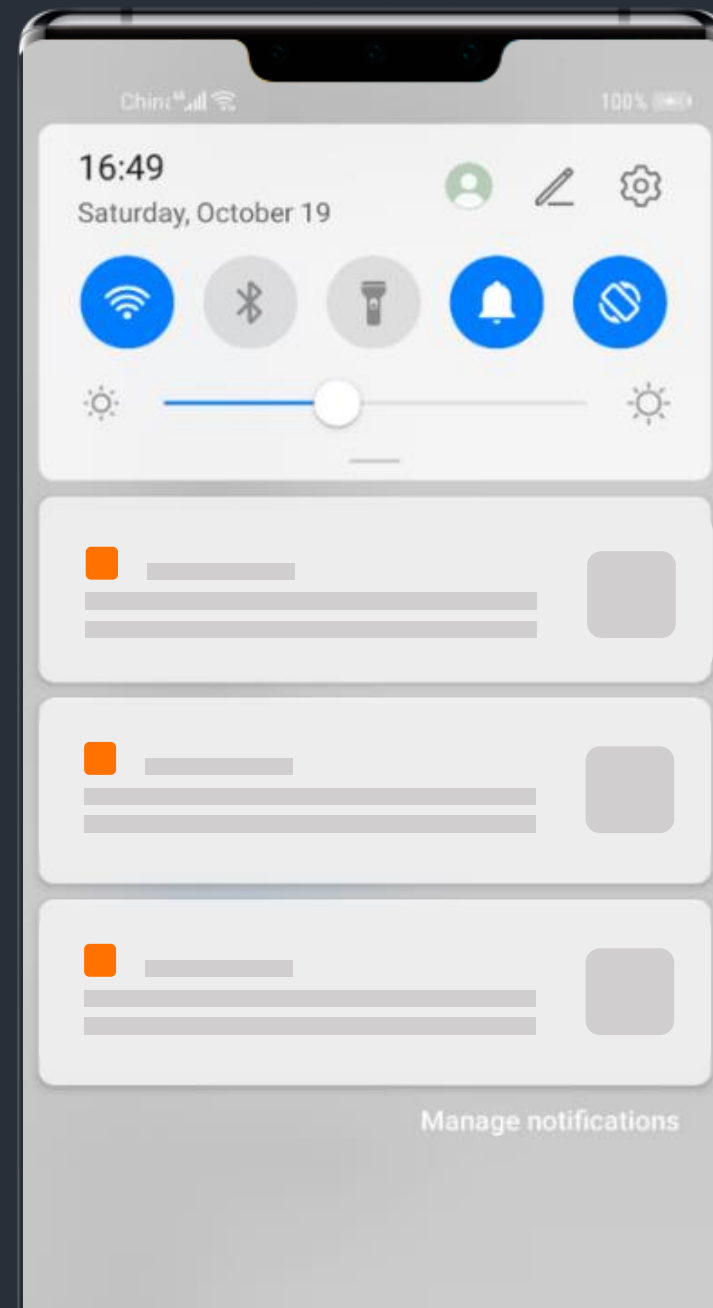




HUAWEI

HMS INSIDE UNIVERSITIES

2021



CONTENIDO



- 1 INTRODUCCIÓN A HMS
- 2 PUSH KIT
- 3 ANALYTICS KIT / DTM / ADS KIT
- 4 LOCATION/ MAP/ SITE
- 5 IAP
- 6 ACCOUNT KIT / IDENTITY
- 7 GAME SERVICE
- 8 REMOTE CONFIGURATION
- 9 ML KIT / SCAN KIT
- 10 PANORAMA KIT / AWARENESS KIT
- 11 SAFE DETECT/ FIDO**
- 12 NEARBY SERVICE



CONTENIDO

Introducción

1

Escenarios de
servicio

3

Preguntas y
respuestas

5

2

Ventajas

4

Pasos para
integrar

6

Referencias

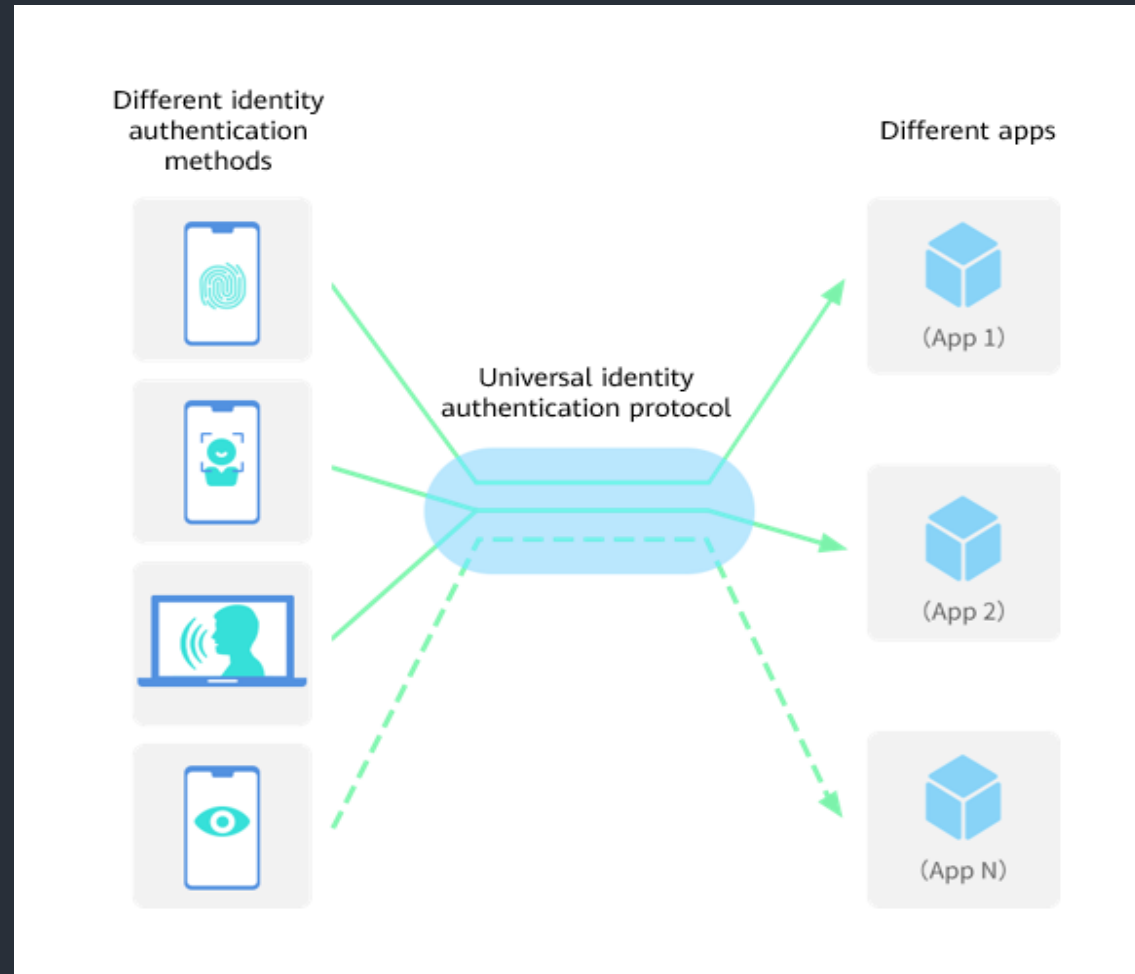


Introducción FIDO2 (Online Identity Verification)

FIDO2 es un estándar de autenticación abierto, alojado por FIDO Alliance. Es la clave para optimizar tanto la seguridad como la conveniencia de la autenticación, y ofrece medios de verificación de identidad en línea como complemento a la autenticación de contraseña.



¿Qué es FIDO2?



Fast Identity Online (FIDO) es un conjunto de protocolos de autenticación de identidad publicados y mantenidos por FIDO Alliance.

FUNCIONALIDADES

Autenticación biométrica local



Autenticación de roaming



Verificación de identidad en línea



FUNCIONALIDADES

Autenticación biométrica local



Autenticación de roaming



Verificación de identidad en línea





Confiabilidad

Garantiza la autenticación con controles de integridad del sistema, verificación de claves y reconocimiento facial 3D mejorado.



Protección de la privacidad

Verifica y almacena los datos de privacidad del usuario localmente en lugar de almacenarlos en la nube.



Solución madura

Ofrece especificaciones de ecosistema completas y aplicaciones fácilmente integradas a una infraestructura de cuenta existente.



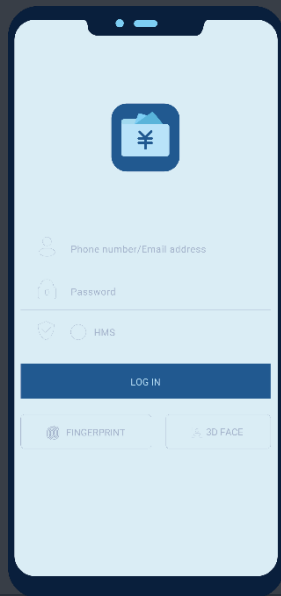
Conveniencia

Facilita la seguridad de las contraseñas usando funciones biométricas o, en su lugar, autenticadores de roaming.

FIDO2 Scenarios

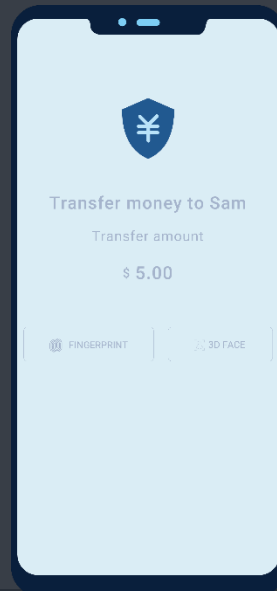
Secure sign-in

Utiliza verificación facial o de huellas dactilares para evitar la filtración de contraseñas, el relleno de credenciales y otros riesgos de contraseñas.



Secure payment

Autentica a los usuarios utilizando sus funciones biométricas para pagos dentro de la aplicación, con o sin contraseña.



Autenticación segura mediante el autenticador FIDO

Admite autenticación en plataformas habilitadas para FIDO2 sobre transportes específicos como USB, NFC y Bluetooth Low Energy (BLE).

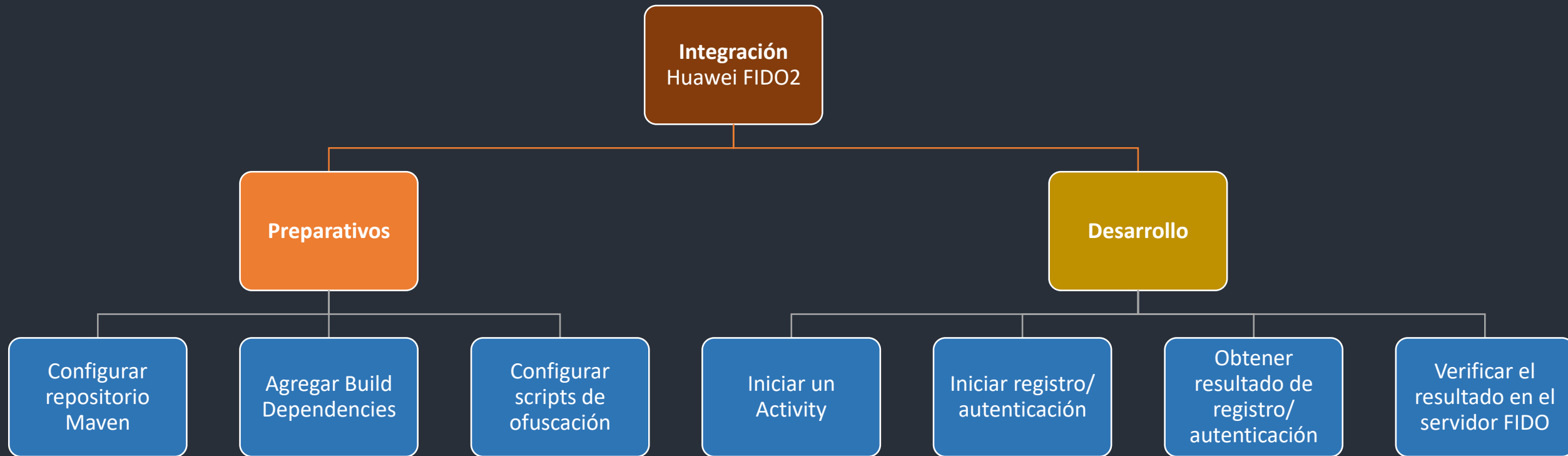


Autenticador FIDO2

Utiliza solo un teléfono Huawei como autenticador para la autenticación FIDO2 en otros dispositivos.



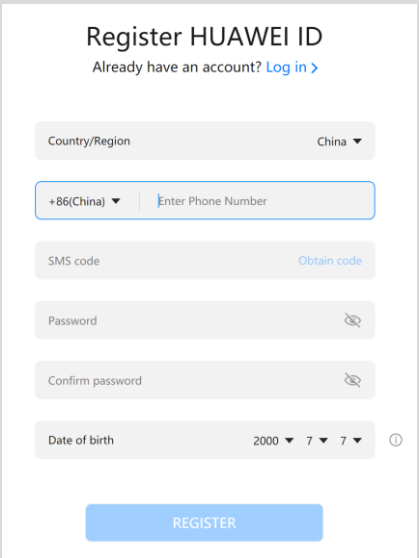
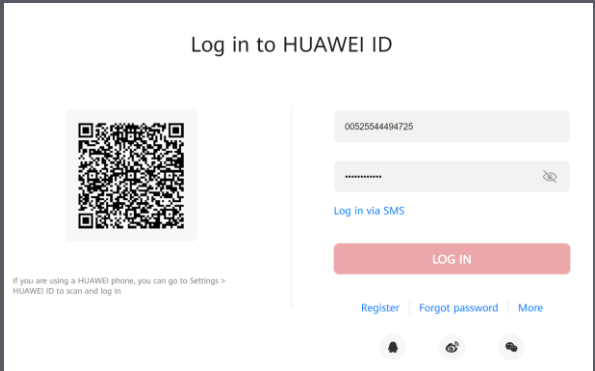
SENCILLO de integrar



Registro

Dirígete a nuestra pagina web

01



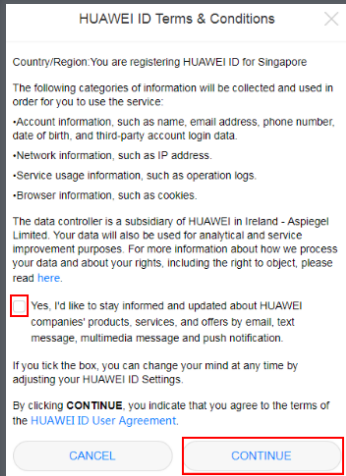
02

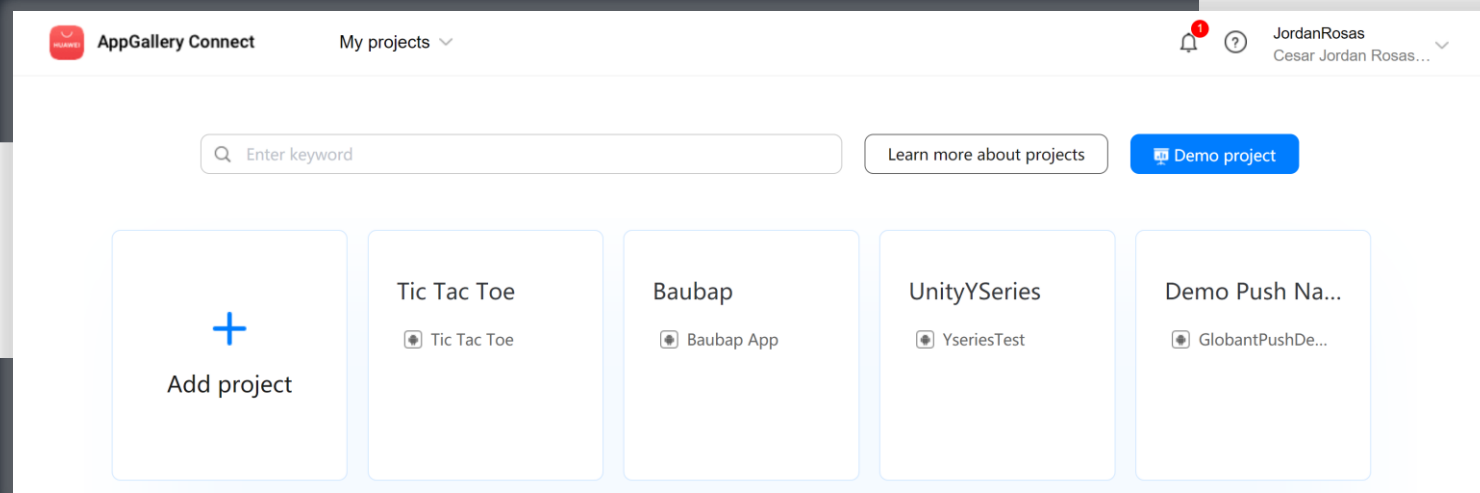
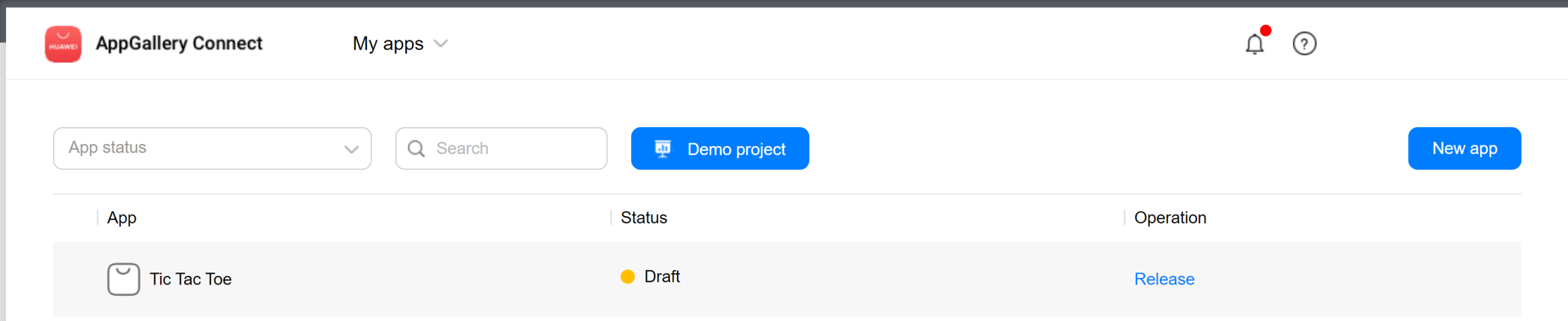
Llena con tus datos

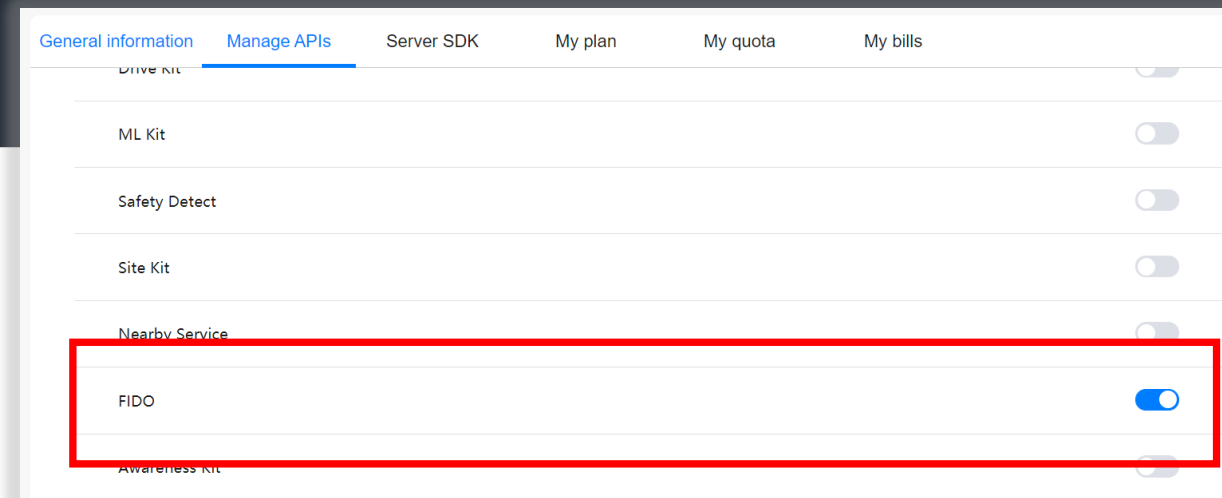
Preparativos

Acepta términos y condiciones

03




1) **Crear** un nuevo proyecto en AGC2) **Generar** una aplicación y agrégala al proyecto



4) **Configurar** las dependencias al repositorio maven

```

1  |
2  | buildscript {
3  |     ext.kotlin_version = "1.3.72"
4  |     repositories {
5  |         google()
6  |         jcenter()
7  |         maven {url 'https://developer.huawei.com/repo/'}
8  |     }
9  |     dependencies {
10 |         
11 |
12 |         classpath 'com.huawei.agconnect:agcp:1.4.1.300'
13 |     }
14 | }
15 |
16 | allprojects {
17 |     repositories {
18 |         google()
19 |         jcenter()
20 |         maven {url 'https://developer.huawei.com/repo/'}
21 |     }
22 | }
23 |
24 | task clean(type: Delete) {
25 |     delete rootProject.buildDir
26 | }

```

Preparativos

4) **Agregar** las dependencias

```
apply plugin: 'com.huawei.agconnect '
```

FIDO2

```
implementation 'com.huawei.hms:fido-fido2:{version}'
```

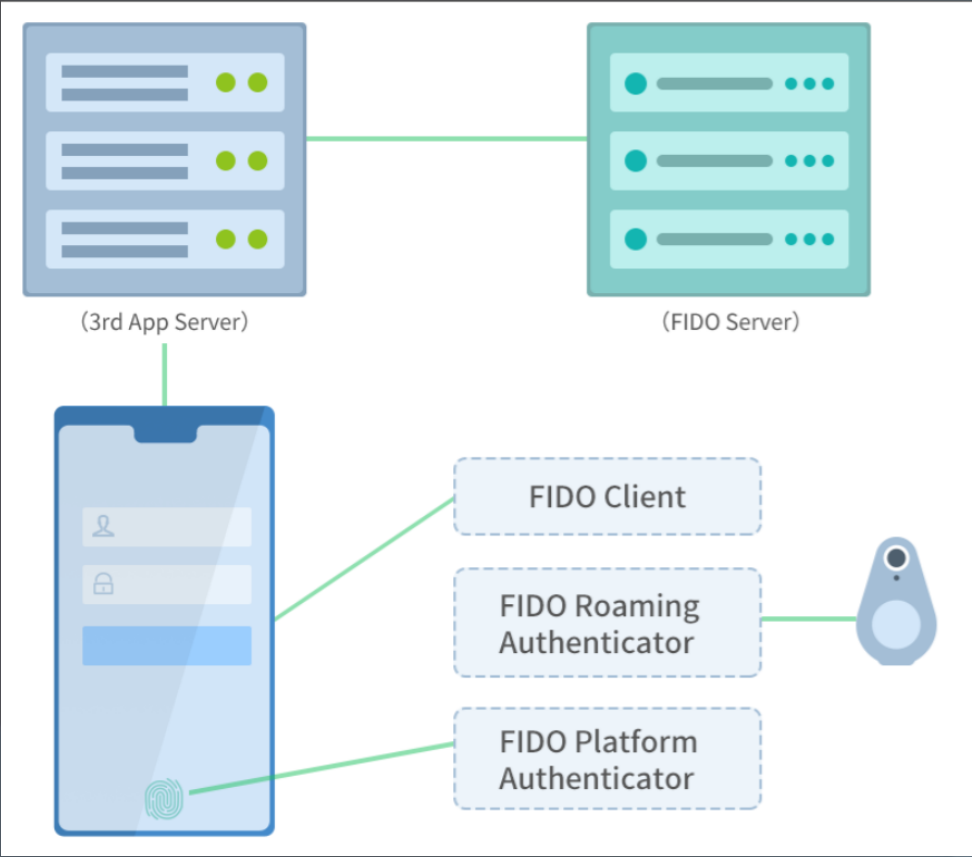
BioAuthn-AndroidX

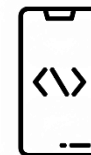
```
implementation 'com.huawei.hms:fido-bioauthn-  
androidx:{version}'
```

BioAuthn

```
implementation 'com.huawei.hms:fido-bioauthn:{version}'
```

FIDO2 incluye dos operaciones: registro y autenticación. Los procesos son similares para las dos operaciones.





1 Inicialice una instancia de Fido2Client.

```
Fido2Client fido2Client = Fido2.getFido2Client(activity);
```

2 Obtenga un valor ‘challenge’ y la política de registro o autenticación del servidor FIDO e inicie una solicitud de autenticación o registro en función de la información obtenida.

```
1.ServerPublicKeyCredentialCreationOptionsResponse response = fidoServer.getAttestationOptions(request);
2.if (!ServerStatus.OK.equals(response.getStatus())) {
3.Log.e(TAG, getString(R.string.reg_fail) + response.getErrorMessage());
4.showError(getString(R.string.reg_fail) + response.getErrorMessage());
5.}
6.PublicKeyCredentialCreationOptions publicKeyCredentialCreationOptions =
7.ServerUtils.convert2PublicKeyCredentialCreationOptions(fido2Client,response);
```



3

Llame a `FIDO2client.getRegistrationintent ()` o `FIDO2client.getAuthenticationIntent ()` para iniciar el registro o la autenticación.

4

Llame a `FIDO2Intent.LaunchFIDO2Activity` en `Fido2IntentCallback` para iniciar la página de registro o autenticación.

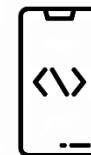
```
1.fido2Client.getRegistrationIntent(registrationRequest, registrationOptions, new Fido2IntentCallback() {  
2.@Override  
3.public void onSuccess(Fido2Intent fido2Intent) {  
4.fido2Intent.launchFido2Activity(Fido2DemoMainActivity.this, Fido2Client.REGISTRATION_REQUEST);  
5.}  
6.  
7.@Override  
8.public void onFailure(int errorCode, CharSequence errString) {  
9.showError(getString(R.string.reg_fail) + errorCode + "=" + errString);  
10.}  
11.});
```

5

Llame a `getFido2RegistrationResponse` o `getFido2AuthenticationResponse` en `activity.onActivityResult` para obtener la respuesta de registro o autenticación.

Desarrollo

Integración de
FIDO2 Client



```
1.switch (requestCode) {
2.// Receive the registration response.
3.case Fido2Client.REGISTRATION_REQUEST:
4.Fido2RegistrationResponse fido2RegistrationResponse =
fido2Client.getFido2RegistrationResponse(data);
5.reg2Server(fido2RegistrationResponse);
6.break;
7.
8.// Receive the authentication response.
9.case Fido2Client.AUTHENTICATION_REQUEST:
10.Fido2AuthenticationResponse fido2AuthenticationResponse =
11.fido2Client.getFido2AuthenticationResponse(data);
12.auth2Server(fido2AuthenticationResponse);
13.break;
14.default:
15.break;
16.}
```

Autenticación de huellas dactilares

Desarrollo

Integración de
BioAuth SDK

1 Crear `FingerprintManager` y `BioAuthnCallback`

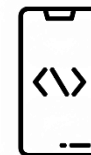
```
1.private FingerprintManager createFingerprintManager() {
2.// Callback.
3.BioAuthnCallback callback = new BioAuthnCallback() {
4.@Override
5.public void onAuthError(int errMsgId, CharSequence errString) {
6.showResult("Authentication error. errorCode=" + errMsgId +
7.,errorMessage=" + errString);
8.}
9.@Override
10.public void onAuthSucceeded(BioAuthnResult result) {
11.showResult("Authentication succeeded. CryptoObject=" +
12.result.getCryptoObject());
13.}
14.@Override
15.public void onAuthFailed() {
16.showResult("Authentication failed.");
17.};
18.return new FingerprintManager(this,
19.Executors.newSingleThreadExecutor(), callback);
19.}
```

2

Realizar autenticación

```
1.fingerprintManager.auth();
```

Autenticación facial 3D



1 Construya `BioAuthnCallback`.

```
1.// Callback.
2.BioAuthnCallback callback = new BioAuthnCallback() {
3.@Override
4.public void onAuthError(int errMsgId, CharSequence errString) {
5.showResult("Authentication error. errorCode=" + errMsgId + ",errorMessage=" + errString
6.+ (errMsgId == 1012 ? " The camera permission may not be enabled." : ""));
7.}
8.
9.@Override
10.public void onAuthHelp(int helpMsgId, CharSequence helpString) {
11.resultTextView
12..append("Authentication help. helpMsgId=" + helpMsgId + ",helpString=" + helpString + "\n");
13.}
14.
15.@Override
16.public void onAuthSucceeded(BioAuthnResult result) {
17.showResult("Authentication succeeded. CryptoObject=" + result.getCryptoObject());
18.}
19.
20.@Override
21.public void onAuthFailed() {
22.showResult("Authentication failed.");
23.}
24.};
```



2 Construya `FaceManager` y llame el método `auth`.

```
1.// Cancellation Signal
2.CancellationSignal cancellationSignal = new CancellationSignal();
3.
4.FaceManager faceManager = new FaceManager(this);
5.
6.// Flags.
7.int flags = 0;
8.
9.// Authentication message handler.
10.Handler handler = null;
11.
12.// Se recomienda establecer CryptoObject en nulo. KeyStore no está asociado con la autenticación
13.// facial en el versión actual. KeyGenParameterSpec.Builder.setUserAuthenticationRequired () debe
14.// establecerse en falso en este escenario.
15.
16.CryptoObject crypto = null;
17.
18.faceManager.auth(crypto, cancellationSignal, flags, callback, handler);
```

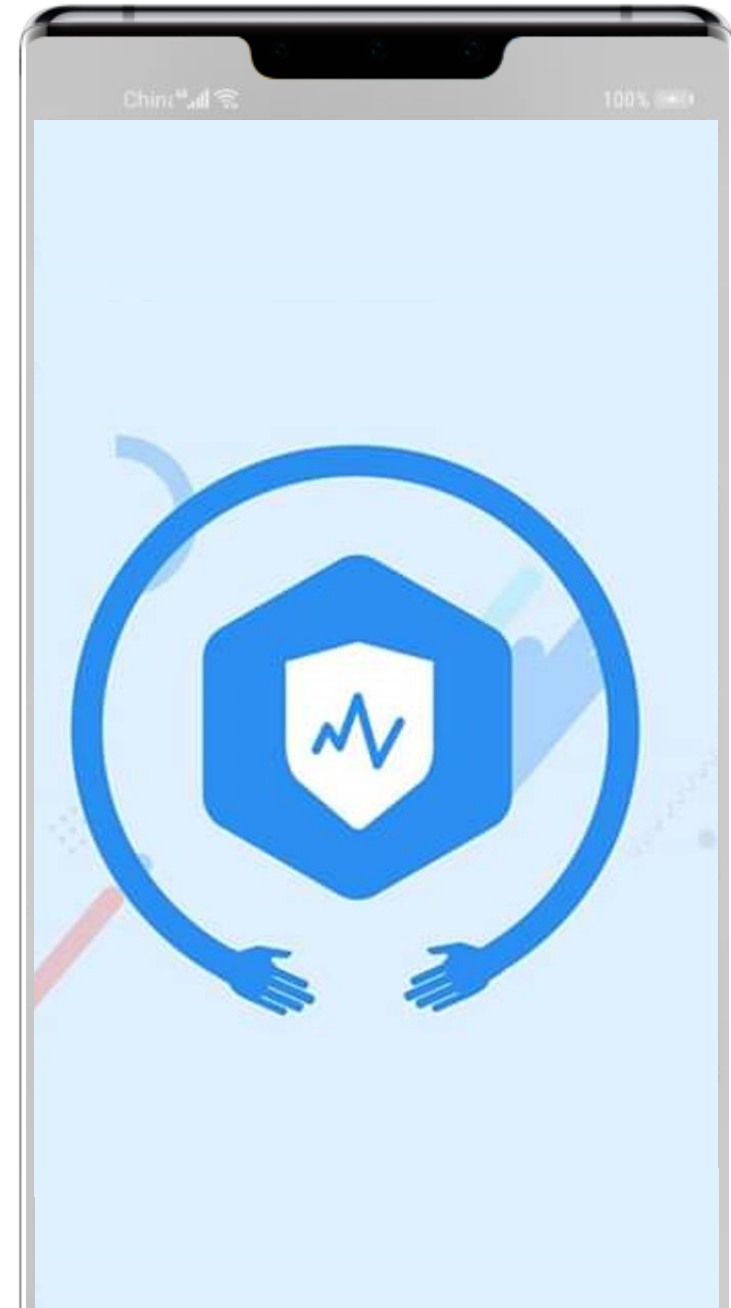


Solo EMUI 10 (API nivel 29) y versiones posteriores admiten la autenticación facial 3D. Además, el hardware del dispositivo debe admitir la autenticación facial 3D. (Mate 20 Pro, Mate 30 Pro, Honor Magic 2)

HMS



SAFETY DETECT



CONTENIDO

Introducción a
Huawei Safety Detect

Interacción

Preguntas y
respuestas

1

3

5

2

4

6

Ventajas

Pasos para
integrar

Referencias

Comprueba si su dispositivo está rooteado, desbloqueado o vulnerable, para que pueda administrar el uso de su aplicación.

Reconoce enlaces potenciales a sitios web maliciosos y determina el tipo de riesgo para que responda de manera adecuada.



Genera una lista marcando aplicaciones maliciosas instaladas en el dispositivo del usuario.

Identifica si el usuario es humano o máquina.

Ventajas de implementar Safety Detect

Seguridad

Proporciona un entorno de ejecución confiable (TEE) para verificar la integridad del sistema.



Conveniencia

Facilita la creación de seguridad en su aplicación con un asistente de integración rápida.



Versatilidad

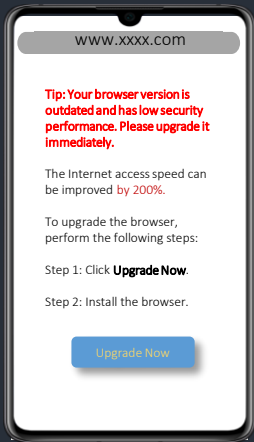
Comprueba la seguridad de una diversidad de aplicaciones: comercio electrónico, finanzas, multimedia y noticias.



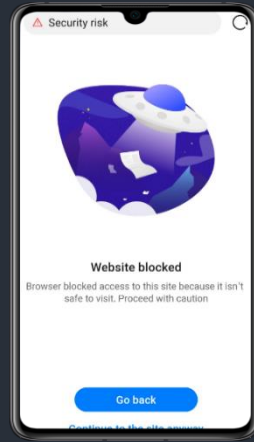
Safety Detect: URLCheck y UserDetect

URLCheck

- 1 Checklist: PHISHING, MALWARE, FakeAlert-Screens
- 2 Deteccion en tiempo real



Alertas falsas



Bloqueo de URL maliciosas

UserDetect

- 1 Identificación de riesgos de ambiente: Root/Simulator/VM/Device change tool/Anonymous IP address
- 2 Análisis de comportamiento: screen touch/sensor behavior analysis



Intercepción de robots

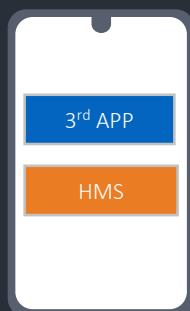


Usuario valido

Safety Detect: SysIntegrity y AppsCheck



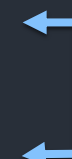
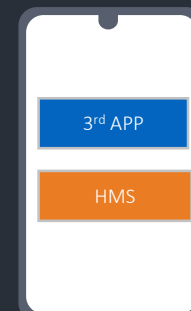
SysIntegrity



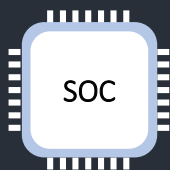
Obtiene el resultado de SysIntegrity



AppsCheck



Obtiene una lista de Apps maliciosas



1

El resultado de la comprobación de la integridad del sistema se basa en un arranque seguro.

2

El resultado de la verificación proviene del entorno de ejecución confiable (TEE).

3

Se utiliza una firma de certificado de servidor para garantizar la integridad del resultado de la verificación.



1

Tasa de detección de virus: 99%

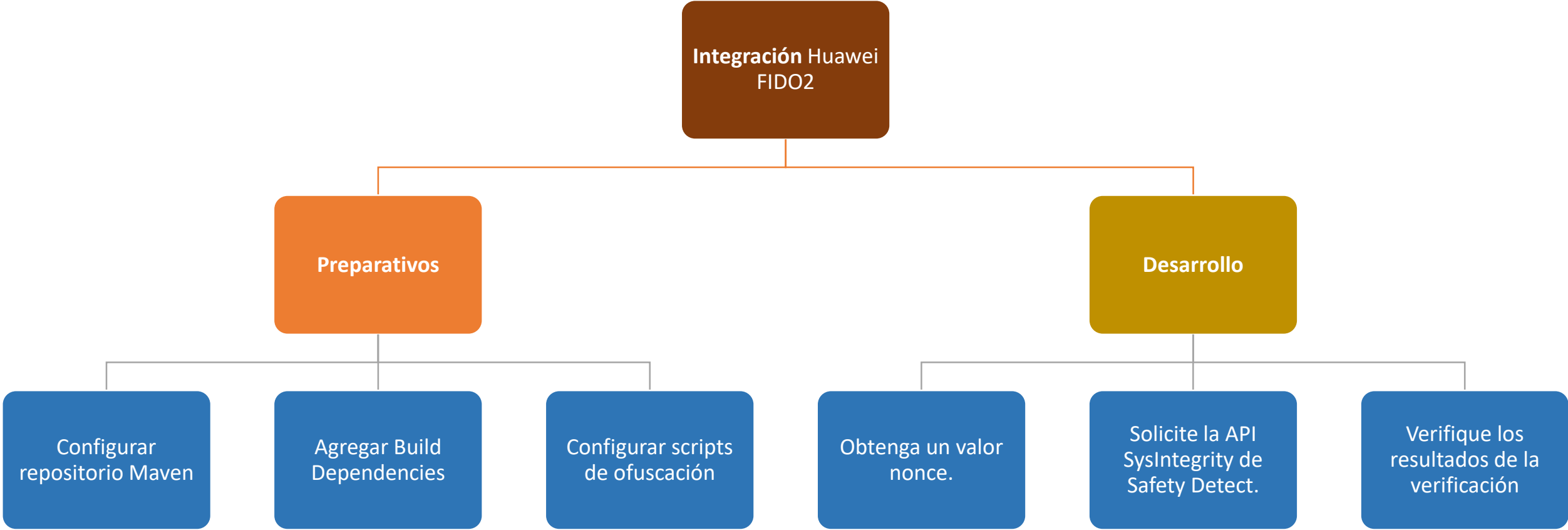
2

Capacidad para detectar amenazas desconocidas según el comportamiento

SENCILLO de integrar




Safety Detect solo funciona en dispositivos Huawei con HMS Core 4.0.0.300 o posterior



4) Configurar las dependencias al repositorio Maven

```

1  |
2  | buildscript {
3  |     ext.kotlin_version = "1.3.72"
4  |     repositories {
5  |         google()
6  |         jcenter()
7  |         maven {url 'https://developer.huawei.com/repo/'}
8  |     }
9  |     dependencies {
10 |         
11 |
12 |         classpath 'com.huawei.agconnect:agcp:1.4.1.300'
13 |     }
14 | }
15 |
16 | allprojects {
17 |     repositories {
18 |         google()
19 |         jcenter()
20 |         maven {url 'https://developer.huawei.com/repo/'}
21 |     }
22 | }
23 |
24 | task clean(type: Delete) {
25 |     delete rootProject.buildDir
26 | }

```

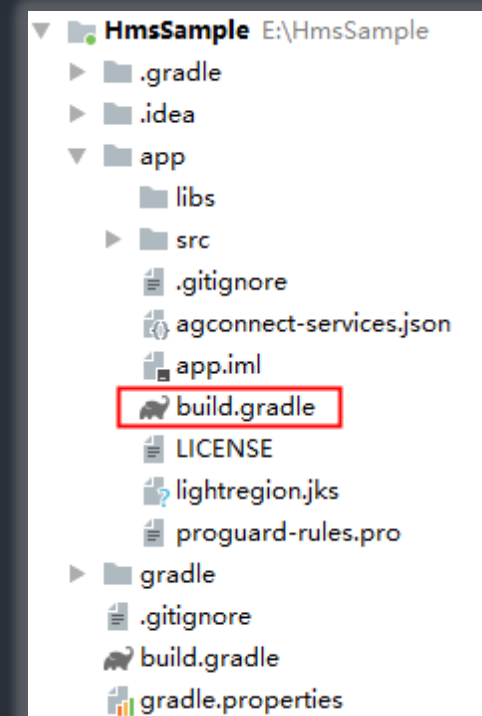


4) Agregar las dependencias a los HMS Game Services

```

dependencies {
    implementation 'com.huawei.hms:safetymdetect:{version}'
}

```





```
private void invokeSysIntegrity() {
    SafetyDetectClient mClient = SafetyDetect.getClient(getActivity());
```

```
    byte[] nonce = ("Sample" + System.currentTimeMillis()).getBytes();
    SysIntegrityRequest sysintegrityrequest = new SysIntegrityRequest();
    sysintegrityrequest.setAppid("3*****");
    sysintegrityrequest.setNonce(nonce);
    sysintegrityrequest.setAlg(alg);
```

```
    Task task = mClient.sysIntegrity(sysintegrityrequest);
    task.addOnSuccessListener(new OnSuccessListener<SysIntegrityResp>() {
        @Override
        public void onSuccess(SysIntegrityResp response) {
            String jwsStr = response.getResult();
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(Exception e) {
            if (e instanceof ApiException) {
                ApiException apiException = (ApiException) e;
                Log.e(TAG, "Error: " +
                    SafetyDetectStatusCodes.getStatusCodeString(apiException.getStatusCode()) +
                    ": " + apiException.getMessage());
            } else {
                Log.e(TAG, "ERROR:" + e.getMessage());
            }
        }
    });
}
```

1

Obtenga un valor nonce.

2

Realice una petición a la API
SysIntegrity de Safety Detect.



3

Verificación del resultado de la verificación en su servidor

HMS Core (APK) envía el resultado de la verificación al servidor de Huawei para verificar el certificado y le devuelve el resultado a través del objeto **SysIntegrityResp**. Puede utilizar el método **getResult** del objeto para obtener la respuesta en formato **JSON Web Signature (JWS)**.

1. Analice el resultado del formato **JWS** para obtener el encabezado, la carga útil y la firma.
2. Obtenga la cadena de certificados del **header** y use el certificado de CA raíz CBG de HUAWEI para verificarla.
3. Verifique el nombre de dominio del certificado hoja en la cadena de certificados. El nombre de dominio correcto es **sysintegrity.platform.hicloud.com**.
4. Obtenga la firma de **signature** y verifíquela.
5. Obtenga el resultado de la verificación de integridad de **payload**. El formato y el ejemplo son los siguientes





Si el valor de **basicIntegrity** es falso en el resultado de la verificación, puede decidir si recordar a los usuarios los riesgos en función de sus requisitos de seguridad.

```
{
  "apkCertificateDigestSha256": [
    "osaUtTsdAvezjQBaw3IhN3/fsc6NQ5KwKuAQXcfrxb4="
  ],
  "apkDigestSha256":
    "vFcmE0uw5s+4tFjXF9rVycxk2xR1rXiZFHuuBFzTVy8=",
  "apkPackageName": "com.example.mockthirdapp",
  "basicIntegrity": false,
  "detail": [
    "root",
    "unlocked"
  ],
  "nonce": "UjJScmEyNGZWbTV4YTJNZw==",
  "timestampMs": 1604048377137,
  "advice": "RESTORE_TO_FACTORY_ROM"
}
```





1 La aplicación integra el SDK de Safety Detect y llama a la API de AppsCheck.

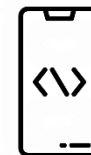
2 La API devuelve una lista de aplicaciones maliciosas a su aplicación.

```
SafetyDetectClient appsCheckClient = SafetyDetect.getClient(MainActivity.this);
Task task = appsCheckClient.getMaliciousAppsList();
task.addOnSuccessListener(new OnSuccessListener<MaliciousAppsListResp>() {
    @Override
    public void onSuccess(MaliciousAppsListResp maliciousAppsListResp) {
        List<MaliciousAppsData> appsDataList = maliciousAppsListResp.getMaliciousAppsList();
        if(maliciousAppsListResp.getRtnCode() == CommonCode.OK) {
            if (appsDataList.isEmpty()) {
                Log.i(TAG, "There are no known potentially malicious apps installed.");
            } else {
                Log.i(TAG, "Potentially malicious apps are installed!");
                for (MaliciousAppsData maliciousApp : appsDataList) {
                    Log.i(TAG, "Information about a malicious app:");
                    Log.i(TAG, "APK: " + maliciousApp.getApkPackageName());
                }
            }
        } else {
            Log.e(TAG, "getMaliciousAppsList failed: "+maliciousAppsListResp.getErrorReason());
        }
    }
}).addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(Exception e) {
        ...
    }
});
```



AppsCheck no requiere internet
para realizar el escaneo de
aplicaciones.





1 Inicializar la API URLCheck

Antes de usar la API **URLCheck**, debe llamar al método **initUrlCheck** para inicializar la API.

```
SafetyDetectClient client = SafetyDetect.getClient(getActivity());  
client.initUrlCheck();
```

2 Solicite una revisión de URL.

Especifique los tipos de amenazas en cuestión como parámetro de entrada de la API URLCheck. Las constantes de la clase **UrlCheckThreat** contienen los tipos de amenazas admitidos.

```
public class UrlCheckThreat {  
    // URLs of this type are marked as URLs of pages containing potentially malicious apps  
    public static final int MALWARE = 1;  
    // URLs of this type are marked as phishing and spoofing URLs.  
    public static final int PHISHING = 3;  
}
```





Inicie una solicitud de verificación de URL. La URL que se va a comprobar contiene el protocolo, el host y la ruta, pero no el parámetro de consulta.

```
String url = "https://developer.huawei.com/consumer/cn/";
SafetyDetect.getClient(this).urlCheck(url, appId, UrlCheckThreat.MALWARE,
UrlCheckThreat.PHISHING).addOnSuccessListener(this, new OnSuccessListener<UrlCheckResponse >(){
@Override
public void onSuccess(UrlCheckResponse urlResponse) {
if (urlResponse.getUrlCheckResponse().isEmpty()) {
// No threat exists.
} else {
// Threats exist.
}
}
}).addOnFailureListener(this, new OnFailureListener() {
@Override
public void onFailure(@NonNull Exception e) {
...
}
});
```

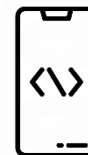




Llame al método `getUrlCheckResponse ()` del objeto `UrlCheckResponse` para obtener la respuesta de verificación de URL. El método devuelve `List <UrlCheckThreat>` que contiene una lista de todos los tipos de amenazas de URL detectadas. Si la lista está vacía, no se ha detectado ninguna amenaza. De lo contrario, puede llamar al método `getUrlCheckResult` de `UrlCheckThreat` para obtener el código de amenaza específico.

```
final EditText testRes = getActivity().findViewById(R.id.fg_call_urlResult);
List<UrlCheckThreat> list = urlCheckResponse.getUrlCheckResponse();
if (list.isEmpty()) {
    testRes.setText("ok");
}
else{
    for (UrlCheckThreat threat : list) {
        int type = threat.getUrlCheckResult();
    }
}
```





3

Cierre la sesión de verificación de URL.

Si su aplicación ya no necesita llamar a la API **URLCheck** o no necesitará usarlo por un tiempo, puede llamar al método **shutdownUrlCheck** para cerrar la sesión de verificación de URL y liberar los recursos relevantes.

```
SafetyDetect.getClient(this).shutdownUrlCheck();
```





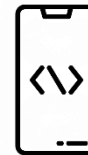
1

Inicialice UserDetect

La API **UserDetect** proporciona la capacidad de detección de comportamiento. Para usar la capacidad, puede llamar a la API **initUserDetect** () para inicializar la detección de usuarios falsos.

```
private void initUserDetect() {
    SafetyDetectClient client = SafetyDetect.getClient(MainActivity.this);
    client.initUserDetect().addOnSuccessListener(new OnSuccessListener<Void>() {
        @Override
        public void onSuccess(Void v) {
            // Indica que la comunicación con el servicio fue exitoso.
        }
    }).addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(Exception e) {
            // Ocurrió un error de comunicación con el servicio.
        }
    });
}
```





2 Para llamar a la API **UserDetect**, realice los siguientes pasos:

Llame al método **userDetection** () de la API **UserDetect** para iniciar una solicitud de detección de usuario falsa. La API devolverá un objeto **responseToken**.

Para iniciar una solicitud de detección, debe llamar al método **userDetection**. Generalmente, este método se activa cuando un usuario toca un control de **IU** (como un botón). Para llamar al método **userDetection**, realice los siguientes pasos:

- Pase el **appId** aplicado como parámetro de entrada del método.
- Agregue las instancias **OnSuccessListener** y **OnFailureListener** como oyentes.
- Anule **onSuccess** y **onFailure** para procesar el resultado.





```
public void onClick(View v) {
    SafetyDetectClient client = SafetyDetect.getClient(getActivity());
    String appId = "your_app_id";
    client.userDetection(appId)
        .addOnSuccessListener(new OnSuccessListener<UserDetectResponse>() {
            @Override
            public void onSuccess(UserDetectResponse userDetectResponse) {
                // Indica que la comunicación con el servicio fue exitoso.
                String responseToken = userDetectResponse.getResponseToken();
                if (!responseToken.isEmpty()) {
                    //Envíe el token de respuesta a su servidor de aplicaciones y llame a la API en la nube de HMS Core en su
                    //servidor para obtener el resultado de detección de usuario falso.
                }
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(Exception e) {
                // Ocurrió un error de comunicación con el servicio.
            }
        });
}
```





3

Llame a una API en la nube (nocaptcha en China continental y verifique fuera de China continental) en función del token de respuesta obtenido en 1 para obtener el resultado de detección del usuario.

Para obtener el resultado de la detección, realice los siguientes pasos en el servidor:

- a. Obtén un token de acceso. Para obtener más información, consulte Autenticación basada en OAuth 2.0.
- b. Llame a la API de la nube para obtener el resultado de la detección. Para obtener más información, consulte Obtención de resultados de detección de usuarios falsos (fuera de China continental). El siguiente es un ejemplo de solicitud:

```
POST https://hirms.cloud.huawei.com/rms/v1/userRisks/verify?appId=***** HTTP/1.1
Content-Type: application/json;charset=utf-8
```

```
{
  "accessToken": "AAWWHI94sgUR2RU5_P1ZptUiwLq7W8XWJ02LxaAPuXw4_H0JFXnB1N-q5_3bw1xVW_SHeDPx_s5bWW-9DjtWZsvcm9CwXe1FHJg0u-D2pcQPcb3sTxDTJeiwEb9WBP1_9w",
  "response": "1_55d74c04eab36a0a018bb7a879a6f49f072b023690cba936"
}
```





4

Puede llamar a la API `shutdownUserDetect ()` para deshabilitar UserDetect y liberar recursos.

```
private void shutdownUserDetect() {  
    SafetyDetectClient client = SafetyDetect.getClient(MainActivity.this);  
    client.shutdownUserDetect().addOnSuccessListener(new OnSuccessListener<Void>() {  
        @Override  
        public void onSuccess(Void v) {  
            // Indica que la comunicación con el servidor fue exitoso.  
        }  
    }).addOnFailureListener(new OnFailureListener() {  
        @Override  
        public void onFailure(Exception e) {  
            // Ocurrió un error de comunicación con el servicio.  
        }  
    });  
}
```





¿Quieres saber mas?



Referencia de API

Consulte nuestra amplia biblioteca de recursos.

<https://developer.huawei.com/consumer/en/doc/development/HMSCore-References/fido2overview-0000001050176660>

<https://developer.huawei.com/consumer/en/doc/development/HMSCore-Guides/dev-process-0000001050990069>



SDK

Descargue la versión más reciente de SDK.

<https://developer.huawei.com/consumer/en/doc/development/HMSCore-Library/sdk-download-0000001050159937>

<https://developer.huawei.com/consumer/en/doc/development/HMSCore-Examples/sample-code-0000001050158985>



Guía de desarrollo

Descargue nuestras instrucciones más recientes para la integración del kit.



Código de muestra

Inspírese con nuestra completa colección de casos de estudio.

