*Detailed extension to Decentralized Privacy-Preserving Proximity Tracing (DP3T) protocol*

# Quarantine Violation Police Alert

System capable of reporting the violation of a positive patient's quarantine to the competent authorities

*Cossentino Edoardo – 06227 00793*
*Di Prisco Giovanni – 06227 00818*
*Manna Fernando – 06227 00749*
*Schettini Marco – 06227 00825*

## About this document

This document proposes a system to prevent violations of the quarantine during the SARS-CoV-2 pandemic. A citizen (infected or not) can be quarantined; this system helps the police authority in monitoring them during the isolation.

The system is based on the DP3T protocol and uses its infrastructure. Our idea adds some new information to DP3T in order to achieve the new goals trying not to violate the security and the privacy of DP3T.

The goal is preventing the violation quarantine without revealing the identity of the person (unless it is necessary to punish an illegal action). The aim is achieved adding to the data that the app backend of DP3T knows normally, some extra anonymous data. The only personal information is owned by a backend server belonging to a police authority and it is used only to punish a citizen that is not respecting the rules.

A special *BLE Beacon* is installed in the house of the quarantined person (this detail will be discussed further) and it is established a continuous exchange of messages between the app backend and what we will call *SmartBeacon*. The smartphone is like a proxy in this context and needs to be near to the SmartBeacon in order to have a successful communication. The messages exchanged between the SmartBeacon and the smartphone and then between the smartphone and the app backend does not contain personal info. If the exchange is successful it is assumed that the person is respecting the quarantine; if something goes wrong, the police authority is informed and can decide what measure should be taken.

The info used in the system are not owned by a unique actor in order to prevent an abuse of the data. The app backend only knows anonymous data (the identifier of the SmartBeacon and a seed associated to a person) and cannot derive some other information, especially sensitive data. The police authority backend server owns some personal info of the person that already knows in a normal situation (names and surnames, home addresses etc.).

# Contents

# Figures

*On the front cover: photo by Markus Spiske from Pexels*

# Project Proposal

In this chapter we are going to explain our proposal of project and to provide several information about the system to well understand its functioning. In several cases some topics, introduced in this chapter, are discussed in more detail in the following sections.

## 1. ASSUMPTIONS

There are some assumptions that need to be mentioned to understand how the system can work properly. These rules must be respected and are the basis of the project:

- It is assumed that everyone has a smartphone and an Internet connection.
- The people are forced to download the app and follow the instructions of the authorities.
- When a person is ready to be quarantined it is accompanied home by the authorities.
- When the authorities are at the patient home, they must be able to test Internet connection and perform the installation of the SmartBeacon.
- The authorities must check that the installation of the SmartBeacon and its initialization with the smartphone of the person are successful.
- It is assumed that the installation of the SmartBeacon is unbreakable (in this case the violation of seals physical affixed to the Beacon should be evident) and its routine cannot be stopped by anyone except the authorities.
- It is assumed that the person cannot fake one of the operations carried out in presence of the authorities.
- The health authority and the police authority are trusted in preserving data.
- The servers involved in the system are considered safe.
- Consider that this project is an extension of the DP3T protocol and therefore suffers from the same shortcomings. For example, the eventuality in which the patient owns another smartphone or leaves his home leaving his smartphone here will not be managed. In such cases we rely on external entities, authorities able to perform periodic reliability checks on the subject in question. In the section *Mitigations for Quarantine Violation* we will explain some opportunities to fix this problem but they will not somewhere explained else because it would be beyond the goals of this project.

## 2. GOALS

The goal of the system is to develop a protocol based on DP3T to help the police authority monitoring the observation of the quarantine by the people, without violate their privacy. Usually these operations require a big amount of resources and a coordination between different actors.

Without a report by a citizen or a great effort by the authorities it would be impossible controlling every person quarantined. The authorities cannot supervise all the streets or knock on the bell of all those in quarantine; at the same time in this difficult period maybe a citizen has other problems than reporting people that are violating their isolations.

With our proposal we take advantage of the protocol DP3T and the fact that everyone has a smartphone. The actors involved only do a few more things than that their normal routine during this emergency. Adding some information in the protocol DP3T and using few means we can supply a big help to the insiders and maybe they can spend their time in most useful operations.

We don’t add information that violates the standard of privacy and security of DP3T, except for the fact that the police authority need to know who to sanction in case of an illegal action (we will discuss this topic in the next chapters).

Another purpose of the project is to calm down all the people letting them know that the quarantined persons cannot walk around freely; in this way it is simpler returning to our normal lives.

The entire installation will be dismantled in 14 days, like DP3T, in case of end of pandemic.

# 3. REQUIREMENTS

There are some physical requirements needed to ensure the correct working of the whole system:

- The smartphone of the quarantined person.
- The SmartBeacon installed in the house of the quarantined person.
- A sufficient expenditure of resources by the police authority to accompany home people who must begin the quarantine and to perform the required operations.
- A sufficient expenditure of resources by the health authority to manage all the data and the secret keys of the people.
- A secure app backend server managed by the producers of the app.
- A secure server owned by the police authority to manage the personal info of the people quarantined.

## 4. ACTORS

There are several actors involved in the system, each of them has a precise function.

The main actor is the **patient**, i.e. the quarantined person. He has the duty to follow the instructions of the police and health authorities and must stay home during the quarantine to avoid sanctions.

The **SmartBeacon** installed in the house of the quarantined person broadcast several data during his functioning. Data are received by the smartphone and can be interpreted only by the app backend.

The **smartphone** of the quarantined person must upload to the app backend the data received from the SmartBeacon during the quarantine to communicate that the person is following the rule.

The **health authority** is the authority supposed to be present in the normal functioning of DP3T. Its routine is the same described in DP3T White Paper except for one operation: in our system, in the moment in which a person is quarantined, the health authority must upload to the app backend several extra-data to make possible monitoring the person.

The **app backend** is a server that has a list of the seeds associated to the quarantined citizens. With those seeds it cannot recover the identity of the persons according to DP3T. The server checks the data received from the smartphones of the quarantined persons to be sure that they are at home. In case of error it will send an alert to the authority backend server.

The **police authority** accompanies the quarantined person at home and performs the activities listed in the assumptions. Moreover, the police authority upload to the its backend server some data linked to the SmartBeacon and the quarantined person that are needed to act in case of a violation. In case of alert reported to the authority backend server has to decide what is the right strategy to act.

The **police authority backend** stores the data necessary only to associate every SmartBeacon to the home in which it has been installed. It has the duty to send a message to the app backend to start the working of the system and then waits for an alert. If the app backend server sends an alert, thanks to the police authority backend server the authorities could be warned.

# Confidentiality, Integrity and Adversaries

In this chapter are described the properties of confidentiality and integrity data that the system wants to achieve.

## 1. CONFIDENTIALITY OF DATA

### A. EPHEMERALITY OF THE DATA

The first goal we want to achieve is the *protection of the users' privacy*. The most important feature of the system (derived from the structure of DP3T) is the ephemerality of the data. It's known that in DP3T it is very difficult to infer information about the identities of the people from the messages exchanged between the smartphones and between each smartphone and the server because all the data contained in these messages cannot be associated to the identity of the real users.

The extra data continuously exchanged that this extension introduced concern the only SmartBeacon and an adversary cannot infer anything about the location of the device. The only actors that have sensitive information are the two authorities that we can consider trusted, as already said in the assumptions.

### B. DISPLACEMENT OF THE DATA

Even if some of the actors involved are trusted, we have decided to not concentrate the data in a single place.

**App backend**. As in DP3T we want that the app backend does not own anything about the identity of the users and their location. The app backend stores only information about the ephemeral codes sent by the smartphones and the IDs of the SmartBeacon. With these data, even if the app backend suffers an attack or, in general, can no longer be considered reliable, it cannot associate the entries of its database to the smartphones or their owners and above all it doesn't know about people home address.

**Police authority backend**. Police authority is trusted and is the unique actor that own personal info. It stores in its database the IDs of the SmartBeacons and some personal info of the quarantined person that the police authority already knows, because it is the norm when a person is quarantined an officer goes to the subject's home to notify him of the quarantine.

Please note this authority cannot associate the information stored to the smartphones of the quarantined persons; in this way a national security agency cannot use this data to monitor citizens also in other circumstances because in there is no association between the physical person and his smartphone.

## 2. INTEGRITY OF DATA

### A. COMMUNICATION BETWEEN SMARTPHONE AND SMARTBEACON

The SmartBeacon periodically broadcasts messages; if and when the smartphone receives them, it appends to these messages other data and sends the complete message to the app backend; if the messages received by the app backend are correct (this topic will be discussed in the next chapter), we can say that the person is at home. The messages sent from the SmartBeacon to the smartphone are encrypted using a symmetric key encryption. The actors that own the secret key are the SmartBeacon and the app backend.

We want that these messages to be authentic and intact. To be sure that anyone (included the smartphone of the quarantined person) cannot modify the messages sent by the SmartBeacon we implement an encryption scheme that achieves **security against CCA**. In this way we ensure the authenticity and the integrity of the messages. The properties of this standard of security will be discussed in the chapter *Confidentiality and Integrity Analysis.*

### B. COMMUNICATION BETWEEN SMARTPHONE AND APP BACKEND AND BETWEEN APP BACKEND AND POLICE AUTHORITY BACKEND.

The exchange of messages between the smartphone and the app backend and between the two servers uses the Internet. We want that no one can sniff these messages or modify them. The communication that takes place using the web is considered safe and privacy-preserving thanks to the use of an **SSL/TLS protocol**. The properties of this standard of security will be discussed in the chapter *Confidentiality and Integrity Analysis*.

# 3. ADVERSARIES

In this chapter we describe the potential attackers that may benefit from the data or simply are interested to violate the operation of the system. For each of these attackers we describe their capabilities and what kind of risk they pose for the system.

## A. THE PATIENT

The quarantined subject can be itself an adversary (maybe the more interested in breaking the system). He could try to send fake messages to the app backend to communicate that it is at home even if it were not so.

## B. BLUETOOTH ADVERSARIES

It is universally recognized that Bluetooth communication, especially in broadcast, suffers from some shortcomings in terms of security, so it is in this situation that we expect the greatest appeal.

- There could be a Bluetooth adversary that performs an attack man in the middle in the continue communication between the smartphone and the SmartBeacon.
- There could be a Bluetooth adversary that can try to replace the true SmartBeacon establishing a connection or broadcasting packets very similar to the packets which come from an original SmartBeacon.
- There could be a Bluetooth adversary that can try to perform a DoS attack to the smartphone sending to it a lot of fake messages that it has to manage.

The patient could belong to one among these categories.

## C. NETWORK ADVERSARIES

Although the progress in cybersecurity has gone above all in the direction of securing the Internet, it is good practice to emphasize that even in this case there are some threats that may be lurking

- There could be an adversary that can try to fake the SSL certificate of the app backend and replace it in the communication between the smartphone and the app backend.
- There could be an adversary that can try to break the security of the SSL/TLS communication between the smartphone and the app backend, or between the app backend and the police authority server to steal sensible data.

The patient could belong to one among these categories.

Moreover, malicious server should be included in this group. For example, against violation or misbehavior of the app backend we can take countermeasures, but it could still infer some information based on the IP address of the smartphones.

## D. TARGETED THREATS TO DEVICES

There could be an adversary that can try to break the security of the smartphones or the servers involved in the system to take control of them or steal sensible data. Among these, a prominent adversary could be a quarantined patient who violates the source code of the application installed on the smartphone or owns a modified version of the provided operative system. In this case we could face to problems of secure coding and software stability that overshoot our purpose.

## E. TARGETED THREATS TO PEOPLE AND SOCIETY

An adversary not so closely linked to the mechanism of this extension of the DP3T protocol should be everyone is interested to eavesdrop data and to collecting them to malicious statistical purposes. Among these possible adversaries we could see epidemiologist interested to exploit the information about the number and the precise location of probably infected people or state-level adversaries interested to sensitive individuals.

# System Design

The presentation of the system design is divided into several phases that follow the timeline of events. The steps will be highlighted from the moment the patient's quarantine status is asserted.

## 1. SETUP

The first step provides that a health authority has judged a patient as infected or has simply sentenced it should not be in contact with other people and therefore it is necessary that it does not leave his home. When the health authority has arranged for the patient to be quarantined, it is already provided by the DP3T protocol that the health authority uploads the tuple ($SK_{T\_PAST}$, T_PAST) to the app backend that can identify the smartphone. The $SK_{T\_PAST}$ is a key, which we will call $SK_{DP3T\_T\_PAST}$, associated with the patient's smartphone at the time of the presumed infection, this time is identified by T_PAST. See the DP3T White Paper for further information. The tuple provided by the DP3T protocol is expanded into ($SK_{DP3T\_T\_PAST}$, T_PAST, $ID_{BEACON}$, $SK_{BEACON}$), a quadruple of data that is inserted into the app backend database.

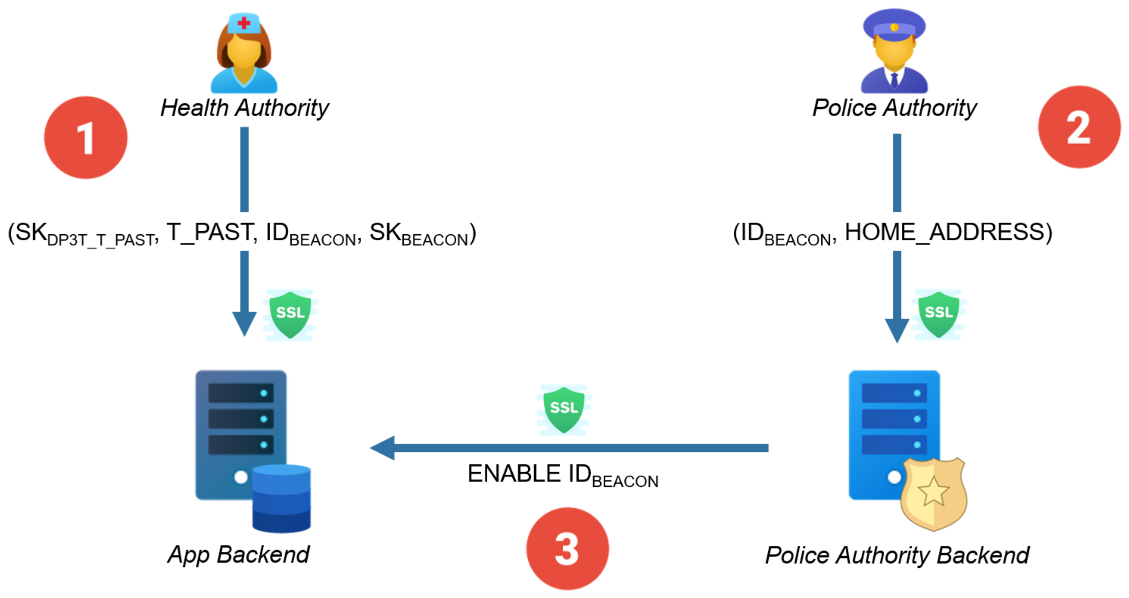


*Figure 1 : Setup*

The two fields with which the tuple provided by the DP3T protocol is expanded are explained below.

**$ID_{beacon}$.** A unique identifier within the implemented mechanism that distinguishes the beacon. It is not possible that a SmartBeacon of a quarantined patient has the $ID_{beacon}$ of another SmartBeacon associated with another patient. This identifier can be created ad-hoc or it can correspond to the Bluetooth UUID of the device.

**$SK_{beacon}$.** It is the secret key that the app backend will get and which, by construction, the SmartBeacon already owns. This key will be used by the SmartBeacon to encrypt messages that we will show later and by the app backend to decrypt them.

When the patient is taken to his home or he is known to be at his home ready to start the quarantine, a member of the police authority provides to go to the aforementioned home to certify the presence of the subject and install the SmartBeacon that will be then fixed to the wall, for example, and seals will be affixed.

The authority in question makes sure, before installing the device, that the Beacon signal is reachable throughout the home and that a stable Internet connection is present in these places of the home. He then proceeds to install the SmartBeacon and switch it on by synchronizing his time (the usefulness of this last step will be illustrated later). This initial configuration of the Beacon must be made inviolable, for example with the aid of a USB or RS-232 terminal device.

The last operation that the police authority must perform, before leaving the home, is to upload the SmartBeacon ID associated with the patient's home address on the spot. The upload is carried out on the police authority backend which keeps this information in its database.

## 2. PHYSICAL PRESENCE EVIDENCE

The method chosen so that the patient's smartphone can testify to being near the SmartBeacon is the forwarding of a message that the SmartBeacon composes and encrypts. The message that the Beacon composes contains a TIMESTAMP, or date and time of the beacon, in an uncommon format, and it is encrypted with the $SK_{BEACON}$, the key previously introduced and which the SmartBeacon owns. The encrypted message is then broadcast via Bluetooth. The patient's smartphone (if it is near the beacon) will capture the package.

Not all Bluetooth packets will be considered good for the purpose, an application field inserted within the payload of the advertising message sent by the Beacon will be exploited so that the packet may be distinguished by other transmissions that the patient could perform at the same time via Bluetooth. This avoids monopolizing the Bluetooth connection of the smartphone.

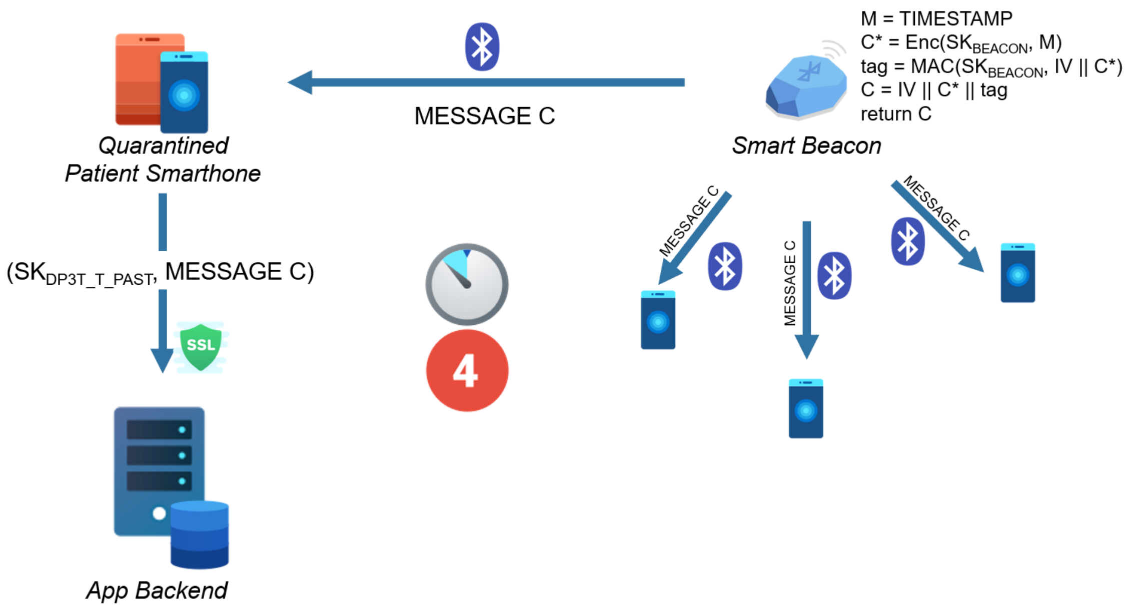


*Figure 2 : Persistence evidence*

The smartphone will then enter the payload of the package received via Bluetooth (then get the encrypted message, called MESSAGE C) into the tuple ($SK_{DP3T\_T\_PAST}$, MESSAGE C).

This operation requires the app to remember $SK_{DP3T\_T\_PAST}$, the $SK_{DP3T}$ key that was owned at the time of the alleged infection and this possession by the smartphone is already supported by the DP3T protocol.

The smartphone then proceeds to send the tuple to the app backend via Internet. The communication takes place in protected mode through an encrypted HTTPS connection after verification of the SSL certificate of the app backend.

The capture of Message C and the forwarding of the latter accompanied by $SK_{DP3T\_T\_PAST}$ takes place periodically, over a very narrow time interval such as 2 minutes. At this time, it seems unlikely that the patient may have moved significantly away from his home where the quarantine was placed.

## 3. BACKEND COMPUTING

The app backend must manage a multitude of messages from all patients in quarantine on the territory it covers, in addition to normal service communications and other queries that do not concern the mechanism we are describing.

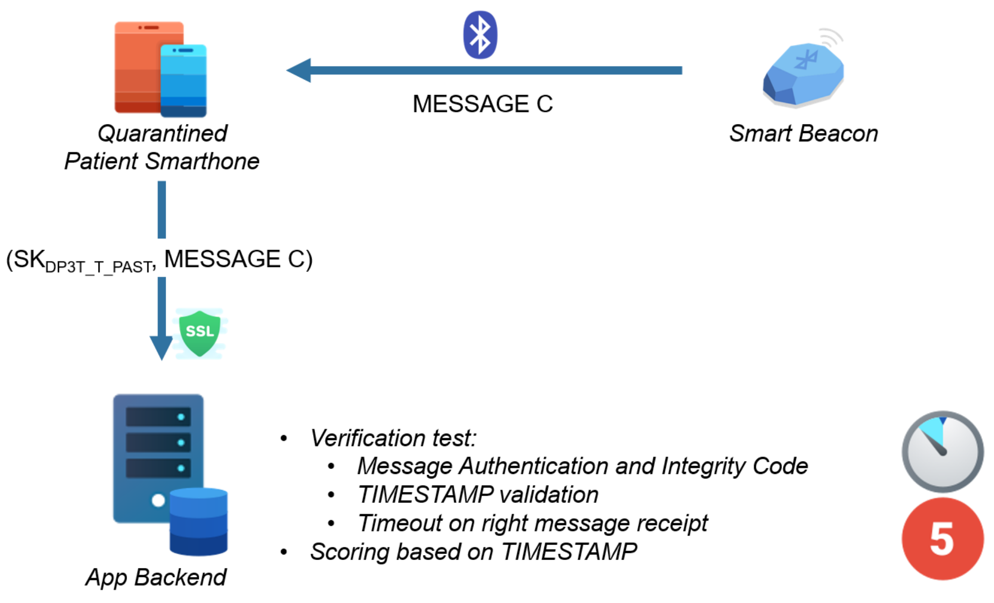
Among the many messages it receives, it can distinguish those associated with our game thanks to the typical identifiers of the application layer of the TCP/IP stack.

So, the messages, well formed, which it expects to receive from the smartphones of the patients under quarantine provides for a tuple ($SK_{DP3T\_T\_PAST}$, MESSAGE C). The app backend searches within its database the information relating to $SK_{DP3T\_T\_PAST}$ of the patient in question to obtain the $SK_{BEACON}$ coding key previously entered by the health authority. With the use of this key, the backend can decrypt the MESSAGE C received. Then a method of verification of this message is triggered so that it can be proved that the patient has been close to the Beacon and therefore has been able to forward the right MESSAGE C.

The smartphone is expected to have been able to send multiple tuples ($SK_{DP3T\_T\_PAST}$, MESSAGE C) in which the MESSAGE C can be related to a Beacon intercepted but not associated to the patient's home (for example belonging to a nearby home where quarantine was also imposed) or it can be composed by an opponent who sends messages with the same application field of the SmartBeacon advertising packet.

If, at the same time, the app backend receives multiple tuples with the same $SK_{DP3T\_T\_PAST}$, it is sufficient that at least one of these tuples passes the verification test of the MESSAGE C. Indeed, there could be some Bluetooth adversaries which emulate, voluntarily or not, the SmartBeacon the same packet application field and send their messages. If among the received messages in the app backend there is the right message, we have no reason to consider the quarantine as violated.

The Backend App must receive at least one tuple ($SK_{DP3T\_T\_PAST}$, MESSAGE C) that passes the verification test in a fixed period which can be set at a slightly longer time than the period with which the smartphone periodically sends the aforementioned tuple. If the correct tuple relating to the smartphone, identified thanks to $SK_{DP3T\_T\_PAST}$, is not received, in the fixed period, with a certain tolerance on time and admittance on the lost packets, the quarantine is considered violated.



MESSAGE C
Quarantined Patient Smarthone
Smart Beacon
(SK_DP3T_T_PAST, MESSAGE C)
SSL
App Backend
• Verification test:
• Message Authentication and Integrity Code
• TIMESTAMP validation
• Timeout on right message receipt
• Scoring based on TIMESTAMP
5

*Figure 3 : App backend computing*

### Verification Test

- The MESSAGE C must pass the integrity verification provided by the MAC system. So, the C MESSAGE is divided into a tag and a MESSAGE C*. The app MAC function checks the tag and, if the authentication is not successful, the tuple test fails.
- The MESSAGE C* is decrypted and a validity check is carried out on the decrypted message which we remember to be a TIMESTAMP. The format of this message must therefore be correct. If the decrypted message does not correspond to the expected format, the test fails.

### Scoring

The decrypted TIMESTAMP can be used by the app backend to calculate a score regarding the reliability of the connection with the smartphone: if the time indicated by the TIMESTAMP does not correspond, considering a defined tolerance, to the time of receipt of the package then we may consider any issues connecting the smartphone to the network.

## 4. ALERT FORWARDING

If the verification test has returned a negative result above all the received tuples and therefore the app backend has sentenced the quarantine as violated, then a message is sent to the police authority backend that can be translated by the latter as an ALERT MESSAGE. This message is accompanied by the $ID_{BEACON}$ so that the police authority backend can retrieve information relating to the patient's home address and carry out an on-the-spot check.

The communication takes place in protected mode through an encrypted HTTPS connection after verification of the SSL certificate of the police authority backend.

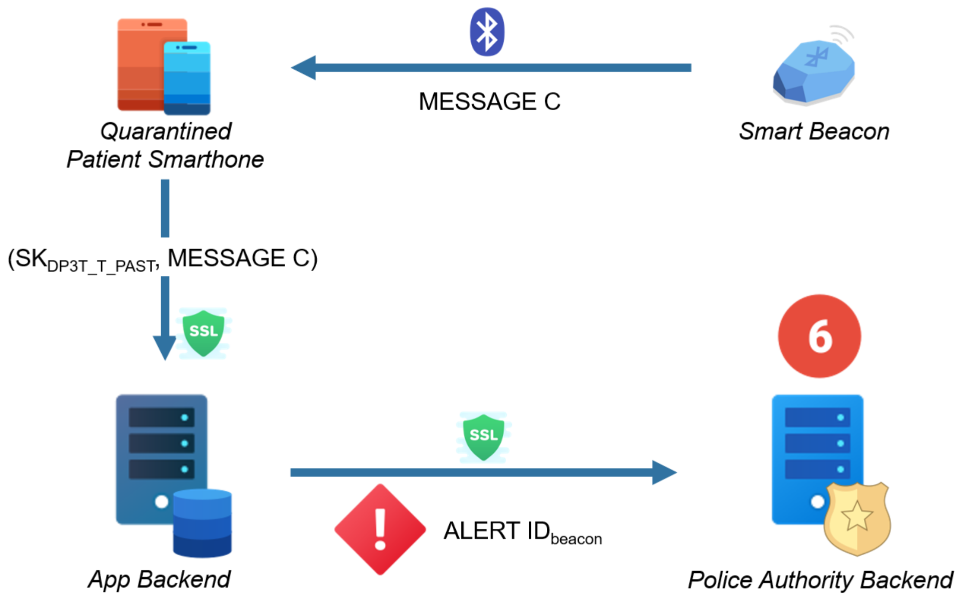


*Figure 4 : Police alert*

# Confidentiality and Integrity Analysis

Following the same representation of the system design, even the analysis of the property of confidentiality and integrity of data will be divided in steps. In each step we will analyze security and privacy.

## 1. SETUP

After a patient has sentenced it should be in quarantine by the health authority, the actions described in the paragraph Setup in chapter *System Design* are put in place.

We remind you that a tuple of data consisting of ($SK_{DP3T\_T\_PAST}$, T_PAST, $ID_{BEACON}$, $SK_{BEACON}$) is uploaded to the app backend database by the health authority while the tuple consisting of ($ID_{BEACON}$, HOME_ADDRESS) is uploaded to the police authority backend server by the police authority. Then an ENABLE message is sent from the police authority backend server to the app backend to start the procedure.

The use of the tuple ($SK_{T\_PAST}$, T_PAST) is already included in the DP3T protocol. The introduction of $ID_{BEACON}$ and $SK_{BEACON}$ does not violate the properties enunciated in the chapter *Confidentiality, Integrity and Adversaries*. From the field $ID_{BEACON}$, the app backend cannot retrieve any information about the location of the SmartBeacon; $SK_{BEACON}$ is the secret key used in the encryption scheme that we are going to explain.

The tuple ($ID_{BEACON}$, HOME_ADDRESS) does not violate the enunciated properties because it is assumed that the police security authority already knows some personal information about the quarantined subject; with this tuple the police security authority cannot make an association with the smartphone of the subject mentioned above.

Concerning security, the communications use HTTPS with TLS 1.2 leveraging one of the following ciphers, which are considered particularly trusted:

- ECDHE-RSA-AES256-GCM-SHA384
- ECDHE-RSA-AES128-GCM-SHA256
- TLS_CHACHA20_POLY1305_SHA256

The cipher suites above have been chosen studying the most used suites by this kind of application and ensure the security of the communication with TLS 1.2.

Let us analyze them in detail:

A. ECDHE-RSA-AES256-GCM-SHA384:

- ***Elliptic-curve Diffie-Hellman Ephemeral (ECDHE)***
  This kind of protocol is used when two actors need to establish a secret key using a public channel (consequently an eavesdropper may read the messages exchanged). The algorithm is a variant of the classic Diffie Hellman protocol, but it is based on the algebraic structure of elliptic curves over finite fields, rather than on Galois fields. The ephemerality in this protocol is because the keys exchanged are temporary and not static.

- ***Rivest-Shamir-Adleman (RSA)***
  RSA is an asymmetric key encryption scheme. There is a public key that is known to everyone (used to encrypt messages) and a private key that must be secret (used to decrypt messages). Also knowing the public key, it is difficult to arise the secret key (using keys of reasonable size). In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, this is the *factoring problem*.

- ***Advanced Encryption Standard 256 (AES256)***
  AES is a widely used practically specification for the encryption of data using the technique of block cipher. 256 stays for a key of 256 bits. In AES, the block size is fixed to 128 bits and the keys may be of 128, 192 or 256 bits. The key size specifies the number of *rounds* of the encryption.

- ***Galois/Counter Mode (GCM)***
  GCM is a widely used operation mode for symmetric key cryptographic block chipper, thanks to its good performance with a reduced use of hardware resources. The operation is an authenticated encryption algorithm designed to provide both data authenticity (integrity) and confidentiality, defined for block ciphers with a block size of 128 bits.

- ***Secure Hash Algorithm 384 (SHA384)***
  SHA indicates a family of several cryptographic hashing functions. SHA produces a fixed-size digest with variable size messages in input. The security is in the fact that this operation is not reversible. 384 stays for the size of the outputted digest (384 bits).

B. ECDHE-RSA-AES256-GCM-SHA256:

The only difference with ECDHE-RSA-AES256-GCM-SHA384 is the size of the digest outputted by SHA (in this case is 256 bits rather than 384).

C. TLS_CHACHA20_POLY1305_SHA256:

- ***Transport Layer Security (TLS)***
  TLS is the cryptographic protocol used to supply secure communication over a network channel.

- ***ChaCha20***
  ChaCha20 is stream cipher designed for high performance in software implementations. It is a refinement of Salsa20 stream cipher.

- ***Poly1305***
  Poly1305 is a message authentication code (MAC) widely used with AES, Salsa20 or ChaCha20.

- ***Secure Hash Algorithm 256 (SHA256)***
  SHA256 as described above.

All the messages over a network channel using the above techniques are considered secure.

In the setup phase the police security authority must also install the beacon. This step is considered secure due to the use of special sealing techniques by the authority.

## 2. PHYSICAL PRESENCE EVIDENCE

In order to prove the respect of the quarantine, the quarantined subject's smartphone has to forward to the app backend the tuple ($SK_{DP3T\_T\_PAST}$, MESSAGE C) at a fixed interval of time, as described in the paragraph "Physical presence evidence" in the chapter *System Design*.

The MESSAGE C is encrypted by the SmartBeacon using a symmetric key encryption scheme that achieves security against a CCA (Chosen-Ciphertext Attack) adversary. A CCA adversary can have access to an oracle and ask to encrypt any message he likes (except the challenge message); security against CCA adversaries is obtained combing an encryption scheme secure against CPA (Chosen-Plaintext Attack) adversaries and a MAC (Message authentication code) scheme. The resulting ciphertext is a combination of an encrypted message and a tag. The CCA adversary, despite his power, cannot create an encrypted message accompanied with a convincing tag. Another important feature of this scheme is his invulnerability against the malleability of the ciphertexts.

The plaintext message, that we remind you it is a TIMESTAMP (e.g. *2020-06-05|11:59:59|UTC+1*), is padded with **PKCS#7** and then encrypted with **AES** algorithm, in Cipher Block Chaining (**CBC**) encryption mode, with $SK_{BEACON}$ obtaining the ciphertext.

$$C^* = Enc(SK_{BEACON}, M)$$

So, the scheme provides to generate the **Hash-Based Message Authentication Code** with the key $SK_{BEACON}$.

$$tag = HMAC(SK_{BEACON}, IV \,||\, C^*)$$

A concatenation of the necessary data is performed to obtain the complete token which the SmartBeacon sends in broadcast. We call this token **MESSAGE C**:

$$IV \parallel C^* \parallel tag$$

In this message we can distinguish the following fields:

- *IV*: 128-bit Initialization Vector used in AES encryption and decryption of the ciphertext; the IV must be chosen uniquely for every token, with a high-quality source of entropy, random selection will do this with high probability.
- *C**, the original input message, padded and encrypted.
- *tag*, 256-bit SHA256 HMAC of the concatenation of IV and C*

The smartphone can recognize the messages broadcasted by the SmartBeacon thanks to an application field in the advertising message of the Bluetooth packet. If an adversary could emulate the structure of the packets broadcasted by the SmartBeacon, he could try to substitute the SmartBeacon in the communication. This adversary could only send a replication of an intercepted message because, due to the encryption scheme mentioned above, he could not create another ciphertext with a correct tag.

The operation of discarding these packets is demanded to the app backend and it will be described in the next section.

The smartphone sends his messages to the app backend using HTTPS (TLS 1.2) with the same parameters described in the previous paragraph.

Like all the others devices using Bluetooth Low Energy, this system is susceptible to an adversary that try to avoid the reception of the messages by the smartphone taking advantage of specific techniques aimed at disrupt the electromagnetic transmissions, a practice known by the name of *jamming*.

## 3. BACKEND COMPUTING

After that the messages sent by the several smartphones (corresponding to as many patients) are received, the app backend performs the operations mentioned in the paragraph "Backend computing" in chapter *System Design*. The communications between the app backend and the smartphones are considered safe due to the use of HTTPS with TLS 1.2.

The app backend receives the various MESSAGE C in as many tuples that we remind you are structured in the format ($SK_{DP3T\_T\_PAST}$, MESSAGE C). According to the procedure described in the previous chapter, for each tuple, the backend tries to perform the decryption of the MESSAGE C sealing off the initialization vector IV, the ciphertext C* and the tag. So, it tests the validity of the message authentication code and goes ahead with the decryption of C*. Then the backend provides to validate the plaintext that should match the format of the message as TIMESTAMP.

A doubt that might arise concerns that it is trivial for an adversary to infer the format (it shouldn't be a secret) and content of the plaintext (the TIMESTAMP represents the date and time in which the message was sent). It is true but not enough to violate the scheme because the use of a message authentication code permits to withstand Chosen Ciphertext Attack and so Chosen Plaintext Attack.

As the backend is responsible to reject the packets that an adversary can create and inject into the communication, it must reject also the packets the adversary sniffs in the communication between the smartphone and the backend and sends to the smartphone through the Bluetooth, emulating a SmartBeacon (maybe in order to permit the patient to leave his home with his smartphone and a mobile fake SmartBeacon) as a variant of the replay attack. In this case the TIMESTAMP contained in the plaintext corresponds to a too old time period. So, the app backend must check this TIMESTAMP with its system clock (considering a tolerance)[*]. If it were the original SmartBeacon that

[*] A similar mechanism can be implemented directly on the smartphone which is not able to decrypt the ciphertext and so to compare the TIMESTAMP: the smartphone could perform the hashing on the MESSAGE C received via Bluetooth and store this hash code in its memory; if the hash code of a new message is already contained in the smartphone, this message is rejected. The use of the hash function instead of the storing of the entire message is useful to save memory and to save time during the compare between old messages and new ones. We chose not

sent the packets with a wrong TIMESTAMP, because of a malfunction, it would still be necessary to notify the authorities for a reconfiguration of the Beacon so the provided ALERT to the police authority backend cannot be considered a fake alert.

The app backend could be susceptible to an only direct attack in which an adversary who exploit a database breach or takes the full control of the app backend itself, violating the security systems of the server. Even in this case the adversary could not infer any information about the identity of the users from the only data stored in the app backend. In the assumptions we assume that the servers involved in the system are secure, any vulnerability in them are beyond the responsibility of this project.

## 4. ALERT FORWARDING

If the verification test performed by the app backend fails, the app backend sends an **ALERT** message to the security authority backend, as described in the paragraph "Alert forwarding" in the chapter *System Design*. The alert is sent over a secure channel using HTTPS with TLS 1.2 and the parameters described in the previous sections.

Along the lines of the app backend server violation, the police authority backend servers are vulnerable to the same type of direct attacks. As said before any vulnerability in the servers are beyond the responsibility of this project. In case of an attacker that takes full control it would be able to have access to a list of personal information and $\text{ID}_{\text{BEACON}}$ associated to them; with only these data it would be impossible to retrieve any information about the smartphone of the users but for a good adversary it's trivial to infer that all these data are related to quarantined patients, adding this information to the already sensitive leaked data.

## 5. MITIGATIONS FOR QUARANTINE VIOLATION

It is reasonable to expect that a non-compliant patient will adopt behaviors that are not permitted by the medical protocol, and therefore violate the imposed quarantine regime, simply leaving his smartphone at home in proximity to the SmartBeacon. In fact, there is

to adopt this solution which exposes the smartphone to a DoS attack which must be managed or at least mitigated.

nothing to prevent the patient from having a second Smartphone, or from using that of a friend or family member.

Considering this, strategies to mitigate the phenomenon are necessary; but before proceeding with the presentation of some solutions, it is necessary to clarify some aspects of fundamental importance.

Although the entire system has in the basic requirements the availability of a smartphone by the patient, nothing is required in terms of advanced features of the device, and it is not possible to claim it, since inevitably not everyone will have a smartphone with fingerprint detection digital, face recognition, or voice recognition.

It comes by itself that some of the solutions we will see will not be applicable in all cases, but will be provided in a scalable way, based on the characteristics of the device and the patient's preference.

Otherwise, we will have the opportunity to use solutions without advanced requirements, which remain a valid alternative to limit the number of offenders although they do not have the same efficiency compared to these mitigations.

Ultimately, however, it will be impossible to completely cancel the phenomenon, regardless of the type of solution used, since even for the most advanced ones, the patient could benefit from external help. Therefore, remaining within the limits of privacy, with the collaboration with the patient, the goal will be to partially limit the number of offenders in the quarantine.

### Fingerprint recognition

For patients who will have a smartphone with a fingerprint detector, there will be the possibility to confirm their proximity status to the device, through simple daily tasks, in which the patient will be invited to submit their fingerprint.

### Face recognition

In the case of devices designed for facial recognition (less common than the impression), the patient may eventually choose to confirm his status using this alternative, leaving the rest of the process unchanged.

### Speaker recognition

This type of recognition could not be so easy to implement not only because of the quality of the device required but also for the high *recall* of the results. The algorithm could submit simple dynamic linguistic tasks (in mother tongue), in order to avoid the use of recorded voices. This solution is difficult to apply, so we will limit ourselves to considering it as a possible future development. On the other hand, the possibility of elementary tasks, such as spelling the name or surname, is not to be excluded.

### CAPTCHA solving

As a general solution, the patient can easily solve one or more CAPTCHAs. Even if there is no guarantee that the patient himself will solve it (among all the mitigations, this is the easier to violate), it is a weak solution to discourage the quarantine violation.

## 6. MITIGATIONS FOR INTERNET CONNECTION ISSUES

It is possible to practice different measures to mitigate the problem of a poor quality of the Internet connection of the smartphone. As already anticipated, the police authority already carries out a check on the quality of the patient's smartphone connection but, if this connection, although present, is unsatisfying, it is possible to implement a mechanism that allows you to avoid false alerts to the police authority backend or a practice that allows to certify the patient's presence at home during the Internet blackout period.

Here are some opportunities to be implemented according to the expected cost of the system architecture:

- A superstructure can be created that allows to store the packages that the smartphone should send to the app backend directly within the SmartBeacon. In this way, once the app backend -which no longer received messages due to the smartphone Internet connection drop- has sent the Alert message to the police security backend and the police authority has arrived on the spot to verify the presence of the patient, it can benefit from a procedure that allows it to deduce that the patient is stayed at home during the Internet blackout period. In any case, the vulnerability due to the communication of the packet that contains weakly sensitive data through communication via Bluetooth is expected.

- A mechanism can be created that involves a greater economic effort: accessorize the SmartBeacon with an Internet connection. In this case, the package that the patient's smartphone was supposed to send through a standalone Internet connection may occasionally benefit from the emergency connection provided by the Beacon. This method solves the inconvenience of the unnecessary visit of the police authority provided by the solution in the previous point but does not solve the problem of the Bluetooth vulnerability.
- A further method which involves reducing the problem of Bluetooth vulnerability but which does not remedy the unnecessary visit by the police authority is to store the packets on the smartphone itself and allow, through a protected procedure, the authority to verify the presence of such packages.

## 7. MITIGATIONS FOR BLUETOOTH ADVERSARIES

An easy attack that Bluetooth communication could be subjected to would be a DoS attack. In fact, an opponent who emulates the Beacon could start sending packets at a very high frequency in order to fill the buffer that the smartphone maintains to store the messages it will have to forward to the app backend. A mitigation to this problem could be the rejection of packets received by the same device at too high a frequency. A hashing check (as already described in the footnote on page 26) could be added to this procedure to deal with a DoS attack in which the opponent performs a complete replay attack and cannot be distinguished from the original SmartBeacon.

## 8. MITIGATIONS FOR BLUETOOTH PACKET LOSS

A further mitigation is represented by the creation of a pyramid scheme for the definition of the sending frequencies and the expectation of receiving packets between SmartBeacon, smartphone and app backend.

This scheme requires that the SmartBeacon send its messages at a reasonably high but not excessive frequency (in order to distinguish daring opponents sending packets too quickly) such as a minute. The smartphone does not forward the received MESSAGE C directly to the app backend, including each one in the tuple ($SK_{DP3T\_T\_PAST}$, MESSAGE C), but rather stores them in a forwarding buffer. It proceeds to send the tuples at a lower frequency, e.g. five minutes. The app backend will then receive, in the best case every five minutes, a message containing multiple tuples to check. A tolerance on the delay of communication between smartphone and app backend, for example up to one minute, can amortize problems connecting the smartphone to the Internet.

In practice, in total every six minutes at most, the app backend receives multiple tuples with different MESSAGEs C, some from opponents and therefore rejected by validation, others from the SmartBeacon. It is sufficient that at least one of the MESSAGEs C obtained passes the verification test in order to consider the quarantine respected. In fact, a six-minute period (purely indicative, to be tightened by also scaling the other devices sending rates) could be considered an interval that makes the patient's departure inadmissible.

In this way, problems related to the loss of packets on a Bluetooth connection and indirectly also the effects of a connection of the Smartphone to the Internet of poor quality are mitigated.

# Source code and configuration files

In this chapter we explain the implementation of a representative simulation of the mechanism we showed in the previous pages. The project implementation is written in Python 3 and we used s

## 1. USED LIBRARIES

In order to implement the various functions used to encrypt the message and to communicate with the different servers we decide to use *Cryptography* and *SSL* libraries. Both are commonly used in this kind of situations. The former provides us functions which automatize the encryptions and the decryptions of a user message. The latter provides us the methods to establish a secure connection specifying a series of parameters as the cyphers used to encrypt the message or the protocol to use for the connection.

### A. *Cryptography*

*Cryptography* includes both high-level recipes and low-level interfaces to common cryptographic algorithms such as symmetric ciphers, message digests, and key derivation functions. [2]

It uses the *Fernet library* that guarantees that a message encrypted using it cannot be manipulated or read without the key. *Fernet* is an implementation of symmetric (also known as "secret key") authenticated cryptography.[3]

The encrypted message follows the Fernet Token format:

- *Version*, 8 bits. This field denotes which version of the format is being used by the token. Currently there is only one version defined, with the value 128 (0x80).
- *Timestamp*, 64 bits. This field is a 64-bit unsigned big-endian integer. It records the number of seconds elapsed between January 1, 1970 UTC and the time the token was created. Instead our ciphertext contains the current date with the current time, using the method *datetime.utcNow.*
- *IV*, 128 bits. It is the 128-bit Initialization Vector used in AES encryption and decryption of the Ciphertext. When generating new fernet tokens, the IV must be chosen uniquely for every token. With a high-quality source of entropy, random selection will do this with high probability.

[2] https://cryptography.io/en/latest/
[3] https://cryptography.io/en/latest/fernet/

- *Ciphertext*, variable length, multiple of 128 bits, the AES block size. It contains the original input message, padded and encrypted.
- *HMAC*, 256 bits SHA256 HMAC, under signing-key, of the concatenation of the fields

  `Version ‖ Timestamp ‖ IV ‖ Ciphertext`

  Note that the HMAC input is the entire rest of the token verbatim, and that this input is *not* base64url encoded.

**Encryption**

Given a key and message, the method generates a fernet token with the following steps, in order:

1. Record the current time for the timestamp field.
2. Choose a unique IV.
3. Construct the ciphertext:
   i. Pad the message to a multiple of 16 bytes (128 bits) per RFC 5652, section 6.3. This is the same padding technique used in PKCS #7 v1.5 and all versions of SSL/TLS (cf. RFC 5246, section 6.2.3.2 for TLS 1.2).
   ii. Encrypt the padded message using AES 128 in CBC mode with the chosen IV and user-supplied encryption-key.
4. Compute the HMAC field as described above using the user-supplied signing-key.
5. Concatenate all fields together in the format above.
6. base64url encode the entire token.

**Decryption**

Given a key and token, in order to verify that the token is valid and recover the original message, the method performs the following steps, in order:

1. base64url decode the token.
2. Ensure the first byte of the token is 0x80.
3. If the user has specified a maximum age (or "time-to-live") for the token, ensure the recorded timestamp is not too far in the past.
4. Recompute the HMAC from the other fields and the user-supplied signing-key.
5. Ensure the recomputed HMAC matches the HMAC field stored in the token, using a constant-time comparison function.
6. Decrypt the ciphertext field using AES 128 in CBC mode with the recorded IV and user-supplied encryption-key.
7. Unpad the decrypted plaintext, yielding the original message.

B. *SSL*

*SSL* library provides access to Transport Layer Security (often known as “Secure Sockets Layer”) encryption and peer authentication facilities for network sockets, both client-side and server-side. This module uses the OpenSSL library.

This library provides us a rapid and complete interface for our purpose, that is the capability of create a thread that is always listening to incoming connection and a function to send a message throw this connection.

It allows us to set a series of parameters:

- SSL Protocol.
- Load and verify of certificates server-side and client-side.
- Ciphers used for the encryption.

## 2. IMPLEMENTED CLASSES

The provided implementation follows the object-oriented programming paradigms, so we introduce the association between the scripts and the code of the classes. With the camel case format, we refer to the classes representing the actors involved in the scenario.

A. backend.py

**Backend class**

Backend is a base class, used to avoid code redundancy for the implementation of app backend and authority backend.
It includes several variables for SSL Connectivity and Certificate Management. In addition, each Backend stores its own Database on a proper JSON file.

**AppBackend class**

AppBackend is a derived class of Backend, it has all methods for Database management, Client-Server data exchange and ciphertext decryption.
Database management requires the entry **{id_phone: [id_beacon, key, datetime]}** addition and retrieving, so a control routine for expired datetime check.

Client-Server methods manage data coming from SmartPhone and HealthAuthority, and data sent to AuthorityBackend.
Finally, ciphertext decryption is performed with keys stored in database's entries.

**AuthorityBackend class**

AuthorityBackend also is a derived class of Backend, again it has all new methods for database management and Client-Server data acquisition, the only addition of this class is the JSON logfile, used to take memory of violation alerts coming from AppBackend.
Database management requires only the addition of a new entry **{id_beacon: info}**, where info may be a properly formatted string, its definition is left to authorities.
Server method manages data coming from AppBackend, which are stored in the logfile, taking memory of the alerts' datetime, the format is **{id_beacon: [datetime_0, datetime_1, …, datetime_n]}**.

B. client.py

**Client class**

Client is a base class, used to avoid code redundancy for HealthAuthority and SmartPhone. It includes few methods for Certificate Acquisition and Client Session.
The Client Session is already defined for all derived clients, basically it sends a properly formatted string to AppBackend. String format changes according to the specific client.

**HealthAuthority class**

HealthAuthority is a derived class of Client, with just a single method.
The routine method simply calls the base class client session, after a proper string formatting.

**SmartPhone class**

SmartPhone is a derived class of Client, with several variables and methods for Status Management and Data Sending/Memorization Routine.
Status Management methods and variables, serve to determine the smartphone's random behavior:

- Device is on/off.
- Bluetooth is on/off.
- Wi-Fi is on/off.

Data Routine changes according to smartphone's Wi-Fi, in online mode Bluetooth packets are simply forwarded to AppBackend with a client session and a proper string format, alternatively, in offline mode packets are registered into a logfile, and the last packet received is immediately sent to AppBackend once Wi-Fi connection is restored.

C. beacon.py

**SmartBeacon class**

SmartBeacon class simply encrypts datetimes for a specific smartphone (only in the code). Even if in the real case a single beacon may encrypt data for multiple devices, to keep it simple in the code, a single beacon may have multiple instances with multiple headers, in order to communicate with different devices.
SmartBeacon's Data Routine may change according to its on/off status, clearly when turned off, nothing happens.

D. main.py

**Demo class**

Demo class serves as a simulation of the project behavior.
Class methods provide initialization, with beacon and smartphone instances creation and registration (or not), periodic data routine, where each pair (Beacon, Smartphone) runs its own behavior (according to its own status), and finally the control routine, where periodically the AppBackend instance, runs a datetime check

on each entry of its database, and eventually alerts AuthorityBackend instance about violations.

## 3. SIMULATED ENVIRONMENT

In the proposed simulation, the user may follow the interaction between an arbitrary number of devices, such as beacons or smartphones, and three central entities, AppBackend, AuthorityBackend and HealthAuthority. It supports only one instance per backend, so basically, we will have just one pair (AppBackend, AuthorityBackend), while a single instance of HealthAuthority is sufficient (eventually an arbitrary number may be used).

This environment runs with 3 different phones, each of them has its own $SK_{DP3T\_T\_PAST}$, while only 2 beacons with 2 different keys, header and info are used, to show how it works when two phones shares a beacon.

Devices initialization is performed just once, and it may be performed periodically, according to some changes in the code. Eventually the user may decide to register or not each single beacon or smartphone on the proper database, to look at the behavior of both backends in such scenery.

Server sessions run on different loop-threads, listening for new connections from clients. Communications are managed through fixed header strings for each client communicating to a server.

Data and control routine run on two different timer-threads. About data routine, the behavior of each device may change according to its own status, so it is possible to see some devices turn off, being offline or out of Bluetooth range.

Timing of different interactions has been selected in order to give a reasonable output during the demo, in practice, of course, things would be different. Anyway, the idea is to give the patient a time period in which send at least one ciphertext to AppBackend before any control routine. During the simulation, each beacon encrypts a plaintext every 3 seconds, which has an expire datetime of 15 seconds. Even the control routine runs every 15 seconds, so basically at least 1/5 of the packets must arrive to AppBackend in order to avoid a violation alert.

The only visible outputs are incoming tokens for AppBackend and violation alerts for AuthorityBackend, while relevant actions performed during the execution are registered in the execution log, stored in the root directory of the project. Other relevant data are stored in the sub directory of their own class.

## 4. HOW TO RUN

The entire project consists of 5 different Python 3.7 scripts:

- backend.py
- client.py
- beacon.py
- utils.py
- main.py

All required libraries are already installed in the project interpreter (a virtual environment is provided in the directory *venv*), so a first way to run the code is to import the root directory of the project in your IDE and run main.py.

If this fails, from command window it is necessary to install just two libraries, *numpy* and *cryptography*. Again, it only needs to run main.py.

Project is scripted to store data in the same directory of main.py script, check there for all logs and other files.