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 @ Tencent's Xuanwu Lab
- hyperchemma#tencent.com
- Embedded Device Security
- Firmware Reverse-Engineering
- Fan of IoT
- Big Fan of #
Who am I?

- Security Researcher @ Tencent's Xuanwu Lab
- hyperchemma#tencent.com
- Embedded Device Security
- Firmware Reverse-Engineering
- Fan of IoT
- Big Fan of 滚滚向前
Agenda

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

Magnetic Card = Tokenization
Magnetic Card & MST

**Magnetic Card:**
- Store data using magnetic particles;
- Physically 3 tracks on card;
- **Track2** is the only one needed for payment;
- 62307448888888888 = 21027777777777777;
- 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:
  - Store data using magnetic particles;
  - Physically 3 tracks on card;
  - Track 2 is the only one needed for payment;
  - 623074488888888888 = 21027777777777777777;
- 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 Samsung Pay;
MST mechanism
MST mechanism

However, Anyone can capture this signal.
Tokenization

- Reliable solution for processing sensitive information;
- Mathematically irreversible;
- 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

- Applets running in SE
- Trustlets For Payment
- Drivers for devices and TrustZone
- Libs for Comm and Crypto
- SamsungPay Apps

More Secure
More Interface
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 Service 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 directory.
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;
• Four methods for SamsungPay:
  – nativeCreateTLCommunicationContext
  – nativeDestroyTLCommunicationContext
  – nativeProcessTACommand
  – nativeGenerateDeviceCertificates
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`  
  • `/dev/mst_ctrl`  
  ▼ MobiCore Driver
  ▼ `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;

<table>
<thead>
<tr>
<th>Function</th>
<th>CmdID</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>turnonMST</td>
<td>1</td>
<td>Used</td>
</tr>
<tr>
<td>turnoffMST</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>sendTrack1</td>
<td>2</td>
<td>Unused</td>
</tr>
<tr>
<td>sendTrack2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>sendTrack3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>sendTest</td>
<td>4</td>
<td>Used In Test APP</td>
</tr>
<tr>
<td>Escape</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
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, MASTERCARD, 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: nativeProcessTACCommand
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 E²PROM, 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
• **Software:**
  - A Card OS inside, Regulated by
  - Java Card runtime;
  - Cryptographic and Hashing;
  - Security Domain;
  - Global Platform API;
  - Card Life Cycle Models;
  - Secure Channel;
**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;

- 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;

• 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

Payment

Local

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

- PAN
- BankID
- Const
- Sequence
- Token
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 = 210210100051295089
6230745372011888888 = 210210100061045672
6230745372011888888 = 210210100071577380
6230745372011888888 = 210210100081608599
6230745372011888888 = 210210100091744699
```

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:
  • \texttt{StartPay(AID)}
  • \texttt{transmitMSTData(ConfigData)}
Payment-Token Generation

StartPay in Trustlet

- checkData
- checkPayMode
- APDU_selectAID
- APDU_getTrackData
- getAuth
- convertData
- openESEDevice
- APDU_StartUseCard
Payment-Token Generation

Ensure authentication complete

StartPay in Trustlet

getAuth

checkData

convertData

openESEDevice

APDU_StartUseCard

APDU_selectAID

APDU_getTrackData

checkPayMode

checkPayMode
Payment-Token Generation

StartPay in Trustlet

Ensure authentication complete

String2hex (AID)

getAuth

checkData

convertData

openESEDevice

APDU_StartUseCard

APDU_selectAID

APDU_getTrackData

checkPayMode

checkPayMode
Payment-Token Generation

StartPay in Trustlet

Ensure authentication complete

String2hex (AID)
Through SPI

getAuth
convertData
openESEDevice
APDU_StartUseCard
checkData
checkPayMode
APDU_selectAId
APDU_getTrackData
Payment-Token Generation

StartPay in Trustlet

- checkData
- getAuth
- convertData
- openESEDevice
- APDU_StartUseCard

- APDU_selectAID
- APDU_getTrackData

- Is NFC or MST mode?
- Ensure authentication complete

- String2hex (AID)
- Through SPI
Payment-Token Generation

StartPay in Trustlet

Ensure authentication complete

Talk to SE

Is NFC or MST mode?

Through SPI

String2hex (AID)

APDU_StartUseCard

APDU_selectAID

APDU_getTrackData

checkData

getAuth

convertData

openESEDevice

checkPayMode

Talk to SE

String2hex (AID)

Through SPI

Ensure authentication complete

StartPay in Trustlet
Payment-Token Generation

transmitMSTData(ConfigData)

- **CheckTrackData**: Validate track data format and charset by TrackNum
- **EncodeTrackData**: Select charset by TrackNum, and encode data to signal (binary stream)
- **AdjustData**: Generate leading and tail zeros, and connect to signal above
- **Send2Device**: Copy result to a shared memory.
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?
02 Card Registration

1. Choose the Samsung Pay icon
2. Register the card you want to use
3. Use the camera to read your card
   - When the camera cannot read your card successfully, enter your card information manually.
4. Enter your card information
   - When the camera reads your card, check and enter the rest of your card information.
5. Agree to the terms of use
6. Card verification
   - You can choose one of the options to verify your card.
7. Enter code registration
8. Enter the RN number registered
9. 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;

- 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;

 ✓ 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);
Apply a virtual card for physical card, with your credentials

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

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

Request OTP

Enroll Card
Apply a virtual card for physical card, with your credentials

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

Request OTP

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

Apply a virtual card for physical card, with your credentials

"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 for your physical card, with your credentials

Collect Issuer info according to your card number

Wait for virtual card download info

Send the OTP back to bank to finish identification

getCard IssuerInfo

Wait Push

Verify OTP

Apply a virtual card for your physical card, with your credentials

Collect Issuer info according to your card number

Wait for virtual card download info

Send the OTP back to bank to finish identification

getCard IssuerInfo

Wait Push

Verify OTP
Registration-Code

```java
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 ctxt=ApplicationContext.getApplicationContext();
boolean err=IUPJniInterface.jIE(ctxt)://libupsamaddon.so initJNIEnvironment
String SessionKey=IUPJniInterface.mkSK();//makeSessionKey
String EncryptedKey=IUPJniInterface.EER(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,"exchgetkey 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.skK(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));
    .....+
}
```
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));
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;
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;

Whole process are protected by session key and SSL
Android 5.1.1 is vulnerable to some root tools;
① Traffic packets for both process are encrypted by random session key, and transferred through SSL;
Registration-Tips

③ Android 5.1.1 is vulnerable to some root tools;

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;

③ MITM for SSL does not work;

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

② To learn more, packets should be decrypted;

③ MITM for SSL does not work;

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

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

② To learn more, packets should be decrypted;

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

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

② To learn more, packets should be decrypted;

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

① Traffic packets for both processes are encrypted by random session key, and transferred through SSL;
② To learn more, packets should be decrypted;
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

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

② To learn more, packets should be decrypted;

③ Android 5.1.1 is vulnerable to some root tools;

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

③ MITM for SSL does not work;

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

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

2. To learn more, packets should be decrypted;

3. MITM for SSL does not work;

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

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;

Thus a secure root is must.
private static final void do(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;
        }
        Log.v(arg8, arg9, arg10);
        goto label_3;
    }
    Log.i(arg8, arg9, arg10);
    goto label_3;
    }
}

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, i);
}

WalletAppConfig.Falseflag = TSMServiceJniInterface.tmservice JNI_iDM();
Registration-Trick1

TSMService

```java
private static final void a(int arg7, String arg8, String arg9, throwable arg10) {
    Class v12;
    String v0_1;
    if (crashLogUtil.a) {
        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;
            }
            default: {
                goto label_3;
            }
        }
    }
    goto label_3;
    label_3:
    Log.v(arg8, arg9, arg10);
    goto label_3;
    label_8:
    Log.i(arg8, arg9, arg10);
    goto label_3;
    label_12:
    Log.d(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, i);
    }
    WalletAppConfig.falseflag = TSMServiceJniInterface.tsmservice_jni_iDM();
}
```

; DATA XREF: .data:DF4C0324+o

MOVSR0, #0
BXLR
Registration-Trick1

private static final void a(int arg7, String arg8, String arg9, throwable arg10) {
    Class v1_2;
    String v0_1;
    if (crashLogUtil.a) {
        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;
            }
            default: {
                goto label_3;
            }
        }
    }
    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, i);
}

WalletAppConfig.Falseflag = TSMServiceJniInterface.tsmService_jni_iDM();

| .text:DF4A02C  iDM | .text:DF4A02C |
| text:DF44A02E |

Jni_iDM=Jni_isDebugEnabled
Registration-Trick2

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

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

if(AgentLog.isDebugEnabled() != 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;
    }
  } else if(v0 == 18765) {
    return 1;
  } else if(v0 == 18760) {
    v3 = v4;
  }
  return v3;
}

root@zenltechni: # getprop | grep ro.debug_level
[ro.debug_level]: [0x4f4c]
root@zenltechni: #

SKMS Agent

libandroid_runtime
.SO->
isProductShipNative()
Registration-Trick2

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

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

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

**SKMS Agent**

```bash
root@zenltechni:~ # getprop | grep ro.debug_level
[ro.debug_level]: [0x4f4c]
```

**unmodifiable**
Registration-Trick2
Registration-Trick2

• SKMS Agent is a pre-installed app,
  Only odex exist;
• SKMS Agent is a pre-installed app, Only odex exit;

• System will execute the native code in odex file instead of dalvik code;
Registration-Trick2

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

• System will execute the native code in odex file instead of dalvik code;

• Let’s modify native code directly;
• 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
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;
• 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

- 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;
- Install Applet;

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

```
"msgCd":"INITIALIZEAPDU",
"UUID":"f6ecffff-6b4a-4fa5-a7f7-fd9cbe172222",
"msgTime":"180604164609",
"resultCode":"00000000",
"cApduSet":<APDUs>,
"serviceName":"**** AMSD BANK1 SSD001 Service"
```
```
"msgCd":"NEXTAPDU",
"UUID":"f6ecffff-6b4a-4fa5-a7f7-fd9cbe172222",
"seId":"411111104700DA3E01005177080777777777",
"msgTime":"180315164610",
"rApduSet":<rAPDUs>
```
Registration-SE Operations

• Update Supplementary Security Domain keys:
  • Update $Key_{\text{default}}$ with $Key_{\text{bank}}$;
  • 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 initialization operations 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 communicate 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 3$^{rd}$ party developer;
Acknowledgement

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