National Cyber Security Centre

High level privacy and security design for NHS COVID-19 Contact Tracing App

Dr Ian Levy, Technical Director, National Cyber Security Centre, UK

Version: 0.1, 3rd May 2020

© Crown Copyright 2020

Contents

Disclaimer – versioning and documentation

Code is truth. This document is correct at the time of writing, but the system is still in development, so there may be detail changes before release.

After release, there will certainly be updates to the system and this document, as we learn more about large scale, real world deployment of the app and the mechanisms by which the virus spreads. We will likely evolve the system security once we understand those real world deployments better.

In the interests of expediency, this paper concentrates on the most important privacy and security challenges involved in building a contact tracing app that is part of a wider public health management effort. There is much more to the security and privacy design, but those issues are not unique to a contact tracing app.

We intend to open source our codebase once the first release is finalised. The documentation accompanying that release will supersede this paper.

There will be multiple releases of the app as the Government's response to the COVID-19 virus progresses. Not all features detailed here will necessarily be in the initial release.

About this paper

This document provides a high-level overview of the security and privacy characteristics of the app that is in development by NHSx, the digital innovation unit of the National Health Service, to help manage the COVID-19 crisis in the UK.

This is not a full description of the entire system, the socio-technical design, epidemiological modelling or the plethora of other work being performed outside of this application development. Nor does this document detail the significant, diverse, expert input to the overall system and oversight of its development.

Instead, this technical paper concentrates only on the most important and unique security and privacy characteristics of the putative app and its infrastructure. We only describe epidemiological and clinical aspects of the system, in order to set context for some technical decisions and trade-offs.

The epidemiological advice and models that the NHS is working from show that self-diagnosis is an important part of managing the spread of the disease, alongside various clinical tests and the wider public health response strategy. The Oxford group responsible for the model publishes much of its work.

Self-diagnosis can reduce by days, the time it takes a potentially infectious person to isolate. This is critical to the management of the spread of the disease, under the assumptions in the UK's model.

There are obvious corollaries to a model that includes self-diagnosis. We explore some of those here, with their current mitigations.

Finally, a contact tracing app cannot work in isolation – it must work in concert with, and be a pathway into, the wider public health response. We do not cover that integration here, but it is in place.

Introduction to the NHS COVID-19 app

The NHS COVID-19 app aims to automate key parts of public health contact tracing by offering a proximity cascade system that can help slow transmission of the COVID-19 virus. This will save lives, reduce pressure on the NHS, help return people to normal life and mitigate damage to the economy.

The app also aims to preserve individual and group privacy, be tolerant to various malicious users and minimise the risks of pseudonymous subgroup reidentification. Importantly, it is driven by and informs expert epidemiological modelling, which in turn drives public policy.

How the NHSx app works

The user-centric description of the app is:

"When I download the app, it keeps an anonymous record of when I've been close to other people (proximity events) .

If I self-diagnose in the app, as displaying COVID-19 symptoms, I can choose to provide my personal record of proximity events to an NHSx system.

The NHSx system can then work out who to notify that they have potentially been in contact with COVID-19. To these people, it can provide the latest advice and, potentially, access to testing.

Analysis of the records of proximity events from people displaying symptoms will allow NHSx to monitor and control the spread of the virus.

Preserving the privacy of users is high priority - personal information is kept to a minimum unless I enter it."

We will describe the main parts of the system and enumerate some of the security and privacy risks, along with their mitigations.

This document is not a full security and privacy analysis. We use common shorthand for cryptographic operations where the context is obvious.

For the purposes of this paper, readers should assume that cybersecurity best practice is being adhered to in the architecture of the microservices and back-end infrastructure, the overall development process and in the operational and security management of the infrastructure. We will show this in due course.

App operation

The app will determine and record proximity events between users of the application, on a rolling basis. It will also allow users to assert a self-diagnosis to the system, by way of a structured set of questions.

Such an assertion allows users to donate their proximity event data to a central system, which will model the proximity events to provide an overall infection risk for each physical encounter.

Should any particular physical interactions meet a threshold, those app users potentially impacted will receive a notification through the app. This will tell them that they may have been close enough, for long enough, to someone who has since developed symptoms of COVID-19, that they are at risk of infection. They will then be signposted to existing NHS guidance.

The app may request some extra information at registration, or when a user asserts that they have symptoms and chooses to donate their data. Currently, this is limited to the first part of a postcode (to help resource planning and disease tracking) and answers to some clinical questions regarding symptoms (to increase sensitivity and selectivity of self-diagnosis). In future, this is likely to be a pathway to clinical testing.

The app will also ask people to release themselves from self-isolation if a notified contact seems not to have caused infections (see later), or if they get a clinical negative test.

Security and privacy criteria

We enforce the following high priority security and privacy requirements:

1) Minimise collection of personal data (ideally, no personal data should be collected).
2) Active user consent is required for any action involving collected data.
3) It should not be possible to track users of the app over time, through the Bluetooth transmissions.
4) It should not be possible for an external observer to associate any Bluetooth transmission with any device-specific information, over and above what they can infer through proximity (e.g. I can see that this ephemeral Bluetooth LE transmission value is linked to Alice's device because I can see that Alice is the only person close to me).

5) It should not be possible to submit spoofed data on behalf of another user.
6) It should not be possible for the recipient of a notification to determine which of the people they have been in contact with has asserted symptoms.
7) The system must be tolerant to the actions of malicious users:
    a. Single user seeking to gain from a false self-diagnosis
    b. Malicious user seeking to cause mass notification in a given area (e.g. trying to shut down a hospital)
    c. Nation state actor seeking to cause panic through mass notification across the country
8) A malicious user must not be able to replay a notification of proximity from the service to another user.

These are not the only security and privacy promises we make, but seem to be the most important to explain early in the app's development.

System architecture overview

The following actors and components exist in the system.

1) **Users of the app**
   In our model, users have consented to have their proximity to other users recorded and we expect them to donate their data when they become symptomatic. Users may be malicious.

2) **Devices on which the app is installed**

3) **The registration service**
   The registration service is used to register a particular installation of the app and to exchange installation-specific and global parameters. The registration service will provide a random, unique and anonymous *InstallationID* (a persistent pseudonym) and the registration process agrees a symmetric key between the device and the registration service. These are stored together in a database. The registration service also provides other system parameters, such as the server public key.

4) **The data collection service**
   When a user chooses to donate their proximity data to the system, they provide the data collection service with their event log, which contains anonymous pseudonyms that they have collected over Bluetooth LE. For each pseudonym, the data collection service can recover the underlying random and anonymous *InstallationID*.

5) **The risk modelling service**
   Given the proximity information (based on normalised Bluetooth LE received signal strength indicators), the risk modelling service can score each encounter in terms of risk of transmission of the virus, since not every proximity event will be equally hazardous. The risk model will be updated as epidemiological and clinical data becomes available.

6) **The notification service**
   Once proximity events have been scored as sufficiently risky, notifications will be generated for each relevant installation, based on the *InstallationID*. Notification queues will be

modelled to ensure they do not have particular undesirable characteristics and then sent for delivery by the notification transport. In our instantiation, the notification transport is Google's FireBase service, which operates across our target ecosystems.

7) **The expert clinicians**
   The clinicians are responsible for ensuring:
   - The risk scoring of contacts is based on an evolving view of the best transmission models available.
   - That the notification cascades make sense in the context of the virus transmission characteristics, enabling malicious events to be detected.
   - The efficacy of any restrictions on the population from the data provided by the system are understood.
   - The aggregate status of users is fed into health system resource planning (from the locality voluntarily provided by users).

In the NHS app, the infrastructure on which the various back end services operate is run in commercial public cloud, with the NHS in control of the code, deployment, operation and administration of the various infrastructure and services (either directly or through contracts).

In our model, the infrastructure provider and the healthcare service can be assumed to be the same entity.

App lifecycle

Here we describe the various stages of the app lifecycle at a high level. More detail of the various protocols involved is provided later.

Installation and Registration

The app registration process exists to set up a relationship between a particular app install instance on a user's mobile phone and the service infrastructure. This includes, setting up asynchronous notifications, exchanging various cryptographic entities and system parameters.

The simplified process is:

- The user downloads the app and initiates registration
- The app contacts the notification transport (Google Firebase in our instantiation) and requests a registration token, which is generated and returned to the app
- The app contacts the registration service and registers its registration token
- The registration service generates[1] an anonymous random GUID (the *InstallationID*), symmetric key used for authentication and activation code. These are stored with the token.
- Registration service sends an *activate* message via the notification transport to the app
- Device contacts the registration service with the activation code, which returns the *InstallationID* and symmetric key.

The registration process does not require the user to submit any personal data and no identifying data is taken from the phone.

[1] It would be better if both client and server contributed to the entropy in the GUID. This may be changed in the future.

The only data required at registration is the first part of the postcode to help with regional healthcare modelling.

Any device-unique cryptographic material is stored in the most secure manner possible, on the host device. For iOS devices, cryptographic material is stored in the Secure Enclave Processor, using Apple standard APIs.

The position on the more variable Android ecosystem is more complex, but the open source Bouncy Castle cryptographic library is used and hardware-backed secure storage is used, if available on the device. Otherwise, best effort software-only protection is used.

BroadcastValue derivation

*BroadcastValue*s are created periodically by the device, using the standard Elliptic Curve Integration Encryption Scheme (ECIES), detailed below.

Our central server has a private $Priv_{Server}$ key and associated public key Pub_{Server} for a fixed system-wide set of parameters over the standard curve, P256. The public key is sent to each device during the initial registration exchange.

At the start of every broadcast period, t, (currently 24 hours – see later for rationale), the device creates a new, ephemeral private key on the curve $Priv_{Device,t}$ and an associated public key $Pub_{Device,t}$

The device then creates a secret value, $Z = ECDH(Pub_{Server}, Priv_{Device,t})$ using its ephemeral private key and the server's long term public key, a simple ECDH key agreement.

We then use a the X9.63 KDF with SHA-256 with input of the ephemeral secret and the server public key to derive two ephemeral symmetric 128 bit values, thus :

$$(K, IV) = KDF(Z, Pub_{Device,t})$$

The plaintext payload we wish to be able to recover in the server is merely

$$m = (StartDate||EndDate||InstallationID||CountryCode)$$

where the date values are 32-bit UNIX epoch dates, the *InstallationID* is 128 bits and the CountryCode is the 16 bit ISO3166-1 country code. The dates are used to perform anti-replay protection and the country code allows for multiple countries to interact, if necessary.

We then proceed to use AES in Galois Counter Mode to compute the pair

$$(C, ICV) = AESGCMENC_K(m, IV)$$

where AESGCMENC is AES encryption in GCM mode, C is the resultant ciphertext of the encryption of the message m and ICV is the GCM-derived integrity check value. K, the cryptovariable, and IV, the initialisation vector, are as derived from the KDF. We use NULL authentication data.

Our *BroadcastValue* is then constructed as the concatenation of the ephemeral public key the device most recently created (along with a tag denoting key format for future use), the ciphertext containing the encrypted dates, *InstallationID* and country code, and the integrity check value:

$$BV = (CountryCode||Pub_{Device,t}, ||C||ICV)$$

giving a total *BroadcastValue* size of 856 bits.

Proximity Event Logging

The *BroadcastValue*, derived from the *InstallationID*, currently has a periodicity of 24 hours. Every 24 hours, a new *BroadcastValue* is calculated by the phone.

The phone advertises a specific service over Bluetooth Low Energy. The service is unique to the NHSx app and only other NHSx apps (or those pretending to be one) will connect to the BLE service on a nearby device.

Once connected, the following full payload is shared over the Bluetooth Low Energy connection

$$P = (BV||TxPower||TxTime||Auth)$$

where $TxPower$ is the BLE transmit power from the sender in dBm, $TxTime$ is the Unix Epoch time of the transmission and $Auth$ is a 16 byte truncation of an HMAC of the other contents of the payload, keyed using the sending device's symmetric authentication key.

Periodically during the connection (currently every 8 seconds, subject to change due to testing and modelling) the Received Signal Strength Indicator (RSSI) for the connection from the BLE stack is recorded.

Once the BLE connection fails – likely due to the phones going out of range – an event is logged, containing the local time, the received BLE payload, the n-tuple of RSSI values and the total time of the encounter.

So, each event in the log consists of the following:

$$Date, RBV, TxPower, TxTime, Auth\ , r_0, \dots, r_n, Extent$$

Where $Date$ is the local system date, RBV is the *BroadcastValue* received from the proximate device over Bluetooth Low Energy, $TxPower$ is the BLE transmit power from the sender in dBm, $TxTime$ is the Unix Epoch time of the transmission from the sender, $Auth$ binds the time, power, BroadcastValue to a particular remote device, r_i is the i^{th} sample of the BLE RSSI value and $Extent$ is the total elapsed time of the continuous encounter.

For clarity, this event log is stored locally on the device, protected by the default per-application policy on the device. Events automatically age off after a set time period, linked to the epidemiology of the virus. Currently, this is set to 28 days.

The periodicity of the *BroadcastValue* can be changed, but 24 hours is attractive as a simple count on the device can give people information to help them understand whether they are following social distancing best practice (e.g. 'Today, you appear to have been very close to 4 people and reasonably close to 20') without the involvement of any external party.

Self-reporting of symptoms

When a user believes they have symptoms, they will be able to 'donate' their event log to the service.

This is a one-time batch operation and does not set up a persistent upload of data. The log is integrity protected by an HMAC, encrypted with the shared symmetric key established at registration and sent to the service infrastructure over TLS-protected channels.

Users will be asked some questions about symptoms, to help with epidemiological analysis.

Recovery of proximate device IDs

The server can look up the device-specific symmetric key and decrypt the submitted event log, providing some authenticity that the log was generated by the asserted device identity.

The log HMAC is checked to provide integrity and the server then proceeds to recover the underlying *InstallationID* for each *BroadcastValue* in each record in the submitted proximity event log.

In the extension case, where multiple countries are collaborating, the plaintext *CountryCode* from the *BroadcastValue* is used to route to a particular service and select the appropriate private key.

Server operation on BroadcastValues

Trivially, the server can recover the underlying *InstallationID* and ISO8601Date from the *BroadcastValue*s sent to it.

For any received *BroadcastValue*, the server extracts the device ephemeral public key $Pub_{Device,t}$ and performs the simple key agreement using ECDH and its own private key $Priv_{Server}$ giving it the ephemeral secret value $Z = ECDH\left(Pub_{Device,t}, Priv_{Server}\right)$.

Note that the period, t, is implicit and requires no synchronisation of counters across devices or infrastructure.

The server can then use its derived shared secret, Z, and the device's ephemeral public key

$$(K, IV) = KDF\left(Z, Pub_{Device,t}\right)$$

and proceed to recover the original message m, from the ciphertext C, contained in the *BroadcastValue*

$$m = AESGCMDEC_K(C||ICV, IV)$$

where AESGCMDEC is AES decryption in GCM mode, the ICV is the integrity check value recovered from the *BroadcastValue* and K, the cryptovariable, and IV, the initialisation vector, are as derived from the KDF.

This results in a set of events containing the fixed (but anonymous and random) *InstallationID* for each device that has been proximate to the submitting device, along with the validity period of the key (to allow replay defence) and the owning country reference (which must match the plaintext value).

The server can now look up the symmetric authentication key for the each *InstallationID* in the records and validate the HMAC authentication of the (*BroadcastValue, TxPower, TxTime*) and ensure that the authenticated transmission time makes sense in the context of the submitted record.

Risk modelling

The server now has an event log consisting of *InstallationID*, an authenticated transmission power from the parter device, an n-tuple of RSSI values and the total time of the encounter.

The device model, recorded on initial registration, is used to normalise RSSI values in each record to give a normalised set of distance measures for each record, regardless of the receiving device radio characteristics.

Each event can now be risk scored by an epidemiological model. Not all proximity events will give rise to significant risk of transmission of infection and this modelling step is used to cut down events that are low risk, from an epidemiological point of view.

The risk model will be improved by epidemiologists as knowledge of the virus characteristics improves and from system data giving feedback on the effectiveness of the model, such as the number of users, warned of proximity risks of particular types, subsequently becoming infected.

Notification cascade

We are now left with a sequence of high-risk (from a viral transmission point of view) events and the associated *InstallationID*s of the devices implicated in those events.

Notifications are queued for release and some cases will need to be triaged by humans before being released. This triage is for reasons of evolving epidemiological understanding, based on the data, as well as analysis *and* the need to filter of suspicious cascades.

The system now uses the notification provider to send an event to each app instance whose *InstallationID* is listed, to generate local notification events, subject to on-device constraints that will be discussed later.

Notification issues

There are a number of scenarios around notification that need to be handled. Here, we provide details of a non-exhaustive set of examples.

1) **The low contact number problem**
   Consider a user of the app, Alice, whose only contacts are her immediate family and her external carer, Chris.

   If Chris then notifies a self-diagnosis, all the people who have been sufficiently proximate to him for sufficient time will then be alerted. This includes Alice.

   We assume that Alice can observe that her immediate family are asymptomatic and can therefore deduce that Chris is now symptomatic and likely infected. This is a breach of Chris's privacy, under our model.

   Since the central system does not necessarily know the contact graph of notified entities (in this case Alice) at the time of notification, suppression of the notification can, subject to a policy decision, be done locally in the app, using simple counting rules (subject to a small population around Alice).

2) **The mass notification problem**
   Consider a malicious user, who can collect broadcast identities from around a particular area, such as a hospital, and record them all.

   They can register a malicious pseudo-device and generate realistic-looking but entirely fake contact events for all the *BroadcastValue*s it has observed. (The mechanism of the chosen notification provider makes this marginally more difficult, but we assume it is zero work here and suggest that ownership of a Raspberry Pi and a decent antenna is all that is required).

Assume the attacker knows enough about the epidemiological model to ensure that all of his fake proximity events are classed as high risk and will generate a notification. Then the malicious user can create an arbitrarily large notification cascade, potentially causing real world impacts.

There are many ways to tackle this problem. One example could be to remotely query some subset of the devices that the attacker claims he was proximate to, in order to determine whether they have the pairwise proximity event. This could have privacy implications, however. For the moment, we have used the transmission risk model in the centralised service, simple counting and the expert centralised human oversight of potentially large, or unusual, notification cascades to mitigate this.

3) **The incorrect submission problem**
   Assume a user submits a self-diagnosis and the notification cascade happens to the high-risk contacts who all self-isolate.

   We can understand the likelihood of the initial self-diagnosis being correct (in advance of any clinical testing results) by observing whether sufficient of the contacts also report symptoms within a small number of days. If not, we can quickly inform people that they no longer need to self-isolate.

4) **The targeted false alerts problem**
   Assume an attacker wishes to target a particular user and cause them to self-isolate. They can get sufficiently close to the target to create a proximity event that will score as high risk. If the attacker self-decalres symptoms, in the process submitting this contact event, the target will be notified to self-isolate.

   This is, to varying degrees, a feature of all digital contact tracing protocols and each will have different potential mitigations. Depending on the properties, over-the-air protocol and the opportunities for ‘sense checking’ interactions, this sort of attack may or may not scale.

   In our model, the ability to centrally corellate and ‘sense check’ putative interactions and the recording of observed radio parameters give us the ability to spot scaled attacks. We cannot spot an attack where an attacker observes a target’s Bluetooth transmissions and creates false proximity records that look reasonable. However, as more people submit contact records, it may be possible to corellate pairwise events. This is future work.

There are other scenarios that require special treatment and there will be other oddities that come to light as the system is scaled. The centralised model we have adopted allows us to manage these in a way that seems to be more difficult for a decentralised model, where the service is essentially a dumb router. This model also allows for more refined epidemiological inferences to be made as the transmission model is adapted to prevailing circumstances.

Second order proximity events, and the resulting cumulative ‘risk from’ and ‘risk to’ users will be available in due course and requires only a change in the centralised epidemiological algorithms. This would allow more targeted notifications where the transmission risk warrants.

In due course, the service may be extended to be one pathway to confirmed and authenticated clinical tests, which would follow a slightly different path, but still preserve privacy and security until a user specifically identifies themselves to the healthcare system.

Reidentification risk

One of the main issues with a centralised system is the potential risks brought about by the collection of the proximity events in one place, giving rise to a graph which describes proximity events between users.

In all large scale social graphs of this sort, there is a risk that seemingly innocuous data can be analysed to identify particular subgroups of the population. This is a well understood problem and papers such as this one by Ohm give a good overview. These reidentification techniques always require some sort of data, other than the persistent pseudonym, to be present.

At the time of writing, our contact graph nodes are labelled only with the persistent pseudonym, postal district (covering several thousand properties) and the various contact events (time, pairwise contact persistent pseudonym and the radio data that serves as a proxy for interaction distance). There is insufficient data here to attract any reidentification risk.

The risk comes as more data is added to the graph, or commingled with it. The app system is deployed on separate infrastructure, managed separately to the rest of the NHS systems. Any connection to other clinical or record systems, for example to link with testing, will be done through privacy preserving gateways. These will be detailed as designs are finalised for functionality. The addition of more data to the graph nodes needs careful consideration.

High level privacy and security criteria

In due course, we expect a more standardised way to compare the various tracing models that will emerge. When it does, we will use that, but in the meantime we offer a high level view of the main characteristics of the NHS app, at the time of writing.

Criterion 1 is satisfied, since, from the description, it is trivial to observe that we collect no personal data.

Criterion 2 is satisfied in two ways. Firstly, collection of proximity event data by Alice's device from Bob's device requires both Alice and Bob to have installed and activated the app, including by confirming a consent statement. Secondly, event data are stored locally, unless and until, they are submitted to the centralised service by a positive action on the part of the user (i.e. asserting they are symptomatic). Locally stored events age off as they become less useful under the infection model in use.

Criterion 3 is satisfied by the construction of the *BroadcastValue*. Given BV_i for a particular device, it is impossible for an external observer to calculate BV_{i-n} or BV_{i+n} for any $n \neq 0$. Similarly, given the entropy in the ephemeral private key y that is used to construct the *BroadcastValue*, there should be no relationship between any set of *BroadcastValue*s created by individual devices.

Criterion 4 is satisfied by virtue of there being no device-specific information in the system at all and the entropy in the ephemeral keypairs generated periodically by the device.

Criterion 5 is satisfied by tying a symmetric key to the *InstallationID* of the device. By observation, an external observer does not have the *InstallationID* of another user – and hence cannot create *BroadcastValue*s that lead to that *InstallationID* being recovered by the server – and also does not posses the device-specific symmetric key with which to encrypt and HMAC the submitted log. Spoofing data from another user requires compromise of that user's device to recover the *InstallationID* and symmetric key.

Criteria 6 and 7 are satisfied by the various controls in place in the calculation of real-world risk from the submitted RSSI data and the extra controls in place in the notification cascade system for 'weird' cascades. However, in reality, criterion 7(a) cannot be individually controlled without breaking overall system privacy promises. However, the impact of this attack is localised and minimal.

Criterion 8 is satisfied through the individual encryption keys per device, which provide authenticity of the notification communication from the server over the notification provider.

Centralised vs Decentralised

This paper does not make a case for either a centralised or decentralised matching model, but seeks only to explain the current design for the NHS systems. However, it seems prudent to document our current understanding of the implications of the use of a decentralised digital contact tracing model in the UK.

We believe that any likely decentralised model (because there are many) will have the following impacts in the UK:

- **Move the health response from 'react to symptoms' to 'react to clinical test results':** We cannot currently find a way to manage malicious notifications, or possible amplification attacks, in a decentralised model without authentication. Consequently, notification must be uniquely tied to an authentic clinical test. This generates a dependency on the digital authentication of clinical testing.
- **Delays to symptom reporting:** Most decentralised systems seem to introduce delays in the reporting of symptoms, sometimes due to characteristics of the crypt design, sometimes due to the data flow requirements.
- **Second order tracing (also known as recursive tracing) seems to have different characteristics:** A decentralised model allows only 'risk to' an individual from their exposures to be understood, while a centralised model allows 'risk from' an infectious individual to others.

These will affect the health response in the context of the epidemiological model used in the UK and it is for health professionals to decide on the likely impact of those changes.