About this file and how it was produced

This file lists the differences between the v1.0 and v1.1 of the ROBERT protocol specification document in a (hopefully) readable manner. It has been automatically produced as follows:

- automatic conversion from latex to text, using:
  `pandoc -s ROBERT-specification-EN-v1_0.tex -o ROBERT-specification-EN-v1_0.txt` (as well as for version 1.1)
  Note that this automatic conversion from latex to text using pandoc has some limits (e.g., the "(sk_S, pk_S)("Registration key-pair")" and similar are missing from section 3.1).
- automatic diff generation using:
  `rfcdiff --html --width 80 robert-specifications-v1.0.txt robert-specifications-v1.1.txt`
  This diff is based on the [IETF rfcdiff tool](#).

We hope this difference file will be of help to the interested reader.

Cheers. The authors

Diff: ROBERT-specification-EN-v1_0.txt - ROBERT-specification-EN-v1_1.txt

ROBERT-specification-EN-v1_0.txt	ROBERT-specification-EN-v1_1.txt
skipping to change at ***line 161***	**skipping to change at** ***line 161***
$EBID_{A,i}$ Ephemeral Bluetooth IDentifier of user $U_A$ at epoch	$EBID_{A,i}$ Ephemeral Bluetooth IDentifier of user $U_A$ at epoch
$i$	$i$
$ECC$ Encrypted Country Code	$ECC$ Encrypted Country Code
$ECC_{A,i}$ Encrypted Country Code used by $U_A$ at epoch $i$	$ECC_{A,i}$ Encrypted Country Code used by $U_A$ at epoch $i$
$epoch\_duration\_sec$ Duration of an epoch in seconds	$epoch\_duration\_sec$ Duration of an epoch in seconds
$ESR\_REQUEST$ Request sent by the $App$ to query the user status	$ESR\_REQUEST$ Request sent by the $App$ to query the user status
$ESR\_REPLY$ Answer sent by the server to users to notify their st	$ESR\_REPLY$ Answer sent by the server to users to notify their st
atus	atus
$HELLO$ Message broadcast by an $App$ via its Bluetooth Low E	$HELLO$ Message broadcast by an $App$ via its Bluetooth Low E
nergy interface	nergy interface
$ID$ Permanent and anonymous identifier associated to each	$ID$ Permanent and anonymous identifier associated to each
registered user	registered user
$ID_A$ Permanent and anonymous identifier of user $U_A$, sto	$ID_A$ Permanent and anonymous identifier of user $U_A$, sto
red by the server	red by the server
$IDTable$ Database maintained by the back-end server	$IDTable$ Database maintained by the back-end server
$K_A$ Shared key between the server and the $App_A$	$K^{enc}_A$ Shared key between the server and the $App_A$, used f
	or encryption of sensitive information.
	$K^{auth}_A$ Shared key between the server and the $App_A$, used f
	or authentication of $App_A$ messages.
$K_G$ Federation Key (shared between the servers of all cou	$K_G$ Federation Key (shared between the servers of all cou
ntries with a federation agreement)	ntries with a federation agreement)
$K_S$ Server Key (used by a server to generate $EBIDs$), st	$K_S$ Server Key (used by a server to generate $EBIDs$), st
ored by the server	ored by the server
$LEE_A$ List of exposed epochs of user $U_A$, stored in $IDTa	$LEE_A$ List of exposed epochs of user $U_A$, stored in $IDTa
ble$	ble$
$LocalProximityList$ Local list on an $App$ where the $HELLO$ messages rec	$LocalProximityList$ Local list on an $App$ where the $HELLO$ messages rec
eived by nearby devices are stored	eived by nearby devices are stored
$M$ Number of epochs between 2 consecutive requests by an	$M$ Number of epochs between 2 consecutive requests by an
App to the back-end server to obtain its list of $(EBID, ECC)$ pairs for the fo	App to the back-end server to obtain its list of $(EBID, ECC)$ pairs for the fo
llowing epochs	llowing epochs
	$(sk_S, pk_S)$ Registration key-pair, an asymmetric key-pair of the
	server, used during registration.
$Srv$ The back-end server	$Srv$ The back-end server
$SRE_A$ Variable that indicates the last epoch when $U_A$ has	$SRE_A$ Variable that indicates the last epoch when $U_A$ has
sent a "Status Request" to the server, stored in $IDTable$	sent a "Status Request" to the server, stored in $IDTable$
$T$ The minimum number of epochs between 2 consecutive $E	$T$ The minimum number of epochs between 2 consecutive $E
SR_REQUEST$	SR_REQUEST$
$Tpststart$ The time when the proximity tracing service has been	$Tpststart$ The time when the proximity tracing service has been
started	started
$UN_A$ Flag indicating if $U_A$ has already been notified to	$UN_A$ Flag indicating if $U_A$ has already been notified to
be at risk of exposure. $UN_A$ is stored in the $IDTable$	be at risk of exposure. $UN_A$ is stored in the $IDTable$
: Glossary of terms and variables used in this	: Glossary of terms and variables used in this
paper[]{label="tab:notations"}	paper[]{label="tab:notations"}
High-Level Description and Assumptions	High-Level Description and Assumptions
skipping to change at ***line 241***	**skipping to change at** ***line 243***
be used by the health authority. In fact, the algorithm will probably be	be used by the health authority. In fact, the algorithm will probably be
different if the App is used to notify users that need to get tested in	different if the App is used to notify users that need to get tested in
priority or if it is used to decide who should stay/go into confinement.	priority or if it is used to decide who should stay/go into confinement.
System Overview	System Overview
---------------	---------------
All the notations used in this paper are summarized in	All the notations used in this paper are summarized in
Table [\[tab:notations\]](#tab:notations){reference-type="ref"	Table [\[tab:notations\]](#tab:notations){reference-type="ref"
reference="tab:notations"}. The proposed system is composed of users who	reference="tab:notations"}. The proposed system is composed of users who
install the Proximity Tracing Application, $App$, and a back-end server,	install the Proximity Tracing Application, $App$, and a back-end server
$Srv$, under the control of the authority. We assume that $Srv$ is	under the control of the authority. We assume that the back-end server
configured with a well-known domain name, certificate and is highly	is configured with a well-known domain name, certificate and is highly
secured.	secured.
Apps interact with the system through the four following procedures:	Apps interact with the system through the four following procedures:
- **Initialization:** When a user wants to use the service, she	- **Initialization:** When a user wants to use the service, she
installs the application, $App$, from an official App store (Apple	installs the application, $App$, from an official App store (Apple
or Google). $App$ then registers to the server that generates a	or Google). $App$ then registers to the server that generates a
permanent identifier (ID) and several Ephemeral Bluetooth	permanent identifier (ID) and several Ephemeral Bluetooth
Identifiers ($EBIDs$). The back-end maintains a table, , that keeps	Identifiers ($EBIDs$). The back-end maintains a table, , that keeps
an entry for each registered ID. The stored information are	an entry for each registered ID. The stored information are
skipping to change at ***line 353***	**skipping to change at** ***line 355***
trivial to identify infected users [@cryptoeprint:2020:399]. The	trivial to identify infected users [@cryptoeprint:2020:399]. The
system's design would be based on the fact that all users which at any	system's design would be based on the fact that all users which at any
point ever saw an infected user's *HELLO* can now use contextual	point ever saw an infected user's *HELLO* can now use contextual
metadata (such as a meeting date and time) to identify the infected	metadata (such as a meeting date and time) to identify the infected
users.	users.
Initialisation	Initialisation
==============	==============
This section describes the initialisation process on the server and the	This section describes the initialisation process on the server and the
application	application.
(Figure [\[fig:initialisation\]](#fig:initialisation){reference-type="ref"	
reference="fig:initialisation"}).	

\centering ![Initialisation of the system: server set up and application registration.[]{label="fig:initialisation"}](figures/init-1node-DE.png){width="11cm"}	
Server Set Up {#sec:serv-init} ------------- The back-end server is initialized at the beginning of the proximity tracing official period under the control of the Authority. In order to be able to determine in which period and epoch the system is, the server stores $T_{ptsstart}$, the time when the proximity tracing service has been started in the country. From this timestamp, the server maintains a counter $i$, initialized to 0, representing the current epoch number	Server Set Up {#sec:serv-init} ------------- The back-end server is initialized at the beginning of the proximity tracing official period under the control of the Authority. In order to be able to determine in which period and epoch the system is, the server stores $T_{ptsstart}$, the time when the proximity tracing service has been started in the country. From this timestamp, the server maintains a counter $i$, initialized to 0, representing the current epoch number
skipping to change at *line 382*	skipping to change at *line 378*
We also assume that the server is configured with: a $L$-bit long key, with $L>=128$ to be defined, only known by the server[^5]. a $L$-bit long key, with $L>=128$ to be defined, shared between all servers in Europe (used for federation, see Section [8](#sec:federation){reference-type="ref" reference="sec:federation"}).	We also assume that the server is configured with: a $L$-bit long key, with $L>=128$ to be defined, only known by the server[^5]. a $L$-bit long key, with $L>=128$ to be defined, shared between all servers in Europe (used for federation, see Section [8](#sec:federation){reference-type="ref" reference="sec:federation"}).
	an asymmetric-key pair, whose private key $sk_S$ is known only from the server, and public key part is known to every App. This key-pair is defined over the elliptic curve NIST-P256, with $pk_S = sk_S.G$, where $G$ is the base point of prime order on the curve defined by the specifications[^6].
a 8-bit long value that codes the country where the server provides service (used for federation, see Section [8](#sec:federation){reference-type="ref" reference="sec:federation"}). These country codes should be known to all international systems where federation is possible. For example, France could use the code "33\", Germany the code \"49\", \...	a 8-bit long value that codes the country where the server provides service (used for federation, see Section [8](#sec:federation){reference-type="ref" reference="sec:federation"}). These country codes should be known to all international systems where federation is possible. For example, France could use the code "33\", Germany the code \"49\", \...
	Note that the server relies on a security module to store and manipulate its secrets and the secrets shared with mobile applications.
In addition, the server maintains a local database, (see Section [3.2](#sec:serv-reg){reference-type="ref" reference="sec:serv-reg"} for details).\ Application Registration (Server Side) {#sec:serv-reg} -------------------------------------- ### The $IDTable$ database For each registered application $App_A$, belonging to an anonymous user $U_A$, the server $Srv$ keeps a record in a local database with the following information:	In addition, the server maintains a local database, (see Section [3.2](#sec:serv-reg){reference-type="ref" reference="sec:serv-reg"} for details).\ Application Registration (Server Side) {#sec:serv-reg} -------------------------------------- ### The $IDTable$ database For each registered application $App_A$, belonging to an anonymous user $U_A$, the server $Srv$ keeps a record in a local database with the following information:
a $L$-bit long key, with $L>=128$ to be defined, shared with $App_A$, that is used to authenticate $App_A$ messages.	a $L$-bit long key, with $L \geq 128$ to be defined, shared with $App_A$, that is used to authenticate $App_A$ messages. a $L$-bit long key, with $L \geq 128$ to be defined, shared with $App_A$, that is used to encrypt sensitive information sent from the server to $A$.
a 40-bit identifier that is unique for each registered App, and generated randomly (random draw process without replacement to avoid collision). This permanent ID is only known to the server.	a 40-bit identifier that is unique for each registered App, and generated randomly (random drawing process without replacement to avoid collisions). This permanent ID is only known to the server.
this flag indicates if the associated user has already been notified to be ("true\") or not ("false\"). It is initialized with value "false\". Once set to "true\", the user is not allowed to make any additional status request. The flag can be reset if the user can prove that she is not at risk anymore (for example by proving that she got a test and the result was negative). a 24-bit value that indicates the last epoch when $U_A$ has sent a "Status Request\". each time one of A's EBID appears in the proximity list of an infected user, the epoch $j$ when the encounter happened between the infected user and $U_{A}$ is added to this list. This list of epochs reflects the exposure of the user (temporal and frequency information) and is used, together with other information (e.g. the duration and proximity of a contact to an infected person), to compute the risk score. Note that a	this flag indicates if the associated user has already been notified to be ("true\") or not ("false\"). It is initialized with value "false\". Once set to "true\", the user is not allowed to make any additional status request. The flag can be reset if the user can prove that she is not at risk anymore (for example by proving that she got a test and the result was negative). a 24-bit value that indicates the last epoch when $U_A$ has sent a "Status Request\". each time one of A's EBID appears in the proximity list of an infected user, the epoch $j$ when the encounter happened between the infected user and $U_{A}$ is added to this list. This list of epochs reflects the exposure of the user (temporal and frequency information) and is used, together with other information (e.g. the duration and proximity of a contact to an infected person), to compute the risk score. Note that a
given epoch $i$ can appear several times in this list if $App_A$ received several *HELLO* messages from infected users during that epoch.	given epoch $i$ can appear several times in this list if several *HELLO* messages sent by $App_A$ appeared in the proximity list(s) of one of several infected users during that epoch.
Application Registration (Application Side) --- A user $U_A$ who wants to install the application on his device must download it from the official Apple or Android stores. After installing the application $App_A$, $U_A$ needs to register to the back-end server. During this registration phase: - A proof-of-work (PoW) system (like a CAPCHA) is implemented in order	Application Registration (Application Side) --- A user $U_A$ who wants to install the application on his device must download it from the official Apple or Android stores. After installing the application $App_A$, $U_A$ needs to register to the back-end server. During this registration phase: - A proof-of-work (PoW) system (like a CAPCHA) is implemented in order
skipping to change at *line 452*	skipping to change at *line 462*
(over a TLS channel) with: - The current epoch value, $i$. - The duration of an epoch, $epoch\_duration\_sec$. - The starting time of the following epoch, $T_{ptsstart} + (i+1)*epoch\_duration\_sec$ (This is required for synchronization with the server).	(over a TLS channel) with: - The current epoch value, $i$. - The duration of an epoch, $epoch\_duration\_sec$. - The starting time of the following epoch, $T_{ptsstart} + (i+1)*epoch\_duration\_sec$ (This is required for synchronization with the server).
- The $\mbox{K}_A$ key, shared with the server.	- The keys $K^{auth}_A$ and $K^{enc}_A$, shared with the server.
- An initial list of $T$ $(EBID_{A,i},ECC_{A,i})$ pairs (see Section [4](#sec:EBID){reference-type="ref" reference="sec:EBID"} for more details).	- An initial list of $T$ $(EBID_{A,i},ECC_{A,i})$ pairs (see Section [4](#sec:EBID){reference-type="ref" reference="sec:EBID"} for more details).
	The keys $K^{auth}_A$ and $K^{enc}_A$ are exchanged by means of a key establishment procedure: - $App_A$ generates an ephemeral key pair $(ske_A, pke_A=ske_A.G)$ and transmits the value $pke_A$ to the server in its registration message - A shared secret $SharedSecret$, the x-coordinate of the point $(ske_A.sk_S).G$, is obtained by $App_A$ and the server as follows:

```
Generation of the Ephemeral Bluetooth Identifiers {#sec:EBID}
=================================================

During registration and then regularly, i.e. every $M$ epochs (value to
be defined), each registered user $U_A$ connects to the server in order
to obtain a list of the $M$ $(EBID_{A,i},ECC_{A,i})$ pairs for the $M$
following epochs. For efficiency, this request can be performed together
with an *Exposure Status Request* (see Section
[6.2](#sec:request-server){reference-type="ref"
reference="sec:request-server"}).

Upon receiving such a request, the server generates and sends to $App_A$
a list of $T$ $(EBID_{A,i},ECC_{A,i})$ pairs for the upcoming $T$
epochs[^6], where:

-$\textrm{EBID}_{A,i}$ (\"Ephemeral Bluetooth IDentifier for A\"):

:   a 64-bit identifier generated for the epoch $i$ as:
    $$EBID_{A,i} = ENC({K}_{S}, i~|~ID_{A})$$ where $ENC$ is a
    block-cipher with 64-bit block size, for example 3-key TripleDES.

-$\textrm{ECC}_{A,i}$ (\"Encrypted Country Code\"):

:   an 8-bit code that indicates, in an encrypted form, the country code
    of ${EBID}_{A,i}$. This field is used for federation purposes and
    can only be decrypted by back-end servers that have federation
    agreements. More specifically. $ECC_{A,i}$ is the country code
    $CC_A$ encrypted using $AES-OFB$ with key $K_G$ and $IV$
    $EBID_{A,i}$, i.e.,
    $$ECC_{A,i} = MSB(AES({K_G},EBID_{A,i}~|~0^{64})) \oplus CC_A$$

Proximity Discovery
===================

In this phase, $App_A$ performs two operations simultaneously:

-   ***HELLO* Message Broadcasting** and,

-   ***HELLO* Message Collection**.

\centering
```

```
    -   $App_A$ computes $ske_A.pk_S$

    -   The server computes $sk_S.pke_A$

-   $App_A$ and the server derive $K^{auth}_A$ as
    $K^{auth}_A = HMAC\_SHA256(SharedSecret,$ \"authentication key\"$)$

-   $App_A$ and the server derive $K^{enc}_A$ as
    $K^{enc}_A = HMAC\_SHA256(SharedSecret,$ \"encryption key\"$)$
Generation of the Ephemeral Bluetooth Identifiers {#sec:EBID}
=================================================

During registration and then regularly, i.e. every $M$ epochs (value to
be defined), each registered user $U_A$ connects to the server in order
to obtain a list of the $M$ $(EBID_{A,i},ECC_{A,i})$ pairs for the $M$
following epochs. For efficiency, this request can be performed together
with an *Exposure Status Request* (see Section
[6.2](#sec:request-server){reference-type="ref"
reference="sec:request-server"}).

Upon receiving such a request, the server generates and sends to $App_A$
an encrypted list of $T$ $(EBID_{A,i},ECC_{A,i})$ pairs for the upcoming
$T$ epochs[^7], where:

-$\textrm{EBID}_{A,i}$ (\"Ephemeral Bluetooth IDentifier for A\"):

:   a 64-bit identifier generated for the epoch $i$ as:
    $$EBID_{A,i} = ENC({K}_{S}, i~|~ID_{A})$$ where $ENC$ is a 64-bit
    block cipher, for example, $SKINNY-64/192$ (a block cipher of block
    size 64 bits and key size 192 bits) [@skinny].

-$\textrm{ECC}_{A,i}$ (\"Encrypted Country Code\"):

:   an 8-bit code that indicates, in an encrypted form, the country code
    of ${EBID}_{A,i}$. This field is used for federation purposes and
    can only be decrypted by back-end servers that have federation
    agreements. More specifically. $ECC_{A,i}$ is the country code
    $CC_A$ encrypted using $AES-OFB$ with key $K_G$ and $IV$
    $EBID_{A,i}$, i.e.,
    $$ECC_{A,i} = MSB(AES({K_G},EBID_{A,i}~|~0^{64})) \oplus CC_A$$

The encryption of the list is performed with the authenticated
encryption algorithm $AES-GCM$, using key $K^{enc}_A$ with a random
96-bit IV and a 128-bit tag.
Proximity Discovery
===================

In this phase, $App_A$ performs two operations simultaneously:

-   ***HELLO* Message Broadcasting** and,

-   ***HELLO* Message Collection**.

\centering
```

skipping to change at *line 540* **skipping to change at *line 575***

```
:   16-bit timestamp (to encode the fine-grain emission time). It
    contains the 16 less significant bits of the current NTP \"Seconds\"
    timestamp of $App_A$ (which represents, for era 0, the number of
    seconds since 0h January 1st, 1900 UTC). Since it is truncated to 16
    bits, it covers a bit more than 18 hours, what is much larger than
    the epoch duration. This field is used to mitigate replay attacks.

-$\textrm{MAC}_{A,i}$:

:   a $HMAC-SHA256(K_A, c_1~|~M_{A,i})$ truncated to 40 bits ($c_1$ is
    the 8-bit prefix \"01\"). This field is used to prevent integrity
    attacks on the *HELLO* messages.

**Note:** The *HELLO* contains a \"country code\" that is encrypted and
is therefore not visible by other devices. It is used in case of
federation by the foreign back-end servers (see
Section [8](#sec:federation){reference-type="ref"
reference="sec:federation"} for more details).

 *HELLO* Message Collection
--------------------------

$App_A$ continuously collects *HELLO* messages sent by nearby devices
running the same application.

Upon receiving $HELLO_{A,i}$, $App_A$:

1.  parses[^7] $HELLO_{A,i}$ to retrieve $ecc_A$ (8 bits), $ebid_A$ (64
    bits), $time_A$ (16 bits) and $mac_A$ (40 bits).

2.  obtains a 32-bit NTP "Seconds" timestamp, $time'_A$.

3.  checks that: $$|time_A - {TRUNC}_{16}(time'_A)|<\delta$$ where
    $\delta$ is a configurable time tolerance (typically a few seconds,
    value to be defined)[^8].

4.  stores, in its   the following pair[^9]: $$(HELLO_{A,i}, time'_A)$$

**Note:** Entries in  are automatically deleted after $CT$ Days (the
value $CT$ needs to be defined with the health authority).

Infected User Declaration {#sec:infected_app_registration}
=========================

\centering
![Infected Node Declaration: upon agreement, a user tested uploads her
 to the
```

```
:   16-bit timestamp (to encode the fine-grain emission time). It
    contains the 16 less significant bits of the current NTP \"Seconds\"
    timestamp of $App_A$ (which represents, for era 0, the number of
    seconds since 0h January 1st, 1900 UTC). Since it is truncated to 16
    bits, it covers a bit more than 18 hours, what is much larger than
    the epoch duration. This field is used to mitigate replay attacks.

-$\textrm{MAC}_{A,i}$:

:   a $HMAC-SHA256(K^{auth}_A, c_1~|~M_{A,i})$ truncated to 40 bits
    ($c_1$ is the 8-bit prefix \"01\"). This field is used to prevent
    integrity attacks on the *HELLO* messages.

**Note:** The *HELLO* contains a \"country code\" that is encrypted and
is therefore not visible by other devices. It is used in case of
federation by the foreign back-end servers (see
Section [8](#sec:federation){reference-type="ref"
reference="sec:federation"} for more details).

 *HELLO* Message Collection
--------------------------

$App_A$ continuously collects *HELLO* messages sent by nearby devices
running the same application.

Upon receiving $HELLO_{A,i}$, $App_A$:

1.  parses[^8] $HELLO_{A,i}$ to retrieve $ecc_A$ (8 bits), $ebid_A$ (64
    bits), $time_A$ (16 bits) and $mac_A$ (40 bits).

2.  obtains a 32-bit NTP "Seconds" timestamp, $time'_A$.

3.  checks that: $$|time_A - {TRUNC}_{16}(time'_A)|<\delta$$ where
    $\delta$ is a configurable time tolerance (typically a few seconds,
    value to be defined)[^9].

4.  stores, in its   the following pair[^10]: $$(HELLO_{A,i}, time'_A)$$

**Note:** Entries in  are automatically deleted after $CT$ Days (the
value $CT$ needs to be defined with the health authority).

Infected User Declaration {#sec:infected_app_registration}
=========================

\centering
![Infected Node Declaration: upon agreement, a user tested uploads her
 to the
```

skipping to change at *line 593* **skipping to change at *line 628***

```
####  Upload Authorization Procedure:

If user $U_C$ is tested and diagnosed at a hospital or medical office
and it is estimated that she was contagious from $ContStart_{C}$ to
$ContEnd_{C}$ (expressed in seconds using the standard NTP \"Seconds\"
system), she is proposed to upload to the server each $(HELLO, Time)$
pair of her  that satisfies: $$ContStart_{C} < Time < ContEnd_{C}$$ In
this document, we do not detail the interactions between $App_C$ and the
health authority. In particular, we do not present the
security/authorization procedure that verifies that only authorized and
positively-tested users are allowed to upload their [^10] (note that
during this upload, $App_C$ does not reveal any of its $EBIDs$ to the
server).
```

```
####  Upload Authorization Procedure:

If user $U_C$ is tested and diagnosed at a hospital or medical office
and it is estimated that she was contagious from $ContStart_{C}$ to
$ContEnd_{C}$ (expressed in seconds using the standard NTP \"Seconds\"
system), she is proposed to upload to the server each $(HELLO, Time)$
pair of her  that satisfies: $$ContStart_{C} < Time < ContEnd_{C}$$ In
this document, we do not detail the interactions between $App_C$ and the
health authority. In particular, we do not present the
security/authorization procedure that verifies that only authorized and
positively-tested users are allowed to upload their [^11] (note that
during this upload, $App_C$ does not reveal any of its $EBIDs$ to the
server).
```

```
#### Upload Mechanism:

A  contains the $EBIDs$ of the devices that the infected user has
encountered in the last $CT$ days. This information together with the
timing information associated with each *HELLO* message could be used to
build the de-identified social/proximity graph of the infected user. The
aggregation of many such social/proximity graphs may lead, under some
```

skipping to change at *line 615*

```
social graphs of the users.

Would that be a concern, it is necessary to \"break\" the link between
any two $EBIDs$ contained in the  to prevent the server from getting get
this information. Therefore, instead of uploading the   our scheme
uploads each of its elements independently.

Different solutions can be envisioned to achieve this goal:

-   The $(HELLO, Time)$ pairs of the  are sent to the server one by one
    using a $Mixnet$[^11]. Upon reception of these messages, the server
    won't be able to associate them with a  if the upload is spread over
    a long period of time.

-   The  is uploaded on a trusted server (for example at a hospital or
    health organization) that mixes all the $(HELLO, Time)$ pairs of all
    infected users' . The back-end server has only access to the exposed
    entries via a specific API provided by the trusted server.

-   The back-end server is equipped with some secure hardware component
    that processes the uploads of the . The back-end server has only
    access to the exposed entries via a specific API provided by the
    secure hardware module.

 Server Operations {#sec:request-server}
------------------

Upon reception of a $h_A=(HELLO_A, Time_A)$, the server:

1.  parses $h_A$ to retrieve $ecc_A$ (8 bits), $ebid_A$ (64 bits), $t_A$
    (16 bits), $mac_A$ (40 bits) and $time'_A$ (32 bits).

2.  decrypts $ecc_A$, using $K_G$, to recover the message country code,
    $cc_A$. If $cc_A$ is different from the server's country code,
    $CC_S$, and corresponds to a valid country code, $h_A$ is forwarded
    to the back-end server managing this country using the federation
    procedure in place (See
    Section [8](#sec:federation){reference-type="ref"
    reference="sec:federation"}). Otherwise it continues.

3.  computes $ENC^{-1}(K_S, ebid_A)$ to retrieve $i_A~|~id_A$, where
    $i_A$ is the epoch number corresponding to $ebid_A$ and $id_A$ is
    the permanent identifier.

4.  verifies that $id_A$ corresponds to the ID of a registered user,
    otherwise $h_A$ is rejected silently.

5.  checks that: $$|t_A - {TRUNC}_{16}(time'_A)|<\delta$$ where $\delta$
    is a configurable time tolerance (typically a few seconds, value to
    be defined). $h_A$ is rejected if this test is not satisfied.

6.  checks that $time'_A$ corresponds to epoch $i_A$:
    $$|(Time_A - T_{tpsstart})/\textrm{epoch}\_\textrm{duration}\_\textrm{sec} -
 i_A|\leq 1$$
    (note that a difference of 1 may happen if the *HELLO* message is
    sent at the very end of the epoch due to transmission and processing
    times). $h_A$ is rejected if this test is not satisfied.

7.  retrieves from , $K_A$, the key associated with $id_A$.

8.  verifies if the MAC, $mac_A$, is valid as follows:
    $$mac_A == HMAC-SHA256(K_A, c_1~|~ecc_A~|~ebid_A~|~t_A)$$ If $mac_A$
    is invalid, $h_A$ is rejected silently.

9.  adds $i_A$ in $LEE_A$[^12].

10. securely erases $(HELLO_A, Time_A)$.

Exposure Status Request (ESR) {#sec:status_request}
=============================

\centering
![Exposure Status Request: the App regularly requests the server to know
if any of its EBID appeared in any *HELLO* messages collected by an
infected users.[]{label="fig:query"}](figures/query.png){width="10cm"}

In order to check whether user $U_A$ is \"at risk\", i.e. if she has
encountered infected and contagious users in the last $CT$ days,
application $App_A$ regularly[^13] sends \"Exposure Status\" Requests
($ESR\_REQUEST$) to the server $Srv$ for $ID_A$. The server then
computes a \"risk score\" value, derived in part from the list $LEE_A$
corresponding to $ID_A$. The server replies with a $ESR\_REPLY$ message
that is set to \"1\" when the user is \"at risk\" (i.e. if the \"risk
score\" is larger than a threshold value) or to \"0\" otherwise.

#### Application processing:

$App_A$ queries the server by sending the following request over an TLS
channel: $$ESR\_REQUEST_{A,i}=[EBID_{A,i} ~|~ Time ~|~ MAC_{A,i}]$$ with
$$MAC_{A,i}= HMAC-SHA256(K_A, c2 ~|~ EBID_{A,i} ~|~ Time )]$$ where:

-   $c_2$: Fixed prefix \"02\" (8 bits).

-   $EBID_{A,i}$: Ephemeral Bluetooth ID of the current epoch $i$ (64
    bits).

-   $Time$: 32-bit timestamp in seconds, corresponding to the
    transmission time.

#### Server processing:
```

```
#### Upload Mechanism:

A  contains the $EBIDs$ of the devices that the infected user has
encountered in the last $CT$ days. This information together with the
timing information associated with each *HELLO* message could be used to
build the de-identified social/proximity graph of the infected user. The
aggregation of many such social/proximity graphs may lead, under some
```

skipping to change at *line 650*

```
social graphs of the users.

Would that be a concern, it is necessary to \"break\" the link between
any two $EBIDs$ contained in the  to prevent the server from getting get
this information. Therefore, instead of uploading the   our scheme
uploads each of its elements independently.

Different solutions can be envisioned to achieve this goal:

-   The $(HELLO, Time)$ pairs of the  are sent to the server one by one
    using a $Mixnet$[^12]. Upon reception of these messages, the server
    won't be able to associate them with a  if the upload is spread over
    a long period of time.

-   The  is uploaded on a trusted server (for example at a hospital or
    health organization) that mixes all the $(HELLO, Time)$ pairs of all
    infected users' . The back-end server has only access to the exposed
    entries via a specific API provided by the trusted server.

-   The back-end server is equipped with some secure hardware component
    that processes the uploads of the . The back-end server has only
    access to the exposed entries via a specific API provided by the
    secure hardware module.

 Server Operations {#sec:request-server}
------------------

Upon reception of a $h_A=(HELLO_A, Time_A)$, the server:

1.  parses $h_A$ to retrieve $ecc_A$ (8 bits), $ebid_A$ (64 bits), $t_A$
    (16 bits), $mac_A$ (40 bits) and $Time_A$ (32 bits).

2.  checks that: $$|t_A - {TRUNC}_{16}(Time_A)|<\delta$$ where $\delta$
    is a configurable time tolerance (typically a few seconds, value to
    be defined). $h_A$ is rejected if this test is not satisfied.

3.  decrypts $ecc_A$, using $K_G$, to recover the message country code,
    $cc_A$. If $cc_A$ is different from the server's country code,
    $CC_S$, and corresponds to a valid country code, $h_A$ is forwarded
    to the back-end server managing this country using the federation
    procedure in place (See
    Section [8](#sec:federation){reference-type="ref"
    reference="sec:federation"}). Otherwise it continues.

4.  computes $ENC^{-1}(K_S, ebid_A)$ to retrieve $i_A~|~id_A$, where
    $i_A$ is the epoch number corresponding to $ebid_A$ and $id_A$ is
    the permanent identifier[^13].

5.  verifies that $id_A$ corresponds to the ID of a registered user,
    otherwise $h_A$ is rejected silently[^14].

6.  checks that $Time_A$ corresponds to epoch $i_A$[^15]:
    $$|(Time_A - T_{tpsstart})/\textrm{epoch}\_\textrm{duration}\_\textrm{sec} -
 i_A|\leq 1$$
    (note that a difference of 1 may happen if the *HELLO* message is
    sent at the very end of the epoch due to transmission and processing
    times). $h_A$ is rejected if this test is not satisfied.

7.  retrieves from , $K_A$, the key associated with $id_A$.

8.  verifies if the MAC, $mac_A$, is valid as follows:
    $$mac_A == HMAC-SHA256(K_A, c_1~|~ecc_A~|~ebid_A~|~t_A)$$ If $mac_A$
    is invalid, $h_A$ is rejected silently[^16].

9.  adds $i_A$ in $LEE_A$[^17].

10. erases $(HELLO_A, Time_A)$.

Exposure Status Request (ESR) {#sec:status_request}
=============================

\centering
![Exposure Status Request: the App regularly requests the server to know
if any of its EBID appeared in any *HELLO* messages collected by an
infected users.[]{label="fig:query"}](figures/query.png){width="10cm"}

In order to check whether user $U_A$ is \"at risk\", i.e. if she has
encountered infected and contagious users in the last $CT$ days,
application $App_A$ regularly[^18] sends \"Exposure Status\" Requests
($ESR\_REQUEST$) to the server $Srv$ for $ID_A$. The server then
computes a \"risk score\" value, derived in part from the list $LEE_A$
corresponding to $ID_A$. The server replies with a $ESR\_REPLY$ message
that is set to \"1\" when the user is \"at risk\" (i.e. if the \"risk
score\" is larger than a threshold value) or to \"0\" otherwise.

#### Application processing:

$App_A$ queries the server by sending the following request over an TLS
channel:

$$ESR\_REQUEST_{A,i} = [EBID_{A,i} ~|~ i ~|~ Time ~|~ MAC_A]$$ with
$$MAC_{A,i} = HMAC-SHA256(K^{auth}_A, c2 ~|~ EBID_{A,i} ~|~ i ~|~ Time)$$

where:

-   $c_2$: Fixed prefix \"02\" (8 bits).

-   $EBID_{A,i}$: Ephemeral Bluetooth ID of the epoch $i$ (64 bits).

-   $i$: epoch of validity of $EBID_{A,i}$, either the current epoch if
    $App_A$ has an $EBID$ for the current epoch, or the latest epoch for
    which $App_A$ has an $EBID$ (24 bits).

-   $Time$: 32-bit timestamp in seconds, corresponding to the
    transmission time.

#### Server processing:
```

```
Upon the reception of a request, $ESR\_REQUEST_{A,i}$, at epoch $i$, the
server:

1.  parses $ESR\_REQUEST_{A,i}$ to retrieve $ebid_A$, $time_A$ and
    $mac_A$.

2.  verifies that $time_A$ is close to its current time.

3.  retrieves the permanent identifier $id_A$ and epoch $i_A$ by
    decrypting $ebid_A$, as $$i_A~|~id_A=ENC^{-1}(K_S, ebid_A)$$

4.  uses $id_A$ to retrieve from ��the associated entries: $K_A$, $UN_A$,
    $SRE_A$, $LEE_A$.

5.  verifies that $$mac_A=={HMAC-SHA256}(K_A, c2 ~|~ ebid_A ~|~ t_A )$$



    If $mac_A$ is incorrect, the message is silently rejected;

6.  verifies that $UN_A$ is $"false"$ (User Notified) in order to check
    that the user has not already received a \"At Risk\" notification.
    If the user has already been notified, the message is silently
    rejected.

7.  verifies that $(i - SRE_A)>=T$, where $T$ is minimum number of
    epochs between two consecutive $ESR\_REQUEST$. Otherwise, the
    message is silently rejected.

####

If the $ESR\_REQUEST_{A,i}$ is valid, the server:

1.  updates $SRE_A$ with the current epoch number, $i$, in .

2.  computes a \"risk score\" value, derived in part from the list
```

```
Upon the reception of a request, $ESR\_REQUEST_{A,i}$, at epoch $i$, the
server:

1.  parses $ESR\_REQUEST_{A,i}$ to retrieve $ebid_A$, $i_A$, $time_A$
    and $mac_A$.

2.  verifies that $time_A$ is close to its current time.

3.  retrieves the permanent identifier $id_A$ and epoch $i'_A$ by
    decrypting $ebid_A$, as $$i'_A~|~id_A=ENC^{-1}(K_S, ebid_A)$$

4.  verifies that $i_A == i'_A$, otherwise the message is rejected.


5.  uses $id_A$ to retrieve from  the associated entries: $K^{auth}_A$,
    $UN_A$, $SRE_A$, $LEE_A$.

6.  verifies that
    $$mac_{A,i}== {HMAC-SHA256}(K^{auth}_A, c2 ~|~ ebid_A ~|~ i_A ~|~ t_A )$$
    If $mac_A$ is incorrect, the message is silently rejected;

7.  verifies that $UN_A$ is $"false"$ (User Notified) in order to check
    that the user has not already received a \"At Risk\" notification.
    If the user has already been notified, the message is silently
    rejected.

8.  verifies that $(i - SRE_A) \geq T$, where $T$ is minimum number of
    epochs between two consecutive $ESR\_REQUEST$. Otherwise, the
    message is silently rejected.

####

If the $ESR\_REQUEST_{A,i}$ is valid, the server:

1.  updates $SRE_A$ with the current epoch number, $i$, in .

2.  computes a \"risk score\" value, derived in part from the list
```

skipping to change at *line 782* | skipping to change at *line 827*

```
neighboring states. This is especially true in the case of Europe, where
freedom of movement is a core value.

We therefore propose the use of a distributed, federated architecture
where countries may operate their own back-ends and develop their own
Apps. This is also practical as we can expect each instance to require
country-specific options with respect to health system integration and
localization.

A detailed specification of a federation protocol remains to be defined
in agreement with all other partners of the PEPP-PT initiative[^14].
However, the proposed structure of the *HELLO* message allows some
elegant solutions. We propose a standard format for all *HELLO* messages
as follows[^15]:

$ECC_{Country} (8 bits) ~|~ EBID (64 bits) ~|~ t (16 bits) ~|~ MAC (40 bits)$

Given this simple definition, a User $Bernard$ from a given country,
let's say France, could visit another country, let's say Germany, and
still be able to use his national application. The protocol works as
follows:

-   When $Bernard$ goes to Germany, his App broadcasts, at each epoch
    $j$, $HELLO_{FR,j}$ messages as defined in Section
```

```
neighboring states. This is especially true in the case of Europe, where
freedom of movement is a core value.

We therefore propose the use of a distributed, federated architecture
where countries may operate their own back-ends and develop their own
Apps. This is also practical as we can expect each instance to require
country-specific options with respect to health system integration and
localization.

A detailed specification of a federation protocol remains to be defined
in agreement with all other partners of the PEPP-PT initiative[^19].
However, the proposed structure of the *HELLO* message allows some
elegant solutions. We propose a standard format for all *HELLO* messages
as follows[^20]:

$ECC_{Country} (8 bits) ~|~ EBID (64 bits) ~|~ t (16 bits) ~|~ MAC (40 bits)$

Given this simple definition, a User $Bernard$ from a given country,
let's say France, could visit another country, let's say Germany, and
still be able to use his national application. The protocol works as
follows:

-   When $Bernard$ goes to Germany, his App broadcasts, at each epoch
    $j$, $HELLO_{FR,j}$ messages as defined in Section
```

skipping to change at *line 855* | skipping to change at *line 900*

```
potentially different EBID schemes. It further allows us to improve upon
the format without requiring changes to the federation protocols.

\appendix
Towards Probabilistic Notifications {#sec:proba}
==================================

As described in previous work
[@canetti2020anonymous; @cryptoeprint:2020:399], all proximity-tracking
schemes are vulnerable to the \"one entry\" attack. In this attack, the
adversary has only one entry, corresponding to $User_T$, in her [^16].
When the adversary is notified \"at risk\", she learns that $User_T$ was
diagnosed COVID-positive. The $ROBERT$ scheme, however, mitigates this
attack by:

-   \(1) Requiring to all users to register (anonymously) to the server.

-   \(2) Not allowing a user that receives an $ESR\_REPLY$ message set to 1
    to query the server anymore.

As a result, a registered user can only perform the attack once and then
```

```
potentially different EBID schemes. It further allows us to improve upon
the format without requiring changes to the federation protocols.

\appendix
Towards Probabilistic Notifications {#sec:proba}
==================================

As described in previous work
[@canetti2020anonymous; @cryptoeprint:2020:399], all proximity-tracking
schemes are vulnerable to the \"one entry\" attack. In this attack, the
adversary has only one entry, corresponding to $User_T$, in her [^21].
When the adversary is notified \"at risk\", she learns that $User_T$ was
diagnosed COVID-positive. The $ROBERT$ scheme, however, mitigates this
attack by:

-   \(1) Requiring to all users to register (anonymously) to the server.

-   \(2) Not allowing a user that receives an $ESR\_REPLY$ message set to 1
    to query the server anymore.

As a result, a registered user can only perform the attack once and then
```

skipping to change at *line 905* | skipping to change at *line 950*

```
question. First, we need to acknowledge that proximity tracing is not
perfect, and that there will be anyway false positives or false
negatives. In this context, is it really problematic to add $5\%$ or
$10\%$ more false positives? Second, the answer to this question may
also depend on what the application is used for. If the App is used to
target users that should get tested, we believe that testing 5 or 10%
more users randomly should be quite acceptable. If the App is used to
notify users to go in quarantine, false positives could be more
problematic\...

A Note on Current PEPP-PT Deployment
====================================

The protocols of the PEPP-PT App and service currently scheduled for
release in Germany diverge from the protocols proposed in this paper.
This is due to the fact the the PEPP-PT initiative originated from a
national project which was the driving force behind the current German
App release. However, efficient and successful containment through
proximity tracing can only happen internationally. As such, the
scheduled release is only a first step and plans to migrate towards this
protocol that supports federation between countries are already in
place.

The major differences between the protocols proposed in this paper and
the current state in Germany, as of today, are:

-   **EBID format**: The current $EBIDs$ issued by the back-end do not
    contain the $ECC$. They also do not contain a $MAC$ or timestamp.

-   **Upload authorization**: The current App and back-end already
    include methods and protocols for upload authorization after
    infection; protocols which are specifically tailored to the German
    health system. Those methods will likely change from country to
```

```
question. First, we need to acknowledge that proximity tracing is not
perfect, and that there will be anyway false positives or false
negatives. In this context, is it really problematic to add $5\%$ or
$10\%$ more false positives? Second, the answer to this question may
also depend on what the application is used for. If the App is used to
target users that should get tested, we believe that testing 5 or 10%
more users randomly should be quite acceptable. If the App is used to
notify users to go in quarantine, false positives could be more
problematic\...
```

```
    country.

The migration path is clear and the systems and protocols will be
harmonized soon.

Server Security Considerations
==============================

This proposal assumes that the back-end server is correctly secured,
implementing the best state-of the art counter-measures and deploying
the required security measures to prevent intrusions [^17].

Its security will be audited, tested and validated by the competent
national bodies. Different measures are already considered like:

-   A secured logging systems (in order to allow regular audits of the
    operations for secured or privacy checks).

-   The use of hardware or at least software security modules for secure
    cryptographic processing, key generation and protection.

Furthermore, an extension of $ROBERT$ that uses a secret sharing scheme
to split the secret keys between the server and the applications is
under consideration for a future version of this document.

[^1]: For simplicity, we consider that $CT$ is fixed and the same for
    all users. In practice, this value could be adapted to each infected
    user by doctors.

[^2]: Federation is also considered for users who are traveling abroad.
    See Section [8](#sec:federation){reference-type="ref"
    reference="sec:federation"}.

[^3]: <https://en.wikipedia.org/wiki/Network_Time_Protocol>

                 skipping to change at line 974
    recommended in the Bluetooth v5.1
    specification [@bluetooth_core_specification5.1 Vol 3, Part C, App.
    A]). We assume that these epochs and rotation periods are
    synchronized.

[^5]: Note that this key can be renewed every few epochs for better
    security. In this case, the server needs to store all the keys that
    were generated during the last $CTK$ epochs, where
    $CTK= 86400*CT/ epoch\_duration\_sec$.

[^6]: Note that the server does not store these pairs.

[^7]: In this paper, we use lowercase for variables that result from a

    parsing operation.

[^8]: Note that since $time_A$ is only 16-bit long, this check is not
    enough to detect replay attacks of *HELLO* messages that were
    generated more than $2^{16}$ seconds (i.e., a bit more than 18
    hours) before $time'_A$. In case $U_A$ is diagnosed positive on
    COVID-19, an additional test is performed by the server (Section
    [7](#sec:status_request){reference-type="ref"
    reference="sec:status_request"}) to further detect these attacks.

[^9]: Other information that could be useful to compute the risk score,
    such as the message's reception power or the user's
    speed/acceleration during the contact, could be added.

[^10]: One solution under study is to consider that the user obtains an
    authorization token from the hospital or the medical office when it
    is diagnosed . The User can then use this authorization token to
    validate its  upload.

[^11]: Since all mobile telecom operators are using NAT, it should be
    studied whether the use of a Mixnet or proxy is really needed.

[^12]: Note that at each epoch $i$, the server cleans up the ${LEE}_A$
    list of each registered $ID_A$ by removing the \"expired\" entries.
    More precisely, all epochs $j$ of ${LEE}_A$ such that
    $(i-j) >= (ct*24*3600/epoch\_duration\_sec)$ are deleted.

[^13]: The queries are sent regularly and at most every $T$ epochs. If a
    user is allowed to perform N queries per day, $T$ is defined as
```

```
Server Security Considerations
==============================

This proposal assumes that the back-end server is correctly secured,
implementing the best state-of the art counter-measures and deploying
the required security measures to prevent intrusions [^22].

Its security will be audited, tested and validated by the competent
national bodies. Different measures are already considered like:

-   A secured logging systems (in order to allow regular audits of the
    operations for secured or privacy checks).

-   The use of hardware or at least software security modules for secure
    cryptographic processing, key generation and protection.

Authenticated Requests
======================

ESR Request is an authenticated request, enabling $App_A$ to make the
server perform an operation on the data associated to user $A$. Other
types of authenticated requests can be defined in a similar manner to
trigger other operations, for example to unregister a user.

Any type of authenticated request is built in the same manner as an
$ESR$ request (see Section [7](#sec:status_request){reference-type="ref"
reference="sec:status_request"}), with the exception that the constant
$c$ used in the MAC computation must be different.

For example, the following values of $c$ could be used:

  Code   Request Type
  ------ ---------------
  1      Hello message
  2      ESR Request
  3      Unregister
  4      DeleteHistory

The server processing part is identical up to step 6 (included). These
steps validate the request. In following steps the operations specific
to the request are implemented.

[^1]: For simplicity, we consider that $CT$ is fixed and the same for
    all users. In practice, this value could be adapted to each infected
    user by doctors.

[^2]: Federation is also considered for users who are traveling abroad.
    See Section [8](#sec:federation){reference-type="ref"
    reference="sec:federation"}.

[^3]: <https://en.wikipedia.org/wiki/Network_Time_Protocol>

                 skipping to change at line 1013
    recommended in the Bluetooth v5.1
    specification [@bluetooth_core_specification5.1 Vol 3, Part C, App.
    A]). We assume that these epochs and rotation periods are
    synchronized.

[^5]: Note that this key can be renewed every few epochs for better
    security. In this case, the server needs to store all the keys that
    were generated during the last $CTK$ epochs, where
    $CTK= 86400*CT/ epoch\_duration\_sec$.

[^6]: See Annex D.1 of NIST FIPS186-4,
    <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>

[^7]: Note that the server does not store these pairs.

[^8]: In this paper, we use lowercase for variables that result from a
    parsing operation.

[^9]: Note that since $time_A$ is only 16-bit long, this check is not
    enough to detect replay attacks of *HELLO* messages that were
    generated more than $2^{16}$ seconds (i.e., a bit more than 18
    hours) before $time'_A$. In case $U_A$ is diagnosed positive on
    COVID-19, an additional test is performed by the server (Section
    [7](#sec:status_request){reference-type="ref"
    reference="sec:status_request"}) to further detect these attacks.

[^10]: Other information that could be useful to compute the risk score,
    such as the message's reception power or the user's
    speed/acceleration during the contact, could be added.

[^11]: One solution under study is to consider that the user obtains an
    authorization token from the hospital or the medical office when it
    is diagnosed . The User can then use this authorization token to
    validate its  upload.

[^12]: Since all mobile telecom operators are using NAT, it should be
    studied whether the use of a Mixnet or proxy is really needed.

[^13]: In the case where $K_S$ renewal is implemented, $K_S$ is selected
    based on $Time_A$. In case the procedure fails, and if $Time_A$ is
    close to a boundary between the validity period of two $K_S$ values,
    the step 4 is performed again with the previous/next value of $K_S$
    [\[fnlabel\]]{#fnlabel label="fnlabel"}.

[^14]: In the case where two $K_S$ values are possible (see footnote
    [\[fnlabel\]](#fnlabel){reference-type="ref" reference="fnlabel"})
    and $h_A$ is rejected, the other key should be tested in Step 4.

[^15]: In the case where two $K_S$ values are possible (see footnote
    [\[fnlabel\]](#fnlabel){reference-type="ref" reference="fnlabel"})
    and the following test fails, goto step 4 to test the other key
    value.

[^16]: In the case where two $K_S$ values are possible (see footnote
    [\[fnlabel\]](#fnlabel){reference-type="ref" reference="fnlabel"})
    and $mac_A$ is incorrect, goto step 4 to test the other key value.

[^17]: Note that at each epoch $i$, the server cleans up the ${LEE}_A$
    list of each registered $ID_A$ by removing the \"expired\" entries.
    More precisely, all epochs $j$ of ${LEE}_A$ such that
    $(i-j) >= (ct*24*3600/epoch\_duration\_sec)$ are deleted.

[^18]: The queries are sent regularly and at most every $T$ epochs. If a
    user is allowed to perform N queries per day, $T$ is defined as
```

$T= 86400/(N * epoch\_duration\_sec)$.	$T= 86400/(N * epoch\_duration\_sec)$.
[^14]: <https://www.pepp-pt.org>	[^19]: <https://www.pepp-pt.org>
[^15]: Actually the adopting countries should only agree to use the 8 first bits of the *HELLO* message as the \"Encrypted Country Code\" that is encrypted using the following 64 bits as an IV of a stream cipher, as detailed in Section [4](#sec:EBID){reference-type="ref" reference="sec:EBID"}.	[^20]: Actually the adopting countries should only agree to use the 8 first bits of the *HELLO* message as the \"Encrypted Country Code\" that is encrypted using the following 64 bits as an IV of a stream cipher, as detailed in Section [4](#sec:EBID){reference-type="ref" reference="sec:EBID"}.
[^16]: This attack can easily be achieved by keeping the Bluetooth interface off, switching it on when the adversary is next her victim and then switching it off again.	[^21]: This attack can easily be achieved by keeping the Bluetooth interface off, switching it on when the adversary is next her victim and then switching it off again.
[^17]: See for example: <https://www.ssi.gouv.fr/entreprise/bonnes-pratiques/poste-de-travail-et-serveurs/>	[^22]: See for example: <https://www.ssi.gouv.fr/entreprise/bonnes-pratiques/poste-de-travail-et-serveurs/>

End of changes. 56 change blocks.

106 lines changed or deleted	*168 lines changed or added*

This html diff was produced by rfcdiff 1.46. The latest version is available from http://tools.ietf.org/tools/rfcdiff/