**From:** gary_davis@apple.com on behalf of Gary Davis <gary_davis@apple.com>

**Sent:** 10 April 2020 11:24

**To:** [redacted]

**Cc:** William Malcolm; ZERDICK Thomas; BUCHTA Anna

**Subject:** Contact Tracing Initiative

Hi [redacted]

I apologise for contacting on a holiday.

William and I were hoping to find 30 minutes of Wojciech's time later today to brief him and whomever is appropriate on an announcement that we plan to make today at 7pm CET regarding a joint initiative between Apple and Google on contact tracing to combat Covid-19.

Let us know if Wojciech and/or the team are available.

Kind regards

William & Gary

**From:** gary_davis@apple.com on behalf of Gary Davis <gary_davis@apple.com>

**Sent:** 10 April 2020 16:10

**To:** [redacted]; WIEWIOROWSKI Wojciech

**Cc:** William Malcolm; [redacted]; BUCHTA Anna

**Subject:** Re: Meeting today

**Attachments:** Contact Tracing - Framework Documentation.pdf; Contact Tracing - Bluetooth Specification.pdf; Contact Tracing - Cryptography Specification.pdf

Dear Wojciech & colleagues

Many thanks for your time.

Here are the technical papers. They will ship at 7pm CET today. Everything is embargoed until then.

Happy to stay in very close contact on this.

Kind regards

Gary

On Apr 10, 2020, at 12:16 PM, [redacted] [redacted] wrote:

3 p.m is fine with webex.

Have a nice afternoon

[redacted]

**From:** Gary Davis <gary_davis@apple.com>
**Date:** 10 April 2020 at 12:47:13 CEST
**To:** [redacted]

[redacted]
**Cc:** William Malcolm
<williammalcolm@google.com>, [redacted]
[redacted]>
**Subject: Re: Meeting today**

Many thanks[redacted]

Can we do 3pm CET?

Can you use this Webex?

[redacted]

Global call-in numbers:

[redacted]

On Apr 10, 2020, at 11:41 AM, [redacted]
[redacted] wrote:

Hi Gary,

Wojciech is happy to talk to you whenever is suitable for you, choose a slot, please.
Concerning the modality how to connect, how would you prefer?
Otherwise we can use WebMeeting.

My colleagues in copy will be ne also participating at the call.

Best,

[redacted]

Sent from my iPhone

**From:** William Malcolm <williammalcolm@google.com>

**Sent:** 10 April 2020 16:25

**To:** Gary Davis

**Cc:** [redacted]; WIEWIOROWSKI Wojciech; [redacted]; [redacted]; BUCHTA Anna

**Subject:** Re: Meeting today

Agreed. Thanks for the time today. Let's keep in close touch.

Best

William

On Fri, 10 Apr 2020 at 15:10, Gary Davis <gary_davis@apple.com> wrote:

> Dear Wojciech & colleagues
>
> Many thanks for your time.
>
> Here are the technical papers. They will ship at 7pm CET today. Everything is embargoed until then.
>
> Happy to stay in very close contact on this.
>
> Kind regards
>
> Gary
>
> > On Apr 10, 2020, at 12:16 PM, [redacted]> wrote:
> >
> > 3 p.m is fine with webex.
> >
> > Have a nice afternoon
> >
> > [redacted]
> >
> > **From:** Gary Davis <gary_davis@apple.com>
> > **Date:** 10 April 2020 at 12:47:13 CEST

**To:**

>

**Cc:** William Malcolm <williammalcolm@google.com>,
[redacted]>

**Subject: Re: Meeting today**

Many thanks [redacted]

Can we do 3pm CET?

Can you use this Webex?

[redacted]

Global call-in numbers:

[redacted]

On Apr 10, 2020, at 11:41 AM, [redacted]> wrote:

Hi Gary,

Wojciech is happy to talk to you whenever is suitable for you, choose a slot, please.
Concerning the modality how to connect, how would you prefer? Otherwise we can use WebMeeting.

My colleagues in copy will be ne also participating at the call.

Best,

[redacted]

Sent from my iPhone

--

William Malcolm | Director, Privacy Legal | williammalcolm@google.com | [redacted]

# Contact Tracing

## Cryptography Specification

Preliminary - Subject to Modification and Extension

April 2020

# Contents

## Overview

This document provides the detailed technical specification for cryptographic key scheduling for a new privacy-preserving Bluetooth protocol to support Contact Tracing. Contact Tracing makes it possible to combat the spread of the COVID-19 virus by alerting participants of possible exposure to someone who they have recently been in contact with, and who has subsequently been positively diagnosed as having the virus. This specification complements the Bluetooth specification that contains further information about the scheduling of the advertisements and the higher-level lifecycle of the protocol.

## External Functions

### Concatenation

We use the symbol || to denote concatenation.

### HKDF

HKDF designates the **HKDF** function as defined by IETF RFC 5869, using the SHA-256 hash function:

Output $\leftarrow HKDF(\text{Key, Salt, Info, OutputLength})$

### HMAC

HMAC designates the **HMAC** function as defined by IETF RFC 2104, using the SHA-256 hash function:

Output $\leftarrow HMAC(\text{Key, Data})$

### Truncation

Truncate defines a truncation function:

Output $\leftarrow \text{Truncate}(\text{Data}, L)$

The `Truncate` function returns the first L bytes of the data. The input data size being greater or equal to *L* is a precondition.

### DayNumber

Provides a number for each 24-hour window. These time windows are based on Unix Epoch Time.

$$\text{DayNumber} \leftarrow \frac{\text{Number of Seconds since Epoch}}{60 \times 60 \times 24}$$

DayNumber is encoded as a 32-bit (`uint32 t`) unsigned little-endian value.

### TimeIntervalNumber

Provides a number for each 10-minute window in a 24-hour window as defined by DayNumber.

This value will be in the $[0,143]$ interval.

$$\text{TimeNumberInterval} \leftarrow \frac{\text{Seconds Since Start of DayNumber}}{60 \times 10}$$

where : Seconds Since Start of DayNumber $\leftarrow$ Number of Seconds since Epoch % (60*60*24)

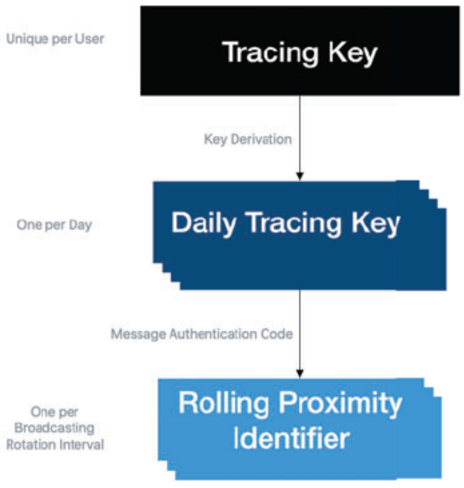
TimeNumberInterval is encoded as a 8-bit (`uint8 t`) value.

### CRNG

CRNG designates a cryptographic random number generator.

Output $\leftarrow CRNG(\text{OutputLength})$.

## Key Schedule for Contact Tracing


Unique per User
Tracing Key
Key Derivation
One per Day
Daily Tracing Key
Message Authentication Code
One per Broadcasting Rotation Interval
Rolling Proximity Identifier

**Tracing Key**

The *Tracing Key* is generated when contact tracing is enabled on the device and is securely stored on the device.

The 32-byte Tracing Key is derived as follows:

$$tk \leftarrow CRNG(32)$$

The Tracing Key never leaves the device.

**Daily Tracing Key**

A *Daily Tracing Key* is generated for every 24-hour window where the protocol is advertising.

From the Tracing Key, we derive the 16-byte Daily Tracing Key in the following way:

$$dtk_i \leftarrow HKDF(tk, NULL, (\text{UTF8("CT-DTK")} \,||\, D_i), 16)$$

where $D_i$ is the DayNumber for the 24-hour window the broadcast is in.

Upon a user testing positive, the Daily Tracing Keys for days where the user could have been affected are derived on the device from the Tracing Key. We refer to that subset of keys as the Diagnosis Keys. If a user remains healthy and never tests positive, these Daily Tracing Keys never leave the device. The use of 16-byte keys allows to limit the server and device requirements for the transfer and storage of the Diagnosis Keys while preserving low false positive probabilities.

**Rolling Proximity Identifier**

The *Rolling Proximity Identifiers* are privacy-preserving identifiers that are sent in Bluetooth Advertisements.

Each time the Bluetooth MAC address changes, we derive a new Rolling Proximity Identifier:

$$RPI_{i,j} \leftarrow \text{Truncate}(HMAC(dkt_i, (\text{UTF8("CT-RPI")} \,||\, TIN_j)), 16)$$

Where:

- $TIN_j$ is the TimeIntervalNumber for the time at which the BLE MAC address changes.

The 16-byte Rolling Proximity Identifier is broadcasted over Bluetooth LE. The use of 16-byte identifiers yields a low probability of collisions and limits the risk of false positive matches, while keeping device storage requirements low.

## Matching Values from Users Tested Positive

Upon a positive test of a user for COVID-19, their Diagnosis Keys and associated DayNumbers are uploaded to the Diagnosis Server. A Diagnosis Server is a server that aggregates the Diagnosis Keys from the users who tested positive and distributes them to all the user clients who are using contact tracing.

In order to identify any exposures, each client frequently fetches the list of Diagnosis Keys. Since Diagnosis Keys are sets of Daily Tracing Keys with their associated Day Numbers, each of the clients are able to re-derive the sequence of Rolling Proximity Identifiers that were advertised over Bluetooth from the users who tested positive. In order to do so, they use each of the Diagnosis Keys with the function defined to derive the Rolling Proximity Identifier. For each of the derived identifiers, they match it against the sequence they have found through Bluetooth scanning. Additional validation can be used to confirm that the advertising happened in a time window comparable to the one expected based on the TimeIntervalNumber.

Matches must stay local to the device and not be revealed to the Diagnosis Server.

## Privacy Considerations

- The key schedule is fixed and defined by operating system components, preventing applications from including static or predictable information that could be used for tracking.
- A user's Rolling Proximity Identifiers cannot be correlated without having the Daily Tracing Key. This reduces the risk of privacy loss from advertising them.
- A server operator implementing this protocol does not learn who users have been in proximity with or users' location unless it also has the unlikely capability to scan advertisements from users who recently reported Diagnosis Keys.
- Without the release of the Daily Tracing Keys, it is not computationally feasible for an attacker to find a collision on a Rolling Proximity Identifier. This prevents a wide-range of replay and impersonation attacks.
- When reporting Diagnosis Keys, the correlation of Rolling Proximity Identifiers by others is limited to 24h periods due to the use of Daily Tracing Keys. The server must not retain metadata from clients uploading Diagnosis Keys after including them into the aggregated list of Diagnosis Keys per day.

## Test Vectors

Test vectors for interoperability testing between implementations of this specification are available upon request in a machine readable format.

# Contact Tracing

## Bluetooth Specification

Preliminary - Subject to Modification and Extension

April 2020

# Contents

## Overview

This document provides the detailed technical specification for a new privacy-preserving Bluetooth protocol to support Contact Tracing. Contact Tracing makes it possible to combat the spread of the COVID-19 virus by alerting participants of possible exposure to someone who they have recently been in contact with, and who has subsequently been positively diagnosed as having the virus. The Contact Detection Service is the vehicle for implementing contact tracing and uses the Bluetooth LE (Low Energy) for proximity detection of nearby smartphones, and for the data exchange mechanism.

## Definitions

- Contact Detection Service - The BLE service for detecting device proximity.
- Tracing Key - A key that is generated once per device.
- Daily Tracing Key - A key derived from the Tracing Key every 24 hours for privacy consideration.
- Diagnosis Key - The subset of Daily Tracing Keys which are uploaded when the device owner is diagnosed positive for the COVID-19 virus.
- Rolling Proximity Identifier - A privacy preserving identifier derived from the Daily Tracing Key and sent in the bluetooth advertisements. It changes every ~15 minutes to prevent wireless tracking of the device.

## Contact Detection Service

Contact Detection is a BLE service registered with the Bluetooth SIG with 16-bit UUID 0xFD6F, it is designed to enable proximity sensing of Rolling Proximity Identifier between devices for the purpose of computing an exposure event.

Devices advertise and scan for the Contact Detection Service by way of its 16-bit service UUID. The Service Data type with this service UUID shall contain a 128-bit Rolling Proximity Identifier that changes periodically.

| **Flags** | | | **Complete 16-bit ServiceUUID** | | | **Service Data - 16 bit UUID** | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Length** | **Type** | **Flags** | **Length** | **Type** | **ServiceUUID** | **Length** | **Type** | **ServiceData** | |
| 0x02 | 0x01 (Flag) | 0x1A | 0x03 | 0x03<br>(Complete 16-bit ServiceUUID) | 0xFD6F<br>(Contact Detection Service) | 0x13 | 0x16<br>(Service Data - 16 bit UUID) | 0xFD6F<br>(Contact Detection Service) | **16 bytes**<br>**Rolling Proximity Identifier** |

## Advertisement Payload

The Contact Detection Service payload shall be ordered as shown below and shall not include other data types.

The Contact Detection Service payload has three sections:

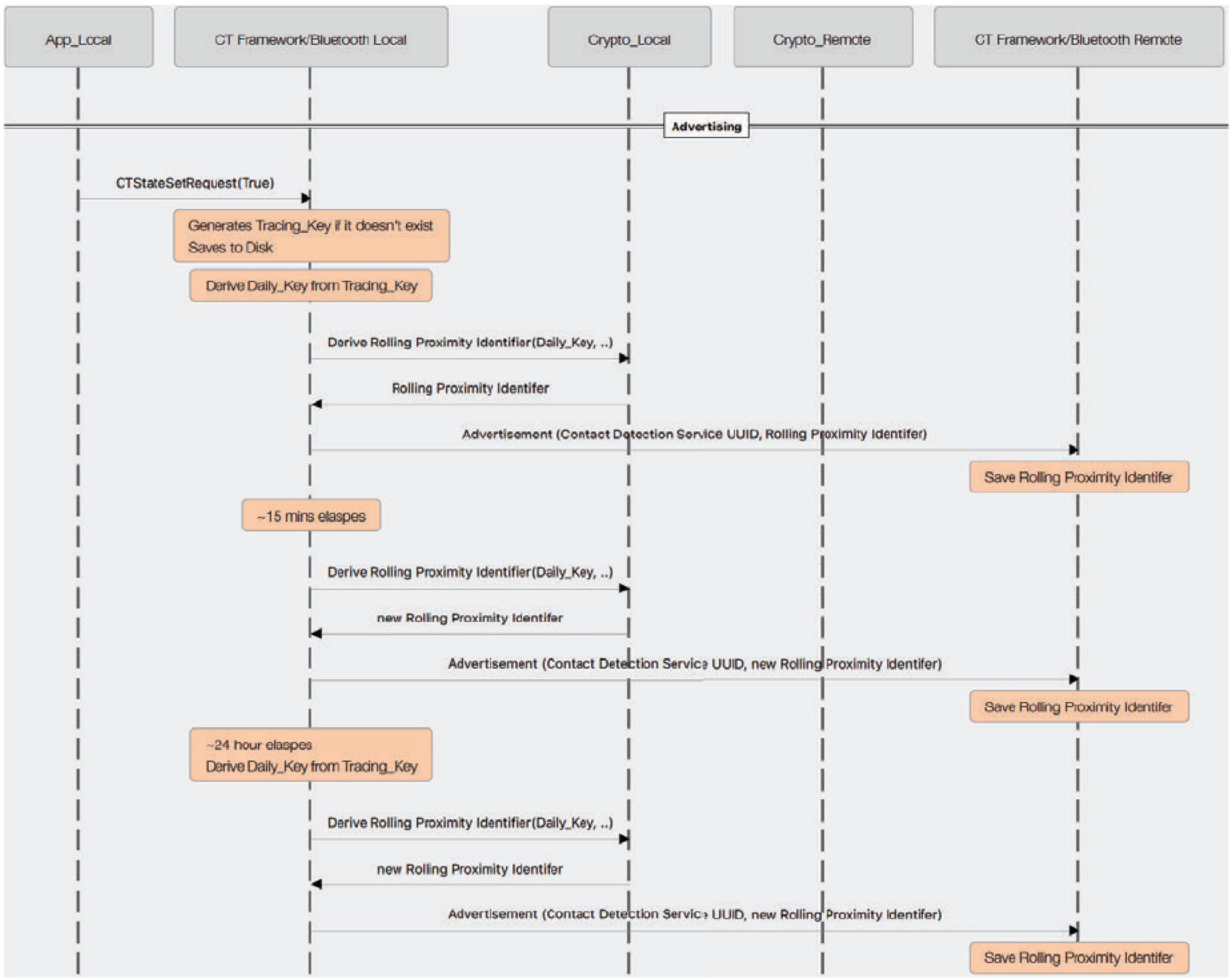
1. Flags Section: LE general discoverable mode (bit 1) shall be set to 1.
2. Complete 16-bit Service UUID Section: the UUID is 0xFD6F, and shall precede the Service Data section.
3. Service Data 16-bit UUID Section: the content of service data payload shall be a 128-bit Rolling Proximity Identifier.

## Advertising Behavior

- Advertisements are to be non-connectable undirected of type `ADV_NONCONN_IND` (Section 2.3.1.3 of 5.2 Core Spec).
- The advertiser address type should be Random Non-resolvable.
- On the platforms supporting Bluetooth Random Private Address with randomized rotation timeout interval, the advertiser address rotation period shall be a random value, greater than 10 minutes and less than 20 minutes.
- The advertiser address and Rolling Proximity Identifier shall be changed synchronously so address and Rolling Proximity Identifier can not be linked.
- A separate advertising instance should be used, if HW allows, to provide advertising reliability and flexibility in choosing optimal interval.
- The advertising interval is subject to change, but is currently recommended to be 200-270 milliseconds.

## Advertising Flow

The following diagram illustrates the flow of advertisement between devices.


App_Local
CT Framework/Bluetooth Local
Crypto_Local
Crypto_Remote
CT Framework/Bluetooth Remote
Advertising
CTStateSetRequest(True)
Generates Tracing_Key if it doesn't exist
Saves to Disk
Derive Daily_Key from Tracing_Key
Derive Rolling Proximity Identifier(Daily_Key, ..)
Rolling Proximity Identifer
Advertisement (Contact Detection Service UUID, Rolling Proximity Identifer)
Save Rolling Proximity Identifer
~15 mins elaspes
Derive Rolling Proximity Identifier(Daily_Key, ..)
new Rolling Proximity Identifer
Advertisement (Contact Detection Service UUID, new Rolling Proximity Identifer)
Save Rolling Proximity Identifer
~24 hour elaspes
Derive Daily_Key from Tracing_Key
Derive Rolling Proximity Identifier(Daily_Key, ..)
new Rolling Proximity Identifer
Advertisement (Contact Detection Service UUID, new Rolling Proximity Identifer)
Save Rolling Proximity Identifer

## Scanning Behavior

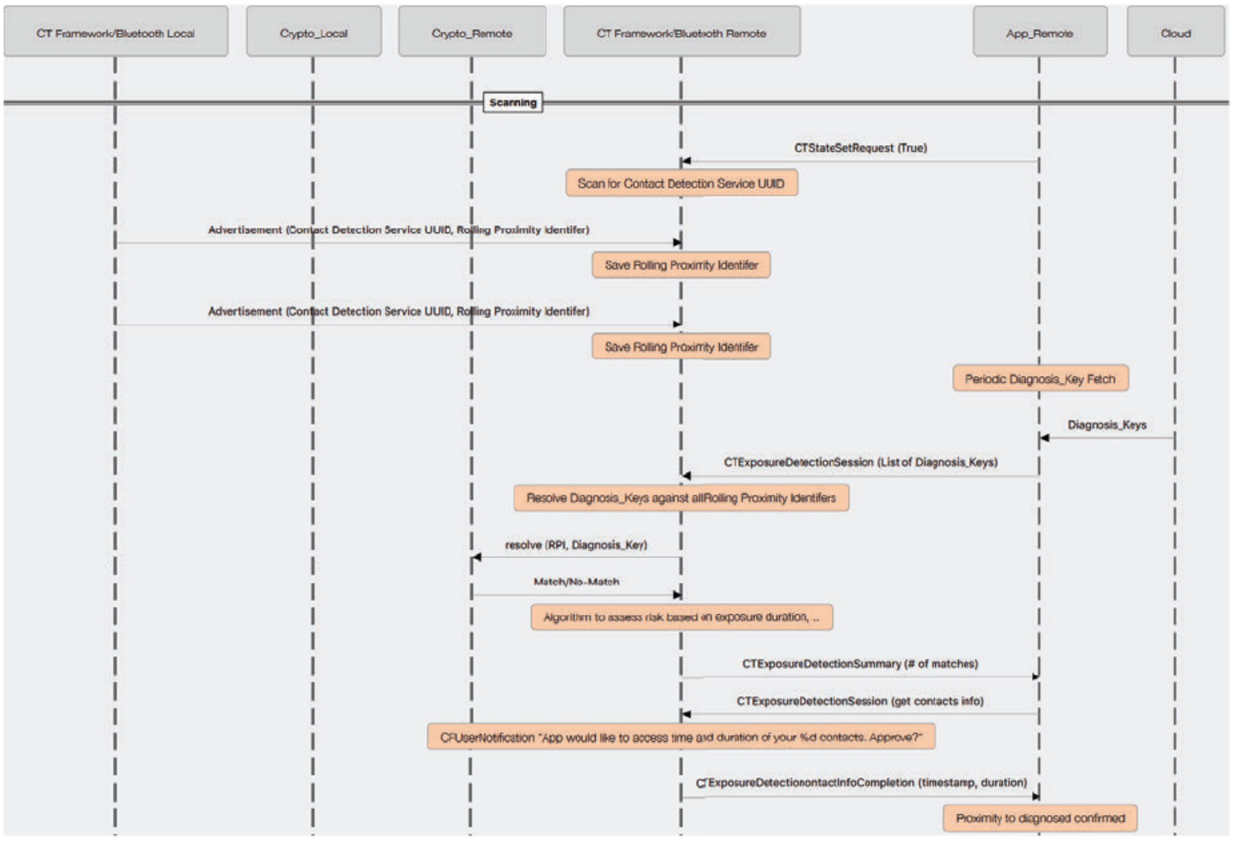
- Discovered Contact Detection Service advertisements shall be kept on the device.
- Scan results shall be timestamped and RSSI-captured per advertisement.
- Scanning interval and window shall have sufficient coverage to discover nearby Contact Detection Service advertisers within 5 mins.
- Scanning strategy that works best is opportunistic (leveraging existing wakes and scan windows) and with minimum periodic sampling every 5 mins.

## Scan Power Considerations

- Platforms running the Contact Detection Service should be designed to account for large volume of advertisers in public spaces that shall rotate their Random Non-resolvable address and Rolling Proximity Identifier frequently.
- Wherever supported by HW and OS, Bluetooth controller duplicate filters and other HW filters should be used to prevent excessive power drain.

## Scanning Flow

The following diagram illustrates the flow and behavior of device scanning.


CT Framework/Bluetooth Local
Crypto_Local
Crypto_Remote
CT Framework/Bluetooth Remote
App_Remote
Cloud
Scanning
CTStateSetRequest (True)
Scan for Contact Detection Service UUID
Advertisement (Contact Detection Service UUID, Rolling Proximity Identifer)
Save Rolling Proximity Identifer
Advertisement (Contact Detection Service UUID, Rolling Proximity Identifer)
Save Rolling Proximity Identifer
Periodic Diagnosis_Key Fetch
Diagnosis_Keys
CTExposureDetectionSession (List of Diagnosis_Keys)
Resolve Diagnosis_Keys against allRolling Proximity Identifers
resolve (RPI, Diagnosis_Key)
Match/No-Match
Algorithm to assess risk based on exposure duration, ..
CTExposureDetectionSummary (# of matches)
CTExposureDetectionSession (get contacts info)
CFUserNotification "App would like to access time and duration of your %d contacts. Approve?"
CTExposureDetectionontactInfoCompletion (timestamp, duration)
Proximity to diagnosed confirmed

## Privacy

Maintaining user privacy is an essential requirement in the design of this specification.  The protocol does this by the following means:

- The Contact Tracing Bluetooth Specification does not require the user's location; any use of location is completely optional to the schema.   In any case, the user must provide their explicit consent in order for their location to be optionally used.
- Rolling Proximity Identifiers change on average every 15 minutes, making it unlikely that user location can be tracked via bluetooth over time.
- Proximity identifiers obtained from other devices are processed exclusively on device.
- Users decide whether to contribute to contact tracing.
- If diagnosed with COVID-19, users consent to sharing Diagnosis Keys with the server.
- Users have transparency into their participation in contact tracing.

# Contact Tracing

## Framework Documentation

## (API)

Preliminary - Subject to Modification and Extension

April 2020

# Contents

## Overview

The ContactTracing Framework is designed to help developers implement a privacy preserving contact tracing solution. It covers two user roles:

1. *Affected User*. A user who reports themself as positively diagnosed as having the virus.
2. *Exposed User*. A user who has notified themself as potentially exposed to an Affected User.

### Affected User

When a user is positively affected, their Daily Tracing Keys should be shared with other uses to alert them to potential exposure. These Daily Tracing Keys are retrieved using `CTSelfTracingInfoRequest`.

### Exposed User

Given a set of positively affected Daily Tracing Keys, the framework allows you to determine whether those Daily Tracing Keys were observed locally by the user, indicating potential exposure. If so, additional information such as date and duration may also be retrieved. Possible observations can be retrieved using `CTExposureDetectionFinishHandler`, and additional information using `CTExposureDetectionContactHandler`.

The following illustration outlines the flow of the ContactTracing Framework for iOS.

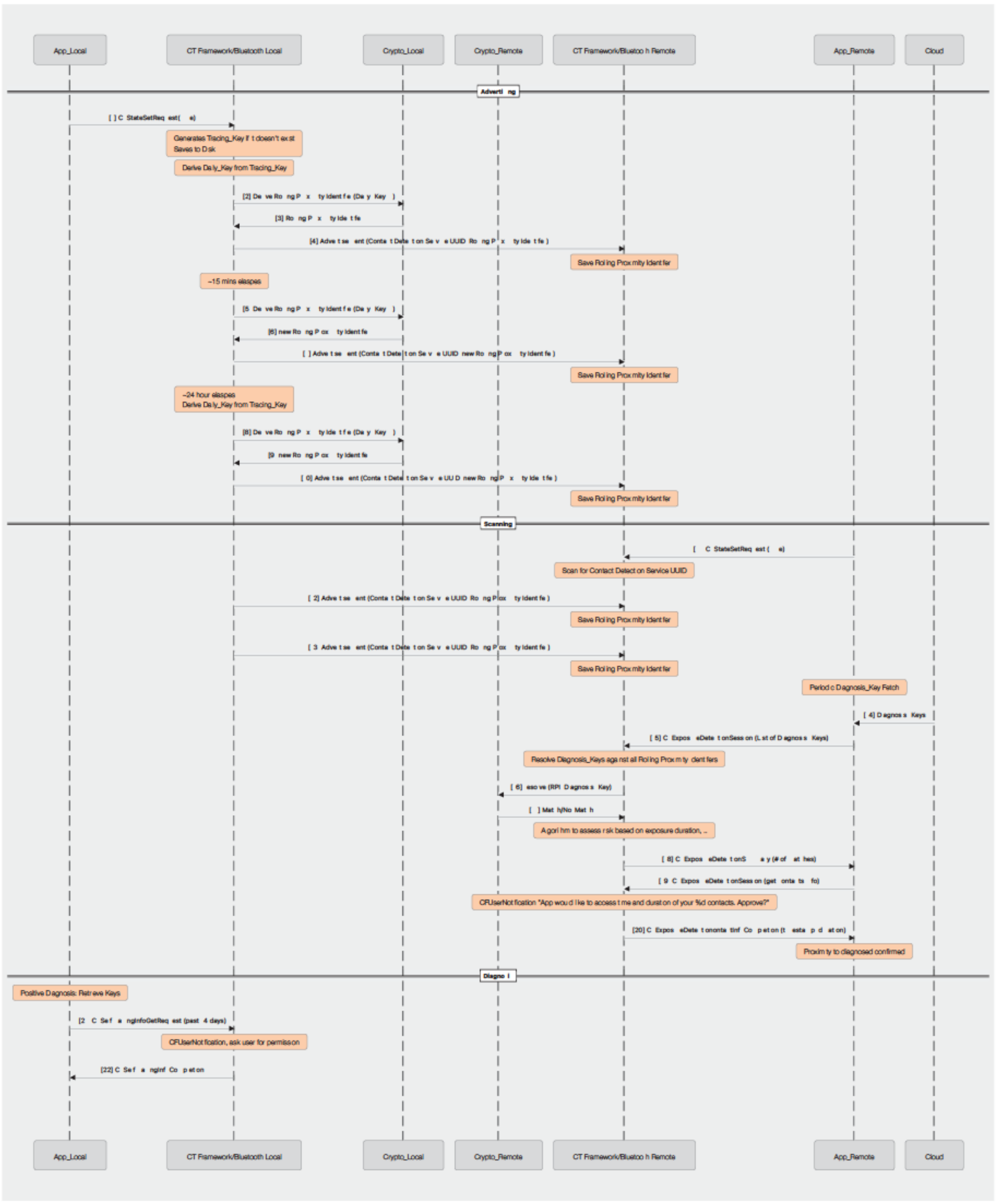

App_Local
CT Framework/Bluetooth Local
Crypto_Local
Crypto_Remote
CT Framework/Bluetoo h Remote
App_Remote
Cloud
Advertising
Generates Tracing_Key if it doesn't exist
Saves to Disk
Derive Daily_Key from Tracing_Key
Save Rolling Proximity Identifier
~15 mins elapses
~24 hour elapses
Derive Daily_Key from Tracing_Key
Scanning
Scan for Contact Detection Service UUID
Periodic Diagnosis_Key Fetch
Resolve Diagnosis_Keys against all Rolling Proximity Identifiers
Algorithm to assess risk based on exposure duration, ..
CFUserNotification "App would like to access time and duration of your %d contacts. Approve?"
Proximity to diagnosed confirmed
Diagnosis
Positive Diagnosis: Retrieve Keys
CFUserNotification, ask user for permission

Figure 1. The Contact Tracing Flow

# CTStateGetRequest

## Overview

Requests whether contact tracing is on or off on the device.

## typedef void ( ^CTErrorHandler )( NSError * _Nullable inError );

The type definition for the completion handler.

## @property CTErrorHandler completionHandler;

This property holds the completion handler that framework invokes when the request completes. The property is cleared upon completion to break any potential retain cycles.

## @property dispatch_queue_t dispatchQueue;

This property holds the the dispatch queue used to invoke handlers on. If this property isn’t set, the framework uses the main queue.

## @property CTManagerState state;

This property contains the snapshot of the state when the request was performed. It’s valid only after the framework invokes the completion handler.

## - (void)perform;

Asynchronously performs the request to get the state, and invokes the completion handler when it's done.

## - (void)invalidate;

Invalidates a previously initiated request. If there is an outstanding completion handler, the framework will invoke it with an error.

Don’t reuse the request after this is called. If you require another request, create a new one.

# CTStateSetRequest

## Overview

Changes the state of contact tracing on the device.

## @property CTErrorHandler completionHandler;

This property holds the completion handler that framework invokes when the request completes. The property is cleared upon completion to break any potential retain cycles.

## @property dispatch_queue_t dispatchQueue;

This property holds the the dispatch queue used to invoke handlers on. If this property isn’t set, the framework uses the main queue.

## @property CTManagerState state;

This property contains the state to set Contact Tracing to. Call the `perform` method to apply the state once set.

## - (void)perform;

Asynchronously performs the request to get the state, and invokes the completion handler when it's done.

## - (void)invalidate;

Invalidates a previously initiated request. If there is an outstanding completion handler, the framework will invoke it with an error.

Don’t reuse the request after this is called. If you require another request, create a new one.

# CTExposureDetectionSession

## Overview

Performs exposure detection bad on previously collected proximity data and keys.

## @property dispatch_queue_t dispatchQueue;

This property holds the the dispatch queue used to invoke handlers on. If this property isn't set, the framework uses the main queue.

## @property NSInteger maxKeyCount;

This property contains the maximum number of keys to provide to this API at once. This property's value updates after each operation complete and before the completion handler is invoked. Use this property to throttle key downloads to avoid excessive buffering of keys in memory.

## - (void) activateWithCompletion:(CTErrorHandler) inCompletion;

Activates the session and requests authorization for the app with the user. Properties and methods cannot be used until this completes successfully.

## - (void)invalidate;

Invalidates the session. Any outstanding completion handlers will be invoked with an error. The session cannot be used after this is called. A new session must be created if another detection is needed.

## - (void) addPositiveDiagnosisKeys:(NSArray <CTDailyTracingKey *> *) inKeys completion:(CTErrorHandler) inCompletion;

Asynchronously adds the specified keys to the session to allow them to be checked for exposure. Each call to this method must include more keys than specified by the current value of <maxKeyCount>.

## typedef void ( ^CTExposureDetectionFinishHandler ) ( CTExposureDetectionSummary * _Nullable inSummary, NSError * _Nullable inError );

The type definition for the completion handler.

## - (void) finishedPositiveDiagnosisKeysWithCompletion: (CTExposureDetectionFinishHandler) inFinishHandler;

Indicates all of the available keys have been provided. Any remaining detection will be performed and the completion handler will be invoked with the results.

## typedef void ( ^CTExposureDetectionContactHandler )( NSArray <CTContactInfo *> * _Nullable inContacts, NSError * _Nullable inError );

The type definition for the completion handler.

## - (void)getContactInfoWithHandler: (CTExposureDetectionContactHandler) inHandler;

Obtains information on each incident. This can only be called once the detector finishes. The handler may be invoked multiple times. An empty array indicates the final invocation of the hander.

# CTExposureDetectionSummary

## Overview

Provides a summary on exposures.

## @property NSInteger matchedKeyCount;

This property holds the number of keys that matched for an exposure detection.

# CTSelfTracingInfoRequest

## Overview

Requests the daily tracing keys used by this device to share with a server.

## Discussion

This request is intended to be called when a user has a positive diagnosis. Once the keys are shared with a server, other users can use the keys to check if their device has been in contact with any positive diagnosis users. Each request will require the user to authorize access.

Keys will be reported for the previous 14 days of contact tracing. The app will also be launched every day after the daily tracing key changes to allow it to request again to get the key for each previous day for the next 14 days.

## typedef void ( ^CTSelfTracingInfoGetCompletion ) ( CTSelfTracingInfo * _Nullable inInfo, NSError * _Nullable inError );

The type definition for the completion handler.

## @property CTSelfTracingInfoGetCompletion completionHandler;

This property invokes this completion handler when the request completes and clears the property to break any potential retain cycles.

## @property dispatch_queue_t dispatchQueue;

This property holds the the dispatch queue used to invoke handlers on. If this property isn't set, the framework uses the main queue.

## - (void)perform;

Asynchronously performs the request to get the state, and invokes the completion handler when it's done.

## - (void)invalidate;

Invalidates a previously initiated request. If there is an outstanding completion handler, the framework will invoke it with an error.

Don't reuse the request after this is called. If you require another request, create a new one.

# CTSelfTracingInfo

## Overview

Contains the Daily Tracing Keys.

## @property NSArray <CTDailyTracingKey *> * dailyTracingKeys;

Daily tracing keys available at the time of the request.

# CTContactInfo

## Overview

Contains information about a single contact incident.

## @property NSTimeInterval duration;

How long the contact was in proximity. Minimum duration is 5 minutes and increments by 5 minutes: 5, 10, 15, etc.

## @property CFAbsoluteTime timestamp;

This property contains the time when the contact occurred. This may have reduced precision, such as within one day of the actual time.

# CTDailyTracingKey

## Overview

The Daily Tracing Key object.

## @property NSData *keyData;

This property contains the Daily Tracing Key information.

## Meeting between Apple, Google and the EDPS – 10.04.20

The meeting was requested by Apple and Google on Friday 10th April and took place the same day.

The representatives of Apple – Gary Davis – and Google – William Malcom – aimed to present to the Supervisor their joint initiative for an interoperable API to enable contact tracking.

The presentation outlined two different phases of the project. A first phase was said to be scheduled to start in Mid-May, when the API would be made available to developers. The second phase would bring the contact tracing system to the operating systems.

The EDPS remarked the importance of ensuring the highest level of data protection, in particular in such a case where the scope of the initiative is global and the risk of adopting lower standards is high.

The EDPS asked whether other data protection authorities were consulted or made aware of the initiative. The EDPS also recommended to inform the European Data Protection Board.