MagicFIDO Android SDK 개발가이드

Version 1.2.13

DREAM SECURITY
Digital Trust

제.개정 이력

개정번호	내용	개정일자
1.2.13	rpContextTable 변경 - startRegistration() - startAuthentication() - startDeregistration()	2020.07.09
1.2.12	setAuthenticatorOptions API 설정 값 추가 - KEY_USE_NUMBER_BLANK	2020.06.23
1.2.11	사용하지 않는 에러코드 제외	2020.03.10
1.2.10	setAuthenticatorOptions API 패스코드 옵션 설정 값 추가 - KEY_SHUFFLE_DISABLE	2020.02.10
1.2.9	Authentication 클래스 API 추가 - setDivideAuthStep() - resumeAuthentication()	2019.08.23
1.2.8	Manifest 등록 내용 변경	2019.07.26
1.2.7	MagicFidoUtil 클래스 setPasscodeRegularExpressionList API 추가	2019.04.26
	setAuthenticatorOptions API 설정 값 추가 - KEY_MAX_LOCK_COUNT	
	FidoResult Code 추가 - ERROR_MAX_LOCK_COUNT_OVER - IGNORE_MAX_LOCK_COUNT_SET	
1.2.6	MagicFidoUtil 클래스 setPasscodeIgnoreList() API 추가	2019.04.15
1.2.5	MagicFidoUtil 클래스 setKeypadAccessibilityOnSpeakerEnable() API 추가	2019.03.27
1.2.4	MagicFidoUtil 클래스 setPasscodeUIType() API 추가	2019.03.20
	안드로이드 스튜디오 라이브러리 등록 방법 변경	
1.2.3	ContextKey 안내 추가	2019.02.21
1.2.2	setAuthenticatorResetCallbackEnable() API 변경 FIDO UI 변경	2019.01.28
1.2.1	setLocalAuthFullScreen() API 추가	2019.01.23
	패스코드 옵션 설정 키값 추가 - KEY_REGULAR_EXPRESSION - KEY_LINE_WIDTH - KEY_LINE_COLOR - KEY_MISMATCH_LINE_COLOR	
	MagicFidoUtil 클래스 API 추가 - setAuthenticatorResetCallbackEnable() 추가	

	- setPasscodeMaxInputAutoNextStepEnable() 추가 - getVersion() 추가	
	Manifest 등록 내용 변경	
1.2.0	FIDO 화면 UI 추가	2018.12.13
1.0.13	setBioVerifyStateCheckEnable() API 추가	2018.11.05
1.0.12	등록, 인증, 해지 완료 후 사용자 ID 가져오는 옵션(getUserID()) 추가 Support-4 라이브러리 삭제	2018.11.02
1.0.11	패스코드 숫자키패드 사용 옵션 추가(setAuthenticatorOption())	2018.09.18
1.0.10	이클립스 라이브러리 등록 방법 제거	2018.09.07
1.0.9	FidoUtils 제거 및 변경, 초기화 Parameter 변경 Deregistration 설정 값 추가에 따른 API 추가 및 변경 setAuthenticatorOption() API 추가	2018.08.03
1.0.8	안드로이드 스튜디오 라이브러리 등록 방법 설명 추가	2018.07.05
1.0.7	targetSdkVersion, 난독화 시 주의사항 추가	2018.04.26
1.0.6	isAvailabileFido() API 대신 MagicFIDOUtil 에서 사용하는 방식으로 변경	2018.02.27
1.0.5	SSLEnable() 함수 추가	2017.11.19
1.0.4	FacetID 획득 방법 설명 추가	2017.11.17
1.0.3	에러코드 추가, deleteFIDOAllUser 리턴 값 수정, isAvailableFIDO 문서 수정.	2017.04.12
1.0.2	결과코드에 대한 유의사항 작성	2017.04.04
1.0.1	ResultCode 추가	2017.01.16
1.0.0	제정	2016.06.07

목 차

1. 개요

1.1 목적

본 문서는 FIDO 기술규격을 준수하여 InApp 방식으로 FIDO 인증기술을 적용할 수 있는 방법을 기술한다.

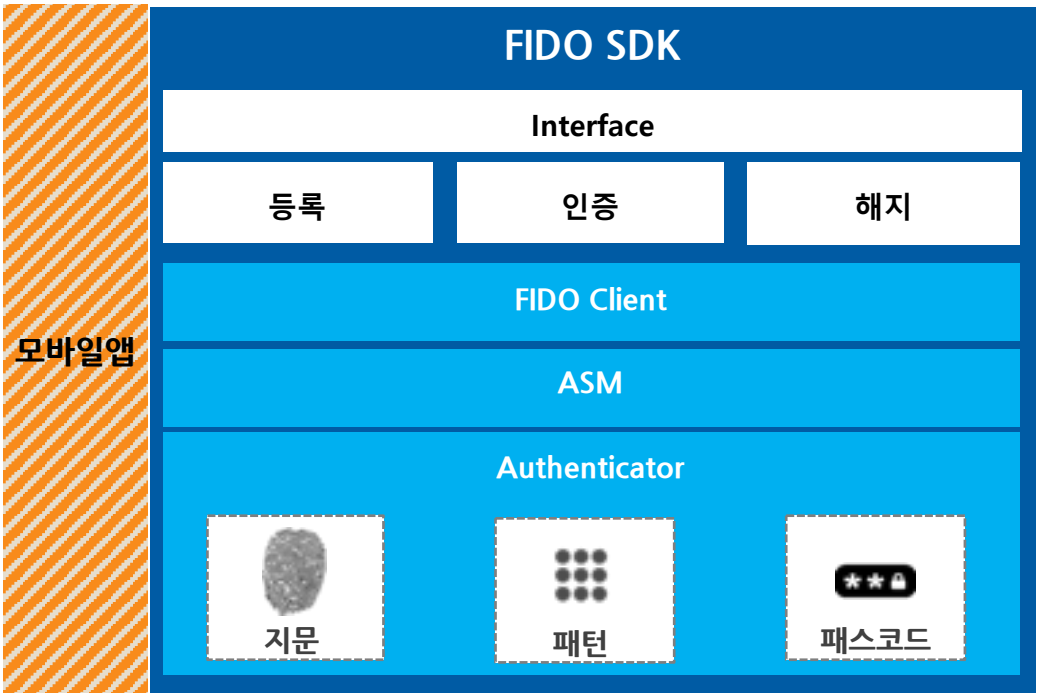
1.2 문서의 구성 및 범위

본 문서는 다음과 같은 사항에 대해 기술하며, 특별히 규정하지 않은 사항에 대해서는 관련 규격을 따른다.

- 패키지 구성 및 설치방법
- API 개발 가이드
- 사용예제

1.3 FIDO 모듈구성

FIDO 모듈은 FIDO SDK 로 구성되며 FIDO 를 적용할 모바일앱은 제공된 SDK 를 이용하여 인증장치 등록, 인증, 인증장치 해지를 간단하게 구현할 수 있다.



[그림 1] FIDO SDK 모듈구성

1.4 서비스 흐름도

FIDO 를 적용하려는 모바일앱에서 FIDO 시스템을 이용해 인증을 수행하려면 다음의 2 가지 과정을 순차적으로 진행해야 한다.

1. FacetID 를 FIDO 서버에 등록
2. keystore 내의 인증서를 FIDO SDK 에 전달 후 인증장치를 FIDO 서버에 등록

위의 과정을 정상적으로 진행했으면 FIDO 를 통해서 인증을 수행할 수 있으며, 사용자가 더 이상 FIDO 를 사용하지 않을 경우 제공된 API 를 통해서 인증장치를 해제할 수 있다.

1.4.1 FacetID 등록 방법

앱을 디버그 빌드하게 되면 기본적으로 Android SDK 설치시 생성되는 개발용 디버그 키스토어를 사용하여 앱을 사인하게 되며, 릴리즈 빌드를 할 경우에는 릴리즈용 키스토어로 하게 된다. 그러면 다른 FacetID 값이 다르기 때문에 테스트서버에는 디버그용 FacetID, 운영서버에는 운영 FacetID 를 등록하는 것을 권고한다.

* 앱내 자바 코드로 키해시 구하기

```
/**
 * RPClient의 FacetID 획득
 * @param context      Android's Context
 * @return FacetID
 */
public static String getFacetID(Context context) {
  try {
        // RPClient PackageInfo 획득
        PackageInfo info = context.getPackageManager().getPackageInfo(
                        context.getPackageName(), PackageManager.GET_SIGNATURES);

        // RPClient를 빌드 할 때 이용한 인증서 획득
        byte[] cert = info.signatures[0].toByteArray();
        InputStream input = new ByteArrayInputStream(cert);

        CertificateFactory cf = CertificateFactory.getInstance("X509");
        X509Certificate c = (X509Certificate) cf.generateCertificate(input);

        // Sha1 Hash(빌드 할 때 이용 한 인증서)
        MessageDigest md = MessageDigest.getInstance("SHA1");

        // FIDO1.0 FacetID 형식에 맞추어 만듦
        String rpfacetinfo = "android:apk-key-hash:"
        + Base64.encodeToString(md.digest(c.getEncoded()),
        Base64.DEFAULT | Base64.NO_WRAP | Base64.NO_PADDING);
        return rpfacetinfo;
```

```
    } catch (PackageManager.NameNotFoundException e) {
            e.printStackTrace();
    } catch (CertificateException e) {
            e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
    }
    return null;
}
```

2. 개발환경 설정

2.1 패키지 구성

- S/W FIDO 를 사용하기 위해, RPClient 프로젝트에 Lib_Magic_FIDO 라이브러리 프로젝트를 링크 또는 추가하여 사용 할 수있다. 아래의 설명은 S/W FIDO 를 제공하기 위한 패키지 구성이다.

구 조	파일명	설 명
Doc	MagicFIDO Android SDK 개발가이드 vx.x.x.pdf	본 문서
Libs	Lib_Magic_FIDO	FIDO SDK 라이브러리
	magicvkeypad	MagicVKeypad 라이브러리
Sample	DREAM_RPClient_Sample	일반 FIDO 연동 샘플 RP Client 프로젝트

2.2 라이브러리 프로젝트 설정

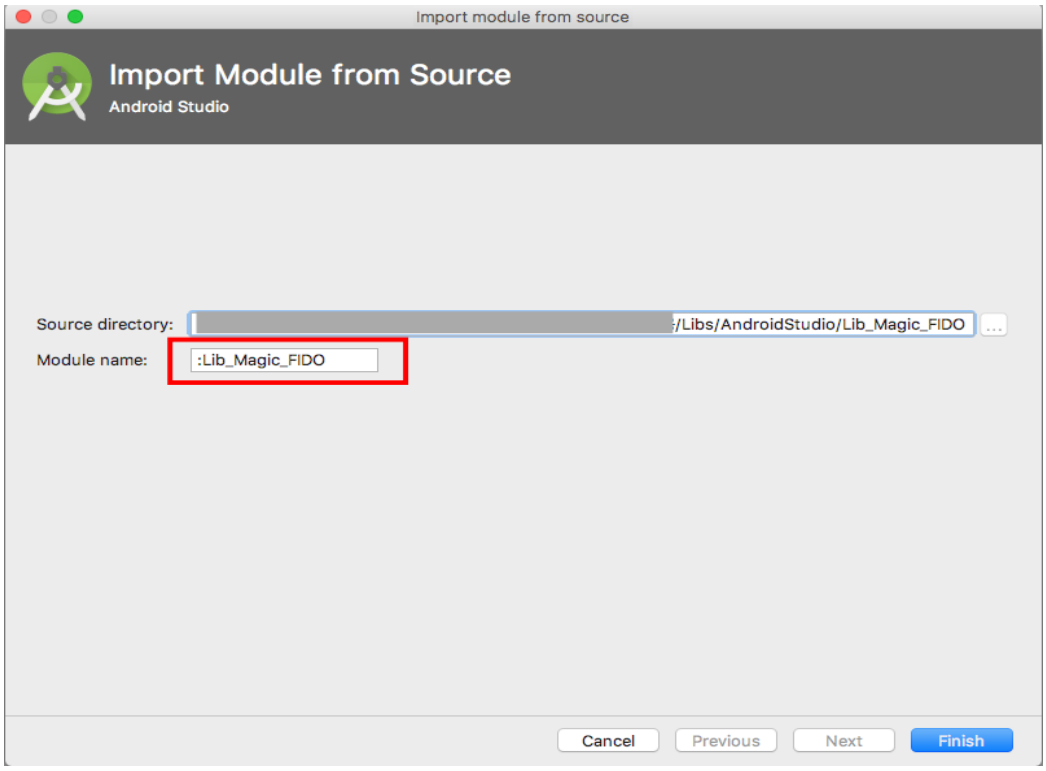
제공되는 SDK 는 라이브러리, 이미지 파일, 레이아웃 등을 쉽게 관리 하기 위해 라이브러리 프로젝트 형태로 제공 된다.

2.2.1 안드로이드 스튜디오 라이브러리 등록

- **Lib_Magic_FIDO**

FIDO 를 사용 할 수 있도록 제공된 SDK 는 라이브러리와 리소스 파일이 포함되어 있으며, RP Client 에 해당 모듈을 추가하여 사용한다.

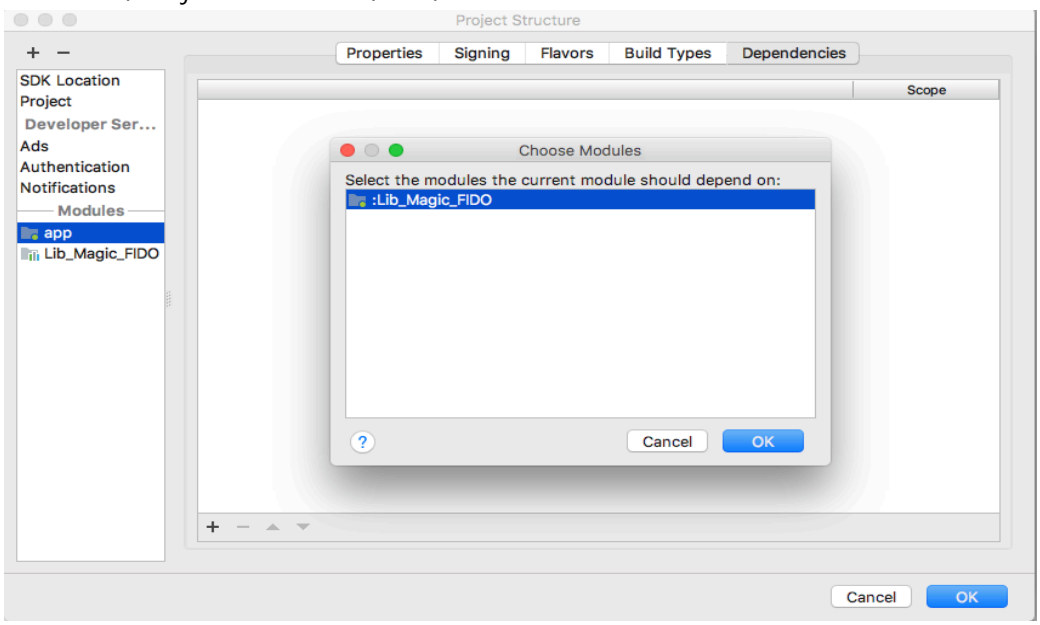
① File - New - Import Module 클릭



② Source directory: Libs/Lib_Magic_FIDO 경로 설정

③ Module name: ':Lib_Magic_FIDO' 나오는 지 확인 후 Finish 버튼 클릭

④ RP Client 의 Project Structure 에 들어감



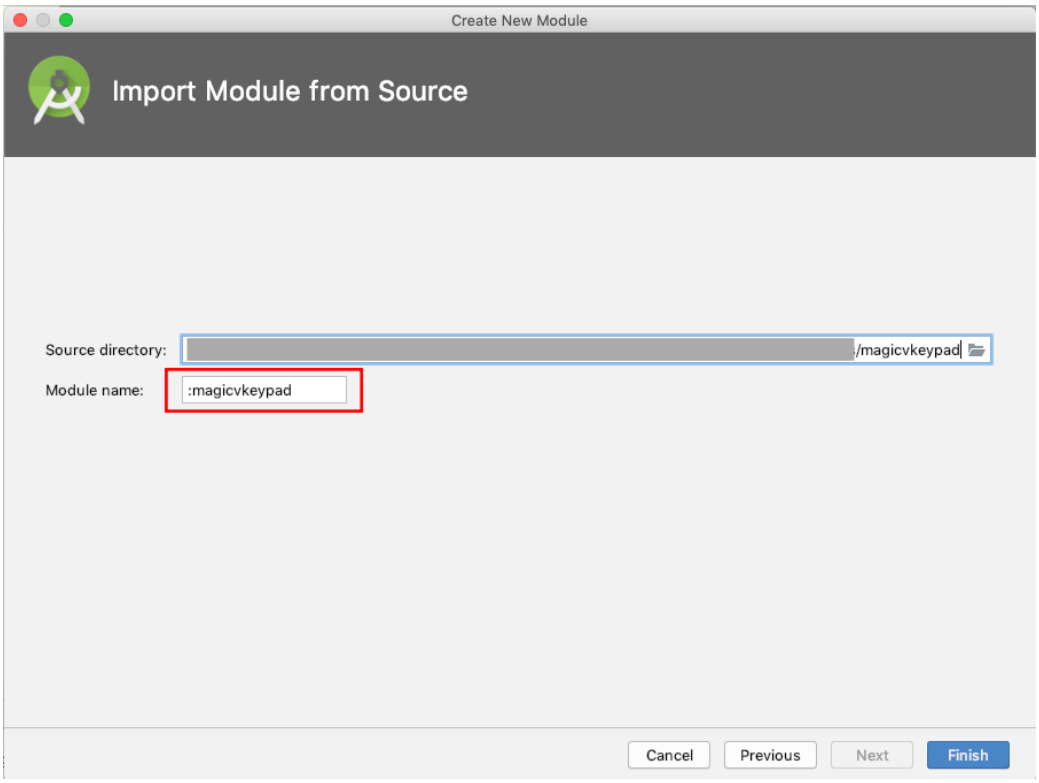
⑤ app – Dependencies – '+' 버튼 클릭 – 3. Module Dependency 클릭

⑥ 추가한 'Lib_Magic_FIDO' 모듈 선택 후 OK 버튼 클릭하여 추가

- **magicvkeypad**

  패스코드 인증장치에서 사용되는 보안 키패드 모듈로, 라이브러리와 리소스 파일이 포함되어 있으며 RP Client 에서 해당 모듈을 추가하여 사용한다..

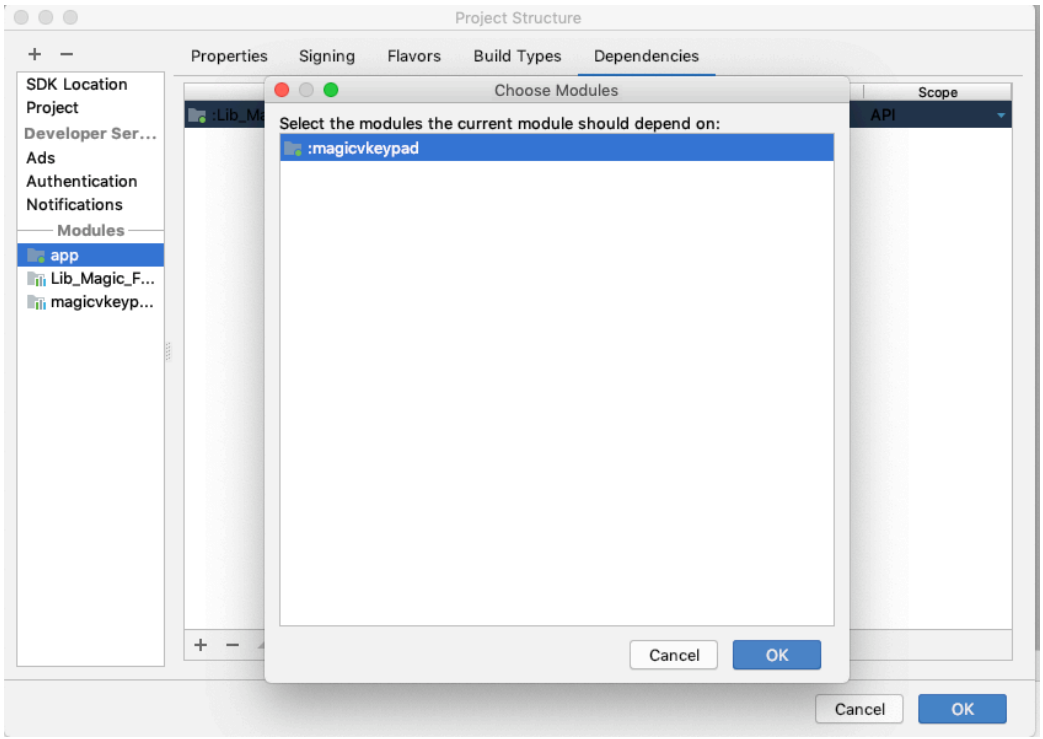
  ① File - New - Import Module 클릭



  ② Source directory: Libs/magicvkeypad 경로 설정

  ③ Module name: ': magicvkeypad'나오는 지 확인 후 Finish 버튼 클릭

  ④ RP Client 의 Project Structure 에 들어감



⑤ app – Dependencies – '+' 버튼 클릭 – 3. Module Dependency 클릭

⑥ 추가한 'magicvkeypad 모듈 선택 후 OK 버튼 클릭하여 추가

2.3 RP Client 에 AndroidManifest 등록

Lib_Magic_FIDO 가 동작 할 수 있도록 아래의 내용들을 RP Client 에 있는 AndroidManifest.xml 에 등록해야 한다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">

   .......
  <!-- FIDO SDK Permission -->
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.READ_PHONE_STATE" />
  <uses-permission android:name="android.permission.USE_FINGERPRINT" />
   <!-- FIDO SDK Activity -->
   <activity
        android:name="com.dream.magic.fido.client.asm.ui.AsmListActivity"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" >
   </activity>
   <activity

android:name="com.dream.magic.fido.authenticator.finger.local.UserLocalVerificationActivit
y"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" >
   </activity>
   <activity
```

```
        android:name="com.dream.magic.fido.authenticator.localActivity.SelectCertActivity"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" >
    </activity>
    <activity

android:name="com.dream.magic.fido.authenticator.finger.local.UserLocal_Normal_Activity"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" >
    </activity>
    <activity
        android:name="com.dreamsecurity.lib_passcode.ui.Passcode_UI"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@android:style/Theme.Translucent.NoTitleBar">
        </activity>
    <activity
        android:name="com.dreamsecurity.lib_passcode.ui.Passcode_UI_Square"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@android:style/Theme.Translucent.NoTitleBar">
    </activity>
    <activity
        android:name="com.dreamsecurity.pattern.DSRegisterPattern"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@style/PatternDialogStyle"></activity>
    <activity
        android:name="com.dreamsecurity.pattern.DSVerifyPattern"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@style/PatternDialogStyle"></activity>
    <activity
        android:name="com.dreamsecurity.pattern.DSChangePattern"
        android:configChanges="keyboardHidden|orientation|screenSize"
        android:theme="@style/PatternDialogStyle"></activity>
    <!-- FIDO SDK Activity -->
  .......
```

2.4 설정시 주의 사항

- **Permission 주의 사항**

1. FIDO 에서는 단말의 고유 번호인 IMEI 값을 확인 하기 위해서 폰상태에 대한 Permission 이 필요하다. 이러한 사유로 아래의 Permission 을 추가해야 하며, Android 6.0 에서는 Permission 에 대한 사용자의 제어가 가능 하기 때문에 FIDO 실행 시 Permission 이 허가가 되어 있는지 확인이 필요하다.

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

2. FIDO 에서는 지문에 대한 Permission 이 필요 하다. 이러한 사유로 아래의 Permission 을 추가해야 한다.

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

- **프로젝트에 적용 시 주의사항**
  - **targetSdkVersion 이 반드시 23 이상이어야 한다.**

• **프로젝트에 적용 후 proguard 난독화 시 주의사항**

- 제공되는 FIDO SDK 는 모두 난독화가 되어 배포되기 때문에 따로 난독화를 진행할 필요가 없으며, 배포된 SDK 를 난독화에서 제외하기 위해 proguard-rules 파일에 아래와 같이 설정해야 한다.

```
-keepattributes Signature
-dontwarn com.dream.magic.**
-keep class com.dream.magic.** {*;}
-dontwarn com.dreamsecurity.**
-keep class com.dreamsecurity.** {*;}
-dontwarn org.apache.http.**
-keep class org.apache.http.** {*;}
-keepattributes InnerClasses
-keep class **.R
-keep class **.R$* {
    <fields>;
}
```

2.5 Android FIDO 연계를 위한 사양 정의

- 안드로이드/iOS 지원단말 및 OS

구분	지원 OS	지원 단말
Android	6.0 ~	지문 인식 탑재 단말 (Swap 방식 단말기 제외)

- 지원 단말 여부에 따른 메뉴 활성화 처리

FIDO 지원단말 확인은 FIDO Client SDK 에서 제공해주는 isAvailableFIDO() API 를 호출해서 리턴 값이 true 이면 FIDO 메뉴를 활성화 하고 false 가 반환되면 해당 메뉴를 사용자에게 보이지 않도록 구성할 수 있다.

3 API Reference

본 API 는 FIDO 를 이용하기 위한 중요기능인 등록, 인증, 해지 기능을 제공하며 이를 원활하게 사용할 수 있도록 유틸, 에러확인 등의 클래스로 구성되어 있다. 등록, 인증, 해지에서 제공되는 API 는 순자적으로 호출되어야 한다.

3.1 Registration Class

3.1.1 Class Overview

인증장치를 FIDO 서버에 등록하기 위한 메시지 생성 및 분석하는 기능을 제공한다.

3.1.2 Summery

Public Constructors	
	Registration (Context context, FIDOCallbackResult callbackResult, String reqURL, String resURL) 클래스 생성자.

Public Methods	
void	setNetworkTimout (int connectTimeOut, int readTimeOut) Server 와의 응답 대기 시간을 설정한다. (Default 30 초)
void	disUseWaitDialog ( ) FIDO 서버와 통신 시 Indicator Dialog 를 Display 하지 않는다.
void	startRegistration (HashTable<Object, Object> rpContextTable) 인증장치 등록 요청 및 진행 한다.
String	getToken ( ) FIDO 서버로부터 전달 받은 Token 을 받는다.
String	getUserID ( ) FIDO 서버로부터 전달 받은 userID 를 받는다.

3.1.3 Public Constructor

public **Registration** (Context context, FIDOCallbackResult callbackResult, String reqURL, String resURL);

클래스 생성자.

Parameters

Context	Application Context

callbackResult	결과를 받을 수 있는 Callback Result
reqURL	정책 요청 FIDO 서버 URL
resURL	등록 요청 FIDO 서버 URL

3.1.4 Public Methods

public void **setNetworkTimout** (int connectTimeOut, int readTimeOut);

Server 와의 응답 대기 시간을 설정한다. (Default 30 초)

Parameters

connectTimeOut	FIDO 서버 접속 Timeout
readTimeOut	FIDO 서버 응답 Timeout

Remarks

startRegistration()을 호출 하기 전에 실행한다. 네트워크 타임아웃을 설정하지 않을 경우, 각각 30 초 시간으로 정해진다.

public void **disUseWaitDialog** ( );

FIDO 서버와 통신 시 Indicator Dialog 를 Display 하지 않는다.

Remarks

함수를 호출하지 않았을 경우에는 기본적으로 자동 사용한다.

public void **startRegistration** (HashTable<Object, Object> rpContextTable);

인증장치 등록 요청 및 진행 한다.

Parameters

rpContextTable	FIDO 서버에서 사용자를 구분하기 위한 HashTable (샘플코드 및 개발가이드의 '4. 예제' 참고)

Remarks

단말기에 따라 인증장치(지문, 패턴, 패스코드) 화면이 나타난다.

username 에는 ' '(공백) 문자 또는 한글을 지원하지 않음.

등록 요청 한 결과를 생성자에서 설정 한 FIDOCallbackResult 으로 리턴 받는다.

RequestCode 는 REQUEST_CODE_REGIST 으로 내려온다.

에러코드 ERROR_ALREADY_REGI_USER 를 받을 경우, 서버에는 요청한 아이디로 이미 등록 되어 있지만 단말에는 실제 값이 없기 때문에 "FIDO 해지" 진행 후 다시 재등록을 요청해야 한다.

ContextKey List

키	나타내는 값
<필수>	
ContextKeys.KEY_USERID	사용자 ID
<옵션>	
ContextKeys.KEY_DEVICEID	단말기 ID
ContextKeys.KEY_DEVICEMODEL	단말기 모델명
ContextKeys.KEY_DEVICEOS	단말기 OS
ContextKeys.KEY_AUTHCHECK	등록 시 인증 코드를 확인할 지 여부("true"로 설정)
ContextKeys.KEY_AUTHCODE	인증 코드

FIDOCallBack 에 대한 유의 사항

```
 private FIDOCallbackResult fidoResult = new FIDOCallbackResult() {
@Override
public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
  if(requestCode == FIDORequestCode.REQUEST_CODE_REGIST){
    if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
      // 결과 코드 0: 등록 성공
    }else{
      // 실패 사유
      switch(fidoResult.getErrorCode()){
        case FidoResult.RESULT_USER_CANCEL:
        // 에러코드 999: 사용자 취소
        break;
        case FidoResult.ERROR_CANNOT_USE_FIDO:
        // 에러코드 1007: 지원하지 않는 단말기
        break;
        case FidoResult.ERROR_UNTRUSTED_FACET_ID:
        // 에러코드 2001: -등록되지 않은 3rd App으로 진행
        //                - FIDO Server에 FacetID 를 등록해야 함.
        break;
        case FidoResult.ERROR_NETWORK_STATE:
        // 에러코드 4001: -네트워크 오류
        //                -fidoResult.getDescription() 을 통하여 자세한 오류 내용을 알 수 있음
        break;
        case FidoResult.PW_NOT_MATCH_BIO_FINGER:
```

```
            // 에러코드 5000: 지문 인증 실패
            break;
            case FidoResult.ERROR_UNKNOWN:
            // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
            break;
        }
    }
  }
});
```

public String **getToken** ( );

FIDO 서버로부터 전달 받은 Token 을 받는다.

Returns

Token

Remarks

인증장치 등록 메시지에 대한 결과가 성공일 경우에 값을 획득 할 수 있다.

public String **getUserID**( );

FIDO 서버로부터 전달 받은 UserID 를 받는다.

Returns

UserID

Remarks

인증장치 등록 메시지에 대한 결과가 성공일 경우에 값을 획득 할 수 있다.

3.2 Authentication Class

3.2.1 Class Overview

사용자 인증을 위한 메시지 생성 및 분석하는 기능을 제공한다.

3.2.2 Summery

Public Constructors	
	Authentication (Context context, FIDOCallbackResult callbackResult, String reqURL, String resURL) 클래스 생성자

Public Methods	
void	setNetworkTimout (int connectTimeOut, int readTimeOut) Server 와의 응답 대기 시간을 설정한다. (Default 30 초)
void	disUseWaitDialog ( ) FIDO 서버와 통신 시 Indicator Dialog 를 Display 하지 않는다.
void	StartAuthentication (HashTable<Object, Object > rpContextTable) 사용자 인증 요청 및 진행 한다.
void	StartAuthentication (HashTable< Object, Object > rpContextTable, FIDOCallbackResult callbackResult) 사용자 인증 요청 및 진행 한다.
String	getToken ( ) FIDO 서버로부터 전달 받은 Token 을 받는다.
String	getUserID ( ) FIDO 서버로부터 전달 받은 userID 를 받는다.
void	setBioVerifyStateCheckEnable (boolean enable) 생체 인증장치의 상태 값 변경에 대한 결과를 받을 수 있도록 설정한다.
void	setDivideAuthStep (boolean divideAuthStepEnable) 인증 과정의 단계를 나눠서 진행할 수 있도록 설정한다.
void	resumeAuthentication (Context context, FIDOCallbackResult callbackResult) 인증 과정의 다음 단계를 진행한다.

3.2.3 Public Constructor

public **Authentication** (Context context, FIDOCallbackResult callbackResult, String reqURL, String resURL);

클래스 생성자

Parameters

context	Context
callbackResult	결과를 받을 수 있는 Callback Result
reqURL	정책 요청 FIDO 서버 URL

resURL	인증 요청 FIDO 서버 URL

3.2.4 Public Methods

public void **setNetworkTimout** (int connectTimeOut, int readTimeOut);

Server 와의 응답 대기 시간을 설정한다. (Default 30 초)

Parameters

connectTimeOut	FIDO 서버 접속 Timeout
readTimeOut	FIDO 서버 응답 Timeout

Remarks

startAuthentication()을 호출 하기 전에 실행한다. 네트워크 타임아웃을 설정하지 않을 경우, 각각 30 초 시간으로 정해진다.

public void **disUseWaitDialog** ( );

FIDO 서버와 통신 시 Indicator Dialog 를 Display 하지 않는다.

Remarks

함수를 호출하지 않았을 경우에는 기본적으로 자동 사용한다.

public void **StartAuthentication** (HashTable<Object, Object> rpContextTable);

사용자 인증 요청 및 진행 한다.

Parameters

rpContextTable	FIDO 서버에서 사용자를 구분하기 위한 HashTable

Remarks

단말기에 따라 60 초 동안 인증장치(지문, 패턴, 패스코드) 화면이 나타난다.

username 에는 ' '(공백) 문자 또는 한글을 지원하지 않음.

인증 요청 한 결과를 생성자에서 설정 한 FIDOCallbackResult 으로 리턴 받는다.

RequestCode 는 REQUEST_CODE_AUTH 으로 내려온다.

에러코드 ERROR_NOT_EXIST_USER_DATA(1009) 을 받을 경우, 서버에는 요청한 아이디로 이미 등록되어 있지만 단말에는 실제 값이 없기 때문에 "FIDO 해지" 진행 후 다시 재등록을 요청해야 한다.

ContextKey List

키	나타내는 값
<필수>	

ContextKeys.KEY_USERID	사용자 ID
<옵션>	
ContextKeys.KEY_DEVICEID	단말기 ID
ContextKeys.KEY_DEVICEMODEL	단말기 모델명
ContextKeys.KEY_DEVICEOS	단말기 OS
ContextKeys.KEY_TRANSACTIONTEXT	Transaction 확인 텍스트
ContextKeys.KEY_SESSIONCHECK	인증시 세션 체크 여부("true"로 설정)

FIDOCallBack 에 대한 유의 사항

```
private FIDOCallbackResult fidoCallbackResult = new FIDOCallbackResult() {
 @Override
 public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
   if(requestCode == FIDORequestCode.REQUEST_CODE_AUTH) {
     if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
       // 결과 코드 0: 등록 성공
     }else{
       // 실패 사유
       switch(fidoResult.getErrorCode()){
       case FidoResult.RESULT_USER_CANCEL:
       // 에러코드 999: 사용자 취소
       break;
       case FidoResult.ERROR_CANNOT_USE_FIDO:
       // 에러코드 1007: 지원하지 않는 단말기
       break;
       case FidoResult.ERROR_NOT_EXIST_USER_DATA:
       // 에러코드 1009: -인증하려는 아이디의 데이터가 존재하지 않는 경우
       //                -동일 아이디로 해지 후 재등록 요청 해야 함
       //                -단말에 사용자의 데이터가 없기 때문에, Deregstration의
       //                 startDeregistration()을 이용하여 해지 후 재등록 진행
       break;
       case 13006:
       // 에러코드 13006: -인증 시 등록 된 아이디가 없는 사유로 서버에서 전달하는 에러코드
       //                 - Regsitration의 startRegistration()을 이용하여 등록 요청 진행
       break;
       case 19019:
       // 에러코드 19019: -아이디 누락으로 인한 실패 사유로 서버에서 전달하는 에러코드
       //                 -인증 요청 할 때, UserName을 입력하지 않음.
       //                  userName을 넣은 후 진행.
```

```
        break;
        case FidoResult.ERROR_UNTRUSTED_FACET_ID:
        // 에러코드 2001: -등록되지 않은 3rd App으로 진행
        //                - FIDO Server에 FacetID 를 등록해야 함.
        break;
        case FidoResult.ERROR_NO_SUITABLE_AUTHENTICATOR:
        // 에러코드 2002: - 적합한 인증장치가 없음
        //                - Server에 등록 되어 있지만, 단말기에 해당하는 데이터가 없음.
        //                - 동일 아이디로 Deregistration의 startDeregistration()을 이용하여
        //                  해지 후 재등록 요청 해야 함.
        break;
        case FidoResult.ERROR_NETWORK_STATE:
        // 에러코드 4001: -네트워크 오류
        //                  -fidoResult.getDescription() 을 통하여 자세한 오류 내용을 알 수 있음
        break;
        case FidoResult.PW_NOT_MATCH_BIO_FINGER:
        // 에러코드 5000: 지문 인증 실패
        break;
        case FidoResult.ERROR_UNKNOWN:
        // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
        break;
        }
      }
    }
});
```

public void **StartAuthentication** (HashTable<Object, Object> rpContextTable, FIDOCallbackResult callbackResult);

사용자 인증 요청 및 진행 한다.

Parameters

rpContextTable	FIDO 서버에서 사용자를 구분하기 위한 HashTable
callbackResult	결과를 받을 수 있는 Callback Result

Remarks

StartAutentication(HashTable<String, String> rpContextTable) API 와 동일한 API 로,
FIDO 결과를 받을 수 있는 Callback 의 변경이 필요 할 때 사용한다.

자세한 내용은 StartAuthentication(rpContextTable) API 참고

public String **getToken** ( );

FIDO 서버로부터 전달 받은 Token 을 받는다.

Returns

Token

Remarks

사용자 인증 메시지에 대한 결과가 성공일 경우에 값을 획득 할 수 있다.

public String **getUserID** ( );

FIDO 서버로부터 전달 받은 UserID 를 받는다.

Returns

UserID

Remarks

인증장치 인증 메시지에 대한 결과가 성공일 경우에 값을 획득 할 수 있다.

public void **setBioVerifyStateCheckEnable** (boolean enable);

생체 인증장치의 상태 값 변경에 대한 결과를 받을 수 있도록 설정한다.

Parameters

enable	활성화 여부 (default : false) true : 활성화 / false : 비활성화

Remarks

생체인증장치(지문)의 값이 FIDO 등록 했을 때와 비교하여 변경 되었을 경우 FIDOCallbackResult 로 상태변경 결과값(FidoResult.*BIO_STATE_CHANGED*) 내려온다.

public void **setDivideAuthStep** (boolean divideAuthStepEnable);

인증 과정의 단계를 나눠서 진행할 수 있도록 설정한다.

Parameters

divideAuthStepEnable	활성화 여부 (default : false) true : 활성화 / false : 비활성화

Remarks

인증 과정을 실제 사용자 인증 화면발생 전/후로 나누어 진행하도록 설정할 수 있다.
StartAuthentication 호출 전에 설정해야한다.

public void **resumeAuthentication** (Context context, FIDOCallbackResult callbackResult);

인증 과정의 다음 단계를 진행한다.

Parameters

context	Context
callbackResult	결과를 받을 수 있는 Callback Result

Remarks

setDivideAuthStep이 true가 아닐 경우 무시되며, startAuthentication API가 호출된 이후 호출되어야 한다.
startAuthentication()의 결과로 FidoResult. RESULT_AUTH_READY_SUCCESS 를 반환하기 전에 resumeAuthentication() API가 호출될 경우 콜백을 주지 않고 바로 이어서 인증이 진행된다.

3.3 Deregistration Class

3.3.1 Class Overview

인증장치를 FIDO 서버에서 해지하기 위한 메시지 생성 및 분석하는 기능을 제공한다.

3.3.2 Summery

Public Constructors	
	Deregistration (Context context, FIDOCallbackResult callbackResult, String reqURL) 클래스 생성자.

Public Methods	
void	setNetworkTimout (int connectTimeOut, int readTimeOut) Server 와의 응답 대기 시간을 설정한다. (Default 30 초)
void	disUseWaitDialog ( ) FIDO 서버와 통신 시 Indicator Dialog 를 Display 하지 않는다.
void	StartDeregistration (HashTable<Object, Object> rpContextTable) 인증장치 해지 요청 메시지를 생성한다.
void	StartDeregistrationAll (HashTable<Object, Object> rpContextTable)

	전체 인증장치 해지 요청 메시지를 생성한다.
String	getToken ( ) 서버로부터 전달 받은 Token 을 받는다.
String	getUserID ( ) FIDO 서버로부터 전달 받은 userID 를 받는다.

3.3.3 Public Constructor

public **Deregistration** (Context context, FIDOCallbackResult callbackResult, String reqURL);

클래스 생성자.

Parameters

Context	Context
callbackResult	결과를 받을 수 있는 Callback Result
reqURL	RP Server 에 최초 요청 URL

3.3.4 Public Methods

public void **setNetworkTimout** (int connectTimeOut, int readTimeOut);

Server 와의 응답 대기 시간을 설정한다. (Default 30 초)

Parameters

connectTimeOut	RP Server 에 접속 대기 시간
readTimeOut	RP Server 에 응답 대기 시간

Remarks

startDeregistration()을 호출 하기 전에 실행한다. 네트워크 타임아웃을 설정하지 않을 경우, 각각 30 초 시간으로 정해진다.

public void **disUseWaitDialog** ( );

FIDO 서버와 통신 시 Indicator Dialog 를 Display 하지 않는다.

Remarks

함수를 호출하지 않았을 경우에는 기본적으로 자동 사용한다.

public void **StartDeregistration** (HashTable<Object, Object> rpContextTable);

인증장치 해지 요청 메시지를 생성한다.

Parameters

rpContextTable FIDO 서버에서 사용자를 구분하기 위한 HashTable

Remarks

username 에는 ' '(공백) 문자 또는 한글을 지원하지 않음.

인증 요청 한 결과를 생성자에서 설정 한 FIDOCallbackResult 으로 리턴 받는다.

RequestCode 는 REQUEST_CODE_DEREG 으로 내려온다.

ContextKey List

키	나타내는 값
<필수>	
ContextKeys.KEY_USERID	사용자 ID
<옵션>	
ContextKeys.KEY_DEVICEID	단말기 ID
ContextKeys.KEY_DEVICEMODEL	단말기 모델명
ContextKeys.KEY_DEVICEOS	단말기 OS

FIDOCallBack 에 대한 유의 사항

```
private FIDOCallbackResult fidoCallbackResult = new FIDOCallbackResult() {
 @Override
public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
  if(requestCode == FIDORequestCode.REQUEST_CODE_AUTH) {
    if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
      // 결과 코드 0: 등록 성공
    }else{
      // 실패 사유
      switch(fidoResult.getErrorCode()){
      case FidoResult.RESULT_USER_CANCEL:
      // 에러코드 999: 사용자 취소
      break;
      case FidoResult.ERROR_CANNOT_USE_FIDO:
      // 에러코드 1007: 지원하지 않는 단말기
      break;
      case 13006:
      // 에러코드 13006: -인증 시 등록 된 아이디가 없는 사유로 서버에서 전달하는 에러코드
      //                 - Regsitration의 startRegistration()을 이용하여 등록 요청 진행
```

```
        break;
        case 19019:
        // 에러코드 19019: -아이디 누락으로 인한 실패 사유로 서버에서 전달하는 에러코드
        //                  -인증 요청 할 때, UserName을 입력하지 않음.
        //                   userName을 넣은 후 진행.
        break;
        case FidoResult.ERROR_UNTRUSTED_FACET_ID:
        // 에러코드 2001: -등록되지 않은 3rd App으로 진행
        //                - FIDO Server에 FacetID 를 등록해야 함.
        break;
        case FidoResult.ERROR_NETWORK_STATE:
        // 에러코드 4001: -네트워크 오류
        //                 -fidoResult.getDescription() 을 통하여 자세한 오류 내용을 알 수 있음
        break;
        case FidoResult.PW_NOT_MATCH_BIO_FINGER:
        // 에러코드 5000: 지문 인증 실패
        break;
        case FidoResult.ERROR_UNKNOWN:
        // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
        break;
        }
      }
    }
});
```

public void **StartDeregistrationAll** (HashTable<Object, Object> rpContextTable);

전체 인증장치 해지 요청 메시지를 생성한다.

Parameters

rpContextTable　　FIDO 서버에서 사용자를 구분하기 위한 HashTable

Remarks

username 에는 ' '(공백) 문자 또는 한글을 지원하지 않음.

인증 요청 한 결과를 생성자에서 설정 한 FIDOCallbackResult 으로 리턴 받는다.

RequestCode 는 REQUEST_CODE_DEREG 으로 내려온다.

FIDOCallBack 에 대한 유의사항은 StartDeregistration() 과 동일하다.

public String **getToken** ( );

FIDO 서버로부터 전달 받은 Token 을 받는다.

Returns

Token

Remarks

인증장치 해지 요청 완료 후 값을 획득 할 수 있다.

public String **getUserID**( );

FIDO 서버로부터 전달 받은 UserID 를 받는다.

Returns

UserID

Remarks

인증장치 해지 메시지에 대한 결과가 성공일 경우에 값을 획득 할 수 있다.

3.4 FIDOCallbackResult Interface

3.4.1 Interface Overview

FIDO 의 등록, 인증, 해지에 대한 수행결과 Callback Interface

3.4.2 Summery

Public Interface	
void	onFIDOResult (int requestCode, boolean result, FidoResult fidoResult); FIDO 수행 결과에 대한 결과

3.5 FIDORequestCode Class

3.5.1 Class Overview

FIDO 를 실행 후 Callback 받아 구별 할 수 있는 RequestCode.

3.5.2 Summery

Public Values	
REQUEST_CODE_REGIST	FIDO 등록(Registration) 실행에 대한 식별 코드
REQUEST_CODE_AUTH	FIDO 인증(Authentication) 실행에 대한 식별 코드
REQUEST_CODE_DEREG	FIDO 해지(Deregistration) 실행에 대한 식별 코드
REQUEST_LOCAL_CHANGE_VALUE	인증장치(패턴, 패스코드)에 대하여 설정값 변경 실행에 대한 식별 코드
REQUEST_LCOAL_RESET	인증장치(패턴, 패스코드)에 대하여 설정값 초기화 실행에 대한 식별 코드

3.6 FIDOResult Class

3.6.1 Class Overview

실행 결과에 대한 클래스

3.6.2 Summary

Values		
RESULT_SUCCESS	0	성공
RESULT_AUTH_READY_SUCCESS	10	사용자 인증 준비 완료
RESULT_USER_CANCEL	999	사용자 취소

ERROR_INVALID_PARAMETER	1001	전달 인자값이 올바르지 않습니다.
ERROR_SET_CHANNEL_BINDING	1002	TLS 인증서가 올바르지 않습니다.
ERROR_INVALID_CERT_PATH	1004	FIDO 가 설치되어 있지 않습니다.
ERROR_JSON_PARSER	1005	JSON 메세지 분석 중 오류가 발생하였습니다.
ERROR_NOT_MATCH_FIDO_CLIENT	1006	FIDO Client 가 설치되어 있지 않습니다.
ERROR_CANNOT_USE_FIDO	1007	FIDO 를 사용 할 수 없는 단말기 입니다.
ERROR_ALREADY_REGI_USER	1008	이미 등록 된 사용자 입니다. 해지 후 재등록하여 사용해 주시기 바랍니다. (Deregistration 의 startDeregistration()을 이용하여 해지 후 재등록 하시기 바랍니다.)
ERROR_NOT_EXIST_USER_DATA	1009	사용자의 데이터가 없습니다. 해지 후 재등록하여 사용해 주시기 바랍니다. (Deregistration 의 startDeregistration()을 이용하여 해지 후 재등록 하시기 바랍니다.)
ERROR_UNACCEPTED_PERMISSION	1010	퍼미션이 허가되지 않았습니다.
ERROR_INVALID_USERNAME	1011	username 에는 ' '(공백) 문자 또는 한글을 지원하지 않음
ERROR_AUTH_REQUEST_NOT_EXIST	1100	인증요청이 없습니다.
ERROR_UNTRUSTED_FACET_ID	2001	인증되지 않은 RP Client 입니다.
ERROR_NO_SUITABLE_AUTHENTICATOR	2002	적합한 인증장치가 설치되어 있지 않습니다. (등록 시 이미 등록 된 아이디 이고, 인증 시 Deregistration 의 startDeregistration()을 이용하여 해지 후 재등록 하시기 바랍니다.)
ERROR_UNSUPPORTED_VERSION	2003	지원하지 않는 FIDO 프로토콜 버전입니다.
ERROR_REMOVE_KEYSTORE_IN_SW	2004	화면 잠금 타입이 변경되어 설정이 초기화 되었습니다. 해지 후 재등록하여 사용해 주시기 바랍니다. (Deregistration 의 startDeregistration()을 이용하여 해지 후 재등록 하시기 바랍니다.)
ERROR_MAX_LOCK_COUNT_OVER	3014	잠김허용횟수초과
IGNORE_LOCK_OPTION_SET	3015	인증장치잠금관련설정거부

		(한번이라도 다른값을 입력하여 틀린 후 설정값을 변경하려고 할 경우 차단, 바른값을 입력 한 후에는 변경 가능)
ERROR_NETWORK_STATE	4001	네트워크 오류 발생 Exception 내용
PW_NOT_MATCH_BIO_FINGER	5000	지문 인증에 실패 하였습니다.
ERROR_FORBIDDEN_FIDO	9000	일시적인 오류로 FIDO 를 사용 할 수 없습니다. 확인 후 다시 이용해 주세요.
ERROR_UNKNOWN	9999	알수 없는 오류 입니다.
ERROR_EMPTY_USERNAME	19019	username 은 필수 입력 해야 함

Public Methods	
String	getDescription ( ) FIDO 실행결과에 대한 설명을 획득한다.
int	getErrorCode ( ) FIDO 실행결과에 대한 결과코드를 획득한다.

3.6.3 Public Methods

public String **getDescription** ( );

FIDO 실행결과에 대한 설명을 획득한다.

Returns

FIDO 결과에 대한 설명

public int **getResultCode** ( );

FIDO 실행결과에 대한 결과코드를 획득한다.

Returns

FIDO 결과에 대한 코드

3.7 MagicFIDOUtil Class

3.7.1 Class Overview

FIDO 를 사용할 수 있는 기본 공통 함수들이 있으며, 인증장치(지문, 패턴, 패스코드)에 대해서 사용 가능 여부 및 인증장치의 상태값 확인, 설정 된 값의 변경 또는 설정 된 값의 초기화 등에 관련된 기능을 제공한다.

3.7.2 Summery

Public Constructors	
	MagicFIDOUtil (Context context) 클래스 생성자.

Public Methods	
FidoResult	getLocalVerifyState (LOCAL_AUTH_TYPE localType) FIDO 인증을 수행하기 위한 단말기의 상태(사용가능 단말, 불가능 단말, 지문등록 여부)를 확인 할 수 있다.
int	getAvailableType ( ) 사용자의 단말기에 사용 가능한 인증장치를 확인 할 수 있다.
boolean	isAvailableFIDO (LOCAL_ATUH_TYPE localType) 인증장치 타입에 따른 사용자의 단말기가 인증장치를 사용할 수 있는지 확인한다.
void	changeUserVeirifcation (LOCAL_ATUH_TYPE localType, FIDOCallbackResult callbackResult) 인증장치(패턴, 패스코드)의 설정값을 변경한다. (Ex. 패턴 변경)
void	resetUserVerification (LOCAL_ATUH_TYPE localType, FIDOCallbackResult callbackResult) 인증장치(패턴, 패스코드)의 설정값을 초기화한다.
void	setSSLEnable (boolean enable) SSL 검증 활성화 상태를 설정한다.
void	setAuthenticatorOptions (LOCAL_ATUH_TYPE localType, HashTable<String, Object> settingTable) 인증장치의 제한사항을 설정한다.
String	getVersion () FIDO 버전을 확인한다.
void	setAuthenticatorResetCallbackEnable(boolean enable) 인증화면에서 재등록 버튼이 보이도록 설정한다.
void	setPasscodeMaxInputAutoNextStepEnable(boolean enable) 패스코드 인증장치 최대길이 입력 시 다음 단계로 진행하도록 설정한다.
void	setPasscodeUIType(FIDO_UI_TYPE type) 패스코드 인증장치의 UI 를 변경할 수 있다.
void	setKeypadAccessibilityOnSpeakerEnable(boolean enable)

	보안키패드의 접근성 Talkback 스피커 모드를 설정한다.
void	setPasscodeIgnoreList(HashTable<String, String> ignore) 패스코드 인증장치의 값으로 허용하지 않을 값들을 설정한다
void	setPasscodeRegularExpressionList (HashTable<String, String> expression) 패스코드 인증장치값의 제약사항에 대한 정규표현식들을 설정할 수 있다.

3.7.3 Public Constructor

public **MagicFIDOUtil** (Context context);

클래스 생성자.

Parameters

Context Application Context

3.7.4 Public Methods

public FidoResult **getLocalVerifyState** (LOCAL_AUTH_TYPE localType);

FIDO 인증을 수행하기 위한 단말기의 상태(사용가능 단말, 불가능 단말, 등록 여부)를 확인 할 수 있다.

Parameters

LOCAL_AUTH_TYPE 확인하려는 인증장치 타입
LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE (지문)
LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE (패턴)
LOCAL_AUTH_TYPE.LOCAL_PACODE_TYPE (패스코드)

Returns

상태	설명
FidoResult.LOCAL_VERIFY_REGISTED	사용자의 Local 인증 값이 등록 된 상태
FidoResult.LOCAL_VERIFY_NOT_REGIST	사용자의 Local 인증 값이 등록 되어 있지 않은 상태(FIDO 지원단말)
FidoResult.LOCAL_VERIFY_UNABLE_DEVICE	Local 인증 사용 불가능(FIDO 지원단말 아님)

public int **getAvailableType** ( );

사용자의 단말기에 사용 가능한 인증장치를 확인 할 수 있다.

Returns

지문, 패턴, 패스코드 인증장치를 사용 가능한지에 대한 결과를 전달한다.

Remarks

* 결과값 확인 방법.

```
// FIDOUtil 객체 생성
MagicFIDOUtil fidoUtils = new MagicFIDOUtil(this, null);
// 사용가능 여부 확인 API 호출
int type = fidoUtils.getAvailableTypes();

// 결과 확인
if (((type & LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE.getLocalType()) ==
LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE.getLocalType()) ||
                                ((type & LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE.getLocalType()) ==
LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE.getLocalType())) {
        Toast.makeText(this, "사용가능 한 단말기 입니다.",Toast.LENGTH_LONG).show();
}else{
        if ((type & LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE.getLocalType()) ==
LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE.getLocalType()){
                FidoResult state =
fidoUtils.getLocalVerifyState(LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE);
                Toast.makeText(this, "사용 불가능한 단말기 입니다.("+state.getErrorCode
+")",Toast.LENGTH_LONG).show();
                                }
        else if((type & LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE.getLocalType()) ==
LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE.getLocalType()) {
                FidoResult state =
fidoUtils.getLocalVerifyState(LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE);
                Toast.makeText(this, "사용 불가능한 단말기
입니다.("+state.getErrorCode+")",Toast.LENGTH_LONG).show();
        }else{
                Toast.makeText(this, "사용 불가능한 단말기 입니다.",Toast.LENGTH_LONG).show();
        }
}
```

public boolean **isAvailableFIDO** (LOCAL_AUTH_TYPE localType);

사용자의 단말기에 사용 가능한 인증장치를 확인 할 수 있다.

Parameters

LOCAL_AUTH_TYPE	확인하려는 인증장치 타입 LOCAL_AUTH_TYPE.LOCAL_FINGERPRINT_TYPE (지문) LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE (패턴) LOCAL_AUTH_TYPE.LOCAL_PACODE_TYPE (패스코드)

Returns

가능하면 true 불가능하면 false

Remarks

* Android OS 제한이 있어, 6.0 이상부터 지원.

public void **changeUserVerification** (LOCAL_AUTH_TYPE localType, FIDOCallbackResult fidoCallback);

인증장치(패턴, 패스코드)의 설정값을 변경한다. (Ex. 패턴 변경)

Parameters

LOCAL_AUTH_TYPE	LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE (패턴) LOCAL_AUTH_TYPE.LOCAL_PACODE_TYPE (패스코드)
callbackResult	결과를 받을 수 있는 Callback Result

Remarks

인증장치 설정값을 변경하면 결과를 FIDOCallbackResult 로 리턴 받는다.

RequestCode 는 REQUEST_LOCAL_CHANGE_VALUE 으로 내려온다.

FIDOCallBack 에 대한 유의 사항

```
private FIDOCallbackResult fidoCallback = new FIDOCallbackResult() {
  @Override
  public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
   if(requestCode == FIDORequestCode.REQUEST_LOCAL_CHANGE_VALUE){
    if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
        // 에러코드 0: 등록 성공
    }else{
        // 실패 사유
        switch(fidoResult.getErrorCode()){
        case FidoResult.ERROR_CANNOT_USE_FIDO:
        // 에러코드 1007: 지원하지 않는 단말기
        case  FidoResult.ERROR_INVALID_PARAMETER:
        // 에러코드 1001: 전달인자값을 잘못 넣음.
        case FidoResult.RESULT_USER_CANCEL:
        // 에러코드 999: 사용자 취소
        case FidoResult.LOCAL_FAIL_VERIFY:
```

```
            // 에러코드 3012: 설정 중 잘못 확인 하여 틀렸을 때
            case FidoResult.LOCAL_FAIL_VERIFY_Locking:
            // 에러코드 3013: Max 만큼 틀려서 인증이 락이 된 상태
            case FidoResult.ERROR_UNKNOWN:
            // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
            showDialog("FAIL", fidoResult.getDescription() + "(" +  fidoResult.getErrorCode() + ")");
            break;
            }
        }
    }
});
```

public void **resetUserVerification** (LOCAL_AUTH_TYPE localType, FIDOCallbackResult fidoCallback);

인증장치(패턴 또는 패스코드)의 설정값을 초기화한다.

Parameters

LOCAL_AUTH_TYPE	LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE (패턴) LOCAL_AUTH_TYPE.LOCAL_PACODE_TYPE (패스코드)
fidoCallback	결과를 받을 수 있는 Callback Result

Remarks

인증장치 설정만을 초기화 하는 기능으로 인증장치에 등록된 사용자를 삭제 하지는 않는다. 해당 기능을 사용할 때에는 StartDeregistration() API 함께 사용하는 것을 권고한다.

인증장치 설정을 초기화 하면 결과를 FIDOCallbackResult 로 리턴 받는다.

RequestCode 는 REQUEST_LOCAL_RESET 으로 내려온다.

FIDOCallBack 에 대한 유의 사항

```
private FIDOCallbackResult fidoCallback = new FIDOCallbackResult() {
  @Override
  public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
   if(requestCode == FIDORequestCode.REQUEST_LOCAL_RESET){
    if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
        // 에러코드 0: 등록 성공
    }else{
        // 실패 사유
        switch(fidoResult.getErrorCode()){
        case FidoResult.ERROR_CANNOT_USE_FIDO:
        // 에러코드 1007: 지원하지 않는 단말기
```

```
            case  FidoResult.ERROR_INVALID_PARAMETER:
            // 에러코드 1001: 전달인자값을 잘못 넣음.
            case FidoResult.RESULT_USER_CANCEL:
            // 에러코드 999: 사용자 취소
            case FidoResult.LOCAL_FAIL_VERIFY:
            // 에러코드 3012: 설정 중 잘못 확인 하여 틀렸을 때
            case FidoResult.LOCAL_FAIL_VERIFY_Locking:
            // 에러코드 3013: Max 만큼 틀려서 인증이 락이 된 상태
            case FidoResult.ERROR_UNKNOWN:
            // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
            showDialog("FAIL", fidoResult.getDescription() + "(" +  fidoResult.getErrorCode() + ")");
            break;
            }
        }
    }
});
```

public static void **setSSLEnable** (boolean enable);

SSL 검증 활성화 상태를 설정한다.

Parameters

enable	검증 여부 (true : 검증 , false : 미검증)

Remarks

운영시에는 SSL 인증서를 검증 해야만 한다.

public void **setAuthenticatorOptions** (LOCAL_ATUH_TYPE localType, HashTable<String, Object> settingTable);

인증장치의 제한사항을 설정한다.

Parameters

LOCAL_AUTH_TYPE	LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE (패턴) LOCAL_AUTH_TYPE.LOCAL_PACODE_TYPE (패스코드)
settingTable	인증장치에 설정 할 사항들의 hashTable

Remarks

인증장치의 제한 사항 설정 옵션

키	설명	입력 값 타입
- 인증장치 공통		
KEY_RETRY_COUNT_TO_LOCK	인증장치 잠김까지 시도 가능 횟수	int
KEY_LOCK_TIME	인증장치가 잠겨있는 시간	int
KEY_MAX_LOCK_COUNT	인증장치 잠김 허용 횟수	int
- 패턴 인증장치		
KEY_LINE_WIDTH	패턴 라인 굵기	float
KEY_LINE_COLOR	패턴 라인 색상	String
KEY_MISMATCH_LINE_COLOR	패턴 불일치 라인 색상	String
- 패스코드 인증장치		
KEY_MIN_LENGTH	패스코드로 등록할 수 있는 최소 길이	int
KEY_MAX_LENGTH	패스코드로 등록할 수 있는 최대 길이	int
KEY_USE_NUMBER_KEYPAD	패스코드 숫자키패드 사용 여부	boolean
KEY_REGULAR_EXPRESSION	패스코드 정규 표현식	String
KEY_SHUFFLE_DISABLE	패스코드 숫자키패드 셔플 비활성화 여부	boolean
KEY_USE_NUMBER_BLANK	키패드 공백 사용 여부(숫자키패드 사용 시)	boolean

<정규표현식 예>

영문,숫자,특수문자 각 1 자 이상, 전체 8 자 이상	^(?=.*[A-Za-z])(?=.*₩d)(?=.*[$@$!%*#?&])[A-Za-z₩d$@$!%*#?&]{8,}$
대문자,소문자,숫자 각 1 자 이상, 전체 8 자 이상	^(?=.*[a-z])(?=.*[A-Z])(?=.*₩d)[a-zA-Z₩d]{8,}$
대문자,소문자,숫자,특수문자 각 1 자 이상, 전체 8 자 이상	^(?=.*[a-z])(?=.*[A-Z])(?=.*₩d)(?=.*[$@$!%*?&])[A-Za-z₩d$@$!%*?&]{8,}

입력받은 정규표현식을 패스코드 설정값의 제한사항으로 적용하는 기능을 지원하며, 정규표현식의 생성과 유효성 검증은 SDK 를 적용하는 앱개발사의 영역이다. 위의 표는 정규표현식의 예시를 제공한다. 패스코드 인증장치의 설정값(MIN_LENGTH, MAX_LENGTH, USE_NUMBER_KEYPAD, REGULAR_EXPRESSION)은 패스코드의 UI Type 이 FIDO_PASSCODE_NORMAL 인 경우만 지원한다.

<예시>

```
Hashtable<String, Object> authOption = new Hashtable<String, Object>();
authOption.put(MagicFIDOUtil.KEY_RETRY_COUNT_TO_LOCK, 5);
authOption.put(MagicFIDOUtil.KEY_LOCK_TIME, 30);
authOption.put(MagicFIDOUtil.KEY_MIN_LENGTH, 5);
authOption.put(MagicFIDOUtil.KEY_MAX_LENGTH, 8);
```

```
authOption.put(MagicFIDOUtil.KEY_USE_NUMBER_KEYPAD, true);
authOption.put(MagicFIDOUtil.KEY_REGULAR_EXPRESSION, "^(?=.*[A-Za-
z])(?=.*₩₩d)(?=.*[$@$!%*#?&])[A-Za-z₩₩d$@$!%*#?&]{5,}$");
magicFIDOUtil.setAuthenticatorOptions(LOCAL_AUTH_TYPE.LOCAL_PACODE_TYPE, authOption);

Hashtable<String, Object> patternOption = new Hashtable<String, Object>();
patternOption.put(MagicFIDOUtil.KEY_RETRY_COUNT_TO_LOCK, 5);
patternOption.put(MagicFIDOUtil.KEY_LOCK_TIME, 30);
patternOption.put(MagicFIDOUtil.KEY_LINE_COLOR, "#0078d7");
patternOption.put(MagicFIDOUtil.KEY_MISMATCH_LINE_COLOR, "#d60000");
patternOption.put(MagicFIDOUtil.KEY_LINE_WIDTH, 3.2f);
magicFIDOUtil.setAuthenticatorOptions(LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE, patternOption);
```

public String **getVersion** ();

FIDO SDK 버전을 확인한다.

Returns

SDK 버전

public String **setAuthenticatorResetCallbackEnable** (boolean enable);

인증화면에서 재등록 버튼이 보이도록 설정한다.

Parameters

enable	설정 여부 (default : false) true : 설정 / false : 미설정

Remarks

생체 인증장치를 제외한 인증장치에서 재등록 버튼이 활성화 되며, 사용자가 재등록 버튼을 누를 경우 FIDOCallbackResult 으로 결과코드 980 을 리턴한다.
패턴, 패스코드 인증장치만 사용 가능하다.

public String **setPasscodeMaxInputAutoNextStepEnable** (boolean enable);

패스코드 인증장치 최대길이 입력 시 다음 단계로 진행하도록 설정한다.

Parameters

enable	설정 여부 (default : false) true : 설정 / false : 미설정

Remarks

비밀번호 최대 길이를 설정 해야한다.

public void **setPasscodeUIType** (FIDO_UI_TYPE type);

패스코드 인증장치의 UI 를 변경할 수 있다.

Parameters

type	설정하려는 패스코드 UI 타입 FIDO_PASSCODE_NORMAL : 기본타입 FIDO_PASSCODE_PIN4 : 4 개의 핀 타입 FIDO_PASSCODE_PIN6 : 6 개의 핀 타입

Remarks

호출하지 않을 경우 기본값은 FIDO_PASSCODE_NORMAL
UI 타입과 상관없이 패스코드 인증장치는 하나의 값만을 가진다.
(ex. PIN4에서 1111로 등록한 경우, NORMAL에서도 1111 이며, PIN6에서는 사용 불가

public void **setKeypadAccessibilityOnSpeakerEnable** (boolean enable);

보안키패드의 접근성 Talkback 스피커 모드를 설정한다.

Parameters

enable	설정 여부 (default : false) true : 설정 / false : 미설정

Remarks

보안키패드의 키 영역은 기본적으로 스피커의 경우 Talkback(Voice Assistant)으로 읽어 주지 않도록 설정되어 있다. 필요한 경우 true로 설정하여 스피커를 통해서도 읽히도록 설정할 수 있다.

public void **setPasscodeIgnoreList** (HashTable<String, String> ignore);

패스코드 인증장치의 값으로 허용하지 않을 값들을 설정한다.

Parameters

ignore	패스코드 인증장치의 값으로 등록할 수 없는 값의 HashTable * Key : 허용하지 않는 스트링

* value : 허용하지 않는 스트링을 포함한 경우의 에러 메세지

Remarks

사용자가 패스코드값 설정 시 ignore 의 key 값으로 전달받은 값을 포함하여 설정할 경우 ingore 의 value 값으로 메세지를 보여주며, 패스코드값 등록을 허용하지 않는다.

기본적으로 지원하는 제한사항도 있으며, 해당 키값과 제한사항은 아래와 같다

키	설명
KEY_EQUAL_CHARACTER	동일한 입력값 (ex: 111111, aaaaaa)
KEY_ASCENDING_DESCENDING	오름차순, 내림차순 (ex: abcdef, 654321)
KEY_2CHARACTER_REPEAT	두 문자 반복 사용 (ex: 373737)
KEY_2EQUAL_CHARACTER_REPEAT	두 동일문자 연속 조합 (ex: 335577)
KEY_NOT_NUMBER_ENGLISH	숫자와 영어를 제외한 다른 문자 사용 (ex: 비밀번호)

<예시>

```
Hashtable<String, String> ignoreList = new Hashtable<String, String >();

ignoreList.put("444", "444 는 허용되지 않습니다");
➔ 패스코드에 11444 를 입력 했을 경우 '444 는 허용되지 않습니다' 문구 출력
ignoreList.put("KEY_EQUAL_CHARACTER" , "동일한 문자는 허용되지 않습니다")
➔ 패스코드에 1111 를 입력 했을 경우 '동일한 문자는 허용되지 않습니다' 문구 출력

magicFidoUtil.setPasscodeIgnoreList(ignoreList);
```

public void **setPasscodeRegularExpressionList** (HashTable<String, String> expression);

패스코드 인증장치값의 제약사항에 대한 정규표현식들을 설정할 수 있다.

Parameters

expression — 패스코드 인증장치의 입력값을 체크할 정규표현식의 해시테이블
- Key : 정규표현식
- Value : 정규표현식에 허용되지 않을 경우 에러 메세지 스트링

Remarks

사용자가 설정한 패스코드값이 expression 의 key 값으로 전달받은 정규표현식에 허용되지 않을 경우 expression 의 value 값으로 메세지를 보여주며, 패스코드값 등록을 허용하지 않는다.

정규표현식의 생성과 유효성 검증은 SDK 를 적용하는 앱개발사의 역영이다.

4 사용 예제

4.1 FIDO 등록 (Registration) 사용 예제

```
public final int ConnectTimeOut             = 1000 * 30;
public final int ReadTimeOut        = 1000 * 30;

//RP SDK 를 사용하기 위한 객체
private Registration      mRegist = null;
private FIDOCallbackResult mFidoCallback = null;

public FIDORegistration(Context context, FIDOCallbackResult fidoCallback){
 mFidoCallback = fidoCallback;
 mRegist = new Registration(context, mFidoCallback, getReqUrl(),getResUrl());
 mRegist.setNetworkTimeout(ConnectTimeOut, ReadTimeOut);
}

public void registFIDO(String userID){
 // RPContext를 위한 설정
 Hashtable<Object, Object> settingValue = new Hashtable<Object, Object>();
 settingValue.put(ContextKeys.KEY_USERID, userID);
 mRegist.startRegistration(settingValue);
}

// 등록 결과 리턴
private FIDOCallbackResult fidoResult = new FIDOCallbackResult() {
@Override
public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
  if(requestCode == FIDORequestCode.REQUEST_CODE_REGIST){
    if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
      // 결과 코드 0: 등록 성공
    }else{
      // 실패 사유
      switch(fidoResult.getErrorCode()){
        case FidoResult.RESULT_USER_CANCEL:
        // 에러코드 999: 사용자 취소
        break;
        case FidoResult.ERROR_CANNOT_USE_FIDO:
```

```
        // 에러코드 1007: 지원하지 않는 단말기
        break;
        case FidoResult.ERROR_UNTRUSTED_FACET_ID:
        // 에러코드 2001: -등록되지 않은 3rd App으로 진행
        //                - FIDO Server에 FacetID 를 등록해야 함.
        break;
        case FidoResult.ERROR_NETWORK_STATE:
        // 에러코드 4001: -네트워크 오류
        //                -fidoResult.getDescription() 을 통하여 자세한 오류 내용을 알 수 있음
        break;
        case FidoResult.PW_NOT_MATCH_BIO_FINGER:
        // 에러코드 5000: 지문 인증 실패
        break;
        case FidoResult.ERROR_UNKNOWN:
        // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
        break;
      }
    }
  }
});
```

4.2 FIDO 인증 (Authentication) 사용 예제

```
public final int ConnectTimeOut          = 1000 * 30;
public final int ReadTimeOut    = 1000 * 30;

//RP SDK 를 사용하기 위한 객체
private Authentication mAuth = null;
private FIDOCallbackResult mFidoCallback = null;

public FIDOAuthentication(Context context, FIDOCallbackResult fidoCallback){
 mFidoCallback = fidoCallback;
 mAuth = new Authentication(context, mFidoCallback, getReqUrl(),getResUrl());
 mAuth.setNetworkTimeout(ConnectTimeOut, ReadTimeOut);
}

public void loginAuthFIDO(String userID){
 startAuthentication(userID, null);
```

```
}

public void transferAuthFIDO(String userID, String transferMessage){
 startAuthentication(userID, transferMessage);
}

private void startAuthentication(String userID, String transferMessage){
 // RPContext를 위한 설정
 Hashtable< Object, Object > settingValue = new Hashtable< Object, Object >();
 settingValue.put(ContextKeys.KEY_USERID, userID);

 if(transferMessage != null){
  settingValue.put(ContextKeys.KEY_TRANSACTIONTEXT, transferMessage);
 }
 mAuth.startAuthentication(settingValue);
}

// 인증 결과 리턴
private FIDOCallbackResult fidoCallbackResult = new FIDOCallbackResult() {
@Override
public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
  if(requestCode == FIDORequestCode.REQUEST_CODE_AUTH) {
    if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
      // 결과 코드 0: 등록 성공
    }else{
      // 실패 사유
      switch(fidoResult.getErrorCode()){
      case FidoResult.RESULT_USER_CANCEL:
      // 에러코드 999: 사용자 취소
      break;
      case FidoResult.ERROR_CANNOT_USE_FIDO:
      // 에러코드 1007: 지원하지 않는 단말기
      break;
      case FidoResult.ERROR_NOT_EXIST_USER_DATA:
      // 에러코드 1009: -인증하려는 아이디의 데이터가 존재하지 않는 경우
      //                -동일 아이디로 해지 후 재등록 요청 해야 함
      //                -단말에 사용자의 데이터가 없기 때문에, Deregstration의
```

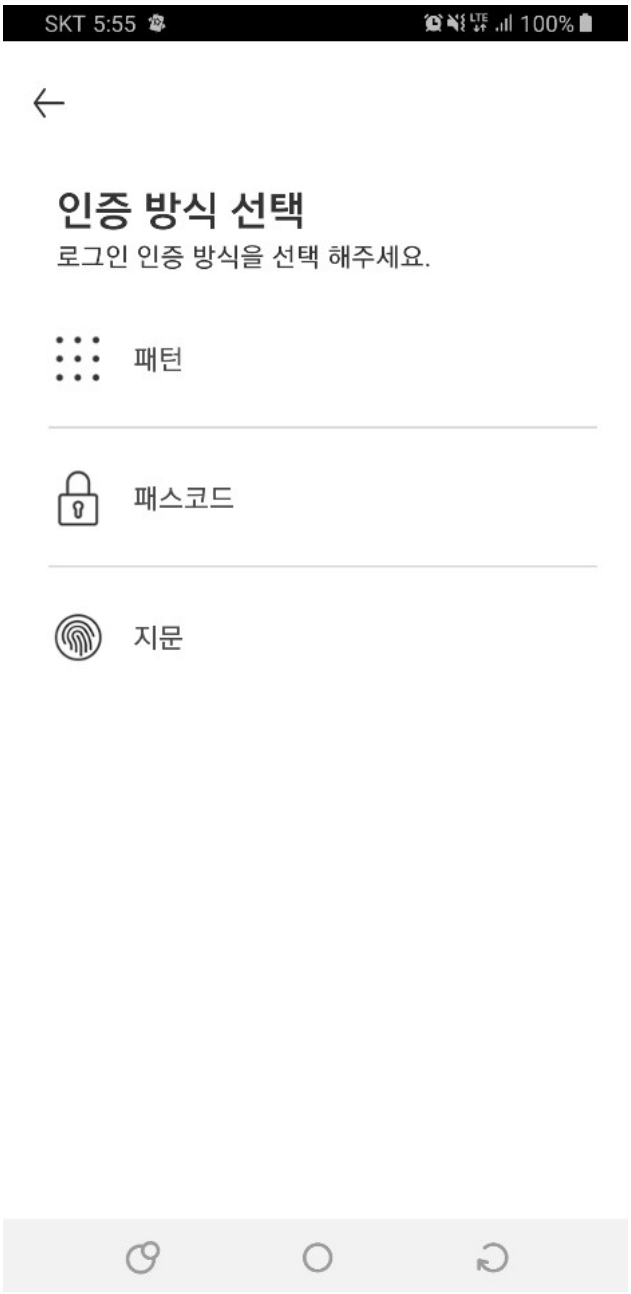
```
        //                  startDeregistration()을 이용하여 해지 후 재등록 진행
        break;
        case 13006:
        // 에러코드 13006: -인증 시 등록 된 아이디가 없는 사유로 서버에서 전달하는 에러코드
        //                 - Regsitration의 startRegistration()을 이용하여 등록 요청 진행
        break;
        case 19019:
        // 에러코드 19019: -아이디 누락으로 인한 실패 사유로 서버에서 전달하는 에러코드
        //                  -인증 요청 할 때, UserName을 입력하지 않음.
        //                   userName을 넣은 후 진행.
        break;
        case FidoResult.ERROR_UNTRUSTED_FACET_ID:
        // 에러코드 2001: -등록되지 않은 3rd App으로 진행
        //                - FIDO Server에 FacetID 를 등록해야 함.
        break;
        case FidoResult.ERROR_NO_SUITABLE_AUTHENTICATOR:
        // 에러코드 2002: - 적합한 인증장치가 없음
        //                - Server에 등록 되어 있지만, 단말기에 해당하는 데이터가 없음.
        //                - 동일 아이디로 Deregistration의 startDeregistration()을 이용하여
        //                  해지 후 재등록 요청 해야 함.
        break;
        case FidoResult.ERROR_NETWORK_STATE:
        // 에러코드 4001: -네트워크 오류
        //                 -fidoResult.getDescription() 을 통하여 자세한 오류 내용을 알 수 있음
        break;
        case FidoResult.PW_NOT_MATCH_BIO_FINGER:
        // 에러코드 5000: 지문 인증 실패
        break;
        case FidoResult.ERROR_UNKNOWN:
        // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
        break;
        }
      }
    }
});
```

4.3 FIDO 해지 (DeRegistration) 사용 예제

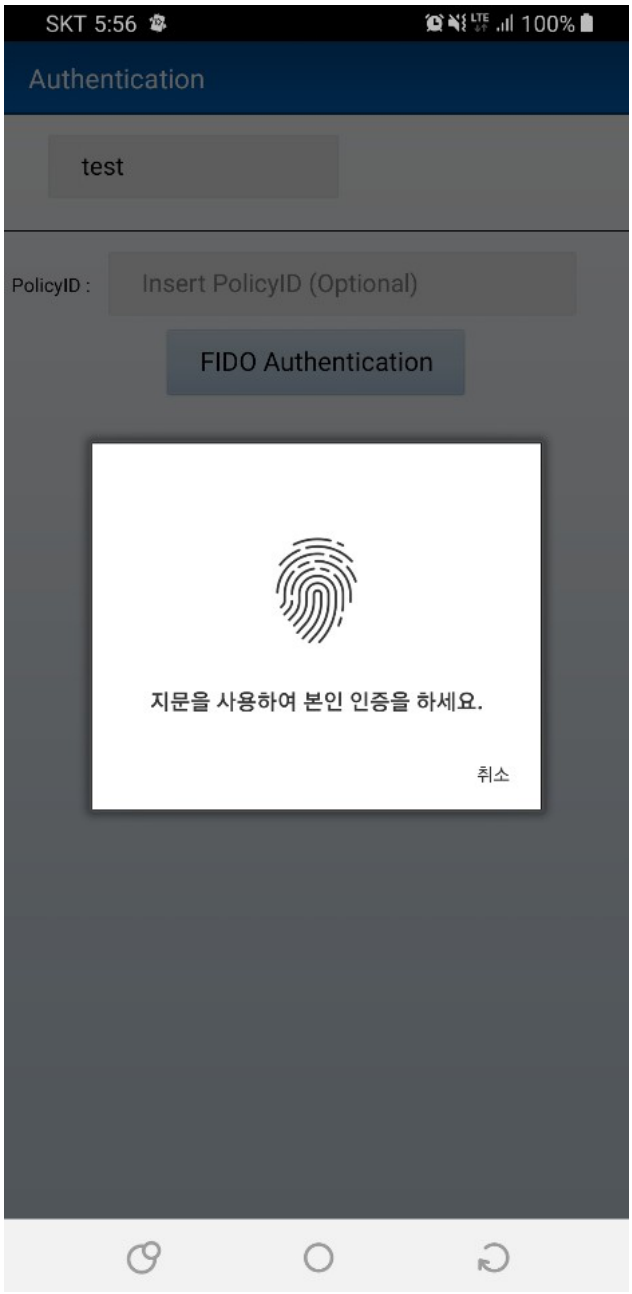
```
public final int ConnectTimeOut              = 1000 * 30;
public final int ReadTimeOut         = 1000 * 30;

//RP SDK 를 사용하기 위한 객체
private Deregistration   mDeregist = null;
private FIDOCallbackResult mFidoCallback = null;

public FIDODeRegistration(Context context, FIDOCallbackResult fidoCallback){
 mFidoCallback = fidoCallback;
 mDeregist = new Deregistration(context, mFidoCallback, getReqUrl());
 mDeregist.setNetworkTimeout(ConnectTimeOut, ReadTimeOut);
}

public void deRegistFIDO(String userID){
 // RPContext를 위한 설정
 Hashtable<String, String> settingValue = new Hashtable<String, String>();
 settingValue.put (ContextKeys.KEY_USERID, userID);
 // 해지할 인증장치 타입(지문, 패턴, 패스코드) 설정
 settingValue.put(ContextKeys.KEY_LOCALTYPE, LOCAL_AUTH_TYPE.LOCAL_PATTERN_TYPE);
 mDeregist.startDeregistration(settingValue);
}

// 해지 결과 리턴
private FIDOCallbackResult fidoCallback = new FIDOCallbackResult() {
  @Override
  public void onFIDOResult(int requestCode, boolean result, FidoResult fidoResult) {
    if(requestCode == FIDORequestCode.REQUEST_CODE_DEREG){
      if(fidoResult.getErrorCode() == FidoResult.RESULT_SUCCESS){
        // 결과 코드 0: 등록 성공
      }else{
        // 실패 사유
        switch(fidoResult.getErrorCode()) {
        case FidoResult.ERROR_CANNOT_USE_FIDO:
        // 에러코드 1007: 지원하지 않는 단말기
        break;
        case FidoResult.ERROR_UNTRUSTED_FACET_ID:
```

```
            // 에러코드 2001: -등록되지 않은 3rd App으로 진행
            //                    -FIDO Server에 FacetID 를 등록해야 함.
            break;
            case FidoResult.ERROR_NETWORK_STATE:
            // 에러코드 4001: -네트워크 오류
            //                    -fidoResult.getDescription() 을 통하여 자세한 오류 내용을 알 수 있음
            break;
            case 15007:
            // 에러코드 15007: -해지 시 등록 된 아이디가 없을 때, 서버에서 내려주는 에러코드
            //                     - 등록 된 아이디가 없기 때문에 Registration의
            //                       startRegistration()을 이용하여 등록 후 이용
            break;
            case FidoResult.ERROR_UNKNOWN:
            // 에러코드 9999: FIDO 로직 수행 중, 알수 없는 에러 발생
            break;
            }
        }
    }
});
```

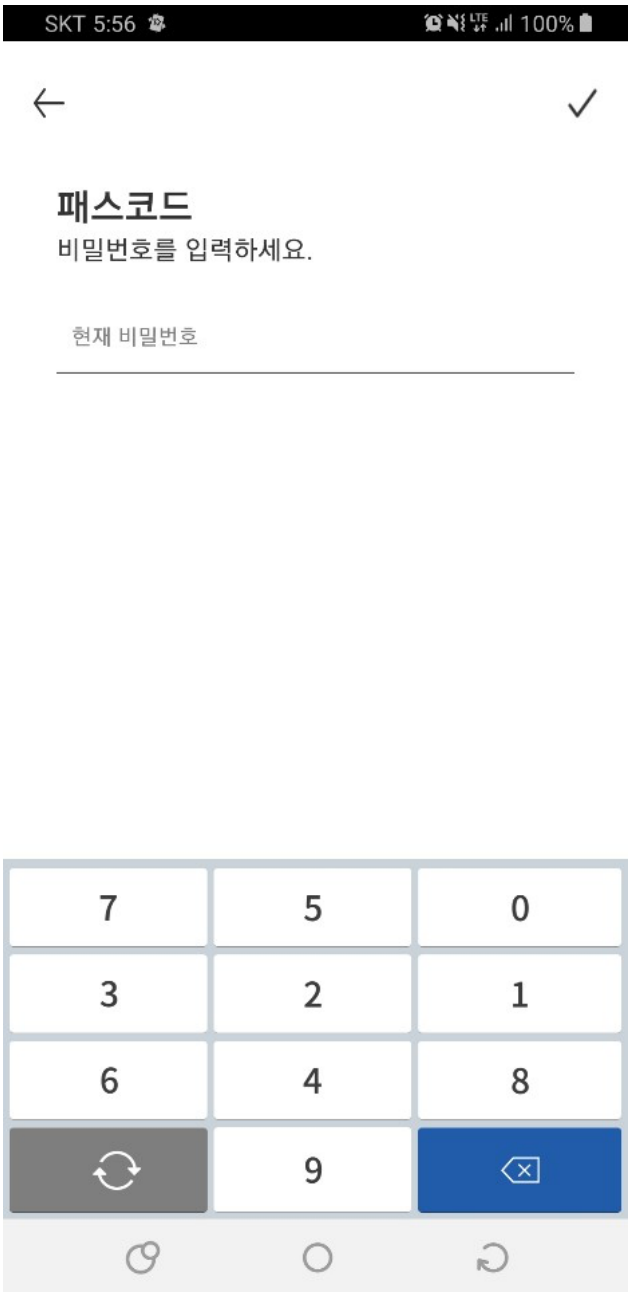
5 FIDO 화면 UI



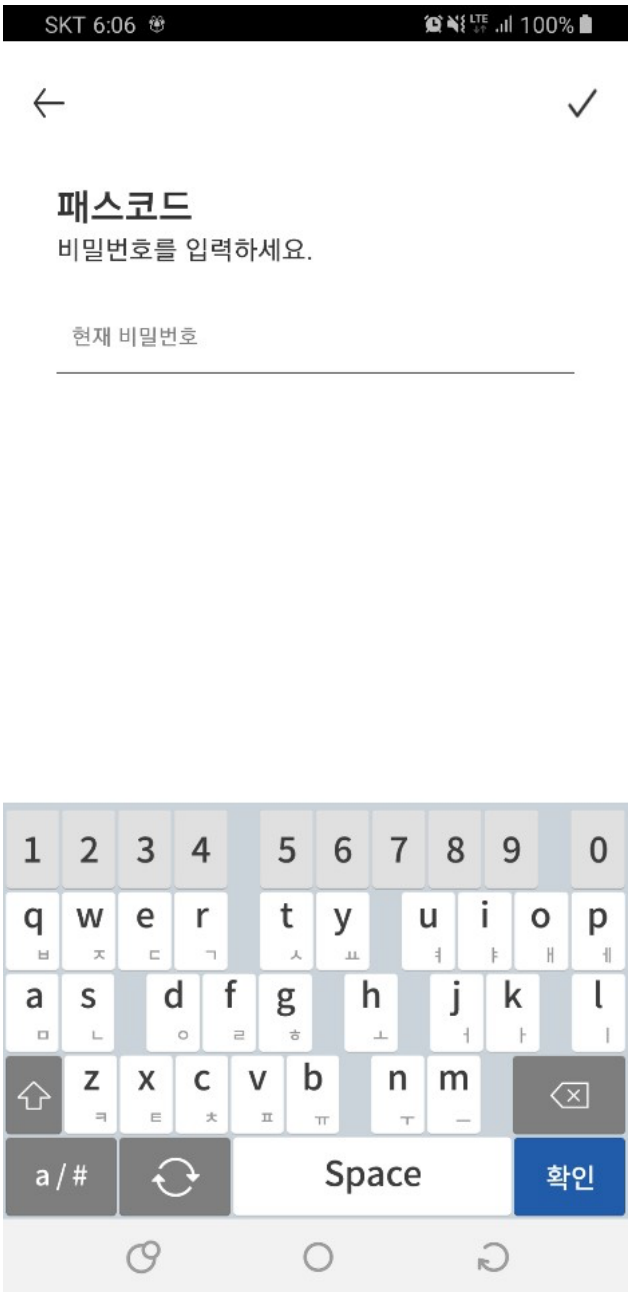
< 인증장치 선택 화면 >



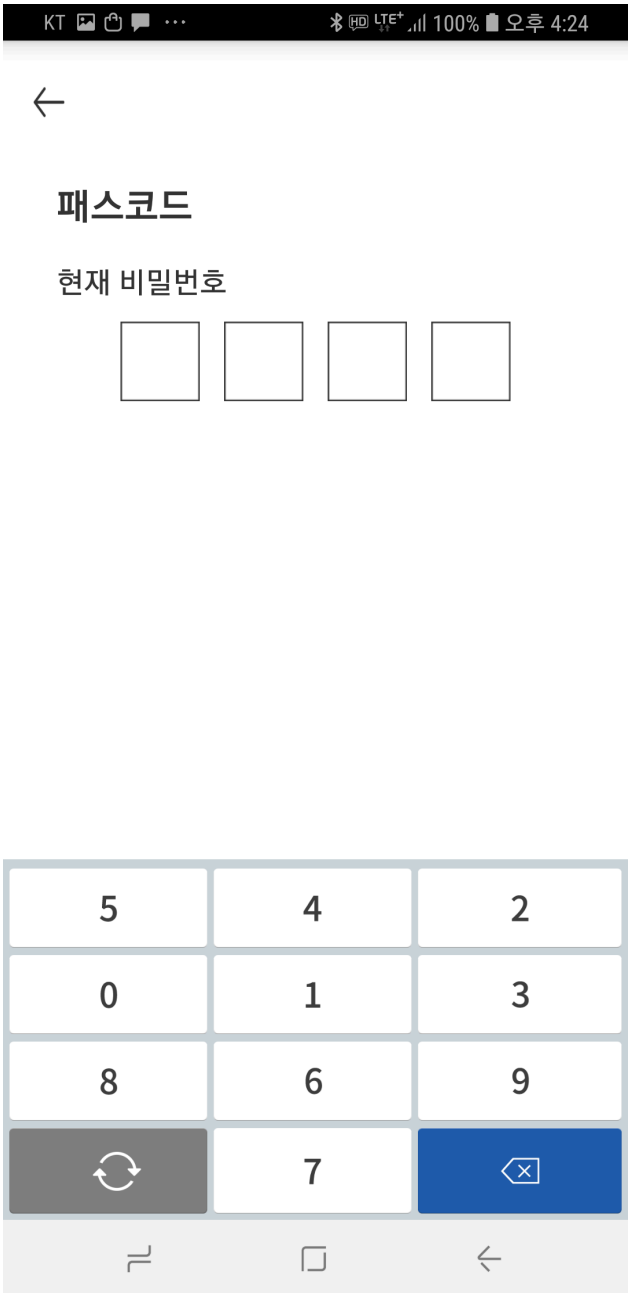
< 지문 인증장치 >



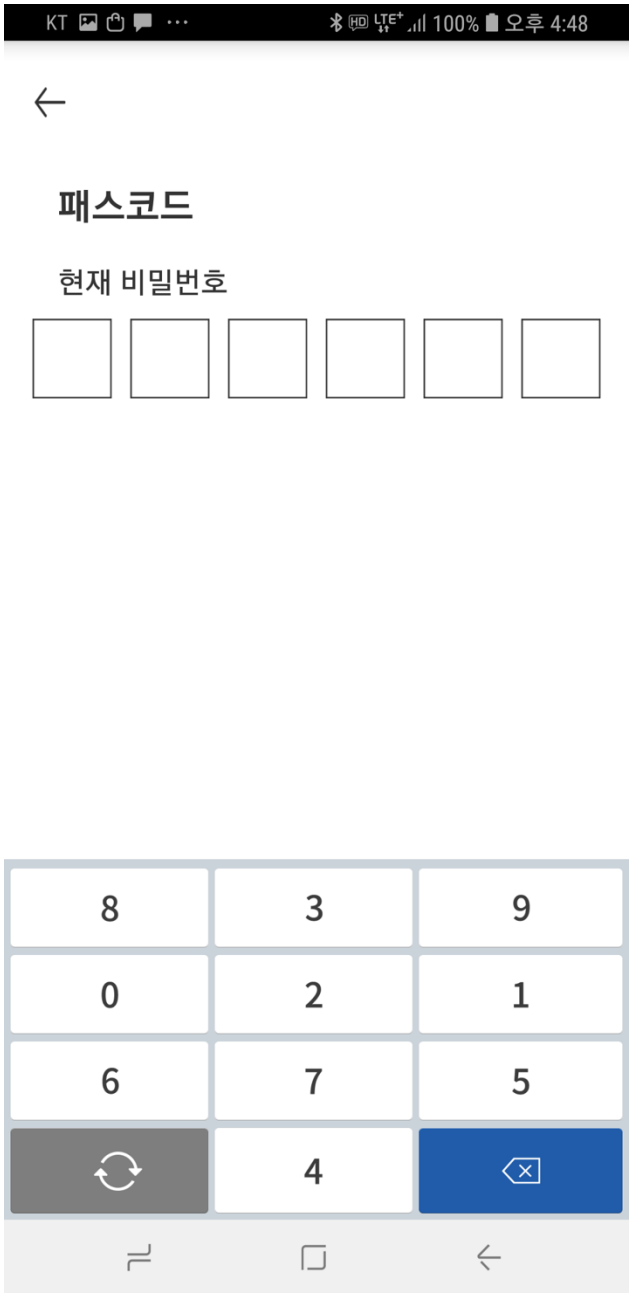
< 패스코드 인증장치 (숫자) >



< 패스코드 인증장치 (문자) >



< 패스코드 인증장치(핀 4) >



< 패스코드 인증장치(핀 6) >

< 패턴 인증장치 >