Exposure Notification

Android API Documentation

Preliminary - Subject to Modification and Extension

May 2020
v1.3

Table of contents

Android Exposure Notification API

This document shows developers how to use the Exposure Notification API provided by Apple and Google to build Android apps for notifying users of possible exposure to confirmed COVID-19 cases.

Note: This document is released in advance of the SDK, itself. Verify that you are reading the correct version of the documentation for the version of the SDK you have.

This document contains the following sections:

- Architecture: provides an understanding of how the system is distributed across different sub-systems.
- Data structures and Methods: explain how to interact with the API from your app.
- Broadcast receivers: explains how to define broadcast receivers in your Android manifest and how to respond to broadcasts received.
- Error handling: describes the error codes to check for when calling methods, and provides suggestions for how to respond to them.
- Reference design: links to the project on GitHub that provides examples of how to interface with the API.
- Glossary: provides definitions of terms specific to Exposure Notification.
- API reference: provides the reference documentation for all of the classes, methods, and fields.
- Revision history: lists the changes made to this document and the API with each release.

Architecture

The Exposure Notification system spans several sub-systems. The following list explains this distribution.

Google Play services

> Bluetooth functionality, including all broadcast and scanning for BLE beacons and local database storage, happens within Google Play services on-device. Your app will need to have the `BLUETOOTH` permission in its manifest, but does not require and cannot include `ACCESS_COARSE_LOCATION`, `ACCESS_FINE_LOCATION`, or `BLUETOOTH_ADMIN`. For more information about restrictions on your app, see the API's Terms of Service.
>
> For your app to work on a device, the device must be running Android version 6.0 (API version 23) or higher.

The following is the full list of the parts of the system that Google Play services handles:

- Manages daily random keys:
    - Generates daily random Temporary Exposure Keys and Rotating Proximity Identifiers (RPIs) based on them.
    - Provides keys to the application for diagnosed users, including an interval number that indicates the key date.
    - Accepts keys from the application for exposure detection, including an interval number that indicates the key date and an transmission risk level.
    - Stores key data in an on-device data store.
- Manages Bluetooth broadcast and collection:
    - Enables broadcast beacons.
    - Scans for other beacons.
    - Stores observed RPIs in an on-device data store.
- Identifies whether the user was in close contact with a confirmed case.
    - Calculates and provides an exposure risk level to the app.
- Presents permission requests to users at the following points:
    - Before starting to scan for and broadcast beacons.
    - Before providing user keys to the app for uploading to the internet-accessible server once the user has been positively diagnosed with COVID-19.

This part of the system has no user interface other than the permission dialogs and notification message, and it runs in a low-power-consuming and privacy-centric way.

Mobile app

Your app does the following:

- Enables users to start and stop broadcasting and scanning.
- Provides Temporary Exposure Keys, key start time number, and key transmission risk level from your internet-accessible server to Google Play services.

- Retrieves keys from the on-device data store and submits them to your internet-accessible server after a user has been confirmed by a medical provider as having tested positive, and the user has provided permission.
- Schedules polling of your internet-accessible server for keys.
- Registers a receiver to receive broadcasts of the `com.google.android.gms.exposurenotification.ACTION_EXPOSURE_STATE_UPDATED` intent, and responds by presenting information to the user. See Broadcast receivers for more information.
- Shows a notification to the user with instructions on what to do next when the user has been exposed to another user who has tested positive for COVID-19.

Authenticated medical provider validation mechanism

Your app must provide functionality that confirms that the user has been positively diagnosed with COVID-19. This functionality controls the release of Temporary Exposure Keys stored on a device whose user is positively diagnosed. The released keys are sent to your app, which then uploads them to your internet-accessible server.

Internet-accessible server

Your app must be able to communicate with an internet-accessible server that you own and that performs the following functions:

- Collects diagnosis keys from users who have been diagnosed with COVID-19.
- When polled, distributes diagnosis keys of confirmed cases to devices.

Data structures

The API contains the data classes described in the following list.

`Status`

This class contains codes that represent the state of the service on the device. For more information on how to use the information in this class, see Error handling.

`ExposureConfiguration`

The `ExposureConfiguration` class contains fields that your app populates before it passes the class object into the `provideDiagnosisKeys()` and `getExposureSummary()` methods:

- `minimumRiskScore`, an integer value that specifies how strong a calculated risk needs to be in order to be recorded.

- `attenuationScores`, an array in which you specify how much risk to associate with the Bluetooth attenuation value from an exposure. The `attenuationWeight` field is reserved for future use.
- `daysSinceLastExposureScores`, an array in which you specify how much risk to associate to an exposure based on the number of days since the exposure happened. The `daysSinceLastExposureWeight` field is reserved for future use.
- `durationScores`, an array in which you specify how much risk to associate with an exposure based on the duration of the exposure. The `durationWeight` field is reserved for future use.
- `transmissionRiskScores`, an array of custom values that specify how much risk to associate with a risk factor you designate. For example, you might associate the Highest Risk value with a user who has recently tested positive, and the Medium Risk for a user who came into contact with a positively-diagnosed person.

These fields are used in the following calculation to determine a `RiskScore`, which must exceed the `minimumRiskScore` value, for the exposure to be recorded:

`RiskScore` = `attenuationScore` * `daysSinceLastExposureScore` * `durationScore` * `transmissionRiskScore`

`TemporaryExposureKey`

The `TemporaryExposureKey` class holds the following:

- `keyData` that is used to generate broadcasts that are collected on the other devices. These connect to provide a record of the interaction between two devices. This information remains on a device until and unless the user tests positive, at which point the user can choose to share that information with the internet-accessible server.
- a `rollingStartNumber` that describes the time at which the key was generated.
- a `rollingPeriod` that stores the expiration date of the key
- a `transmissionRiskLevel`, which specifies the level of risk of cross-exposure during the interaction between devices.

`ExposureSummary`

The `ExposureSummary` class stores the data in the following list for a set of exposures to the same confirmed case:

- The number of days since the last match to a diagnosis key (`daysSinceLastExposure`).
- The number of matched diagnosis keys (`matchedKeyCount`).
- The highest transmission risk level of all exposure incidents (`maximumRiskScore`).

- An array that contains the sum of all durations for all exposures when radio signal attenuation was <= 50 (index 0) and when the attenuation was > 50 (index 1) (`attenuationDurations`). The durations are aggregated across all exposures to the same device.
- A summation of all risk scores of all exposure incidents to the same confirmed case (`summationRiskScore`).

Your app can present this information to users.

`ExposureInformation`

The `ExposureInformation` class stores information about an interaction with another device. It stores the day that the interaction occurred (`dateMillisSinceEpoch`), the duration of the exposure in five-minute increments (`durationMinutes`), the time-weighted average of the attenuation (`attenuationValue`), and a transmission risk value (`transmissionRiskLevel`). You can use this information to gauge a level of risk of the exposure to filter out low-risk interactions, such as sub-five-minute interactions with a low signal strength attenuation that occurred 13 days earlier.

Each time this information is requested using `getExposureInformation()`, Google Play services displays a notification to the user.

Methods

The API provides functionality using the methods shown in the following list. These are provided in a roughly chronological order of usage by an app.

Return values from methods are wrapped in a `Task` object to enable asynchronous operations.

These methods might produce errors. Make sure you're following best practices by addressing errors as they occur. For more information on errors, see Error handling.

`start()`

Tells Google Play services to start the broadcasting and scanning process. The first time that this method is called after installation of the app, it prompts Google Play services to display a dialog box, where the user is asked to give permission to broadcast and scan.

`isEnabled()`

Indicates if exposure notifications are running.

`stop()`

Disables broadcasting and scanning. You can call this directly, and it is also called when users uninstall the app. When it's called as part of the uninstallation process, the database and keys are deleted from the device.

`getTemporaryExposureKeyHistory()`

Retrieves key history from the data store on the device for uploading to your internet-accessible server. Calling this method prompts Google Play services to display a dialog, requesting permission from the user to gather and upload their exposure keys.

`provideDiagnosisKeys()`

Takes an `ExposureConfiguration` object. Inserts a list of files that contain key information into the on-device database. Provide the keys of confirmed cases retrieved from your internet-accessible server to the Google Play service once requested from the API. Information about the file format will be released soon.

`getExposureSummary()`

Retrieves the `ExposureSummary` object that matches the token from `provideDiagnosisKeys()` that you provide to the method. The `ExposureSummary` object provides a high-level overview of the exposure that a user has experienced.

`getExposureInformation()`

Provides a more in-depth version of the information provided by `getExposureSummary()`. Passing in a token from `provideDiagnosisKeys()` for an exposure key provides a list of `ExposureInformation` objects, from which you can gauge the level of risk of the exposure with the user.

Google Play services displays a notification to the user each time this method is invoked.

Broadcast receivers

Your app should register a receiver to receive broadcasts of the `ACTION_EXPOSURE_STATE_UPDATED` intent. This intent is broadcast when diagnosis keys have been compared with the keys on the device, which follows calls to `provideDiagnosisKeys()`. Respond to the broadcast by presenting information to the user. The following code shows a sample of how to do this:

```
public class ExposureNotificationBroadcastReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent) {
    String action = intent.getAction();
    if (ExposureNotificationClient.ACTION_EXPOSURE_STATE_UPDATED
        .equals(action))
      {
        // Handle action
      }
  }
}
```

When declaring the receiver, you must add the `com.google.android.gms.nearby.exposurenotification.EXPOSURE_CALLBACK` permission in the receiver declaration in your Android manifest.

The following code snippet shows the entry for your manifest:

```
<receiver
    android:name=".exposurenotification.app.ExposureNotificationBroadcast"
    android:permission="com.google.android.gms.nearby.exposurenotification
                        .EXPOSURE_CALLBACK">
    <intent-filter>
        <action android:name="com.google.android.gms.exposurenotification
                              .ACTION_EXPOSURE_STATE_UPDATED" />
    </intent-filter>
</receiver>
```

For more information on broadcasts, see Broadcasts overview on https://developer.android.com.

Error handling

Method calls in your app may produce errors. In those cases, use `Task.getException()`, cast the returned object to the `APIException` class, and then get the `Status` via `getStatus()` method, which provides information for troubleshooting. For example, the status might indicate that the user hasn't opted-in, the service isn't running, or that there is insufficient storage on the device to complete an operation.

The following list contains the error statuses to address:

- **`SUCCESS`**. This status is returned when your operation was successful.
- **`FAILED_REJECTED_OPT_IN`**. This status is returned when the app user has not provided permission for the app to scan for and broadcast beacons.
- **`FAILED_SERVICE_DISABLED`**. This status is returned when the user has disabled the service. See `stop()` for more information.
- **`FAILED_BLUETOOTH_SCANNING_DISABLED`**. This status is returned when the Bluetooth capabilities of the device are disabled.
- **`FAILED_TEMPORARILY_DISABLED`**. The service has been temporarily disabled by the user but will resume again shortly.
- **`FAILED_INSUFFICENT_STORAGE`**. This status indicates that the device the app is running on does not have sufficient space to store the keys being downloaded from your internet-accessible server. Inform the user of their need to clear up space on the device.
- **`FAILED_INTERNAL`**. This status indicates that a general error has occurred. Retry the operation to see if it is due to a transient error.

Reference design

For examples of how to integrate with the Exposure Notifications API, see the reference design source code hosted on GitHub.

Glossary

BLE beacons

> Bluetooth Low Energy (BLE) beacons enable Bluetooth-equipped devices to share information when they are within range of each other.

Diagnosis key

The Temporary Exposure Keys provided by the server that have been confirmed to have a positive diagnosis of COVID-19. Used by the app to check for exposure.

Rotating Proximity Identifiers (RPIs)

A random ID derived from the device's Temporary Exposure Key. The RPI is generated on the device and a new RPI is generated every 10-20 minutes.

Temporary Exposure Key

A key that is randomly generated once every 24 hours. It remains on the device for up to 14 days. In the event there is a positive diagnosis of COVID-19, and upon granting permission from the user, these keys are provided to the app.

Transmission Risk

A value defined by your app that specifies how much risk is associated with a given exposure based on a value of importance to your health authority. This enables you to add an additional, non-API-specified value to the computation of total risk.

API reference

```
/**
 * Starts BLE broadcasts and scanning based on the defined protocol.
 *
 * If not previously started, this shows a user dialog for consent to start
 * exposure detection and get permission.
 *
 * Callbacks regarding exposure status will be provided via a BroadcastReceiver.
 * Clients should register a receiver in their AndroidManifest which can handle the
 * following action:
 * <ul>
 *   <li><code>
 *       com.google.android.gms.exposurenotification.ACTION_EXPOSURE_STATE_UPDATED
 *       </code>
 * </ul>
 *
 * This receiver should also be guarded by the <code>
 * com.google.android.gms.nearby.exposurenotification.EXPOSURE_CALLBACK</code>
 * permission so that other apps are not able to fake this broadcast.
 */
Task<Void> start();
```

```
@IntDef({...})
@interface Status {
  int SUCCESS = 0;
  int FAILED_REJECTED_OPT_IN = 1;
  int FAILED_SERVICE_DISABLED = 2;
  int FAILED_BLUETOOTH_SCANNING_DISABLED = 3;
  int FAILED_TEMPORARILY_DISABLED = 4;
  int FAILED_INSUFFICENT_STORAGE = 5;
  int FAILED_INTERNAL = 6;
}

/**
 * Exposure configuration parameters that can be provided when initializing the
 * service.
 *
 * These parameters are used to calculate risk for each exposure incident using
 * the following formula:
 *
 * <p><code>
 *   RiskScore = attenuationScore
 *       * daysSinceLastExposureScore
 *       * durationScore
 *       * transmissionRiskScore
 * </code>
 *
 * <p>Scores are in the range 0-8. Weights are in the range 0-100.
 */
class ExposureConfiguration {
  /**
   * Minimum risk score. Excludes exposure incidents with scores lower than this.
   *
   * Defaults to no minimum.
   */
  int minimumRiskScore;

  /**
   * Scores for attenuation buckets. Must contain 8 scores, one for each bucket
   * as defined below:
   *
   * <p><code>{@code
   * attenuationScores[0] when Attenuation > 73
   * attenuationScores[1] when 73 >= Attenuation > 63
   * attenuationScores[2] when 63 >= Attenuation > 51
   * attenuationScores[3] when 51 >= Attenuation > 33
   * attenuationScores[4] when 33 >= Attenuation > 27
   * attenuationScores[5] when 27 >= Attenuation > 15
   * attenuationScores[6] when 15 >= Attenuation > 10
```

```
     * attenuationScores[7] when 10 >= Attenuation
     * }</code>
     */
    int[] attenuationScores;

    /**
     * Weight to apply to the attenuation score. Must be in the range 0-100.
     *
     * Reserved for future use.
     */
    int attenuationWeight;

    /**
     * Scores for days since last exposure buckets. Must contain 8 scores, one for
     * each bucket as defined below:
     *
     * <p><code>{@code
     * daysSinceLastExposureScores[0] when Days >= 14
     * daysSinceLastExposureScores[1] when Days >= 12
     * daysSinceLastExposureScores[2] when Days >= 10
     * daysSinceLastExposureScores[3] when Days >= 8
     * daysSinceLastExposureScores[4] when Days >= 6
     * daysSinceLastExposureScores[5] when Days >= 4
     * daysSinceLastExposureScores[6] when Days >= 2
     * daysSinceLastExposureScores[7] when Days >= 0
     * }</code>
     */
    int[] daysSinceLastExposureScores;

    /**
     * Weight to apply to the days since last exposure score. Must be in the
     * range 0-100.
     *
     * Reserved for future use.
     */

    int daysSinceLastExposureWeight;

    /**
     * Scores for duration buckets. Must contain 8 scores, one for each bucket as
     * defined below:
     *
     * <p><code>{@code
     * durationScores[0] when Duration == 0
     * durationScores[1] when Duration <= 5
     * durationScores[2] when Duration <= 10
     * durationScores[3] when Duration <= 15
```

```
   * durationScores[4] when Duration <= 20
   * durationScores[5] when Duration <= 25
   * durationScores[6] when Duration <= 30
   * durationScores[7] when Duration  > 30
   * }</code>
   */
  int[] durationScores;

  /**
   * Weight to apply to the duration score. Must be in the range 0-100.
   *
   * Reserved for future use.
   */
  double durationWeight;

  /**
   * Scores for transmission risk buckets. Must contain 8 scores, one for each
   * bucket as defined below:
   *
   * <p><code>{@code
   * transmissionRiskScores[0] when RISK_SCORE_LOWEST
   * transmissionRiskScores[1] when RISK_SCORE_LOW
   * transmissionRiskScores[2] when RISK_SCORE_LOW_MEDIUM
   * transmissionRiskScores[3] when RISK_SCORE_MEDIUM
   * transmissionRiskScores[4] when RISK_SCORE_MEDIUM_HIGH
   * transmissionRiskScores[5] when RISK_SCORE_HIGH
   * transmissionRiskScores[6] when RISK_SCORE_VERY_HIGH
   * transmissionRiskScores[7] when RISK_SCORE_HIGHEST
   * }</code>
   */
  int[] transmissionRiskScores;

  /**
   * Weight to apply to the transmission risk score. Must be in the range 0-100.
   *
   * Reserved for future use.
   */
  double transmissionRiskWeight;
}

/** Risk level defined for an {@link TemporaryExposureKey}. */
@IntDef({...})
@interface RiskLevel {
  int RISK_LEVEL_INVALID = 0;
  int RISK_LEVEL_LOWEST = 1;
  int RISK_LEVEL_LOW = 2;
  int RISK_LEVEL_LOW_MEDIUM = 3;
```

```
  int RISK_LEVEL_MEDIUM = 4;
  int RISK_LEVEL_MEDIUM_HIGH = 5;
  int RISK_LEVEL_HIGH = 6;
  int RISK_LEVEL_VERY_HIGH = 7;
  int RISK_LEVEL_HIGHEST = 8;
}

/** A key generated for advertising over a window of time. */
class TemporaryExposureKey {
  /** The randomly generated Temporary Exposure Key information. */
  byte[] keyData;

  /**
   * A number describing when a key starts.
   * It is equal to startTimeOfKeySinceEpochInSecs / (60 * 10).
   */
  long rollingStartIntervalNumber;

  /**
   * A number describing how long a key is valid.
   * It is expressed in increments of 10 minutes (e.g. 144 for 24 hours).
   */
  long rollingPeriod;


  /** Risk of transmission associated with the person this key came from. */
  @RiskLevel int transmissionRiskLevel;
}
```

```
/**
 * Disables advertising and scanning. Contents of the database and keys will
 * remain.
 *
 * If the client app has been uninstalled by the user, this will be automatically
 * invoked and the database and keys will be wiped from the device.
 */
Task<Void> stop();
```

```
/**
 * Indicates whether exposure notifications are currently running for the
 * requesting app.
 */
Task<Boolean> isEnabled();
```

```
 /**
  * Gets {@link TemporaryExposureKey} history to be stored on the server.
  *
  * This should only be done after proper verification is performed on the
  * client side that the user is diagnosed positive.
  *
  * The keys provided here will only be from previous days; keys will not be
  * released until after they are no longer an active exposure key.
  *
  * This shows a user permission dialog for sharing and uploading data to the
  * server.
  */
Task<List<TemporaryExposureKey>> getTemporaryExposureKeyHistory();
```

```
/**
 * Provides a list of diagnosis key files for exposure checking. The files are to
 * be synced from the server. Diagnosis keys older than the relevant period will be
 * ignored.
 *
 * Diagnosis keys will be stored and matching will be performed in the near future,
 * after which you'll receive a broadcast with the
 * {@link #ACTION_EXPOSURE_STATE_UPDATED} action.
 *
 * The diagnosis key files must be signed appropriately. Exposure configuration
 * options can be provided to tune the matching algorithm. A unique token for this
 * batch can also be provided, which will be used to associate the matches with
 * this request as part of {@link #getExposureSummary} and
 * {@link #getExposureInformation}.
 *
 * After the result Task has returned, keyFiles can be deleted.
 */
Task<Void> provideDiagnosisKeys(
    List<File> keyFiles, ExposureConfiguration configuration, String token);
```

```
/**
 * Gets a summary of the exposure calculation for the token, which should match
 * the token provided in {@link #provideDiagnosisKeys}.
 */
Task<ExposureSummary> getExposureSummary(String token);

/**
 * Summary information about recent exposures.
 *
 * The client can get this information via {@link #getExposureSummary}.
 */
```

```
class ExposureSummary {
  /**
   * Days since last match to a diagnosis key from the server. 0 is today, 1 is
   * yesterday, etc. Only valid if {@link #getMatchedKeysCount} > 0.
   */
  int daysSinceLastExposure;

  /** Number of matched diagnosis keys. */
  int matchedKeyCount;

  /**
   * The highest risk score of all exposure incidents, it will be a value 0-4096.
   */
  int maximumRiskScore;

  /**
   * Array of durations in seconds at certain radio signal attenuations.
   *
   * Array index 0: Sum of durations for all exposures when attenuation was <= 50.
   *
   * Array index 1: Sum of durations for all exposures when attenuation was > 50.
   *
   * These durations are aggregated across all exposures and capped at 30 minutes.
   */
  int[] attenuationDurations;

  /** The summation of risk scores of all exposure incidents. */
  int summationRiskScore;
}

/**
 * Gets detailed information about exposures that have occurred related to the
 * provided token, which should match  the token provided in
 * {@link #provideDiagnosisKeys}.
 *
 * When multiple {@link ExposureInformation} objects are returned, they can
 * be:
 * <ul>
 *   <li>Multiple encounters with a single diagnosis key.
 *   <li>Multiple encounters with the same device across key rotation boundaries.
 *   <li>Encounters with multiple devices.
 * </ul>
 *
 * Records of calls to this method will be retained and viewable by the user.
 */
Task<List<ExposureInformation>> getExposureInformation(String token);
```

```
/**
 * Information about an exposure, meaning a single diagnosis key
 * over a contiguous period of time specified by durationMinutes.
 *
 * The client can get the exposure information via {@link #getExposureInformation}.
 */
class ExposureInformation {
  /** Day level resolution that the exposure occurred. */
  long dateMillisSinceEpoch;

  /** Length of exposure in 5 minute increments, with a 30 minute maximum. */
  int durationMinutes;

  /**
   * The signal strength attenuation score which goes into
   * {@link #getTotalRiskScore}. See {@link ExposureConfiguration} for more details
   * about this score.
   */
  int attenuationValue;

  /** The transmission risk associated with the matched diagnosis key. */
  @RiskLevel int transmissionRiskLevel;

  /**
   * The total risk calculated for the exposure. See {@link ExposureConfiguration}
   * for more information about what is represented by the risk score, it will be
   * a value 0-4096.
   */
  int totalRiskScore;

  /**
   * Array of durations in seconds at certain radio signal attenuations.
   *
   * Array index 0: Sum of durations for all exposures when attenuation was <= 50.
   *
   * Array index 1: Sum of durations for all exposures when attenuation was > 50.
   */
  int[] attenuationDurations;
}
```

Revision history

V1.3 - May 7, 2020

- Move providing `ExposureConfiguration` to `provideDiagnosisKeys()` instead of `start()`, allowing different options for different batches
- Pass signed keys files into `provideDiagnosisKeys()`, instead of raw keys
- Requires a token to be provided to `getExposureInformation()` and `getExposureSummary()`, which matches a token provided to `provideDiagnosisKeys()`
- Include `attenuationDurations` in `ExposureSummary` and `ExposureInformation`
- Remove `ACTION_REQUEST_DIAGNOSIS_KEYS` broadcast, keys can be provided to `provideDiagnosisKeys()` at any point
- Update formula for calculating risk score
- Include a `summationRiskScore` in the `ExposureSummary`
- Documentation updates to reflect changes in the API in v1.3
- Removed references to the section “App requirements”
- Added a section on broadcast handlers
- Added a section on error handling
- Added polling of server for keys and handling broadcasts to the list of responsibilities of your app
- Added a section for the reference design
- Replaced instances of “tracing key” with “exposure key”
- Updated glossary definitions for accuracy
- Updated the minimum supported version from API 21 (Android 5.0) to API 23 (Android 6.0)

v1.2 - April 29, 2020

- Add conceptual material to the document
- Add risk level to `TemporaryExposureKey`
- Add risk score to `ExposureSummary` and `ExposureInformation`
- Rename `MatchingOptions` to `ExposureConfiguration`
    - Allow clients to pass in risk score parameters
- Remove handleIntent and `ExposureNotificationCallback` in favor of registering a `BroadcastReceiver`
- Pass `FileDescriptors` with signatures instead of raw keys to `provideDiagnosisKeys`
- Removed `resetTemporaryExposureKey`, which will instead be handled internally

v1.1 - April 21, 2020

- Renamed from Contact Tracing -> Exposure Notification
- Update `start()`
    - Allow sending in matching options, including `attenuationValueThreshold` and `durationMinutesThreshold`

    - Update documentation
- Update `ExposureNotificationCallback` javadocs and method naming to better conform to Android conventions
- Renamed `DailyTracingKey` to `TemporaryExposureKey`
    - date to `rollingStartNumber` per crypto spec
- Renamed `startSharingTemporaryTracingKeys` to `getTemporaryExposureKeyHistory` and return the keys directly
- Renamed `hasContact` to `getExposureSummary` and added `ExposureSummary`
- Changed `ContactInfo` to `ExposureInformation`
    - Added `attenuationValue`
    - Documented maximum `durationMinutes` to 30.
- Updated return types for all methods to conform with Google Play services style conventions
- Updated documentation on `provideDiagnosisKeys` that older keys will be ignored
- Added methods for a client to `resetAllData` or `resetTemporaryExposureKey`
- General doc cleanup

v1.0 - April 10, 2020

- Initial draft