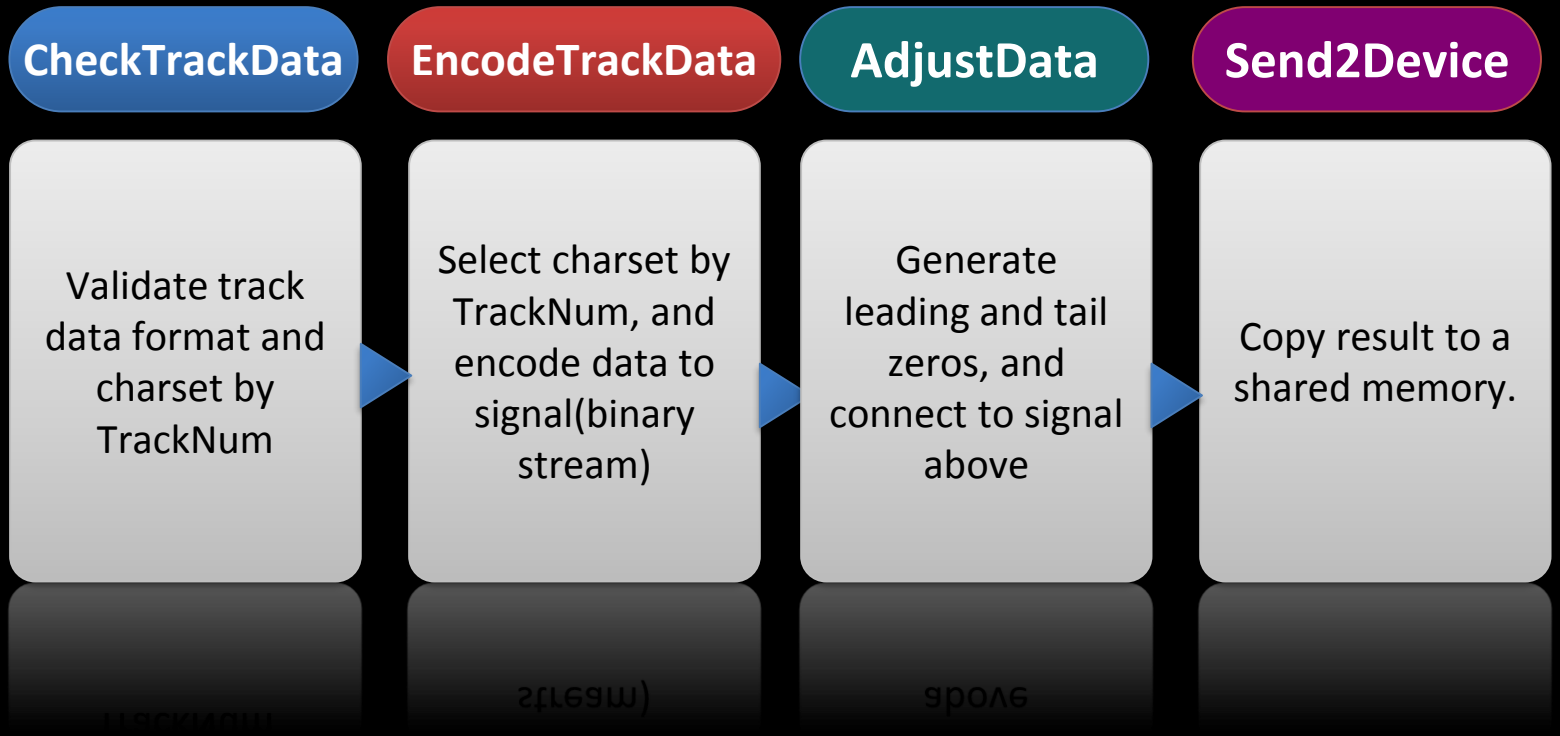


Payment-Token Generation



Payment-Token Generation

transmitMSTData(ConfigData)



Payment-Summary

- Token can be easily captured;
- Token is valid for transaction at that time;
- Invalid or expired if used;
- Synchronized by seqnum can be a problem;
- Algorithm is inside SE.

Payment-Summary

- Token can be easily captured;
- Token is valid for transaction at that time;
- Invalid or expired if used;
- Synchronized by seqnum can be a problem;
- Algorithm is inside SE.

Can we get the algorithm and generate valid token OFF the phone?

SAMSUNG pay

02 Card Registration

Choose the Samsung Pay icon

Register the card you want to use

Use the camera to read your card

* When the camera cannot read your card successfully, enter your card information manually.

Enter your card information

* When the camera reads your card, check and enter the rest of your card information.

Agree with the terms of use

Card Verification

* You can choose one of options to verify your card.

Enter code registration

Enter the PIN number registered

Complete your card registration

Registration-Code

- ✓ Environment check while launch;
- ✓ Highly relied on KNOX;
- ✓ Check server certificate while using SSL;
- ✓ Encrypt Packets while transaction;
- ✓ Obfuscate dalvik code;
- ✓ Check Signature in native lib;
- ✓ Obfuscate native algorithm work flow;

Registration-Code

- ✓ Environment check while launch;
- ✓ Highly relied on KNOX;
- ✓ Check server certificate while using SSL;
- ✓ Encrypt Packets while transaction;
- ✓ Obfuscate dalvik code;
- ✓ Check Signature in native lib;
- ✓ Obfuscate native algorithm work flow;

- x Log all actions into logcat;

Registration-Code

- ✓ Environment check while launch;
- ✓ Highly relied on KNOX;
- ✓ Check server certificate while using SSL;
- ✓ Encrypt Packets while transaction;
- ✓ Obfuscate dalvik code;
- ✓ Check Signature in native lib;
- ✓ Obfuscate native algorithm work flow;

- x Log all actions into logcat;

- x Even the decrypted https packets;

Registration-Code

- ✓ Environment check while launch;
- ✓ Highly relied on KNOX;
- ✓ Check server certificate while using SSL;
- ✓ Encrypt Packets while transaction;
- ✓ Obfuscate dalvik code;
- ✓ Check Signature in native lib;
- ✓ Obfuscate native algorithm work flow;

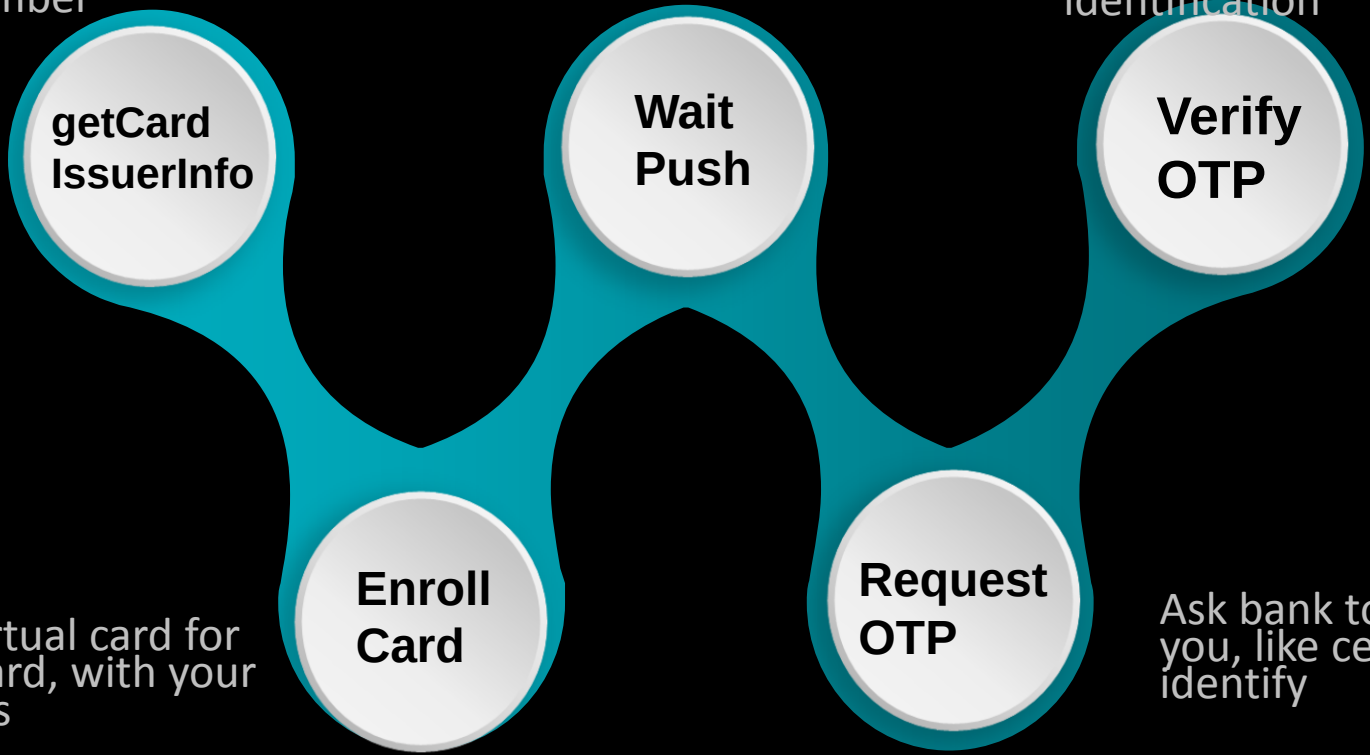
- x Log all actions into logcat;
- x Even the decrypted https packets;
- x Other information (Next Page);

Registration-Code

Collect Issuer info according to your card number

Wait for virtual card download info

Send the OTP back to bank to finish identification



Apply a virtual card for physical card, with your credentials

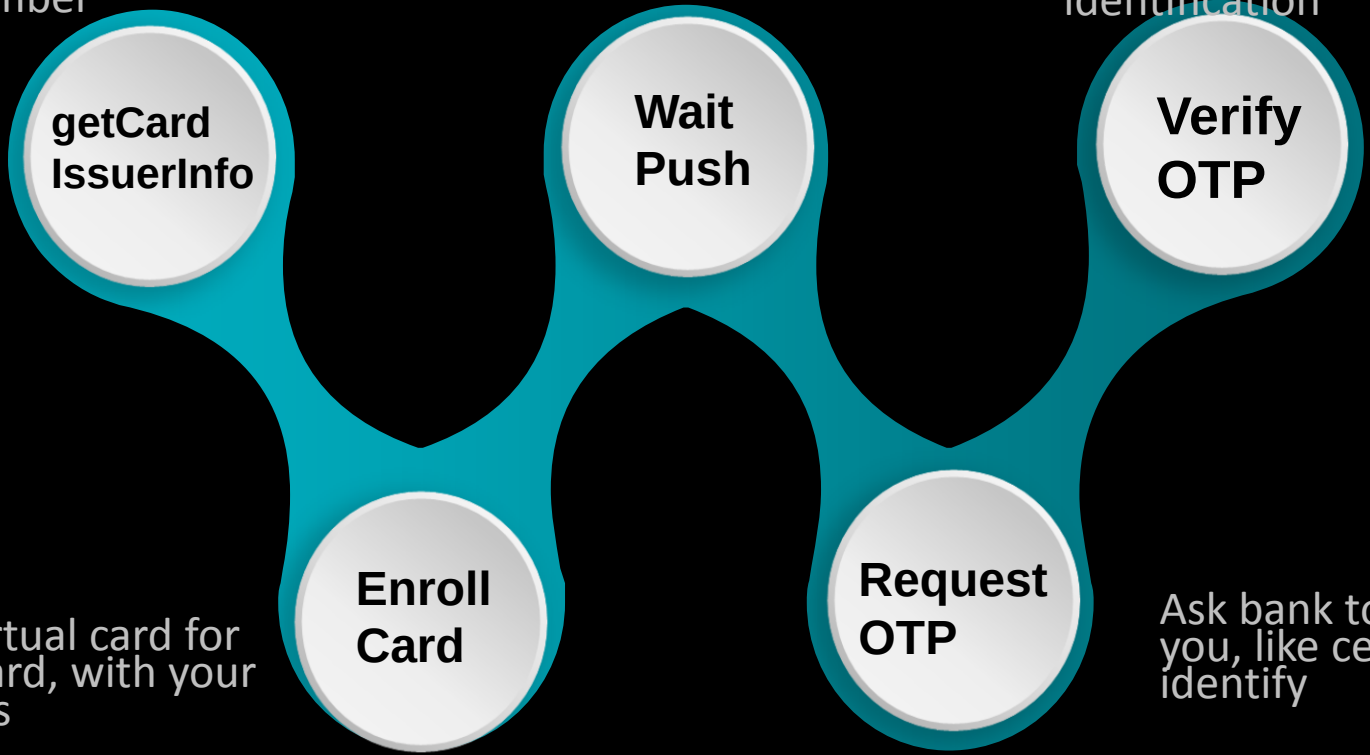
Ask bank to send OTP to you, like cellphone, to identify

Registration-Code

Collect Issuer info according to your card number

Wait for virtual card download info

Send the OTP back to bank to finish identification



Apply a virtual card for physical card, with your credentials

Ask bank to send OTP to you, like cellphone, to identify

Registration-Code

Collect Issuer info according to your card number

getCard IssuerInfo

Wait for virtual card download info

Wait Push

Send the OTP back to bank to finish identification

Verify OTP


```
"msgId":48122704803397632,
"timestamp":1480321175213,
"action":"tsmLib",
"data":{
    "tsmLibData":{
        "event":"DOWNLOAD",
        "sign":"sign",
        "ssid":"d35f4cb6-aa42-4e90-a7e3-a70e7dec6e45"
    },
    "tsmId":"CUP",
    "virtualCardIds":["0a9918c3aa1c428c879b63aaac69af8d"]
}
```


Apply a virtual card fo
physical card, with you
credentials

bank to send OTP to
like cellphone, to
tify

Registration-Code

Collect Issuer info according to your card number

getCard IssuerInfo

Wait for virtual card download info

Wait Push

Send the OTP back to bank to finish identification

Verify OTP


```
"msgId":48122704803397632,
"timestamp":1480321175213,
"action":"tsmLib",
"data":{
    "tsmLibData":{
        "event":"DOWNLOAD",
        "sign":"sign",
        "ssid":"d35f4cb6-aa42-4e90-a7e3-a70e7dec6e45"
    },
    "tsmId":"CUP",
    "virtualCardIds":["0a9918c3aa1c428c879b63aaac69af8d"]
}
```


Apply a virtual card fo
physical card, with you
credentials

bank to send OTP to
like cellphone, to
tify

Registration-Code

```
String[] pubkey = new String[1];
int ErrorCode=mSrv.getPubKey(1000,pubkey); //get Exchg PubKey
Log.i(TAG,"get public key with ErrorCode="+Integer.toString(ErrorCode)+" and PubKey is "+pubkey[0]);
Context ctx=this.getApplicationContext();
boolean err=IUPJniInterface.iJE(ctx); //libuptsmaddon.so initJniEnvironment
String SessionKey=IUPJniInterface.mSK();//makeSessionKey
String EncryptedKey=IUPJniInterface.rER(pubkey[0],SessionKey);//rsaEncryptor
Log.i(TAG,"Call mSK ret="+SessionKey+", Call rER ret="+EncryptedKey);
int xchg_ret=mSrv.exchangeKey(EncryptedKey,pubkey);//exchgkey, return data into pubkey。
Log.i(TAG,"exchangekey ret="+Integer.toString(xchg_ret)+"Return key is "+pubkey[0]);
String strl=IUPJniInterface.dMG(pubkey[0]);//decryptMsG
Log.i(TAG,"Call dMG ret="+strl);
IUPJniInterface.sSK(strl);
Log.i(TAG,"Call sSK");
IUPJniInterface.uSKT(fakePackname,strl);
Log.i(TAG,"Call uSKT");
try
{
    bret=IUPJniInterface.cSKV(fakePackname); //check SessionKey Valid
    Log.i(TAG,"Get flag again");
    if (bret==true)
    {
        Log.i(TAG,"Key Exchange succeed, Try to call init again!");
        int new_ret=mSrv.init(new InitRequestParams(),new myTSMCallback(this,0,0));
        .....+
    }
```

Init Connection

Registration-Code

```
ExecuteCmdRequestParams Paracmd=new ExecuteCmdRequestParams();
String encryptedSign=IUPJniInterface.eMG(sign);
Log.i(TAG,"encryptedSign ret= "+encryptedSign);
Paracmd.setSign(encryptedSign);
String encryptedSsid=IUPJniInterface.eMG(ssid);
Log.i(TAG,"encryptedSsid ret= "+encryptedSsid);
Paracmd.setSsid(encryptedSsid);
String encryptedReserved=IUPJniInterface.eMG("");
Log.i(TAG,"encryptedReserved ret= "+encryptedReserved);
Paracmd.setReserve(encryptedReserved);
int ret=mSrv.executeCmd(Paracmd,new myTSMCallback(this,0,0),null); //do command
Log.i(TAG,"call executeCmd ret="+Integer.toString(ret));
```

Data from Push Msg

Registration-Download

SE Initialization

- **Initial only ONCE, at the first time of use;**
- **Done by SKMS(Samsung) and TSM(Bank);**
- **New Supplementary Security Domain(SSD) Created;**

Virtual Card Applet Download

- **Download and Install Applet of Virtual Card;**
- **Store corresponding data to SE;**
- **Belong to New SSD;**
- **While Activated, the applet can represent your physical bank card;**

Registration-Download

SE Initialization

- **Initial only ONCE, at the first time of use;**
- **Done by SKMS(Samsung) and TSM(Bank);**
- **New Supplementary Security Domain(SSD) Created;**

•Whole process are protected by session key and SSL

Virtual Card Applet Download

- **Download and Install Applet of Virtual Card;**
- **Store corresponding data to SE;**
- **Belong to New SSD;**
- **While Activated, the applet can represent your physical bank card;**

Registration-Tips

Registration-Tips

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

④ Instead of cracking SSL, we have to probe the internals;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

④ Instead of cracking SSL, we have to probe the internals;

Thus a secure root is must

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

④ Instead of cracking SSL, we have to probe the internals;

Thus a secure root is must

① SamsungPay is launched with Android 6.0.1;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

④ Instead of cracking SSL, we have to probe the internals;

Thus a secure root is must

① SamsungPay is launched with Android 6.0.1;

② However SamsungPay works fine on 5.1.1;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

④ Instead of cracking SSL, we have to probe the internals;

Thus a secure root is must

① SamsungPay is launched with Android 6.0.1;

② However SamsungPay works fine on 5.1.1;

③ Android 5.1.1 is vulnerable to some root tools;

Registration-Tips

② To learn more, packets should be decrypted;

① Traffic packets for both process are encrypted by random session key, and transferred through SSL;

③ MITM for SSL does not work;

④ Instead of cracking SSL, we have to probe the internals;

Thus a secure root is must

④ Root privilege can be gained temporarily;

③ Android 5.1.1 is vulnerable to some root tools;

① SamsungPay is launched with Android 6.0.1;

② However SamsungPay works fine on 5.1.1;

Registration-Code


```
HttpEntity v0_10;
Bitmap v0_9;
Header v1_3;
String v0_8;
HttpResponse v1_2;
HttpResponse v8;
Object v0_3;
String v1_1;
HttpPost v2;
StringEntity v0_1;
UPTSMlog.a("sendMessage:" + arg9 + ", " + arg11 + ", " + arg10);
if(TextUtils.isEmpty(((CharSequence)arg9))) {
    throw new IOException();
}
```

```
HttpEntity v16 = v17.getEntity();
if(v16 != null) {
    v18 = EntityUtils.toString(v16);
    AgentLog.d("requestHttpPost response : " + v18);
}
else {
}
```

Registration-Trick1

TSMService

```
private static final void a(int arg7, String arg8, String arg9, Throwable arg10) {
    Class v1_2;
    String v0_1;
    if(crashLogUtil.d) {
        switch(arg7) {
            case 2: {
                goto label_8;
            }
            case 3: {
                goto label_10;
            }
            case 4: {
                goto label_12;
            }
            case 5: {
                goto label_14;
            }
            case 6: {
                goto label_16;
            }
        }

        goto label_3;
    label_8:
        Log.v(arg8, arg9, arg10);
        goto label_3;
    label_12:
        Log.i(arg8, arg9, arg10);
        goto label_3;
    label_10:
        Log.d(arg8, arg9, arg10);
```

```
crashLogUtil.init(((Context)this), WalletAppConfig.falseflag);
```

```
public static void init(Context arg1, boolean arg2) {
    crashLogUtil.c = arg2;
    crashLogUtil.d = arg2;
    crashLogUtil.workFolder = UPTsmUtils.getWorkFolder(arg1, 1);
}
```

```
WalletAppConfig.falseflag = TSMServiceJniInterface.tsmservice_jni_iDM();
```

Registration-Trick1

TSMService

```
private static final void a(int arg7, String arg8, String arg9, Throwable arg10) {
    Class v1_2;
    String v0_1;
    if(crashLogUtil.d) {
        switch(arg7) {
            case 2: {
                goto label_8;
            }
            case 3: {
                goto label_10;
            }
            case 4: {
                goto label_12;
            }
            case 5: {
                goto label_14;
            }
            case 6: {
                goto label_16;
            }
        }

        goto label_3;
    label_8:
        Log.v(arg8, arg9, arg10);
        goto label_3;
    label_12:
        Log.i(arg8, arg9, arg10);
        goto label_3;
    label_10:
        Log.d(arg8, arg9, arg10);
```

```
crashLogUtil.init(((Context)this), WalletAppConfig.falseflag);
```

```
public static void init(Context arg1, boolean arg2) {
    crashLogUtil.c = arg2;
    crashLogUtil.d = arg2;
    crashLogUtil.workFolder = UPTsmUtils.getWorkFolder(arg1, 1);
}
```

```
WalletAppConfig.falseflag = TSMServiceJniInterface.tsmservice_jni_iDM();
```

```
.text:DF44A02C iDM                                  ; DATA XREF: .data:DF4C0324↓o
.text:DF44A02C                  MOVS    R0, #0
.text:DF44A02E                  BX      LR
```

Registration-Trick1

TSMService

```
private static final void a(int arg7, String arg8, String arg9, Throwable arg10) {
    Class v1_2;
    String v0_1;
    if(crashLogUtil.d) {
        switch(arg7) {
            case 2: {
                goto label_8;
            }
            case 3: {
                goto label_10;
            }
            case 4: {
                goto label_12;
            }
            case 5: {
                goto label_14;
            }
            case 6: {
                goto label_16;
            }
        }

        goto label_3;
    label_8:
        Log.v(arg8, arg9, arg10);
        goto label_3;
    label_12:
        Log.i(arg8, arg9, arg10);
        goto label_3;
    label_10:
        Log.d(arg8, arg9, arg10);
```

```
crashLogUtil.init(((Context)this), WalletAppConfig.falseflag);
```

```
public static void init(Context arg1, boolean arg2) {
    crashLogUtil.c = arg2;
    crashLogUtil.d = arg2;
    crashLogUtil.workFolder = UPTsmUtils.getWorkFolder(arg1, 1);
}
```

```
WalletAppConfig.falseflag = TSMServiceJniInterface.tsmservice_jni_iDM();
```

```
.text:DF44A02C iDM                                  ; DATA XREF: .data:DF4C0324↓o
.text:DF44A02C                  MOVS    R0, #0
.text:DF44A02E                  BX      LR
```

Jni_iDM=Jni_isDebugMode

Registration-Trick2

SKMS Agent

```
public static void d(String arg1) {
    if((AgentLog.DBG) && !AgentLog.IS_LEVEL_LOW) {
        Log.d("SKMSAgent", arg1);
    }
}
```

```
boolean v0 = Debug.isProductShip() != 1 ? true : false;
AgentLog.DBG = v0;
```

libandroid_runtime .so-> isProductShipNative()

```
if(AgentLog.getDebugLevel() != 0) {
    v1 = false;
}

AgentLog.IS_LEVEL_LOW = v1;
```

```
public static int getDebugLevel() {
    int v4 = 2;
    int v3 = 0;
    String v2 = SystemProperties.get("ro.debug_level", "Unknown");
    Log.i("SKMSAgent", "DBG: " + AgentLog.DBG);
    Log.i("SKMSAgent", "IS_LEVEL_LOW: " + AgentLog.IS_LEVEL_LOW);
    if(!v2.equals("Unknown")) {
        int v5 = 2;
        try {
            int v0 = Integer.parseInt(v2.substring(v5), 16);
            if(v0 == 20300) {
                return v3;
            }
        }
        catch(NumberFormatException v1) {
            return v3;
        }

        if(v0 == 18765) {
            return 1;
        }

        if(v0 == 18760) {
            v3 = v4;
        }
    }

    return v3;
}
```

```
root@zenltechn:/ # getprop | grep ro.debug_level
[ro.debug_level]: [0x4f4c]
root@zenltechn:/ #
```

Registration-Trick2

SKMS Agent

```
public static void d(String arg1) {
    if((AgentLog.DBG) && !AgentLog.IS_LEVEL_LOW) {
        Log.d("SKMSAgent", arg1);
    }
}
```

```
boolean v0 = Debug.isProductShip() != 1 ? true : false;
AgentLog.DBG = v0;
```

libandroid_runtime .so-> isProductShipNative()

```
if(AgentLog.getDebugLevel() != 0) {
    v1 = false;
}

AgentLog.IS_LEVEL_LOW = v1;
```

```
public static int getDebugLevel() {
    int v4 = 2;
    int v3 = 0;
    String v2 = SystemProperties.get("ro.debug_level", "Unknown");
    Log.i("SKMSAgent", "DBG: " + AgentLog.DBG);
    Log.i("SKMSAgent", "IS_LEVEL_LOW: " + AgentLog.IS_LEVEL_LOW);
    if(!v2.equals("Unknown")) {
        int v5 = 2;
        try {
            int v0 = Integer.parseInt(v2.substring(v5), 16);
            if(v0 == 20300) {
                return v3;
            }
        }
        catch(NumberFormatException v1) {
            return v3;
        }

        if(v0 == 18765) {
            return 1;
        }

        if(v0 == 18760) {
            v3 = v4;
        }
    }

    return v3;
}
```

```
root@zenltechn:/ # getprop | grep ro.debug_level
[ro.debug_level]: [0x4f4c]
root@zenltechn:/ #
```

unmodifiable

Registration-Trick2

SKMS Agent

```
public static void [illegible](String arg1) {
    if((AgentLog.DBG) && !AgentLog.IS_LEVEL_LOW) {
        Log.d("SKMSAgent", arg1);
    }
}
```

```
boolean v0 = Debug.isProductShip() != 1 ? true : false;
AgentLog.DBG = v0;
```

libandroid_runtime .so-> isProductShipNative()

```
if(AgentLog.getDebugLevel() != 0) {
    v1 = false;
}

AgentLog.IS_LEVEL_LOW = v1;
```

```
public static int getDebugLevel() {
    int v4 = 2;
    int v3 = 0;
    String v2 = SystemProperties.get("ro.debug_level", "Unknown");
    Log.i("SKMSAgent", "DBG: " + AgentLog.DBG);
    Log.i("SKMSAgent", "IS_LEVEL_LOW: " + AgentLog.IS_LEVEL_LOW);
    if(!v2.equals("Unknown")) {
        int v5 = 2;
        try {
            int v0 = Integer.parseInt(v2.substring(v5), 16);
            if(v0 == 20300) {
                return v3;
            }
        }
        catch(NumberFormatException v1) {
            return v3;
        }

        if(v0 == 18765) {
            return 1;
        }

        if(v0 == 18760) {
            v3 = v4;
        }
    }

    return v3;
}
```

Return 0

```
root@zenltechn:/ # getprop | grep ro.debug_level
[ro.debug_level]: [0x4f4c]
root@zenltechn:/ #
```

unmodifiable

Registration-Trick2

Registration-Trick2

- SKMS Agent is a pre-installed app, Only odex exsit;

Registration-Trick2

- SKMS Agent is a pre-installed app, Only odex exsit;
- System will execute the native code in odex file instead of ~~dalvik~~ code;

Registration-Trick2

- SKMS Agent is a pre-installed app, Only odex exsit;
- System will execute the native code in odex file instead of ~~dalvik~~ code;
- Let’s modify native code directly;

Registration-Trick2

- SKMS Agent is a pre-installed app, Only odex exsit;
- System will execute the native code in odex file instead of ~~dalvik~~ code;
- Let's modify native code directly;

```
0x00033f94: d10103ff  sub sp, sp, #0x40 (64)
0x00033f98: a90257f4  stp x20, x21, [sp, #32]
0x00033f9c: a9037bf6  stp x22, x30, [sp, #48]
0x00033fa0: aa0003f5  mov x21, x0
0x00033fa4: b90003e0  str w0, [sp]
0x00033fa8: aa0103f6  mov x22, x1
0x00033fac: b9400aa0  ldr w0, [x21, #8]
0x00033fb0: b9419c14  ldr w20, [x0, #412]
0x00033fb4: 34000214  cbz w20, #+0x40 (addr 0xffa579b0) --- patch nop //D503201F
0x00033fb8: b9400aa1  ldr w1, [x21, #8]
0x00033fbc: b941a034  ldr w20, [x1, #416]
0x00033fc0: 350001b4  cbnz w20, #+0x34 (addr 0xffa579a4) --- patch nop//D503201F
0x00033fc4: b9400aa0  ldr w0, [x21, #8]
0x00033fc8: b9401400  ldr w0, [x0, #20]
0x00033fcc: b9472400  ldr w0, [x0, #1828]
0x00033fd0: b40001e0  cbz x0, #+0x3c (addr 0xffa579ac)
0x00033fd4: aa0003f4  mov x20, x0
0x00033fd8: aa1403e1  mov x1, x20
0x00033fdc: aa1503e0  mov x0, x21
0x00033fe0: b9400c00  ldr w0, [x0, #12]
0x00033fe4: aa1603e2  mov x2, x22
0x00033fe8: b9415400  ldr w0, [x0, #340]
0x00033fec: f9401c1e  ldr x30, [x0, #56]
0x00033ff0: d63f03c0  blr x30
```

Registration-Trick2

Registration-Trick2

- Dm-verity is enabled, we can’t change files on System partition;

Registration-Trick2

- Dm-verity is enabled, we can’t change files on System partition;

- Files in dalvik-cache are also odex file;

Registration-Trick2

- Dm-verity is enabled, we can’t change files on System partition;

- Files in dalvik-cache are also odex file;

- System will load dalvik-cache if odex not exist in app dir;

Registration-Trick2

- Dm-verity is enabled, we can’t change files on System partition;

- Files in dalvik-cache are also odex file;

- System will load dalvik-cache if odex not exist in app dir;

- Remove odex will NOT trigger dm-verity;

Registration-Trick2

- Dm-verity is enabled, we can’t change files on System partition;

- Files in dalvik-cache are also odex file;

- System will load dalvik-cache if odex not exist in app dir;

- Remove odex will NOT trigger dm-verity;

- NO integrity check for native code;

Registration-strategy

- Enable packets log strategy:
  - Modify odex native code;
  - Rename to system@priv-app@SKMSAgent@SKMSAgent.apk@classes.dex
  - Write to dalvik-cache directory;
  - Remove original odex file under root privilege;
  - Patch Applied!

Registration-SE Operations

7 Steps of Registration

SE Initialization

–Create Supplementary Security Domain;

–Update Supplementary Security Domain keys;

–Install ARC-C Application;

–Personalize AMSD and Write SEID;

–Add Access Rules for CRS;

–Install CARDS Applet;

Applet Download

–Install Applet;

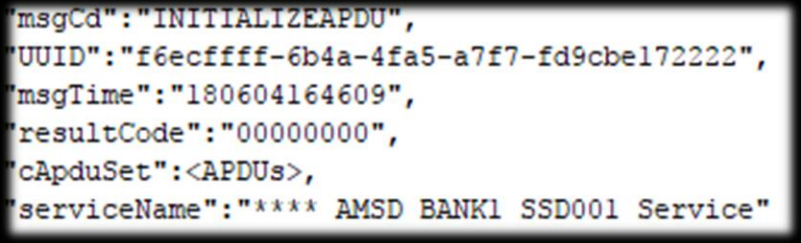
SKMS Agent

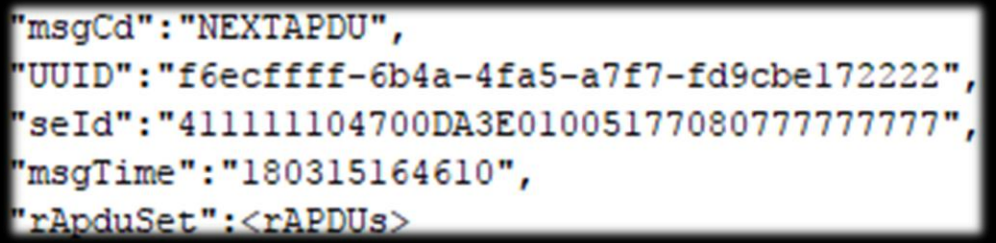
TSMService

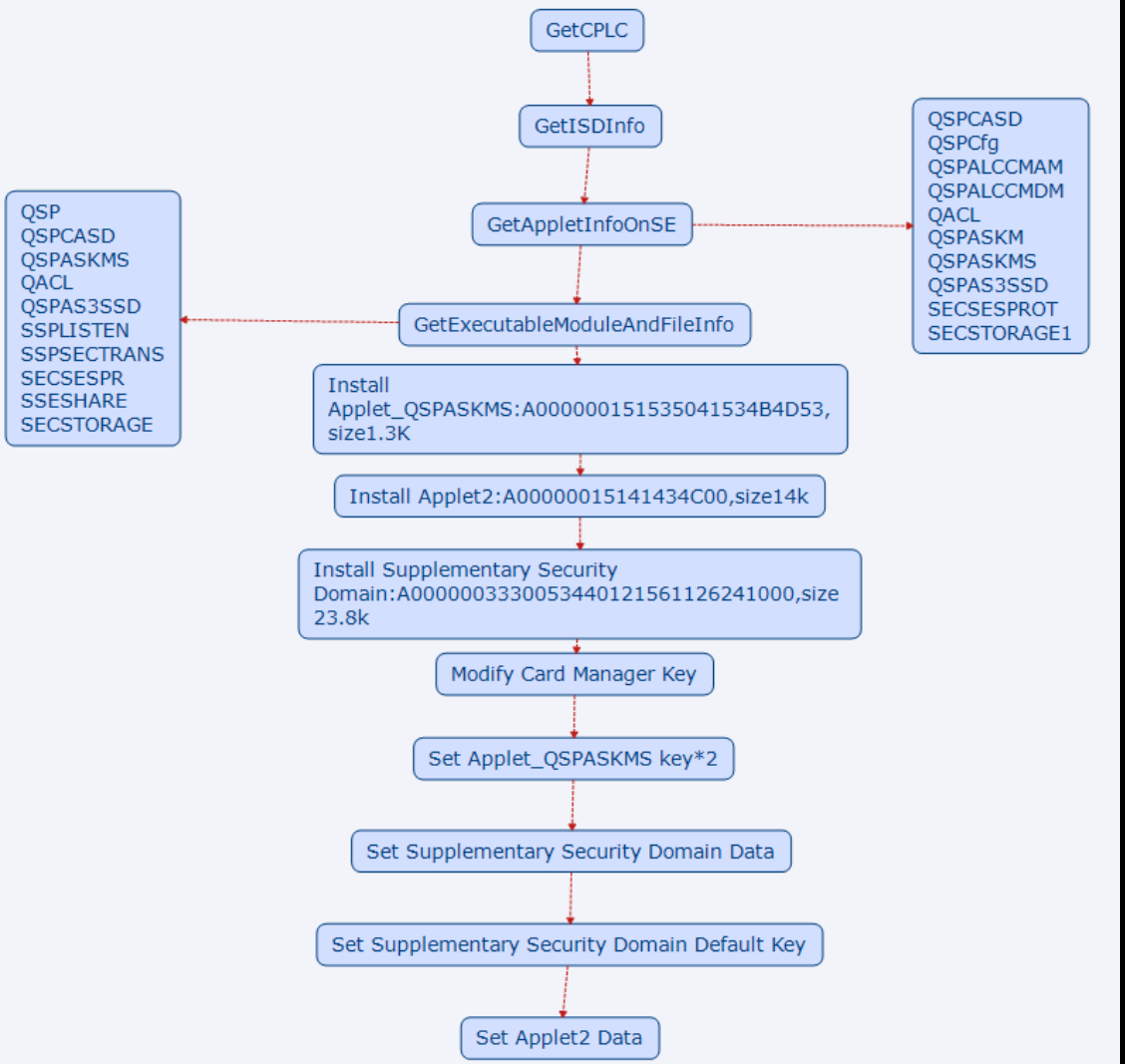
- All packets are transmitted through Secure Channel;
- 3 keys involved: Key_{isd}, $Key_{default}$ and Key_{bank};

Registration-SE Operations

- Create Supplementary Security Domain:
  - Done by SKMS Agent and Samsung Server;
  - Use Key_{isd} to set up Secure Channel, encrypted by Triple DES;
  - Only Samsung and SE know Key_{isd};
  - Working in privilege Security Domain—Issuer Security Domain;
  - At the end of this stage, $Key_{default}$ is set for new domain;






GetCPLC
GetISDInfo
GetAppletInfoOnSE
QSPCASD
QSPCfg
QSPALCCMAM
QSPALCCMDM
QACL
QSPASKM
QSPASKMS
QSPAS3SSD
SECSESPROT
SECSTORAGE1
GetExecutableModuleAndFileInfo
QSP
QSPCASD
QSPASKMS
QACL
QSPAS3SSD
SSPLISTEN
SSPSECTRANS
SECSESPR
SSESHARE
SECSTORAGE
Install Applet_QSPASKMS:A000000151535041534B4D53, size1.3K
Install Applet2:A00000015141434C00,size14k
Install Supplementary Security Domain:A000000333005344012156112624100 0,size23.8k
Modify Card Manager Key
Set Applet_QSPASKMS key*2
Set Supplementary Security Domain Data
Set Supplementary Security Domain Default Key
Set Applet2 Data

Registration-SE Operations

- Update Supplementary Security Domain keys:
  - Update $Key_{default}$ with Key_{bank};
  - Working in supplementary Security Domain;
- Install ARC-C Application:
  - ARA-C(Access Rule Application Client);
  - Hardware-based Access Control Mechanism, allow specific android app to access SE;
  - Hash of certificate is written into;
- Personalize AMSD and Write SEID:
  - AMSD(Authorized Mode Secured Domain, AMSD);
  - Bank assigns an SEID for SE, and write it into SE;

Registration-SE Operations

- Add Access Rules for CRS:
  - CRS(Contactless Registry Service)
  - Application selection rules on the contactless interface(for NFC);
- Install CARDS Applet:
  - Seems Core of Bank implementation,around 11K;
  - After Installation, few initializaiton opertions are done by ISO7816 standard cmds instead of secure channel:
    - CREATE FILE
    - UPDATE BINARY
    - GET CHALLENGE
    - SET PIN

Registration-SE Operations

- Install Applet:
  - Applet for generating tokens, around 53K;
  - Different cards may share the same blob, but different data;
  - The entity that trustlets comminucate with in TrustZone;
  - The whole blob is encrypted, no more detail known until one of the keys gained: Key_{isd}, $Key_{default}$ and Key_{bank}

Registration-Summary

- All traffic packets are encrypted;
- Information leaks also exist;
- Tokens are generated inside SE by certain applet;
- Applets and their config data are stored through Secure Channel, no plain text data exposed;
- Secure Channel is secured by cryptographic key;

Registration-Summary

- All traffic packets are encrypted;
- Information leaks also exist;
- Tokens are generated inside SE by certain applet;
- Applets and their config data are stored through Secure Channel, no plain text data exposed;
- Secure Channel is secured by cryptographic key;

Your WALLET is secured properly!

Black Hat Sound Bytes

- We detailed all process of SamsungPay from userland to TrustZone;
- Key_{isd} is critical for the whole payment system, once leak, attacker can do whatever they want;
- Other two keys are also important to understand the mechanism inside SE;
- SamsungPay will stay secure until these keys leaked/gained;
- Mistake and design faults are made by Samsung and 3rd party developer;

Acknowledgement

- My leader: tombkeeper
- reeseliu for drawing sketch and document translation
- rudywang, jacksonma,huimingliu
- All team members in Xuanwu Lab


Q&A