fpEventManager.java

public class fpEventManager implements Fp\_native.OnReportListerner {

public static final String TAG = "fpEventManager";

private Context mContext;

private static fpEventManager instance;

public static final boolean DEFAULT = FpTouchEventHandle.DEFAULT\_VALUE;

private FpTouchEventHandle mEh;

public fpEventManager(Context c) {

mContext = c;

mEh = FpTouchEventHandle.getInsance(c);

}

public static fpEventManager getInsance(Context c) {

if(instance == null) {

instance = new fpEventManager(c);

}

return instance;

}

public void disable(boolean b) {

FpTouchEventHandle.getInsance(mContext).disable(b);

}

public void setOccluded(boolean occ) {

FpTouchEventHandle.getInsance(mContext).setOccluded(occ);

}

public boolean start() {

if(Fp\_native.native\_init(0) < 0) {

return false;

}

new Thread(new Runnable() {

@Override

public void run() {

while(true) {

try {

int event = Fp\_native.native\_update();

if (event > 0) {

Log.i(TAG, "event:"+event);

mEh.fp\_handleKeyEvent(event);

}

}

catch (Exception e) {

e.printStackTrace();

}

}

}

}).start();

return true;

}

public boolean start(int s) {

Log.i(TAG, "start:" + s);

if(s == 0) {

start();

}

else {

Fp\_native fn = new Fp\_native();

fn.init(1, fpEventManager.this);

}

return true;

}

public boolean OnReport(int e) {

Log.i(TAG, "OnReport:" + e);

mEh.fp\_handleKeyEvent(e);

return false;

}

public static void updatePreference(Context c){

FpTouchEventHandle.getInsance(c).updatePreference(c);

}

}

FpTouchEventHandle.java

public class FpTouchEventHandle {

private static final String TAG = "FpTouchEventHandle";

private static int HW\_EVT\_DOWN = 1;

private static int HW\_EVT\_UP = 2;

private static final int EVT\_ONCE\_TOUCH = 0;

private static final int EVT\_DOUBLE\_TOUCH = 1;

private static final int EVT\_LONG\_TOUCH = 2;

private static long lastFingerDownTime=0;

private static long lastFingerUpTime = 0;

private static final int FINGER\_CLICK\_TIME = 300;

private static final int FINGER\_LONG\_PRESS\_TIME = 600;

private static final int FINGER\_DOUBLE\_CLICK\_TIME = 700;

private Context mContext;

private Vibrator mVb;

private static FpTouchEventHandle instance;

private Timer mTimer;

private final static String ACTION\_LOCK\_SCREEN =

"tinno.intent.action.DO\_KEYGUARD\_LOCKED";

private final static String ACTION\_CAM\_OPEN =

"android.intent.action.ACTION\_SHUTDOWN\_FLASH";

private TelephonyManager mTelemanager;

//private AudioProfileManager mProfileManager;

private final static String PROFILE\_MEETING = "mtk\_audioprofile\_meeting";

private final static String PROFILE\_SILENT = "mtk\_audioprofile\_silent";

private boolean mDisable = false;

private boolean has\_down;

private static boolean is\_Open;

private boolean is\_CamOpen = false;

//private boolean isLockScreen = false;

public static final boolean DEFAULT\_VALUE =

MainFeatureOptions.getTouchBackDefaultStatus();

public static final boolean DEFAULT\_PROJECT\_SUPPORTED\_VALUE =

MainFeatureOptions.isBackFingerprint();

public static final String PRE\_FILES = "pre\_fingerprint";

public static final String PRE\_SENSOR\_BUTTON\_ONOFF =

"pre\_sensor\_button\_onoff";

private static final String AUDIO\_PROFILE\_SERVICE = "audioprofile";

private IAudioProfileService mIAudioProfileService;

private boolean mOccluded = false;

public FpTouchEventHandle(Context c) {

mContext = c;

mVb = (Vibrator) c.getSystemService(Context.VIBRATOR\_SERVICE);

c.getSystemService(Context.AUDIO\_PROFILE\_SERVICE);

mTelemanager =

(TelephonyManager) c.getSystemService(Service.TELEPHONY\_SERVICE);

updatePreference(c);

registerScreenActionReceiver();

getAudioProfileService();

}

public IAudioProfileService getAudioProfileService()

{

if(mIAudioProfileService == null) {

mIAudioProfileService = IAudioProfileService.Stub.asInterface(

ServiceManager.getService(AUDIO\_PROFILE\_SERVICE));

if(mIAudioProfileService == null) {

Log.e(TAG, "-----getAudioProfileService fail!-----");

}

}

return mIAudioProfileService;

}

public static FpTouchEventHandle getInsance(Context c) {

if(instance == null) {

instance = new FpTouchEventHandle(c);

}

return instance;

}

public void disable(boolean b) {

mDisable = b;

}

public void setOccluded(boolean occ) {

mOccluded = occ;

}

public void fp\_handleKeyEvent( int evt) {

if(DEFAULT\_PROJECT\_SUPPORTED\_VALUE){

Log.i(TAG, "Sensor button has been closed!");

return;

}

if(!is\_Open) {

Log.i(TAG, "Sensor button has been closed!");

return;

}

final boolean isCall = (mTelemanager.getCallState()

== TelephonyManager.CALL\_STATE\_OFFHOOK);

if(isDeviceLocked()) {

if(mOccluded && !isCall){

sendDownAndUpKeyEvents(evt);

return;

}

Log.i(TAG, "isLockScreen mOccluded:"+mOccluded);

return;

}

if(mDisable) {

Log.i(TAG, "Disable!");

return;

}

if(isFingerprintEnrolling()){

Log.i(TAG, "isFingerprintEnrolling...!");

return;

}

if(isCall) {

final String cls = getTop();

if(cls.contains("com.android.incallui.InCallActivity")) {

Log.i(TAG, "Incall, so,return!");

return;

}

}

if(isDeviceLocked()) {

Log.i(TAG, "isDeviceLocked!");

return;

}

sendDownAndUpKeyEvents(evt);

}

private void sendDownAndUpKeyEvents(int msgtype) {

//Log.i(TAG, " " + "HW msgtype = "+ msgtype);

if(msgtype == HW\_EVT\_DOWN)

{

final long downTime = SystemClock.uptimeMillis();

lastFingerDownTime = downTime;

}

else if(msgtype == HW\_EVT\_UP)

{

final long upTime = SystemClock.uptimeMillis();

if(upTime -lastFingerDownTime < FINGER\_CLICK\_TIME)

{

String cls = null;

if(is\_CamOpen){

cls = getTop();

}

if(("com.myos.camera.CameraLauncher2".equals(cls)

||"com.myos.camera.SecureCameraActivity".equals(cls)))

{

Log.i(TAG, "do capture!");

simulateKeyStroke(KeyEvent.KEYCODE\_VOLUME\_DOWN);

vibrateForUser(30);

}

else

{

if(is\_CamOpen)

{

Log.i(TAG, "Camera has exit!");

is\_CamOpen = false;

}

if(is\_Open) {

simulateKeyStroke(KeyEvent.KEYCODE\_BACK);

if(isEnableVib())

{

vibrateForUser(MainFeatureOptions.getFingerVibTime());

}

}

}

}

}

}

private boolean isEnableVib()

{

final boolean hf =

EmCommon.getIntForUserEM(mContext.getContentResolver(),

Settings.System.HAPTIC\_FEEDBACK\_ENABLED, 1, UserHandleEM.USER\_CURRENT) != 0;

if(!WrapperManager.isMtkPlateform()/\*Features.is\_mtk\*/) {

return hf;

}

try {

if(getAudioProfileService() != null) {

final String key = getAudioProfileService().getActiveProfileKey();

final boolean enable = !(PROFILE\_MEETING.equals(key)

|| PROFILE\_SILENT.equals(key));

return hf && enable;

}

return hf;

}

catch (Exception e) {

e.printStackTrace();

}

return false;

}

private void simulateKeyStroke(final int KeyCode) {

new Thread(new Runnable() {

@Override

public void run() {

try {

Instrumentation in = new Instrumentation();

in.sendKeyDownUpSync(KeyCode);

} catch (Exception e) {

Log.e(TAG, "simulateKeyStroke:"+e);

}

}

}).start();

}

private boolean isScreenOn() {

PowerManager mPowerManagerEx =

(PowerManager)mContext.getSystemService(Context.POWER\_SERVICE);

boolean isScreenOn = false;

isScreenOn = mPowerManagerEx.isScreenOn();

return isScreenOn;

}

private boolean isDeviceLocked(){

return EncryptServiceManager.isKeyguardLocked(mContext);

}

private boolean isFingerprintEnrolling(){

String currentStackTop = getTop();

return currentStackTop.contains("FingerprintEnrollEnrolling");

}

public void vibrateForUser(int times) {

Vibrator vibrator = mContext.getSystemService(Vibrator.class);

if (vibrator != null) {

final long[] pattern = new long[] {0, times};

vibrator.vibrate(pattern, -1);

}

}

private String getTop() {

final ActivityManager am =

(ActivityManager) mContext.getSystemService(Context.ACTIVITY\_SERVICE);

ActivityInfo aInfo = null;

List<RunningTaskInfo> list = am.getRunningTasks(1);

if (list.size() != 0) {

RunningTaskInfo topRunningTask = list.get(0);

final String top = topRunningTask.topActivity.getClassName();

//Log.d(TAG,"Top:"+top);

return top;

} else {

return "";

}

}

private void registerScreenActionReceiver() {

final IntentFilter filter = new IntentFilter();

filter.addAction(Intent.ACTION\_SCREEN\_OFF);

filter.addAction(Intent.ACTION\_SCREEN\_ON);

//filter.addAction(ACTION\_LOCK\_SCREEN);

filter.addAction(Intent.ACTION\_USER\_PRESENT);

filter.addAction(ACTION\_CAM\_OPEN);

filter.setPriority(1002);

mContext.registerReceiver(receiver, filter);

}

private final BroadcastReceiver receiver = new BroadcastReceiver() {

@Override

public void onReceive(final Context context, final Intent intent) {

final String action = intent.getAction();

Log.d(TAG,"--[onReceive]--: "+action);

final int uid = ActivityManagerEM.getCurrentUser();

if(Intent.ACTION\_USER\_PRESENT.equals(action)) {

disable(false);

//isLockScreen = false;

}

else if(Intent.ACTION\_SCREEN\_OFF.equals(action)) {

if(uid != 0){ return; }

disable(true);

}

else if(Intent.ACTION\_SCREEN\_ON.equals(action)) {

final String cls = getTop();

Log.i(TAG, "getTop:"+cls);

if(cls.contains("com.google.android.setupwizard")

|| cls.contains("com.android.settings.wifi.WifiSetupActivity")) {

disable(false);

}

else if(!isDeviceLocked()) {

disable(false);

}

else {

if(uid != 0){ return; }

disable(true);

}

}

/\*else if(ACTION\_LOCK\_SCREEN.equals(action)) {

//disable(true);

isLockScreen = true;

}\*/

else if(ACTION\_CAM\_OPEN.equals(action)) {

if(MainFeatureOptions.isTouchSensorCaptureSupported()) {

is\_CamOpen = true;

}

}

}

};

public static boolean updatePreference(Context c){

Context useCtx = null;

try {

useCtx = c.createPackageContext("com.ape.encryptmanager",

Context.CONTEXT\_IGNORE\_SECURITY);

}

catch (NameNotFoundException e) {

Log.i(TAG, "NameNotFoundException:"+e);

return false;

}

SharedPreferences mPreferences = useCtx.getSharedPreferences(PRE\_FILES,

(Context.MODE\_WORLD\_READABLE |Context.MODE\_MULTI\_PROCESS));

is\_Open =

mPreferences.getBoolean(PRE\_SENSOR\_BUTTON\_ONOFF, DEFAULT\_VALUE);

Log.i(TAG, "updatePreference-is\_Open:"+is\_Open);

return true;

}

}

Fp\_native.java

public class Fp\_native {

private static final String TAG = "Fp\_native-cls";

private OnReportListerner mOnReportListerner = null;

static

{

System.loadLibrary("fp\_dev\_jni");

}

public static native int native\_init(int set);

public static native int native\_update();

public static native int native\_setObj(Fp\_native f);

public static boolean init(int s, OnReportListerner l) {

if(native\_init(s) < 0) {

Log.i(TAG, "native\_init:fail");

return false;

}

Fp\_native fn = new Fp\_native();

fn.native\_setObj(fn);

fn.setOnReportListerner(l);

return true;

}

//native call

public int onEventReport(int e) {

mOnReportListerner.OnReport(e);

return 0;

}

public interface OnReportListerner {

boolean OnReport(int event);

}

public void setOnReportListerner(OnReportListerner l) {

mOnReportListerner = l;

}

}

EncryptServiceManager.java

public EncryptServiceManager(Context context, Handler handler) {

Log.d(TAG, "new EncryptServiceManager");

mContext = context;

mHandler = handler;

mFingerPrintDataUtils = new FingerPrintDataUtils(context);

mFingerprintManager =

new FingerprintManagerEM(context);

if(mFingerprintManager != null){

Message m = new Message();

m.what = MessageType.TINNO\_MSG\_SERVICE\_CONNECTED;

mHandler.sendMessage(m);

}

}

private EnrollmentCallbackEM mEnrollmentCallback = new EnrollmentCallbackEM() {

@Override

public void onEnrollmentProgress(int remaining) {

if (mEnrollmentSteps == -1) {

mEnrollmentSteps = remaining;

}

mEnrollmentRemaining = remaining;

mEnrollmentProgress = (FINGER\_ENROLL\_TOTAL\_TIMES - remaining)

\* (100.0/FINGER\_ENROLL\_TOTAL\_TIMES);

TinnoFingerprintData MessageData =

new TinnoFingerprintData(-1, (int)mEnrollmentProgress, 0);

if (mListener != null) {

mListener.onEnrollmentProgressChange(mEnrollmentSteps, remaining);

}

if(remaining == 0){

int newFingerId = getCurrentEnrolledFingerId();

if(newFingerId == -1) {

Log.d(TAG, "newFingerId== -1, mEnrollmentCallback err!");

return;

}

MessageData.setFingerid(newFingerId);

}

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_ENROLL\_CORRECT, MessageData));

Log.d(TAG, "remaining = " + remaining

+ "| mEnrollmentProgress = " + mEnrollmentProgress

+ "fingerId = "+ MessageData.getFingerid());

}

@Override

public void onEnrollmentHelp(int helpMsgId, CharSequence helpString) {

TinnoFingerprintData MessageData =

new TinnoFingerprintData(-1, (int)mEnrollmentProgress, 0);

if(helpMsgId == MessageType.FINGERPRINT\_ACQUIRED\_ENROLL\_DUPLICATE){

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_ENROLL\_DUPLICATE, MessageData));

}else if(helpMsgId

== MessageType.FINGERPRINT\_ACQUIRED\_ENROLL\_TOO\_NEARBY){

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_ENROLL\_NO\_EXTRAINFO, MessageData));

}

}

@Override

public void onEnrollmentError(int errMsgId, CharSequence errString) {

if (mListener != null) {

mListener.onEnrollmentError(errString);

}

}

};

private final Runnable mTimeoutRunnable = new Runnable() {

@Override

public void run() {

cancelEnrollment();

}

};

private int startEnrollment(byte [] token) {

mEnrollHandler.removeCallbacks(mTimeoutRunnable);

mEnrollmentSteps = -1;

mEnrollmentCancel = new CancellationSignal();

mFingerprintManager.enroll(token, mEnrollmentCancel,

0, mEnrollmentCallback);

mEnrolling = true;

return 0;

}

private void cancelEnrollment() {

mHandler.removeCallbacks(mTimeoutRunnable);

if (mEnrolling) {

mEnrollmentCancel.cancel();

mEnrolling = false;

mEnrollmentSteps = -1;

}

}

public interface Listener {

void onEnrollmentHelp(CharSequence helpString);

void onEnrollmentError(CharSequence errString);

void onEnrollmentProgressChange(int steps, int remaining);

}

public int getCurrentEnrolledFingerId() {

final List<Fingerprint> enrolled\_items

= mFingerprintManager.getEnrolledFingerprints();

final int fpEnrolledCount = enrolled\_items.size();

final List<FingerPrintItem> database\_items

= mFingerPrintDataUtils.getFingerPrintDatas();

final int fpDatabaseCount = database\_items.size();

for(int i = 0; i < fpEnrolledCount; i++){

int fpEnrolledId = enrolled\_items.get(i).getFingerId();

int j;

Log.d(TAG, "fpEnrolledId = " + fpEnrolledId);

for(j =0; j<fpDatabaseCount; j++){

Log.d(TAG, "DbId = " + database\_items.get(j).getFingerPrintData());

if(fpEnrolledId == database\_items.get(j).getFingerPrintData()){

break;

}

}

if(j == fpDatabaseCount){

Log.d(TAG, "find fp id:" + fpEnrolledId);

return fpEnrolledId;

}

}

Log.d(TAG, "Not found finger id!" );

return -1;

}

public Fingerprint getFingerPrintById(int fingerId){

final List<Fingerprint> items = mFingerprintManager.getEnrolledFingerprints();

final int fpCount = items.size();

for(int i=0; i<fpCount; i++){

if(fingerId == items.get(i).getFingerId())

return items.get(i);

}

return null;

}

public void showAppIcon(Context context){

PackageManager p = context.getPackageManager();

ComponentName cName =

new

ComponentName(APE\_FINGERPRINT\_APP\_NAME,APE\_FINGERPRINT\_CLASS\_NAME);

if(MainFeatureOptions.isNeedShowAppIconSupported(context)){

p.setComponentEnabledSetting(cName, PackageManager.COMPONENT\_ENABLED\_STATE\_ENABLED, PackageManager.DONT\_KILL\_APP);

}else{

p.setComponentEnabledSetting(cName, PackageManager.COMPONENT\_ENABLED\_STATE\_DISABLED, PackageManager.DONT\_KILL\_APP);

}

}

private void deleteFingerPrint(Fingerprint fingerPrint) {

mFingerprintManager.remove(fingerPrint, mRemoveCallback);

}

private FingerprintManagerEM.RemovalCallbackEM mRemoveCallback

= new FingerprintManagerEM.RemovalCallbackEM() {

@Override

public void onRemovalSucceeded(Fingerprint fingerprint) {

}

@Override

public void

onRemovalError(Fingerprint fp, int errMsgId, CharSequence errString) {

}

};

public void resetFingerFailedAttemps(){

Log.d(TAG, "resetFingerFailedAttemps ####");

if (mFingerprintManager != null) {

byte[] token = null;

mFingerprintManager.resetTimeout(token);

mTinnoFingerprintLockout = false;

}

}

private FingerprintManagerEM.AuthenticationCallbackEM mAuthenticationCallback = new FingerprintManagerEM.AuthenticationCallbackEM() {

@Override

public void onAuthenticationFailed() {

Log.d(TAG, "onAuthenticationFailed");

vibrateFingerprintError(mContext);

TinnoFingerprintData MessageData = new TinnoFingerprintData(-1, "verify failed");

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_VERIFY\_FAILED, MessageData));

};

@Override

public void onAuthenticationSucceeded(AuthenticationResult result) {

AuthenticationResultEM resultEM = new AuthenticationResultEM(result);

int fingerid = resultEM.getFingerprint().getFingerId();

//result.getCryptoObject();

TinnoFingerprintData MessageData = new TinnoFingerprintData(

fingerid, 0);

vibrateFingerprintSuccess(mContext);

Log.d(TAG, "onAuthenticationSucceeded, fingerid = " + fingerid);

//Ramiel P7611AE\_SUG\_CN\_6.0PGFAAAES-667 +++

if(SimPinLockNoUseFp()==SIM\_STATE\_PIN\_REQUIRED)

Log.d(TAG, "Ramiel SimPinLockNoUseFp()=SIM\_STATE\_PIN\_REQUIRED DO NOT send succeeded");

//Ramiel P7611AE\_SUG\_CN\_6.0PGFAAAES-667 ---

if(isScreenON(mContext)){

//Ramiel P7611AE\_SUG\_CN\_6.0PGFAAAES-667 +++

if(SimPinLockNoUseFp()!=SIM\_STATE\_PIN\_REQUIRED)

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_VERIFY\_SUCCESS, MessageData));

//Ramiel P7611AE\_SUG\_CN\_6.0PGFAAAES-667 ---

}else{

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_VERIFY\_IDENTIFY\_WAKEUP\_MATCHED, MessageData));

}

cancelOperation(EncryUtil.FINGERPRINT\_REGISTER\_VERIFY\_TYPE,EncryUtil.TAG\_VERIFY\_SUCCESS);

}

@Override

public void onAuthenticationHelp(int helpMsgId, CharSequence helpString) {

Log.d(TAG, "onAuthenticationHelp , helpMsgId = " + helpMsgId + " | helpString = " + helpString);

TinnoFingerprintData MessageData = new TinnoFingerprintData(

helpMsgId, (String)helpString);

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_VERIFY\_HELP, MessageData));

}

@Override

public void onAuthenticationError(int errMsgId, CharSequence errString) {

if(getVerifyCounter() == 0) {

return;

}

Log.d(TAG, "onAuthenticationError , errMsgId = " + errMsgId + " | errString = " + errString);

TinnoFingerprintData MessageData = new TinnoFingerprintData(

errMsgId, (String)errString);

if(errMsgId == FingerprintManager.FINGERPRINT\_ERROR\_LOCKOUT){

mTinnoFingerprintLockout = true;

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_VERIFY\_ERROR\_LOCKOUT, MessageData));

vibrateFingerprintLockOut(mContext);

}/\*else{ //Removed by yinglong.tang.

mHandler.sendMessage(mHandler.obtainMessage(

MessageType.TINNO\_MSG\_SYSTEM\_ERROR, MessageData));

}\*/

}

@Override

public void onAuthenticationAcquired(int acquireInfo) {

Log.d(TAG, "onAuthenticationAcquired , acquireInfo = " + acquireInfo);

}

};

public void cancelOperation(int type, final int tag) {

Log.i(TAG, "Entry cancelOperation.");

switch(type){

case EncryUtil.FINGERPRINT\_REGISTER\_VERIFY\_TYPE:

mHandler.postDelayed(new Runnable() {

@Override

public void run() {

doCancel(tag);

}

}, 0);

break;

case EncryUtil.FINGERPRINT\_REGISTER\_ENROLL\_TYPE:

cancelEnrollment();

break;

}

}

public int verifyStart(final int delay, final int tag) {

if(isEnrolledFingerprints()){

final int userId = ActivityManagerEM.getCurrentUser();

if (isUnlockWithFingerPrintPossible(userId)) {

mHandler.postDelayed(new Runnable() {

@Override

public void run() {

doAuthenticateListening(tag);

}

}, delay);

}

return 0;

}

Log.i(TAG, "verifyStart fail:no enrolled fingerprint!");

return -1;

}

private void doAuthenticateListening(int tag){

if (isDeviceLocked(mContext) && isOccluded()) {

return;

}

if(fingerVerifyCount == 1) {

Log.i(TAG, "Warning: Please call doCancel first!");

return;

}

synchronized (EncryptServiceManager.this) {

final int uid = ActivityManagerEM.getCurrentUser();

fingerVerifyCount = 1;

lastVerifyTag = tag;

Log.i(TAG, "doAuthenticateListening():"+EncryUtil.parseTag(tag));

mFingerprintCancelSignal = new CancellationSignal();

mFingerprintManager.authenticate(null, mFingerprintCancelSignal,

0, mAuthenticationCallback, null, uid);

}

Log.i(TAG, "doAuthenticateListening exit.");

}

private void doCancel(int tag){

/\* //Removed by yinglong.tang

if(fingerVerifyCount == 0){

Log.i(TAG, "Warning: Authenticate alreadly cancel!");

return;

}

\*/

if(!checkCancelPolicy(tag, lastVerifyTag)) {

Log.i(TAG, "checkCancelPermission:fail:" + EncryUtil.parseTag(tag) + " cannot stop:" + EncryUtil.parseTag(lastVerifyTag));

return;

}

synchronized (EncryptServiceManager.this) {

Log.i(TAG, "doCancel():"+EncryUtil.parseTag(tag));

if(mFingerprintCancelSignal != null){

mFingerprintCancelSignal.cancel();

mFingerprintCancelSignal = null;

}

fingerVerifyCount = 0;

}

Log.i(TAG, "doCancel exit.");

}

//Authenticate cancel policy.

private boolean checkCancelPolicy(int cancel\_tag,int auth\_tag) {

if(cancel\_tag == EncryUtil.TAG\_KEYGUARD\_BINDER\_CALL

|| cancel\_tag == EncryUtil.TAG\_VERIFY\_SUCCESS

|| cancel\_tag == EncryUtil.TAG\_SIM\_STATE\_CHANGE

|| cancel\_tag == EncryUtil.TAG\_VERIFY\_ERR\_LOCKOUT) {

return true;

}

if(cancel\_tag == auth\_tag) {

return true;

}

else {

switch (auth\_tag) {

case EncryUtil.TAG\_KEYGUARD\_BINDER\_CALL:

case EncryUtil.TAG\_SCREEN\_ON\_OFF:

case EncryUtil.TAG\_AIRPLANE\_MODE\_CHANGED:

case EncryUtil.TAG\_DO\_UNLOCK\_SCREEN:

case EncryUtil.TAG\_COUNTDOWN\_TIMER\_ON\_FINISH: {

switch (cancel\_tag) {

case EncryUtil.TAG\_USER\_PRESENT:

return true;

}

}

}

}

return false;

}

public static int getVerifyCounter(){

int state = EncryptService.getFingerprintdState(EncryptService.MSG\_FINGERPRINTD\_STATE);

if (state == 0 && fingerVerifyCount != 0 ) {

try {

RuntimeException here = new RuntimeException("here");

here.fillInStackTrace();

android.util.Log.i(TAG, "getVerifyCounter statck is", here);

} catch (Exception e) {

Log.i(TAG, "getVerifyCounter Exception:" + e.toString());

}

}

if(state == 0) {

fingerVerifyCount = 0;

}

return fingerVerifyCount;

}

public boolean isUnlockWithFingerPrintPossible(int userId) {

return mFingerprintManager != null && !isFingerprintDisabled(userId);

}

private boolean isFingerprintDisabled(int userId) {

final DevicePolicyManager dpm =

(DevicePolicyManager) mContext.getSystemService(Context.DEVICE\_POLICY\_SERVICE);

return dpm != null && (dpm.getKeyguardDisabledFeatures(null/\*, userId\*/)

& DevicePolicyManager.KEYGUARD\_DISABLE\_FINGERPRINT) != 0;

}

public int enrollStart(byte [] token) {

return startEnrollment(token);

}

public int[] getFpIndexList() {

try {

int[] data = null;

List<Integer> fpIndexList = new ArrayList<Integer>();

List<Fingerprint>

list = mFingerprintManager.getEnrolledFingerprints();

if (list != null && list.size() > 0) {

for (int i = 0; i < list.size(); i++) {

fpIndexList.add(list.get(i).getFingerId());

}

}

if (fpIndexList != null && fpIndexList.size() > 0) {

data = new int[fpIndexList.size()];

for (int i = 0; i < fpIndexList.size(); i++) {

data[i] = fpIndexList.get(i);

}

}

return data;

} catch (Exception e) {

LogUtil.d(TAG, "Exception :" + e.toString());

return null;

}

}

public int removeCredential(int enrollIndex) {

try{

Fingerprint currentFinger = getFingerPrintById(enrollIndex);

deleteFingerPrint(currentFinger);

return 0;

}catch(Exception e){

LogUtil.d(TAG, "Exception :" + e.toString());

return -1;

}

}

public boolean isFingerprintLockout(){

return mTinnoFingerprintLockout == true;

}

public void setFingerprintLockoutValue(boolean isLockout){

mTinnoFingerprintLockout = isLockout;

}

public static void vibrateFingerprintError(Context c) {

if(c == null) { return; }

Vibrator vibrator = (Vibrator)c.getSystemService(Context.VIBRATOR\_SERVICE);

if (vibrator != null) {

vibrator.vibrate(FP\_ERROR\_VIBRATE\_PATTERN, -1);

Log.d("RamielFP", "vibrateFingerprint 1 !");

}

else {

Log.d(TAG, "vibrateFingerprintError vibrator is null!");

}

}

public static void vibrateFingerprintSuccess(Context c) {

if(c == null) { return; }

Vibrator vibrator = (Vibrator)c.getSystemService(Context.VIBRATOR\_SERVICE);

if (vibrator != null) {

vibrator.vibrate(FP\_LOCKOUT\_VIBRATE\_PATTERN, -1);

Log.d("RamielFP", "vibrateFingerprint 2 !");

}

else {

Log.d("RamielFP", "vibrateFingerprintSuccess vibrator is null!");

}

}

public static void vibrateFingerprintLockOut(Context c) {

if(c == null) { return; }

Vibrator vibrator = (Vibrator)c.getSystemService(Context.VIBRATOR\_SERVICE);

if (vibrator != null) {

vibrator.vibrate(FP\_LOCKOUT\_VIBRATE\_PATTERN, -1);

Log.d("RamielFP", "vibrateFingerprint 3 !");

}

else {

Log.d(TAG, "vibrateFingerprintLockOut vibrator is null!");

}

}

public static boolean isScreenON(Context c) {

if(c == null) { return false; }

PowerManager

pm = (PowerManager)c.getSystemService(Context.POWER\_SERVICE);

if(pm == null) { return false; }

boolean isScreenOn;

if (VERSION.SDK\_INT >= VERSION\_CODES.KITKAT\_WATCH) {

isScreenOn = pm.isInteractive();

} else {

isScreenOn = pm.isScreenOn();

}

Log.i(TAG," isScreenON = " + isScreenOn);

return isScreenOn;

}

public static boolean isDeviceLocked(Context c){

if(c == null) { return false; }

KeyguardManager

km = (KeyguardManager)c.getSystemService(Context.KEYGUARD\_SERVICE);

if(km != null){

return km.isDeviceLocked();//ActivityManagerEM.getCurrentUser()

}

return false;

}

public static boolean isKeyguardLocked(Context c){

if(c == null) { return false; }

KeyguardManager

km = (KeyguardManager)c.getSystemService(Context.KEYGUARD\_SERVICE);

if(km != null){

return km.isKeyguardLocked();

}

return false;

}

public boolean isEnrolledFingerprints() {

List<Fingerprint> enrolled\_items =

mFingerprintManager.getEnrolledFingerprints();

return (enrolled\_items.size() > 0);

}

public void setOccluded(boolean isOccluded) {

mOccluded = isOccluded;

}

public boolean isOccluded() {

return mOccluded;

}

}

EncryptService.java

public void callMakeCallFromFingerLaunch(Context context, String phoneNumber) {

Intent intent;

if(PhoneNumberUtils.isEmergencyNumber(phoneNumber)){

intent = new Intent("android.intent.action.CALL\_EMERGENCY", Uri.parse("tel:" + phoneNumber));

}else{

intent = new Intent("android.intent.action.CALL", Uri.parse("tel:" + phoneNumber));

}

intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK

| Intent.FLAG\_ACTIVITY\_REORDER\_TO\_FRONT);//FLAG\_ACTIVITY\_CLEAR\_TASK//FLAG\_ACTIVITY\_CLEAR\_TOP

context.startActivity(intent);//AsUser(intent, UserHandleEM.CURRENT\_OR\_SELF);

//mVerifyHandler.sendEmptyMessageDelayed(MessageType.TINNO\_MSG\_UNLOCK\_SCREEN, 5);

}

private Handler mVerifyHandler = new Handler(){

@Override

public void handleMessage(Message msg) {

super.handleMessage(msg);

int what = msg.what;

TinnoFingerprintData MessageData = (TinnoFingerprintData)msg.obj;

switch(what){

case MessageType.TINNO\_MSG\_SERVICE\_CONNECTED:

break;

case MessageType.TINNO\_MSG\_VERIFY\_SUCCESS:

case MessageType.TINNO\_MSG\_VERIFY\_IDENTIFY\_WAKEUP\_MATCHED:

onIdentifyRsp(what,MessageData.getFingerid(), -1, null);

break;

case MessageType.TINNO\_MSG\_VERIFY\_FAILED:

case MessageType.TINNO\_MSG\_SYSTEM\_ERROR:

case MessageType.TINNO\_MSG\_VERIFY\_ERROR\_LOCKOUT:

onIdentifyRsp(what,-1, MessageData.getMsgId(), MessageData.getMsgStr());

break;

case MessageType.TINNO\_MSG\_START\_APP:

Bundle bundle = msg.getData();

String packageName = bundle.getString(KEY\_PACKAGE\_NAME);

String className = bundle.getString(KEY\_CLASS\_NAME);

callStartAppFromFingerLaunch(mContext,

packageName, className);

break;

case MessageType.TINNO\_MSG\_UNLOCK\_SCREEN:

unLockScreenForQuickBoot();

break;

case MessageType.TINNO\_MSG\_MAKE\_CALL:

if (mTelemanager.getCallState()

== TelephonyManager.CALL\_STATE\_OFFHOOK) {

Log.i(TAG, "Ramiel(1) doScreenUnLock ");

doScreenUnLock();

break;

}

Bundle bundle2 = msg.getData();

mPhoneNumber = bundle2.getString(KEY\_PHONE\_NUMBER);

Log.i(TAG, "Ramiel(2) doScreenUnLock ");

doScreenUnLock();

if(mMakeCallRunnable != null){

this.removeCallbacks(mMakeCallRunnable);

}

this.postDelayed(mMakeCallRunnable = new Runnable() {

@Override

public void run() {

callMakeCallFromFingerLaunch(mContext, mPhoneNumber);

}

}, 10);

break;

case MessageType.TINNO\_MSG\_START\_VIDEO\_APP\_DELAY:

Bundle videoBundle = msg.getData();

String videoPkgName

= videoBundle.getString(KEY\_PACKAGE\_NAME);

String videoClsName

= videoBundle.getString(KEY\_CLASS\_NAME);

try {

Intent intent = new Intent(Intent.ACTION\_MAIN,null);

intent.setClassName(videoPkgName, videoClsName);

intent.addFlags(Intent.FLAG\_ACTIVITY\_NEW\_TASK | Intent.FLAG\_ACTIVITY\_RESET\_TASK\_IF\_NEEDED);

intent.addCategory(Intent.CATEGORY\_LAUNCHER);

mContext.startActivity(intent);

}

catch (Exception e) {

Log.e(TAG, "TINNO\_MSG\_START\_VIDEO\_APP\_DELAY shit!:" + e);

}

break;

}

}

};

public boolean fingerVerifyStart(int delay, int tag){

Log.d(TAG, " \*\*\*\*\*\*\* GOTO fingerVerifyStart! \*\*\*\*\*\*\* delay:" + delay);

if(mVerifyServiceManager != null){

return (mVerifyServiceManager.verifyStart(delay, tag) != -1);

}

return true;

}

public void onIdentifyRsp(int result, int fingerid, int msgId, String msgstr){

Log.i(TAG, "onIdentifyRsp:"

+ " result: " + result

+ " fingerid: " + fingerid

+ " msgId: " + msgId

+ " msgstr: " + msgstr);

switch(result)

{

case MessageType.TINNO\_MSG\_VERIFY\_SUCCESS:

case MessageType.TINNO\_MSG\_VERIFY\_IDENTIFY\_WAKEUP\_MATCHED:

Log.d(TAG,"~~~~ onIdentifyRsp success ~~~~~");

if(mVerifyRunnable != null){

mVerifyHandler.removeCallbacks(mVerifyRunnable);

mVerifyRunnable = null;

}

//if unlock success, remove this callback.

if(mScreenOnOffCheckdVerifyRunnable != null){

mVerifyHandler.removeCallbacks(mScreenOnOffCheckdVerifyRunnable);

mScreenOnOffCheckdVerifyRunnable = null;

}

if(startQuickBootFunc(fingerid)){

break;

}

if(unLockScreen(fingerid)){

break;

}

Log.d(TAG,"~~~~ restart finger verify! ~~~~~");

fingerVerifyStart(0,EncryUtil.TAG\_VERIFY\_SUCCESS);

break;

case MessageType.TINNO\_MSG\_VERIFY\_FAILED:

case MessageType.TINNO\_MSG\_SYSTEM\_ERROR:

showFingerUnMatchMessage(msgId);

break;

case MessageType.TINNO\_MSG\_VERIFY\_ERROR\_LOCKOUT:

onIdentifyErrorLockScreen();

break;

default:

Log.d(TAG,"Shit! what happened ???");

break;

}

}

private void startAppFromQuickBoot(String packageName, String className) {

// start app

Log.i(TAG, " begin to startAppFromQuickBoot! : " +packageName);

Message msg = mVerifyHandler.obtainMessage();

Bundle bundle = new Bundle();

bundle.putString(KEY\_PACKAGE\_NAME, packageName);

bundle.putString(KEY\_CLASS\_NAME, className);

msg.setData(bundle);

msg.what = MessageType.TINNO\_MSG\_START\_APP;

mVerifyHandler.sendMessageDelayed(msg, SHOW\_PASSWORD\_ACTIVITY\_DELAY);

}

private void makeCallFromQuickBoot(String phoneNumber) {

//make call

Log.i(TAG, " begin to makeCallFromQuickBoot! : " +phoneNumber);

Message msg = mVerifyHandler.obtainMessage();

Bundle bundle = new Bundle();

bundle.putString(KEY\_PHONE\_NUMBER, phoneNumber);

msg.setData(bundle);

msg.what = MessageType.TINNO\_MSG\_MAKE\_CALL;

mVerifyHandler.sendMessageDelayed(msg, SHOW\_PASSWORD\_ACTIVITY\_DELAY);

}

public boolean startQuickBootFunc(int fingerId)

{

final QuickBootData quickBootData =

mFingerPrintDataUtils.getQuickBootDataFromFingerData(fingerId);

String packageName,className;

String phoneNumber;

int quickBootStartOnOff;

if(quickBootData == null) {

Log.e(TAG, " startQuickBootFunc -quickBootData is null! " );

return false;

}

if(quickBootData.isQuickBootDataEmpty()){

Log.i(TAG, "isQuickBootDataEmpty is null!");

return false;

}

if(!quickBootData.isAppEmpty()){

className = quickBootData.getClassName();

packageName = quickBootData.getPackageName();

startAppFromQuickBoot(packageName, className);

return true;

}

if(!quickBootData.isPhoneEmpty()){

phoneNumber = quickBootData.getPhoneNumber();

makeCallFromQuickBoot(phoneNumber);

return true;

}

Log.i(TAG, "Shit! may be has same error ???");

return false;

}

fp\_native.cpp

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <errno.h>

#include "jni.h"

#include <pthread.h>

#include "fp.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#undef LOG\_TAG

#define LOG\_TAG "[Tinnofingerprint-jni]"

#define TAG LOG\_TAG

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct JNI\_data {

JavaVM \*g\_vm;

JNIEnv \*g\_env;

jobject g\_thiz;

jclass clazz;

jmethodID cb;

int loop;

} ;

static JNI\_data mJNI\_data;

static JNI\_data \*mJdata\_p = &mJNI\_data;

static pthread\_t mThd;

//java class name

static const char \*classPathName = "com/ape/encryptmanager/event/Fp\_native";

int get\_ipo\_state(void)

{

char value[100];

memset(value, 0, sizeof(value));

\_\_system\_property\_get("sys.ipo.shutdown", value);

LOGD(LOG\_TAG "%s:%s : ipo.shutdown: %s\n", \_\_FILE\_\_,\_\_func\_\_, value);

return (strcmp(value, "1") == 0);

}

static void \*monitor\_thread(void \*args) {

int evt = 0;

JNI\_data \*mJdata = (JNI\_data \*)args;

LOGD(TAG "[%s]-Entry--\n",\_\_func\_\_);

if(mJdata == NULL) {

LOGE("err: JNI\_data is NULL!--/n");

goto exit;

}

mJdata->g\_vm->AttachCurrentThread(&mJdata->g\_env, NULL);

do {

if(fp\_update\_finger\_state(&evt) < 0){

continue;

}

if((evt == 1) || (evt == 2)) {

mJdata->g\_env->CallIntMethod(mJdata->g\_thiz, mJdata->cb, evt);

}

}

while(mJdata->loop);

mJdata->g\_env->DeleteGlobalRef(mJdata->g\_thiz);

mJdata->g\_vm->DetachCurrentThread();

exit:

LOGD(TAG "[%s]-Exit--\n",\_\_func\_\_);

pthread\_exit(NULL);

return NULL;

}

static int native\_init(JNIEnv \*env, jobject object, jint set) {

LOGD(TAG "[%s]-Entry--\n",\_\_func\_\_);

if(mJdata\_p == NULL)

{

LOGD(TAG "mJdata\_p is NULL\n");

goto error;

}

if(mJdata\_p->loop)

{

LOGD(TAG "thread-loop runing!--\n");

goto error;

}

if(fp\_dev\_init() < 0)

{

LOGD("fp\_dev\_init : fail!--/n");

goto error;

}

mJdata\_p->loop = set;

return 0;

error:

LOGE("%s: err!/n",\_\_func\_\_);

return -1;

}

static int native\_update(JNIEnv \*env, jobject object) {

int ret = 0;

fp\_update\_finger\_state(&ret);

return ret;

}

static int native\_setObj(JNIEnv \*env, jobject object,jobject object1) {

int ret = 0;

if(mJdata\_p->loop == 0) {

return ret;

}

if(env->GetJavaVM(&mJdata\_p->g\_vm) != 0) {

LOGE("GetJavaVM NULL!--/n");

goto error;

}

mJdata\_p->g\_thiz = env->NewGlobalRef(object1);

if(mJdata\_p->g\_thiz == NULL) {

LOGE("mJdata\_p->g\_thiz is NULL!--/n");

goto error;

}

mJdata\_p->clazz = env->GetObjectClass(mJdata\_p->g\_thiz);

if(mJdata\_p->clazz == NULL) {

LOGE("GetObjectClass is NULL!--/n");

goto error;

}

mJdata\_p->cb = env->GetMethodID(mJdata\_p->clazz, "onEventReport", "(I)I");

if(mJdata\_p->cb == NULL) {

LOGE("GetMethodID is NULL!--/n");

goto error;

}

mJdata\_p->g\_env = env;

if(pthread\_create(&mThd, NULL, monitor\_thread, (void \*)mJdata\_p)) {

LOGE("pthread\_create : fail!--/n");

goto error;

}

return ret;

error:

LOGE("%s: err!/n",\_\_func\_\_);

return -1;

}

static JNINativeMethod methods[] = {

{ "native\_init", "(I)I", (void \*)native\_init },

{ "native\_update", "()I", (void \*)native\_update },

{ "native\_setObj", "(Lcom/ape/encryptmanager/event/Fp\_native;)I", (void \*)native\_setObj },

};

/\*

\* Register several native methods for one class.

\*/

static int registerNativeMethods(JNIEnv\* env, const char\* className,

JNINativeMethod\* gMethods, int numMethods) {

jclass clazz;

clazz = env->FindClass(className);

if (clazz == NULL) {

LOGE("clazz is NULL!n");

return JNI\_FALSE;

}

if (env->RegisterNatives(clazz, gMethods, numMethods) < 0) {

LOGE("RegisterNatives fail: %s,%d/n",className,numMethods);

return JNI\_FALSE;

}

return JNI\_TRUE;

}

static int registerNatives(JNIEnv\* env) {

if (!registerNativeMethods(env, classPathName,

methods, sizeof(methods) / sizeof(methods[0]))) {

return JNI\_FALSE;

}

return JNI\_TRUE;

}

typedef union {

JNIEnv\* env;

void\* venv;

} UnionJNIEnvToVoid;

jint JNI\_OnLoad(JavaVM\* vm, void\* reserved) {

UnionJNIEnvToVoid uenv;

uenv.venv = NULL;

jint result = -1;

JNIEnv\* env = NULL;

if (vm->GetEnv(&uenv.venv, JNI\_VERSION\_1\_4) != JNI\_OK) {

goto bail;

}

env = uenv.env;

if (registerNatives(env) != JNI\_TRUE) {

LOGE("JNI\_OnLoad-fail!/n");

goto bail;

}

result = JNI\_VERSION\_1\_4;

bail:

return result;

}

fp.c

#include <string.h>

#include <fcntl.h>

#include <stdlib.h>

#include <stdio.h>

#include <errno.h>

#include <stdint.h>

#include <sys/stat.h>

#include <sys/ioctl.h>

#include <linux/input.h>

#include <poll.h>

#include <semaphore.h>

#include <signal.h>

#include "fp.h"

#undef LOG\_TAG

#define LOG\_TAG "[Tinnofingerprint-fp.c] "

#define TAG LOG\_TAG

static char \_product\_info[512] = {0x00};

static char \_platform\_info[512] = {0x00};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int fp\_vendor = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int cmp\_plf(char \* str)

{

return strcasecmp(\_platform\_info,str);

}

int cmp\_prd(char \* str)

{

return strcasecmp(\_product\_info,str);

}

int fp\_dev\_init(void)

{

int ret = -1;

LOGD(TAG"[guomingyi][%s]\n", \_\_func\_\_);

getFpInfo();

\_\_system\_property\_get("ro.target",\_product\_info);

\_\_system\_property\_get("ro.board.platform", \_platform\_info);

LOGD(TAG "[%s]--->product\_info:[%s]!\n",\_\_func\_\_,\_product\_info);

LOGD(TAG "[%s]--->platform\_info:[%s]!\n",\_\_func\_\_,\_platform\_info);

switch(fp\_vendor)

{

case VENDOR\_GOODIX:

ret = goodix\_dev\_init();

break;

case VENDOR\_ELAN:

ret = elan\_dev\_init();

break;

case VENDOR\_SILEAD:

ret = silead\_dev\_init();

break;

default:

LOGD("Err:[%s]/n", \_\_func\_\_);

break;

}

return ret;

}

int fp\_dev\_uninit(void)

{

int ret = -1;

LOGD(TAG"[guomingyi][%s]\n", \_\_func\_\_);

switch(fp\_vendor)

{

case VENDOR\_GOODIX:

ret = goodix\_dev\_uninit();

break;

case VENDOR\_ELAN:

ret = elan\_dev\_uninit();

break;

case VENDOR\_SILEAD:

ret = silead\_dev\_uninit();

break;

default:

LOGD("Err:[%s]/n", \_\_func\_\_);

break;

}

return ret;

}

int fp\_dev\_resume(void)

{

int ret = -1;

LOGD(TAG"[guomingyi][%s]\n", \_\_func\_\_);

switch(fp\_vendor)

{

case VENDOR\_GOODIX:

ret = goodix\_dev\_resume();

break;

case VENDOR\_ELAN:

break;

case VENDOR\_SILEAD:

break;

}

return ret;

}

int fp\_update\_finger\_state(int \*s)

{

int ret = -1;

switch(fp\_vendor)

{

case VENDOR\_GOODIX:

ret = goodix\_update\_finger\_state(s);

break;

case VENDOR\_ELAN:

ret = elan\_update\_finger\_state(s);

break;

case VENDOR\_SILEAD:

ret = silead\_update\_finger\_state(s);

break;

default:

LOGD("Err:[%s]/n", \_\_func\_\_);

break;

}

return ret;

}

int getFpInfo(void)

{

char buf[50];

int fd = -1;

int v = UNKNOW;

int ret;

LOGD(TAG "%s \n", \_\_func\_\_);

if(fp\_vendor != UNKNOW)

{

LOGD(TAG "[%s]:fp\_vendor:%d\n", \_\_func\_\_, fp\_vendor);

return fp\_vendor;

}

memset(buf, 0, sizeof(buf));

if((fd = open(FP\_DEV\_ATTR, O\_RDONLY)) > 0)

{

ret = read(fd, buf, sizeof(buf));

LOGD(TAG "read file result:%d,fp\_drv: %s\n", ret, buf);

if(strcmp(buf, FP\_DRV\_GOODIX) == 0) {

LOGD(TAG "match: /drv/%s Ok!\n",FP\_DRV\_GOODIX);

v = VENDOR\_GOODIX;

}

else

if(strcmp(buf, FP\_DRV\_SILEAD) == 0) {

LOGD(TAG "match: /drv/%s Ok!\n",FP\_DRV\_SILEAD);

v = VENDOR\_SILEAD;

}

else

if(strcmp(buf, FP\_DRV\_ELAN) == 0) {

LOGD(TAG "match: /drv/%s Ok!\n",FP\_DRV\_ELAN);

v = VENDOR\_ELAN;

}

close(fd);

}

fp\_vendor = v;

return v;

}